

Marko Palviainen, Timo Laakko & Juha Kolari

Visual WML – a development tool for WAP applications



Visual WML

– a development tool for WAP applications

Marko Palviainen, Timo Laakko and Juha Kolari

VTT Information Technology



ISBN 951-38-5807-3 (soft back ed.)

ISSN 1235-0605 (soft back ed.)

ISBN 951-38-5808-1 (URL: <http://www.inf.vtt.fi/pdf/>)

ISSN 1455-0865 (URL: <http://www.inf.vtt.fi/pdf/>)

Copyright © Valtion teknillinen tutkimuskeskus (VTT) 2000

JULKAISIJA – UTGIVARE – PUBLISHER

Valtion teknillinen tutkimuskeskus (VTT), Vuorimiehentie 5, PL 2000, 02044 VTT
puh. vaihde (09) 4561, faksi (09) 456 4374

Statens tekniska forskningscentral (VTT), Bergsmansvägen 5, PB 2000, 02044 VTT
tel. växel (09) 4561, fax (09) 456 4374

Technical Research Centre of Finland (VTT), Vuorimiehentie 5, P.O.Box 2000, FIN-02044 VTT, Finland
phone internat. + 358 9 4561, fax + 358 9 456 4374

VTT Tietotekniikka, Käyttäjäkeskeinen tietotekniikka, Sinitaival 6, PL 1206, 33101 TAMPERE
puh. vaihde (03) 316 3111, faksi (03) 317 4102

VTT Informationsteknik, Användarorienterad kommunikationsteknologi,
Sinitaival 6, PB 1206, 33101 TAMMERFORS
tel. växel (03) 316 3111, fax (03) 317 4102

VTT Information Technology, Human Interaction Technology,
Sinitaival 6, P.O.Box 1206, FIN-33101 TAMPERE, Finland
phone internat. + 358 3 316 3111, fax + 358 3 317 4102

VTT Tietotekniikka, Palveluverkot, Tekniikantie 4 B, PL 1203, 02044 VTT
puh. vaihde (09) 4561, faksi (09) 456 7028

VTT Informationsteknik, Internet, Teknikvägen 4 B, PB 1203, 02044 VTT
tel. växel (09) 4561, fax (09) 456 7028

VTT Information Technology, Networks, Tekniikantie 4 B, P.O.Box 1203, FIN-02044 VTT, Finland
phone internat. + 358 9 4561, fax + 358 9 456 7028

Technical editing Maini Manninen

Otamedia Oy, Espoo 2001

Palviainen, Marko, Laakko, Timo & Kolari, Juha. Visual WML – a development tool for WAP applications. Espoo 2001. Technical Research Centre of Finland, VTT Tiedotteita – Meddelanden – Research Notes 2068. 55 p.

Keywords Wireless Markup Language, XML, Extensive Markup Language, Wireless Application Protocol, WAP application development, mobile internet, Web services, usability

Abstract

The Visual WML system is a result of the WML-Browser project (March 1999 – April 2000) of VTT Information Technology.

The mobile Internet is said to provide the users location and time independent data services – usage wherever and whenever. As the specifications are becoming more stable in the near future, the diversity of the terminals will increase to fulfil the different user needs. The increasing terminal diversity and the increasing population of the mobile users will add the demand for more and more personal usage profiles. Personal multimedia will be the key application area, and it also poses several challenges to the usability of services and applications. Ideally, the future solutions are based on open standards.

The Wireless Application Protocol (WAP) architecture provides an extensible environment for application development for mobile communication devices.

With the advent of WAP technology, Internet services in a large extent are coming within the reach of mobile users. At the same time, the variety of different kinds of mobile terminals and networks are increasing. For instance, the forthcoming mobile communication channels (GPRS, UMTS) will increase the transmission capacity, and wireless local area networks (Wireless LAN) will offer fast and easy access to wired internet, corporate and home environment LANs and their services. Hence, in particular, tools for WAP application development are required.

Visual WML is a tool targeted for editing, creating and browsing WML (Wireless Markup Language) documents. It is also possible to emulate different kinds of WAP devices by utilizing User Agent Profiles.

Preface

This work was carried out in VTT Information Technology within the "WML Browser" project between March 1999 and April 2000. The main goal of the project was to build a software that is capable of browsing WML (Wireless Markup Language) and emulating different WAP (Wireless Application Protocol) terminals, and allows the WML developer to build and edit WML documents by using visual components. Also, the project group decided to use the existing technologies and well-defined specifications and standards as much as possible. The developed software was named Visual WML.

WAP architecture enables the interconnection of the wireless data network and the wired data network. WAP technology brings the Internet content and advanced data services to digital cellular phones and other wireless devices. For integration, WAP specifies a number of specifications that define the layers of the WAP protocol stack. The highest layer of the WAP protocol stack is the WAP application environment (WAE) that specifies different content formats and services for user agents, e.g. WAP phones, to implement.

The most important content format of WAE is WML that is used for content browsing. WML is an XML (Extensible Markup Language) application. XML provides an easy and flexible way to describe and deliver a structured information over the WWW (World Wide Web).

Contents

Abstract.....	3
Preface.....	4
List of symbols.....	7
1. Introduction.....	9
1.1 Background.....	9
1.2 Wireless Application Protocol (WAP).....	9
1.3 VTT WAP.....	10
1.4 Standards and Tools used.....	11
1.5 Programming and hardware environment.....	11
1.6 System requirements.....	12
2. Visual WML components and architecture.....	13
2.1 Architecture.....	14
2.2 Components.....	14
3. Parsing and editing of an XML document.....	16
3.1 XML.....	16
3.1.1 Applying XML.....	16
3.1.2 Document Type Definition.....	17
3.1.3 Wireless Markup Language (WML).....	18
3.2 Document Object model.....	19
3.3 SAX.....	20
3.4 XML parsing.....	20
4. WML Editor.....	21
4.1 Tree view.....	21
4.1.1 Implementation aspects.....	22
4.2 Card view.....	24
4.2.1 Implementation.....	25
4.3 Code view.....	27
4.3.1 Implementation aspects.....	27
4.4 Help Tool.....	27
4.4.1 Implementation.....	28
5. Phone Editor.....	30
5.1.1 Implementation aspects.....	32

6.	WML browser – phone simulator	33
6.1.1	Technical implementation	34
7.	User Agent Profile.....	36
7.1	Introduction.....	36
7.2	The architecture	37
7.3	UAProf schema and base vocabulary	37
7.4	RDF.....	38
8.	Examples	39
8.1	Building a new WML deck.....	39
8.2	Adding elements	40
8.3	Attribute editing	42
8.4	Adding actions	42
8.5	Saving	43
9.	User requirements and design process	45
9.1	Methods	45
9.1.1	Design process.....	45
10.	Conclusions	48
10.1	Overall evaluation results	48
10.2	Technical results	48
10.3	Future work.....	49
11.	Summary	51
	Acknowledgements.....	52
	References.....	53

List of symbols

DOM	Document Object Model
DTD	Document Type Definition
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol [RFC2068]
UAPROF	User Agent Profile [UAPROF]
UI	User Interface
W3C	World Wide Web Consortium
WAE	Wireless Application Environment [WAE0]
WAP	Wireless Application Protocol
WDP	Wireless Datagram Protocol [WDP]
WML	Wireless Markup Language [WML]
WSP	Wireless Session Protocol [WSP]
WTP	Wireless Transaction Protocol [WTP]
WTA	Wireless Telephony Application [WTA]
WTLS	Wireless Transport Layer Security [WTLS]
XML	Extensible Markup Language [XML]

1. Introduction

The Visual WML system to be described in this report is an application development tool for editing, creating, and browsing WML documents. It is also possible to emulate a variety of devices by utilizing User Agent Profiles.

1.1 Background

The Visual WML system is the result of the "WML-Browser" project (March 1999 – April 2000) of VTT Information Technology.

The project's major aim was to develop a WML development tool for browsing and editing WML documents. The initial goals and requirements included the following:

- to develop a WML browser tool
- to support the editing of WML documents
- to enable easy transforming of HTML documents to WML documents
- to simulate different kinds of mobile terminals based on the user agent profile information.

The developed WML tool was later named "Visual WML".

1.2 Wireless Application Protocol (WAP)

The Wireless Application Protocol (WAP) architecture provides an extensible environment for application development for mobile communication devices [WAPA, WAEO].

With the advent of WAP technology, Internet services in a large extend are coming within the reach of mobile users. Hence, in particular, tools for WAP application development are required.

1.3 VTT WAP

One of the initial aims of the WML Browser project was to support other research projects of VTT (cf. <http://www.vtt.fi/tte/projects/WAP/>). Figure 1.1 illustrates the current WAP research and development system architecture.

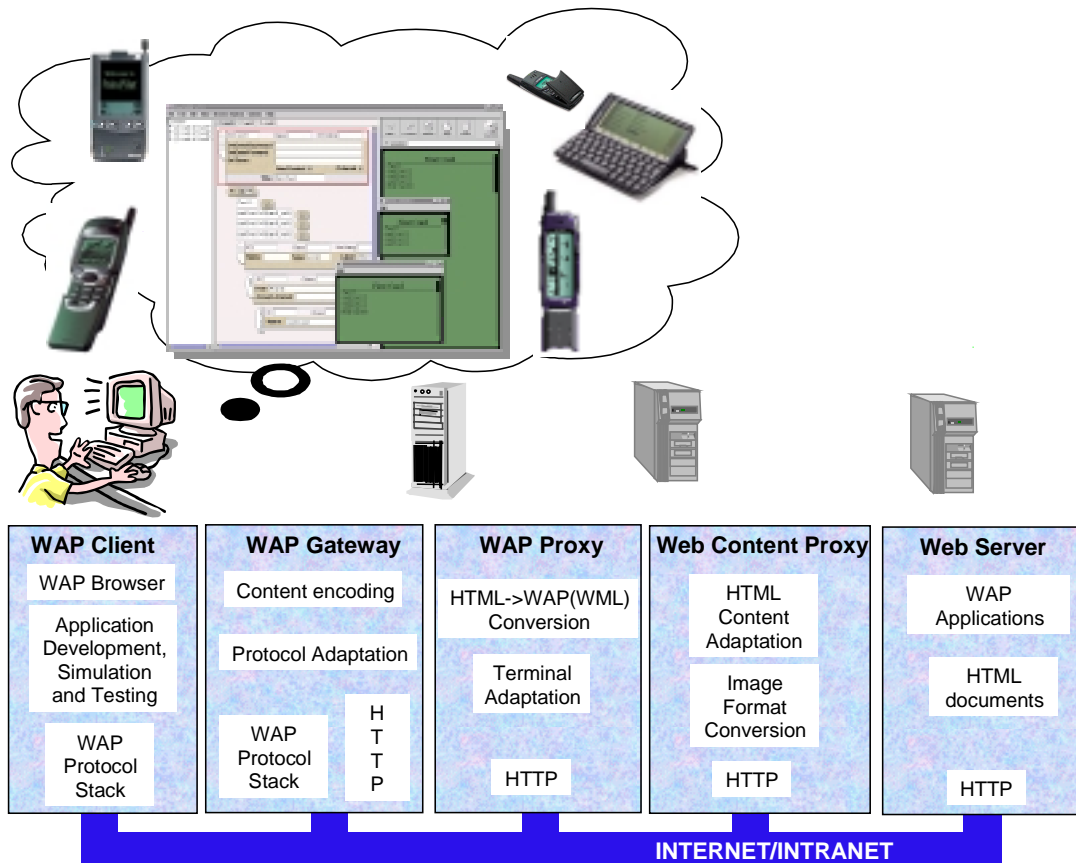


Figure 1.1. WAP research and development system architecture.

VTT Information Technology has developed a WAP research and development system which enables experimentation with new multimedia technologies and ideas, and their implementations in the mobile user's Internet environment before the method is standardised and incorporated into commercial products. The system covers the entire chain from content servers through presentation format and data transfer protocol modifications to the user's browser and terminal [WWW00, HCI99, LEP99]. It includes the key components needed for the development and testing of new services as well as a development tool for application developers.

The results of the WML browser project includes some major components of the WAP client side shown in the figure.

The first WAP terminals came onto the market in 1999, and in future the range of mobile devices and users is likely to expand rapidly. However, this is dependent on the Internet implementation being made sufficiently user-friendly and useful from the perspective of both the service provider and the mobile user. Usability research is an essential aspect of VTT's WAP development work [WWW00] as also in this WML-Browser project.

The modular structure of VTT Information Technology's WAP architecture allows its modification as the WAP specifications are continuously developed. This also facilitates the transfer and timely exploitation of components in the changing environments of partners according to their individual needs.

1.4 Standards and Tools used

The principal idea of the project was to use the existing technologies, and well-defined specifications and standards as much as possible.

The WML version 1.1 (approved in June 1999) represents the culmination of WAP Forum members' efforts to resolve technical issues and ambiguities in WAP Version 1.0 specifications. However, the version 1.1 is not backward compatible with the older version 1.0 (April 1998). The developed Visual WML system supports the WML version 1.1. A freely available validating XML parser [XML4J] for Java including standard DOM interfaces for accessing the document has been used.

1.5 Programming and hardware environment

The JDK 1.2.2, Java development kit from Sun Microsystems, was used on the Windows NT 4.0 platform to build the application. The user interface of the program was implemented with Java Swing components. Visual WML is runnable anywhere the Java virtual Machine, version 1.2.2, is installed.

The development work has made in Windows NT, but the system is also portable to other operating systems. During the project the program was tested, successfully, on the Linux platform (Red Hat 6.1).

1.6 System requirements

The following are the minimum requirements for a computer system to run VisualWML:

- Windows NT 4.0, Windows 98 or Linux
- 200 MHz Pentium or faster processor.
- 64 MB of RAM or more.
- Display resolution 1024x768 or higher
- Java™ Runtime Environment 1.2.2 or later must be installed.

You need an Internet connection in order to test services available on the Internet. To read HTML documents into VisualWML, you need an access to separate HTML to WML conversion proxy server.

2. Visual WML components and architecture

Because the WML syntax (look at Chapter 3.1) is somewhat similar to HTML, it is possible to create WML decks also with a standard text editor. But Visual WML makes it much easier because of its visual user interface for defining elements and attributes, which at the same time can be seen visualized in WML-browsers. A WML deck can be seen in three different views (Figure 2.1). On the left a *tree view*, in the center a *card view* and on the right the *browser view* of the deck is visualized.

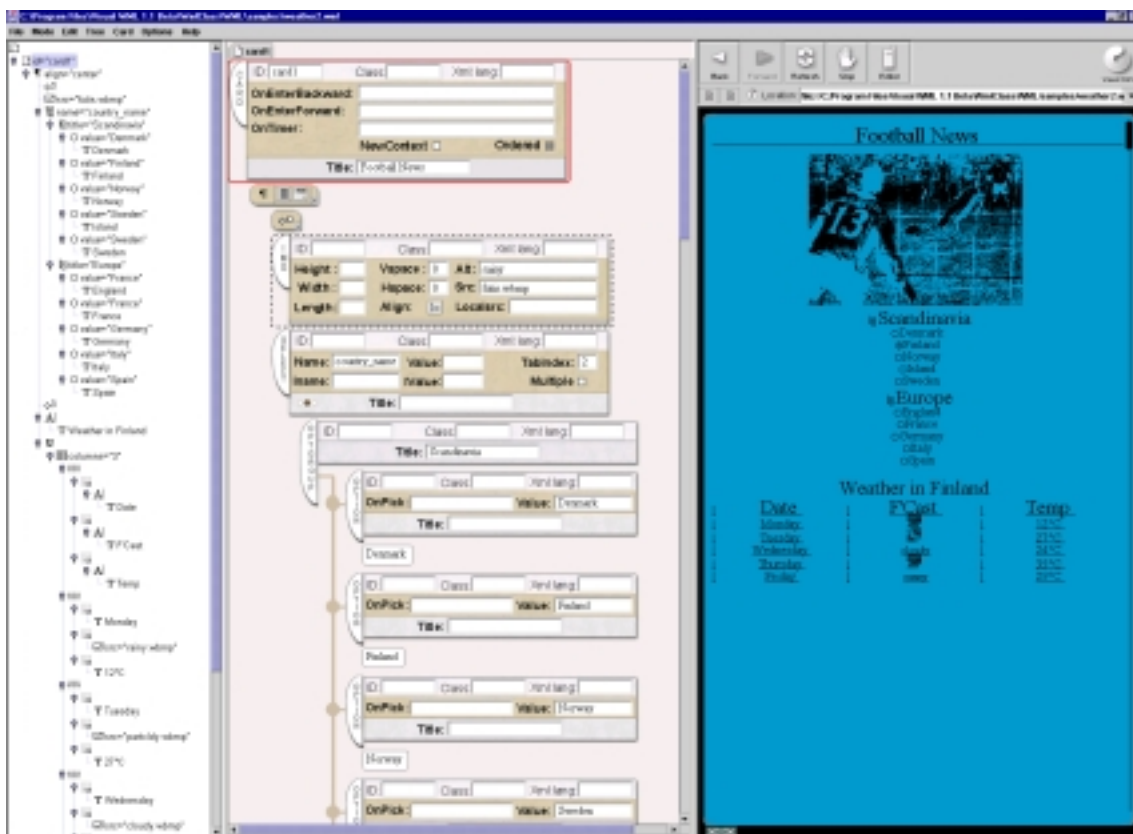


Figure 2.1. WML-Browser main window.

Visual WML ensures that the edited deck syntax is valid at all times. WML-deck can be loaded from local disc or from the web. It is also possible to import HTML pages, because system uses VTT WAP Proxy to transform HTML to WML.

After the deck is imported, it can be edited and seen visualized in a phone simulator in real time. In Visual WML it is also possible to define new phone models and simulate decks in several different simulators at the same time. In the simulator views it is possible to test deck's functionality too.

2.1 Architecture

Figure 2.2 shows the Visual WML architecture. On the main level Visual WML has three components, which are parser, editor and browser. The parser part (chapter 3) takes care of WML document importing (and exporting too), the editor part (chapter 4) handles WML-document editing and the browser part simulates (chapter 6) and visualizes the current deck.

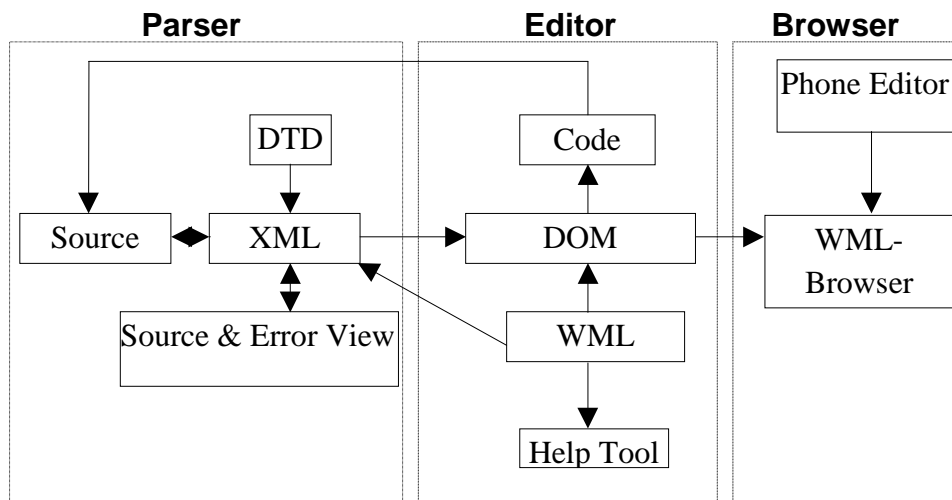


Figure 2.2. Visual WML architecture.

2.2 Components

Visual WML has been implemented with Java, and its components (classes) have been divided into packages that are related to the Visual WML architecture (cf. Figure 2.2) as shown in Figure 2.3. WML-package takes care of controlling all the other packages in the parser, editor, and browser parts. More detailed descriptions of these packages are in chapters 3, 4, and 6, respectively.

Parser	Editor	Browser
	WML	
Parser	TreeTool	PhoneEditor
	CardTool	PhoneSimulator
	CodeView	WMLBrowser
	Help	
	Menus	

Figure 2.3. Visual WML packages.

3. Parsing and editing of an XML document

3.1 XML

The Extensible Markup Language (XML) [XML, LEV98] is the format for structured documents and data on the Web. Also, XML is a simplified subset of the Standard Generalized Markup Language [SGML].

XML uses markup tags (<...>), but unlike HTML, XML tags describe the content, rather than the presentation of that content. HTML tag names are limited to a predefined set and are primarily used to indicate how the enclosed text data is to be displayed. The set of XML tag names are used to indicate what the enclosed data means. For example tags like , <big> and <i> define the style for the text in HTML, they don't tell much about what kind of information they include.

In XML style sheets define the style of the document. With the extensible style sheet language (XSL) [XSL] is possible to attach style e.g. fonts, spacing and font styles to XML, and it is even possible to choose which elements are visualized.

XPointer [Xpointer] and XLink [Xlink] define a standard way to represent links between resources. In addition to simple links, like HTML's <a> tag, XML has mechanisms for links between multiple resources and links between read-only resources. XPointer describes how to address a resource, XLink describes how to associate two or more resources.

3.1.1 Applying XML

Almost all information we deal with every day can be represented as structured. When things are well organized, it is easy to find the information you are looking for. Let's look at the following example (see Figure 3.1) of a simple product catalog which contains a few products grouped into categories. Each product can be described with certain attributes. It can be seen that the catalog has a structure which can be easily represented in XML. With an appropriate application it is now it is easy to get or update information about the products.

<p><i>Product catalog</i></p> <p>Monitors : Sony Multiscan (\$500) Nokia Multigraph (\$550) ...</p> <p>Keyboards: Keytronic Wintronic (\$50) ...</p>
<pre> <?xml version="1.0"?> <catalog> <!-- This is the root element --> <category id="monitors"> <product> <id>12234</id> <model>Nokia Multigraph 477x</model> <price>\$500</price> </product> <product> <id>123456</id> <model>Sony Multiscan</model> <price>\$550</price> </product> </category> <category id="keyboards"> <product> <id>134</id> <model>Keytronic wintronic</model> <price>\$50</price> </product> </category> </catalog> </pre>

Figure 3.1. A little product catalog coded in XML.

3.1.2 Document Type Definition

DTD [DTD] (Document Type Definition) describes the legal structure of an XML document. It is optional, but its use is recommended, especially if XML documents are used in an application that reads documents from several content editors.

```

<!-- Typical usage:
<?xml version="1.0"?>
<!DOCTYPE catalog SYSTEM "catalog.dtd">
<catalog>    ...    </catalog>
-->

<!ELEMENT catalog (category)+>
<!ELEMENT category (product)*>
<!ATTLIST category id CDATA #REQUIRED>

<!ELEMENT product (id, model, price)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT model (#PCDATA)>
<!ELEMENT price (#PCDATA)>

```

Figure 3.2. The DTD for the catalog document shown in Figure 3.1.

For example (see the above figure), DTD declares which elements can be the children of an element and which attributes an element can have.

For defining the structure of an XML document, it is necessary to use symbols in DTD. For example *(category)+* means that the *catalog* element must have one or more *category* elements and *(product)** means that the *category* element can be empty or have one or more *product* elements. The *product* element must have three children in the following order: *id*, *model* and *price*.

Attributes are also defined in DTD. For instance, the *category* element has an attribute named *id*, whose type is CDATA. The REQUIRED keyword means that the attribute must always be used. Note that it could be possible to leave out the *id* attribute and put it as a child element of a *category* element, in the same way as the product element's *id* attribute.

3.1.3 Wireless Markup Language (WML)

Wireless Markup Language (WML) [WML] is an application of XML. Hence, all the rules explained in the earlier sections about XML hold true also in WML. Let's look at the following example of Figure 3.3:

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
    "http://www.wapforum.org/DTD/wml12.dtd">
<wml>
  <card ordered="true" newcontext="false">
    <p align="left">
      <b>This is an example <br/>
      </b>
      <a href="www.wapforum.org/AnotherSample.wml">
        An another sample.
      </a>
    </p>
  </card>
</wml>

```

Figure 3.3. A sample WML document.

The code is somewhat similar to HTML, but is not as complex. In particular, WML is optimized for mobile usage and terminals, and has additional features, for instance, to support the navigation. All the elements are declared in DTD (wml12.dtd). The result in a user's device should look like the following (Figure 3.4).

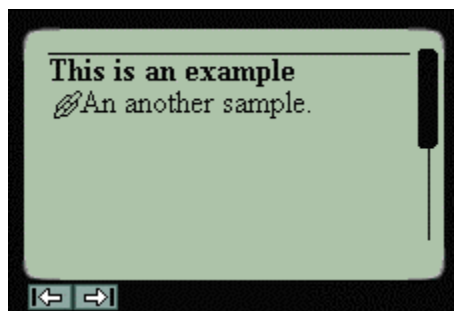


Figure 3.4. Visual presentation of the above example (Figure 3.3) by Visual WML.

3.2 Document Object model

The DOM (Document Object Model) [DOM] is a programming API for XML (and HTML) documents. It defines the logical structure of documents and the way to access and manipulate a document. The World Wide Web Consortium (W3C) has defined the DOM specifications in OMG IDL (Object Management Group, Interface Definition Language) [OMGI].

DOM enables programmers to create and build documents, navigate their structure, and add, modify and delete elements and content in a standard way. One important objective for the Document Object Model is to provide a standard programming interface that can be used in a wide variety of environments and applications. The DOM can be used with any programming language.

The DOM specification has currently two levels. The level 1 specification defines basic interfaces in the Core specification and HTML features in the HTML part. Level 2 defines interfaces for CSS, events, filters, iterations and range. The level 2 proposal is still waiting for its acceptance as a W3C recommendation.

An XML-DOM parser must implement the DOM interfaces in order to provide a standard way for clients to access XML documents (see Section 3.4). When an XML parser reads an XML document, it builds a DOM tree in the memory. The DOM tree consists of a root node, nodes and leaf nodes. These nodes can be manipulated with the DOM interfaces that the XML-DOM parser implements.

The Node interface is the primary data type for the entire Document Object Model. It represents a single node in the document tree. Almost all the other interfaces extend the Node interface and inherit its methods e.g. Element, Document and Attr interfaces, but not all methods are appropriate in inheriting interfaces and they can return null pointers or cause exceptions.

3.3 SAX

SAX [SAX], the Simple API for XML, is a standard interface for event-based XML parsing, developed collaboratively by the members of the XML-DEV mailing list. SAX allows application writers to write applications that use XML parsers, but are independent of the parser used.

The SAX mechanism processes XML data in the form of a text stream and alerts the application whenever something interesting is encountered. An event-based API reports parsing events (such as the start and the end of elements) directly to the application through callbacks.

3.4 XML parsing

A software module that reads XML documents is called an XML processor or, more commonly, an XML parser. The DOM-based XML processor is used to read XML documents and to provide access to their content and structure by implementing DOM interfaces. For high-speed processing of the XML, the SAX parser is appropriate e.g. with SAX events you can do some filtering for documents.

There are a great variety of parsers in the market for different languages and for different document types. When choosing a parser, it is necessary to do some planning. When doing software, which is very time critical, a parser for C language should be chosen.

However, Java itself provides an easy programming environment and the portability for an application. For instance, Java provides an URL class loader mechanism that enables downloading of Java code and dynamic object creation.

The IBM's XML4J [XML4J] is an XML parser for Java that was used in the project, because it provides many advanced features. It is important that the parser conforms to the XML 1.0 Recommendation and associated standards and specifications (DOM 1.0, SAX 1.0, and Namespaces). The parser version discussed in this report is the XML4J 2.0.11 [XML4J].

4. WML Editor

4.1 Tree view

The *tree view* of the editor shows WML elements as a tree structure (Figure 4.1). In the *tree view* it is possible to add and remove WML-elements. It is also possible to use edit commands (cut, copy and paste) to manipulate the tree. The view is synchronized with the other editor views - the *card view* and *code view* (see Section 4.2 and 4.3) and browser windows. So, all the changes can be seen in real time in other windows.

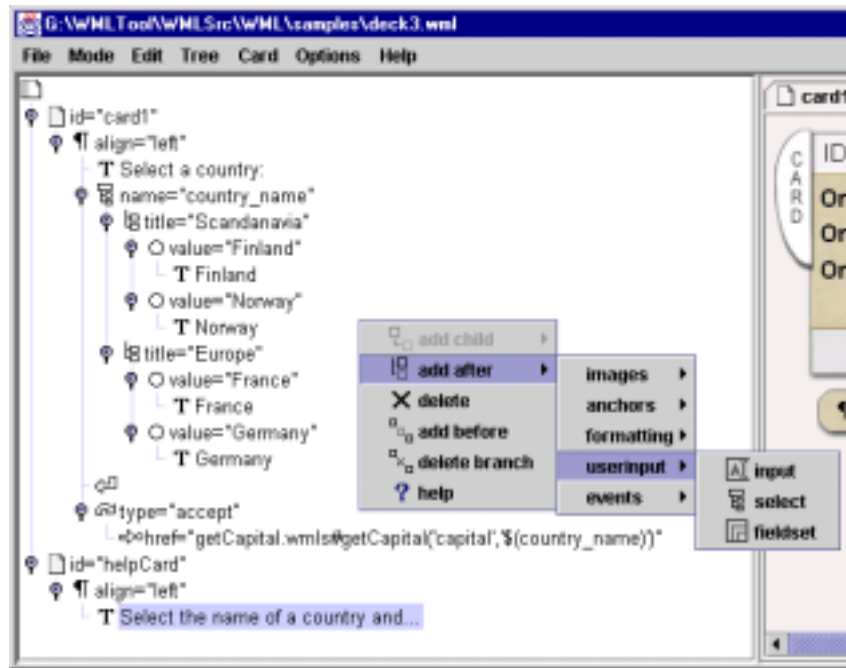


Figure 4.1. Adding WML element in tree view.

Visual WML takes care of WML document validity so that it is not possible to do edit operations that are against DTD. For example, if you want to add a new WML-element Visual WML shows which elements it is possible to a particular place.

Also, Visual WML forces the user to define implied attributes and childrens to the added element. If the user adds a text element in the *tree view* the text string can be edited in a special text editor. The text editor has normal edit operations and it is also possible to load text files from disc to the text element.

4.1.1 Implementation aspects

JavaSwing.JTree class does the implementation of the *tree view* of an XML document. With the JTree, you can display hierarchical data such as XML documents. A JTree object doesn't actually contain data, it simply provides a view of the data. The data can be stored in the DefaultTreeModel class, which you can dynamically edit and that data can the JTree use for displaying. The TreeCellRend class is used for customizing a JTree's display, e.g. you can add your own icon's in the tree leafs.

Look at the figure 4.2 that represents a simple UML class model of the JTree related classes. As you can see the JTree class has methods such as *collapse()* and *expand()*, these methods has only to do with the visual view of the data.

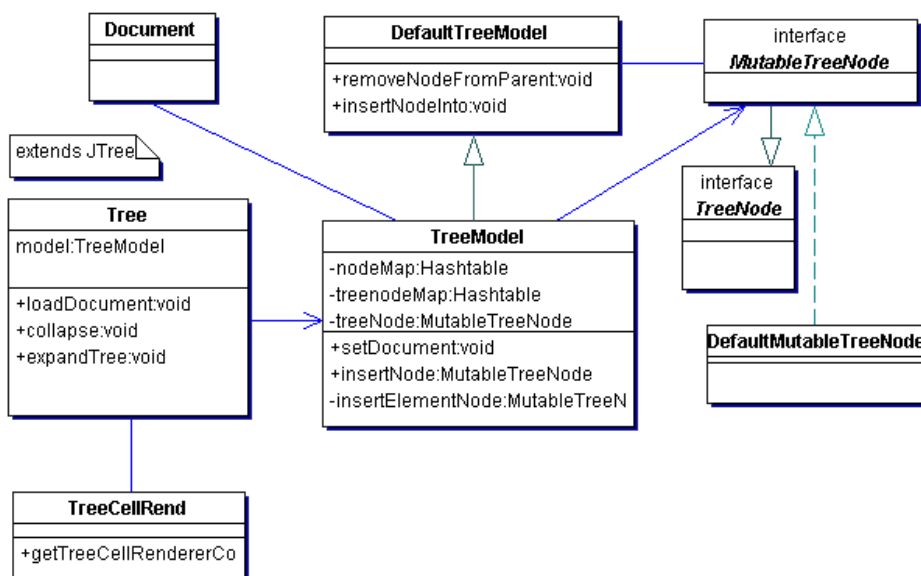


Figure 4.2. The class hierarchy of the Tree View implementation.

The *loadDocument(Document document)* method starts the building process by calling the *setDocument* method in the TreeModel class. The org.w3c.Document is gone through node by node in the TreeModel class while building the DefaultTreeModel structure.

The TreeModel class constructs a tree data structure by using the MutableTreeNode interface, which is very similar to the org.w3c.Node. The TreeNode represents a single node in a *tree view* and defines the requirements for an object that can be used as a tree node in a JTree. The subclass MutableTreeNode defines the requirements for a tree node object that can change by adding or removing child nodes, or by changing the contents of a user object stored in the node.

The *MutableTreeNode insertElementNode(Node newNode, MutableTreeNode where, int w)* method (see Figure 4.3) is called every time when a new element is encountered. Before this method the root element must be inserted in the *TreeModel* and after that the root element is passed to this method, e.g. *insertElementNode(Node theFirstChild, MutableTreeNode theRoot, int childCountofTheRootElement)*.

All elements are gone through in this method and every element is passed in the *insertNode(String what, MutableTreeNode where, int index)* method where a new node (*MutableTreeNode*) is made. The *insertNodeInto(node, where, index)* method inserts a new node in the *TreeModel*. The text nodes also use this method, but before they must be handled in the *insertTextNode* method.

The *nodeMap* and the *treenodeMap* (*HashTable*) are used for making the relationship between the *MutableTreeNode* and *org.w3c.Node*. For example if you like to know which node in the *JTree* corresponds to the *org.w3c.Node*, use the *nodeMap*'s *get(Object obj)* method as the following *Node node = (Node)nodeMap.get(treeNode)*. The *treenodeMap* is used for getting the corresponding *JTree* node. This way it is easy to edit simultaneously the data the *TreeModel* and the *org.w3c.Document* holds.

```

MutableTreeNode insertNode(String what,MutableTreeNode where)
{
    MutableTreeNode node = new DefaultMutableTreeNode(what);
    insertNodeInto(node, where, index);
    return node;
}

MutableTreeNode insertElementNode(Node newNode, MutableTreeNode where)
{
    int w = where.getChildCount();
    String name = Makename(newNode);

    MutableTreeNode element = insertNode(name, where, w);
    nodeMap.put(element, newNode);
    treenodeMap.put(newNode, element);

    NodeList children = newNode.getChildNodes();
    int len = (children != null)?children.getLength():0;

    for (int i = 0; i < len; i++)
    {
        Node node = children.item(i);
        switch (node.getNodeType())
        {
            case Node.TEXT_NODE:
                insertTextNode(node, element,
                    element.getChildCount());
                break;

            case Node.ELEMENT_NODE:
                insertElementNode(node, element);
                break;
        }
    }
    return element;
}

```

Figure 4.3. A piece of code of the *TreeModel*'s building process.

4.2 Card view

The *card view* visualizes WML elements and attributes. In the *card view* the deck is visualized as a tapped panel, which has one card in every panel (Figure 4.4). In the *card view* it is possible to do all the same operations (add, remove, cut, copy elements and edit text elements in text editor) which are possible in *tree view*. The *card view* differs from the *tree view* because in the *card view* the values of element attributes can be changed too.

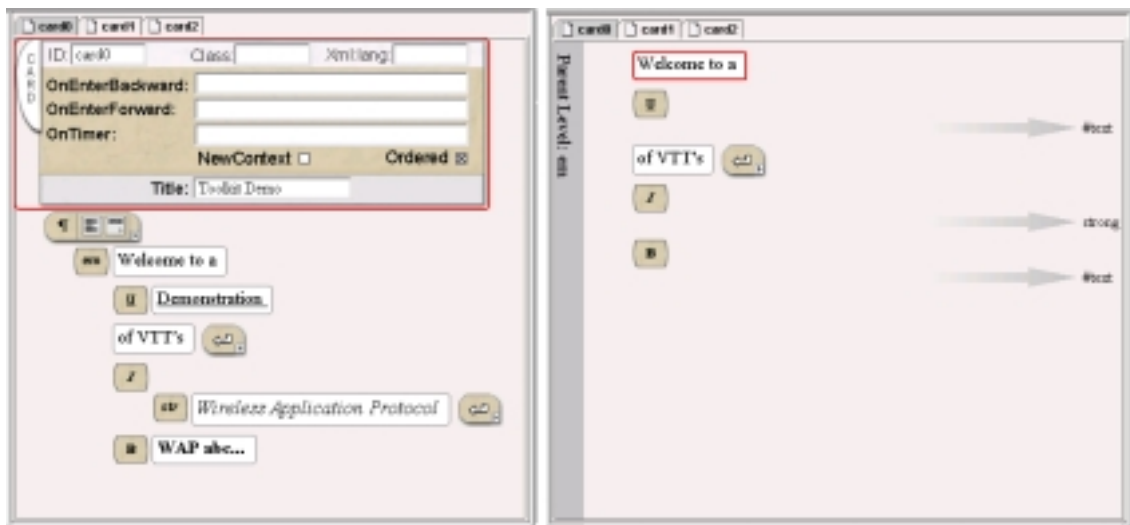


Figure 4.4. Card view. On the left show mode is all elements and on the right show mode is one level.

The *card view* has three different view modes. Modes are all elements, one level and one element. So it is possible to see all elements, one level in hierarchy or one element on the *card view* at same time. This is an important feature especially if the deck is very long. In Figure 4.4 on the right card view's show mode is one level. On the left is shown parent level and arrows show child level elements. By clicking these it is possible to move lower or higher level in hierarchy.

4.2.1 Implementation

Figure 4.6 shows the *card view* architecture. CardPanel is a JTabbedPane, which contains cards. Card is one card (JPanel) in CardPanel and it contains current card WML elements and drawing and control methods for elements. It inherits class named CardFunctionality, which contains control methods for attributes editing and methods

for handling different show modes. Card contains WMLElements which is a pointer to all card WML elements. Object WMLElement is a unit which contains a single DOM element (is WML element or text). It has coordinate information and methods for coordinates counting for WMLElements.

It uses classes in packages (Anchors_images_timers, Decks and cards, Text_formatting, Tasks, Events, User_input and Variables) for drawing and handling different kind of WML elements attribute coordinates. For example CardBase (Figure 4.5) contains image of current WMLElement, coordinates of attributes and methods to get attributes basis on coordinates. Each class in these packages has same structures as in this CardBase example. CardBase inherits methods from class ElementBase, which has methods for attribute and text drawing and methods for handling coordinates.

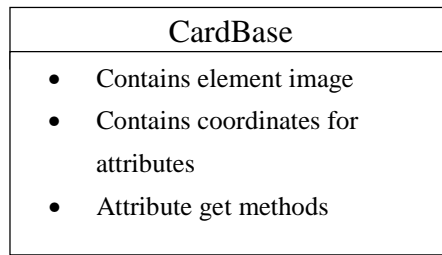


Figure 4.5. One element (card) base in package Decks and cards.

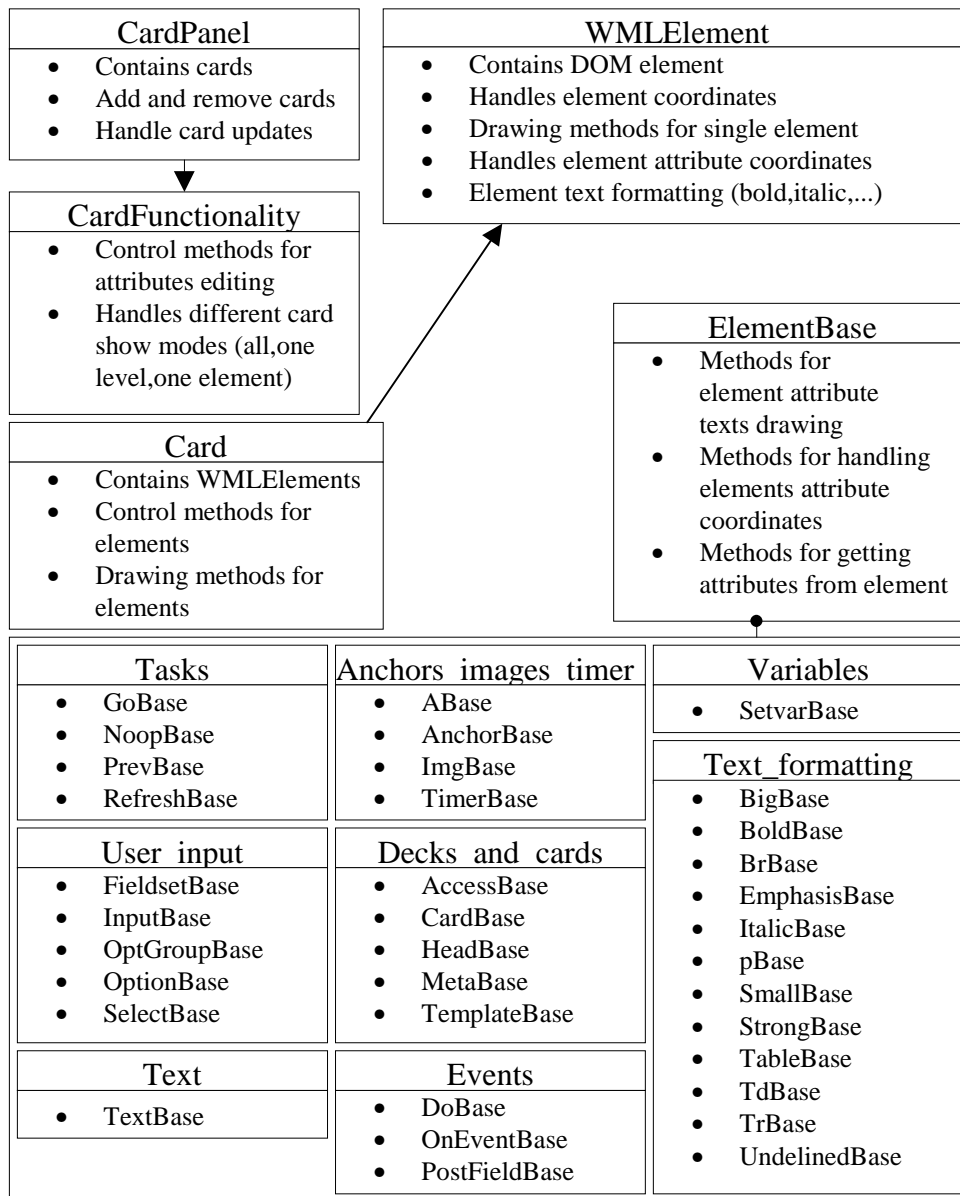


Figure 4.6. Technical implementation of card view.

4.3 Code view

In the *code view* (Figure 4.7) it is possible to edit WML-code as in normal text editor. After changes are made it is possible to save and recompile the deck. If there were errors in the code, *code view* shows in which rows in the WML-code the errors are.



Figure 4.7. Code view.

4.3.1 Implementation aspects

Code view is implemented (in class ViewWMLCodeFrame) using java classes JFrame, JTextArea and JList. JFrame contains both code and error part. Code part is JTextArea and error part is JList, which contains errors. When error is clicked Listlistener is activated and right place will be scrolled in JTextArea. Both the code and error come from parser.

4.4 Help Tool

The Visual WML system has XML-based help pages. With the *help tool* (Figure 4.8), it is possible to get information about the usage of the system, and explanations of different WML-elements and attributes. Also, it is possible to find attributes and elements with index search. The help provides descriptions of all elements and attributes and also there can be found examples how to use these elements.



Figure 4.8. Help tool.

4.4.1 Implementation

The WmlHelp is the main class and it has sub components the HelpTree and the StyledTextPane that do the actual displaying (Figure 4.9), all components use the Java standard Swing architecture.

The HelpTree provides a view to the structure of a help document. While the user traverses the tree the StyledTextPane shows the help information in that particular node. The super class JTextPane supports embedded components such as JLabels and the styled text, and it uses the document interface for holding its data.

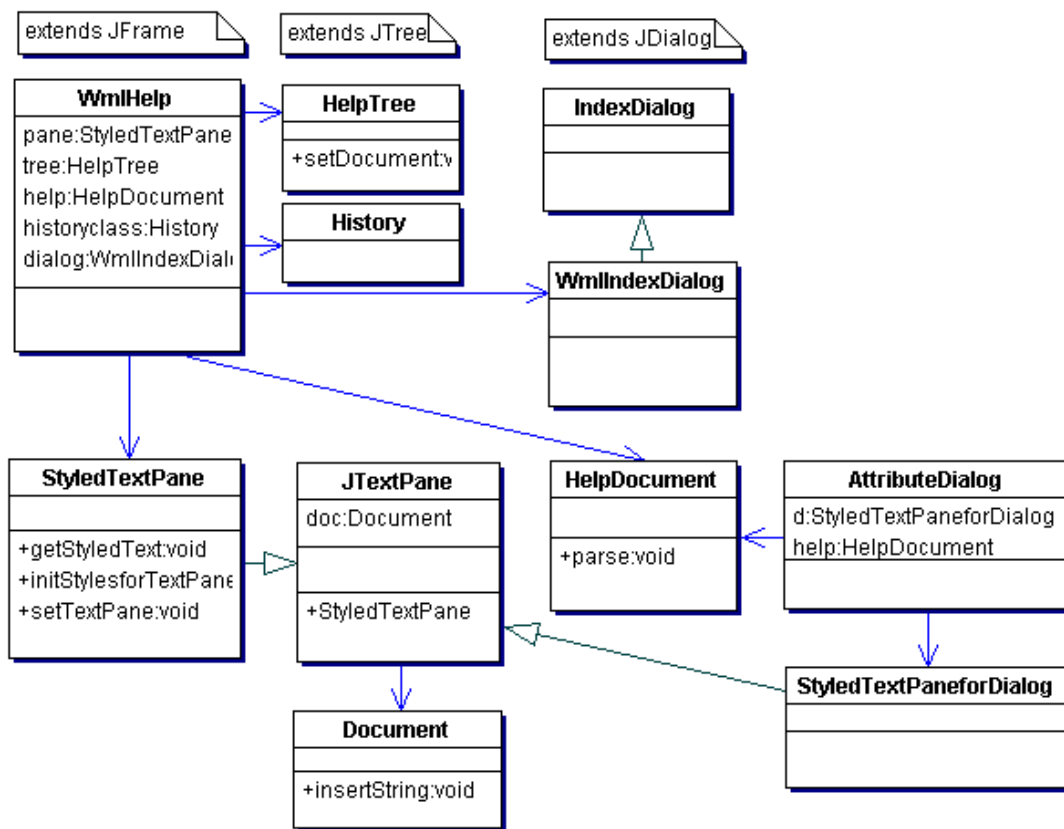


Figure 4.9. The class hierarchy of Help Tool.

The HelpDocument class gets and parses the WML help document, which is encoded in XML. The History class is used for caching the nodes of the document while the user traverse the tree.

WmlIndexDialog provides a mechanism for searching the information in the WML help document. The IndexDialog builds a familiar index search dialog, which an application writer can subclass and do his implementation with a data he wish to use. The attributeDialog is only for displaying the attribute information of a particular element and it uses the StyledTextpaneforDialog.

5. Phone Editor

With the phone editor tool (Figure 5.1) of Visual WML it is possible to define and edit WAP phone models. Also, it is possible to see in real time what the browser looks with the new edited settings (cf. Figure 5.3).



Figure 5.1. Phone editor.

Phone editor settings are divided in the three main categories, which are the User Agent Profile (UAPROF), the element settings, and the simulator settings part. In UAPROF part it is possible to define browser general settings like a vendor, model, etc. and, also, it is possible to define what are the features the browser supports (for example, it is image or sound capable and it's screensize, character and colordepth settings too). It is also possible to import and export UAPROF files.

In element settings part, it is possible to define font styles to different elements. Before font styles can be selected, the device's fontlist must be selected from fontlist selector (Figure 5.2). In the simulator settings there are settings such as background and textcolor. Also it is possible to set event buttons (onenterforward and onenterbackward) visible and enable edit in the browser view.

5.1.1 Implementation aspects

PhoneEditor technical implementation is shown in Figure 5.4. Class PhoneEditor is a JFrame which contains visual java swing components, like JTextFields, JComboBoxes, etc. These can manipulate phone settings (in class PhoneSettings). In phone editor UAProfiles can be imported or exported. Class UAProfManager takes care of these tasks. In phone editor it is possible to define current phone fontlist and that is possible in font selector dialog (class FontSelector).

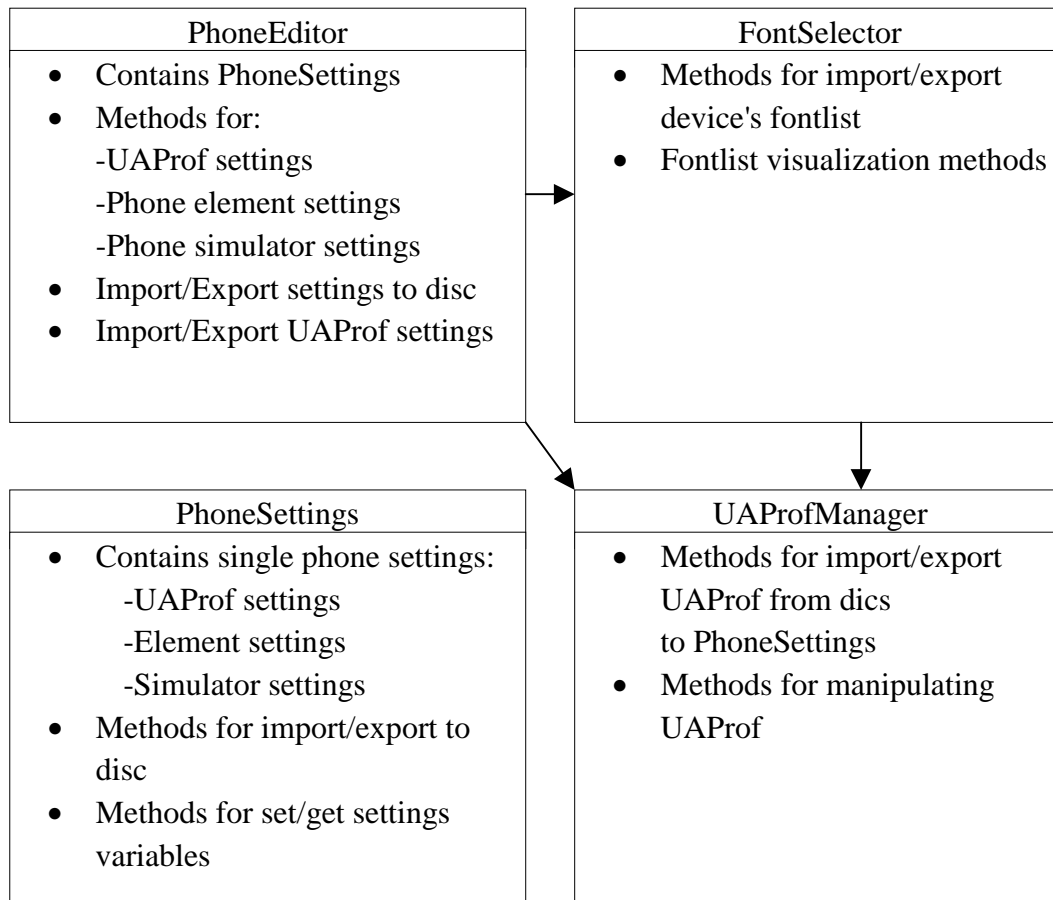


Figure 5.4. PhoneEditor class hierarchy.

6. WML browser – phone simulator

Browsing is possible in the *browser view* of the Visual WML main window or in the separate phone simulator (browser) windows (Figure 6.1) of the system. In the WML browser view, there is a control panel on the top.

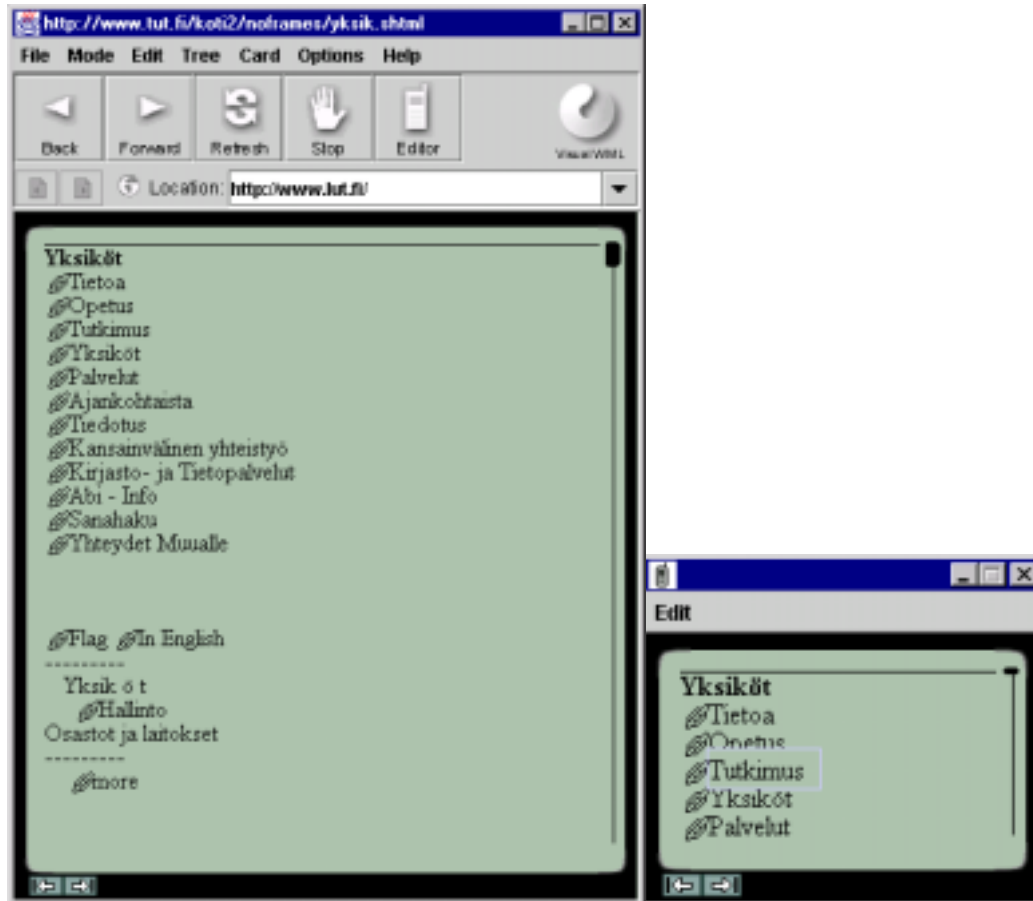


Figure 6.1. Visual WML browser part on the left. On the right is a simulator window.

Browser window represent WML elements with text and icons. It is possible to edit these texts if editable mode is selected. Element's functionality is possible to test by clicking the element icons. For example by clicking a *go* element icon (which is represented with an arrow) it is possible to navigate to new deck. Browsers are synchronized to other parts of Visual WML and all edit operations, which are made to deck, are updated to the others Visual WML windows. It is possible to test events such as a *timer* event with these icons. On the right in the browser window is bar which makes it possible to scroll rows. For testing *onenterbackward* and *onenterforward* events there are two buttons on bottom of window.

By writing a URL it is possible to load new locations (WML or HTML pages which are transformed to WML) to browser. In this panel there are also buttons which make it possible to move in navigation history between decks and cards.

6.1.1 Technical implementation

In Figure 6.2 WMLBrowser and Phone architecture are shown. Both have PhoneScreen, which implements phone simulator visualization. Difference between phone and browser part is that the browser contains PhoneControlPanel (panel which have buttons and location field) and AnimationPanel (which contains animated Visual WML logo) too.

To make visualizations PhoneScreen uses PhoneSettings class, which contains current phone (or browser) settings (fonts, screensize, etc...). PhoneScreen makes phoneElements from WMLElements and visualizes them in rows. PhoneScreen inherits class PhoneFunctionality which contains methods for WML elements functionalities and WML tasks. It has also methods, which make it possible to edit texts in simulator window. Class PhoneFunctionality contains variables (class Variable), which takes care of variables defined in WML deck.

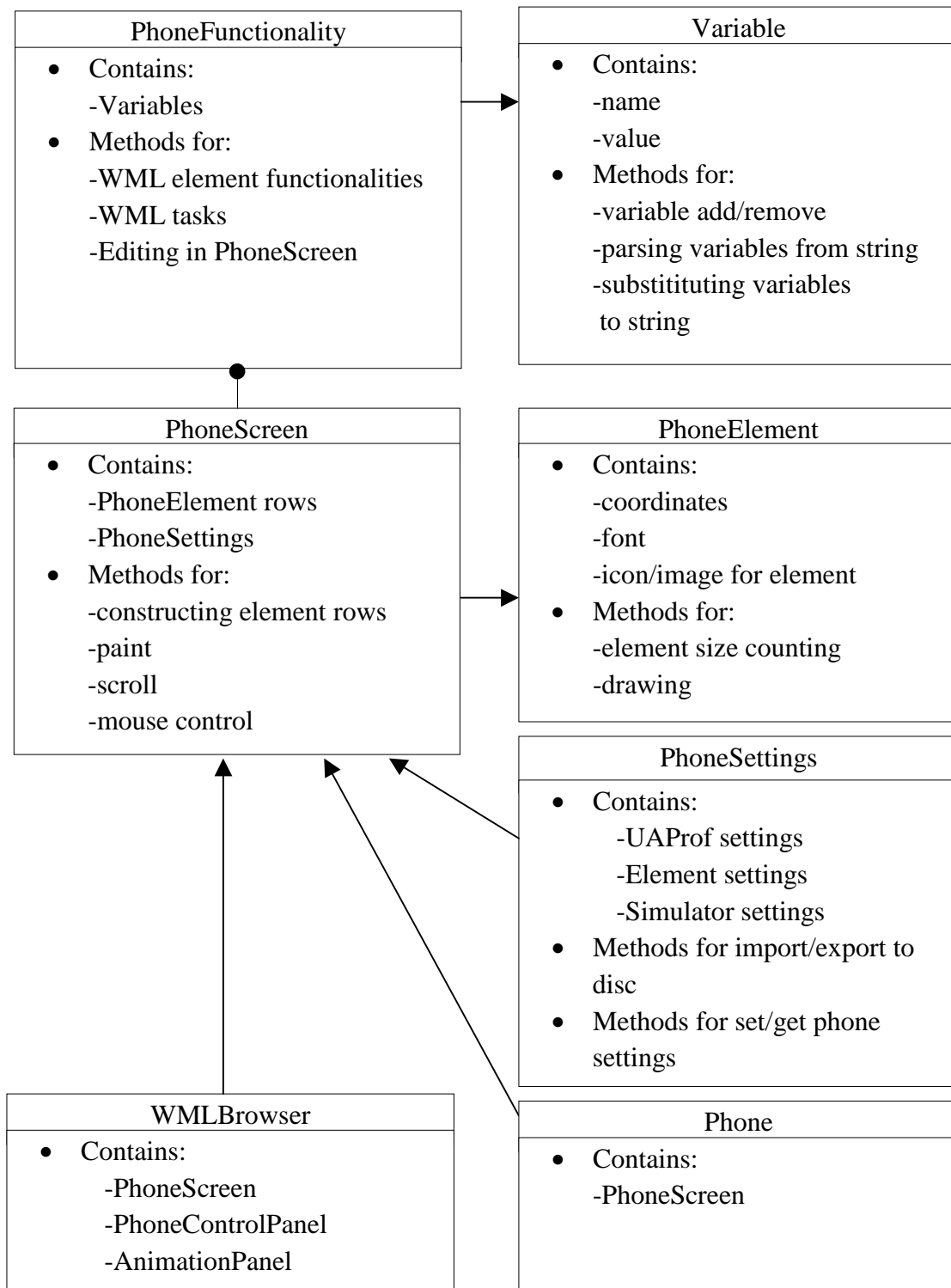


Figure 6.2. WML-Browser and Phone architecture.

7. User Agent Profile

7.1 Introduction

The WAP architecture has brought the World Wide Web to mobile devices. In the near future there will very probably be a wide range of mobile terminals with different capabilities and characteristics. Also, there may be available a variety of different kinds of alternative network connections (mobile and wireless local area networks) for a mobile device. The consequence of this is that clients may receive content that they cannot display or store or the content can take too long to download over the network to the client. How to adapt the content to these devices (and networks)?

The Composite Capabilities/Preferences Profile (CC/PP) framework [CC/PP,CC/PP_R] creates a structured format for how a client device tells an origin server about its capabilities and preferences (Figure 7.1). For expressing a user agent's profile the CC/PP uses Resource Description Framework (RDF) [RDF], which is an application of XML and which defines a simple model for describing interrelationships among resources in terms of named properties and values. The User Agent Profile (UAProf) [UAPROF] uses the CC/PP model to describe the characteristics of a user agent by defining a set of components and attributes.

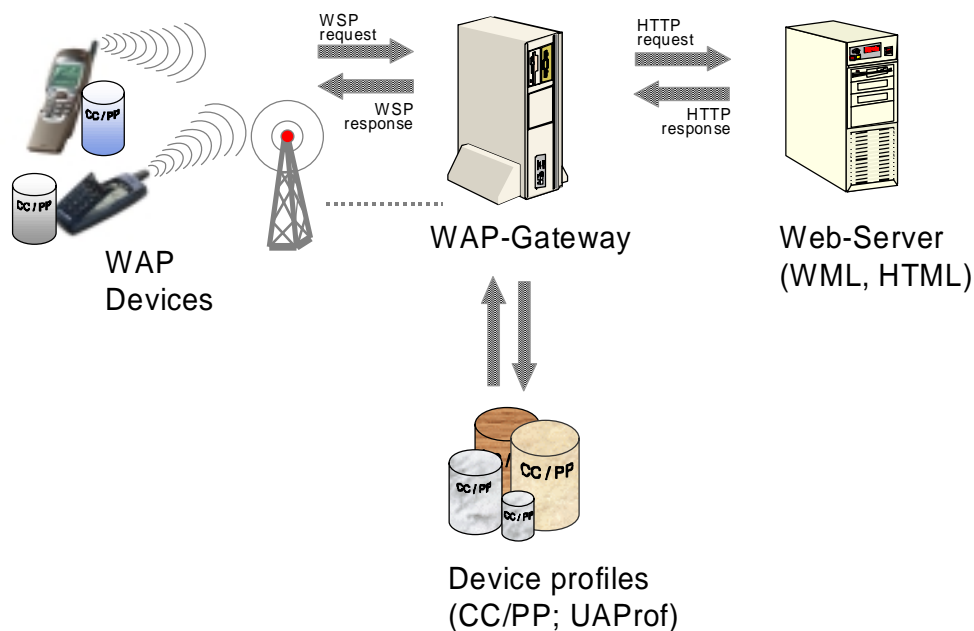


Figure 7.1. The User Agent Profile use case.

7.2 The architecture

The WAP User Agent Profile specification [UAProf] enables the end-to-end flow of a User Agent Profile between the WAP client, the intermediate network points, and the origin server. Capability and Preference Information (CPI) is transmitted in the headers of WSP [WSP] and HTTP1.1 (with HTTP Extension Framework) protocols. WAP gateway translates the WSP requests into HTTP 1.1 requests.

The CPI consists of information from the device hardware, user agent software, and user preferences and network characteristics. The device may not have all this information, but it may publish a single URI that point's e.g. to the device manufacturer and the WAP gateway resolves that URI and retrieves the information from the manufacturer host. The WAP gateway forwards the request from the client to the origin server and includes the profile information in HTTP header (see Figure 7.1). On the way to the origin server the profile may pass through one or more proxies. The origin server extracts the profile information and resolves all indirect references to information stored at other network elements. Then, the origin server formats the requested content into the appropriate format using the UAPROF and generates the HTTP response. The WAP gateway translates the HTTP response into the WSP response.

The UAProf may be cached in the WAP gateway. The WAP gateway applies the profile on all requests from the client, and, further, it may also add information to the profile such as additional network information.

7.3 UAProf schema and base vocabulary

The schema for the User Agent Profile's has five key components that are *HardwarePlatform*, *SoftwarePlatform*, *BrowserUA*, *NetworkCharacteristics* and *WapCharasteristics*. Each of these resources has a collection of properties that describe the component. The *HardwarePlatform* describe the hardware characteristics of the terminal device such as screen size, model etc. The *SoftwarePlatform* has properties that describe the operating system software such as OSVersion etc. The *BrowserUA* has a collection of attributes to describe the HTML browser application. The *NetworkCharasteristics* has information about the network's capability e.g. current network bearerservice and a supported security. The *WapCharacteristics* has properties to describe WAP capabilities on the device e.g. WmlDeckSize and WapVersion.

The following example (Figure 7.2) describes the HardwarePlatform component. The schema uses RDF model that is described in the next subsection 7.4.

```

<rdf:Description ID="HardwarePlatform">
  <rdf:type resource="http://www.w3.org/TR/PR-rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#Component"/>
  <rdfs:label>Component: HardwarePlatform</rdfs:label>
  <rdfs:comment>
    The HardwarePlatform component contains attributes that describe
    the device's hardware characteristics, such as display size,
    character set, alpha-numeric capable,etc.
  </rdfs:comment>
</rdf:Description>

```

Figure 7.2. The component definition of the HardwarePlatform.

In the next example (Figure 7.3), ImageCapable, VoiceInputCapable and Keyboard are properties that override the default ones.

```

<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:prf="http://www.wapforum.org/UAPROF/ccppschemal.0#">

<rdf:Description ID="TerminalHardware">
  <prf:Component>
    <rdf:type resource=
      "http://www.wapforum.org/UAPROF/ccppschemal.0/#HardwarePlatform"/>
    <prf:Defaults rdf:resource="http://www.xyz.com/profiles/1234" />

    <!-- override the ImageCapable property and add
      VoiceInputCapable and Keyboard properties -->
    <prf:Imagecapable>Yes</prf:Imagecapable>
    <prf:Keyboard>predictive</prf:Keyboard>
    <prf:VoiceInputCapable>Yes</prf:VoiceInputCapable>
  </prf:Component>
</rdf:Description>
</rdf:RDF>

```

Figure 7.3. A snippet of a profile.

7.4 RDF

The Resource Description Framework (RDF) [RDF] is being developed by the World Wide Web Consortium (W3C). Its aim is to provide the foundation for metadata interoperability across different resource description communities. RDF provides interoperability between applications that exchange metadata and is targeted for many application areas including resource description, site-maps, content rating, electronic commerce etc.

RDF uses XML as a common syntax for the exchange and processing of metadata. The RDF syntax is used to store instances of the RDF model into machine-readable files and to communicate these instances among applications. In this work, for parsing RDF the SiRPAC [SiRPAC] parser is used.

8. Examples

This chapter provides examples of creating and manipulating a WML deck using the Visual WML system.

8.1 Building a new WML deck

The following steps illustrate how to build a new WML deck and how to add elements to it.

First, go to the File menu and click on the new Deck item. After that the program asks you to specify the file name, you must type the name on the File Dialog box (see Figure 8.1). After that click on the save button. Now you have created a new WML deck and the program should look like in the Figure 8.2.

As you can see there is just two elements in your WML document, which are the *card* element and the up most WML element.

The *card* element is a required element in a WML document, it divides a WML deck into the single units that are shown on the device's screen one of each time.

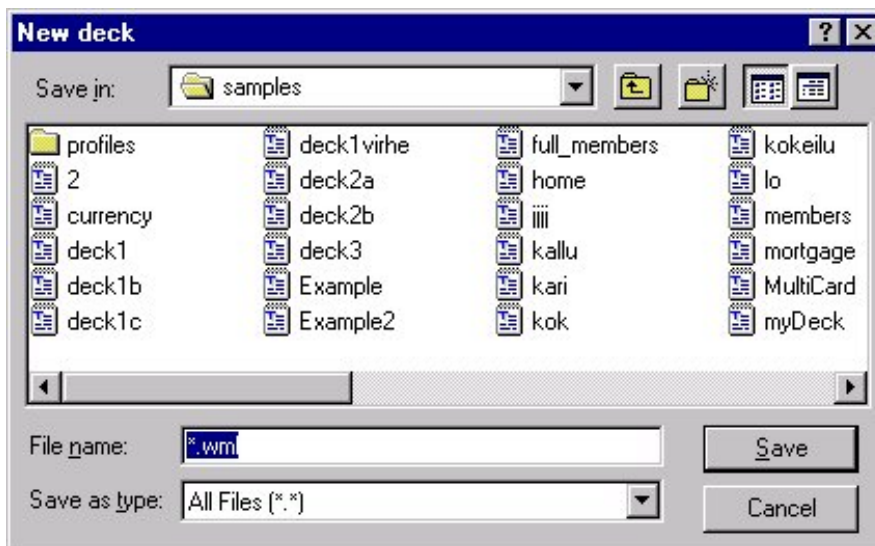


Figure 8.1. File dialog.

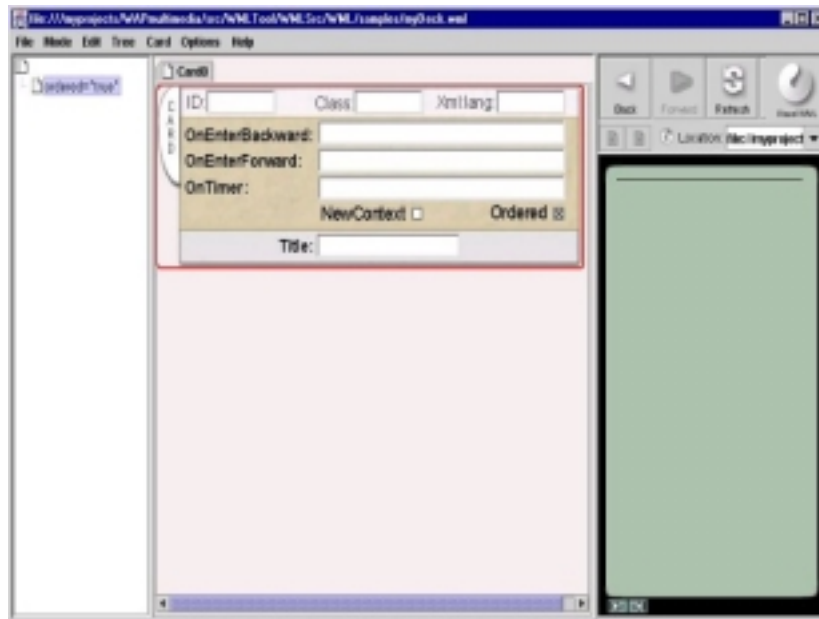


Figure 8.2. A new WML deck.

8.2 Adding elements

You can easily add new elements in your document without doing any grammatic errors. The only way to add elements is to use the right-most (or secondary) mouse button. Let's start with adding a *p* element in the *card* element (see Figure 8.3). First click (right button) on the *card* element in the *tree view* and choose the menu item add child -> formatting -> *p* and click on the *p* menu item. Now you should have succesfully add a new element in your document. If you want to display something in the *card0*, you must add a *text* node. Click on the *p* element and choose a menu item add child -> *text* and click on it, now the *text editor* should appear. With the *text editor* you can edit text nodes of your document. Write e.g. "Hello" and close the text editor (File->close), note that you don't have to save anything. Now the text is in the document. You could have done the previous add operations the same way on the *card view*, just try it.

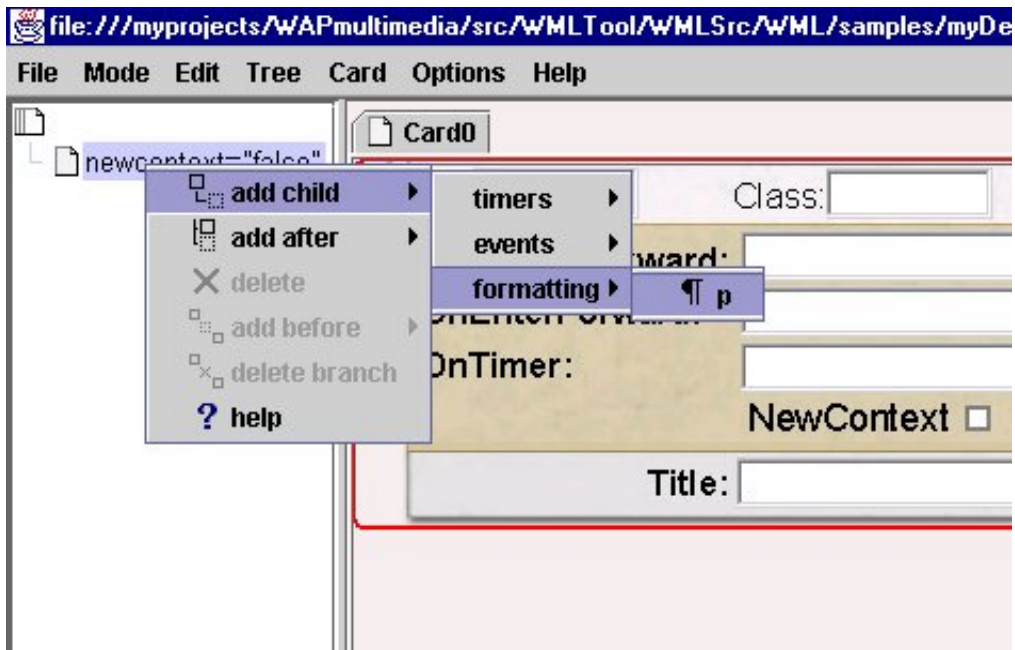


Figure 8.3. Adding a *p* element.

The next step is to add another *card* element to the document. Now we can try the add after menu item, which add elements to the same level. If you like to put a new *card* element before card0 you should use the add child menu item on the root element. Add the *card* element as shown in the Figure 8.4.

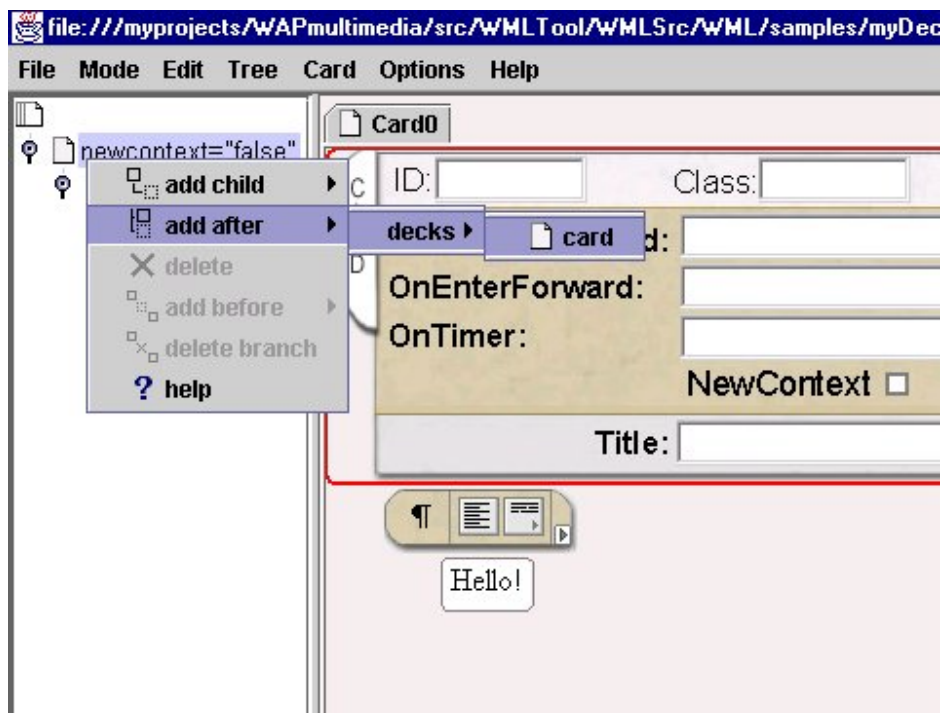


Figure 8.4. Adding a *card* element.

8.3 Attribute editing

You can specify a new attribute or change its value on the *card view* (in the middle of the program). Let's add a new value for the card element's (card0) id and title attributes (see figure 8.2). The id attribute is necessary if you want to navigate between cards. Just click on the text field and write e.g. "card0" (id) and "Hello card" (title).

Do the same for the card1 (id = "card1" and title = "Buy card") and add the *p* element and e.g. the text "Bye!".

Now your program should look like in the Figure 8.5. Look how the *WML Browser* (on the right) keeps updated as you edit the document.

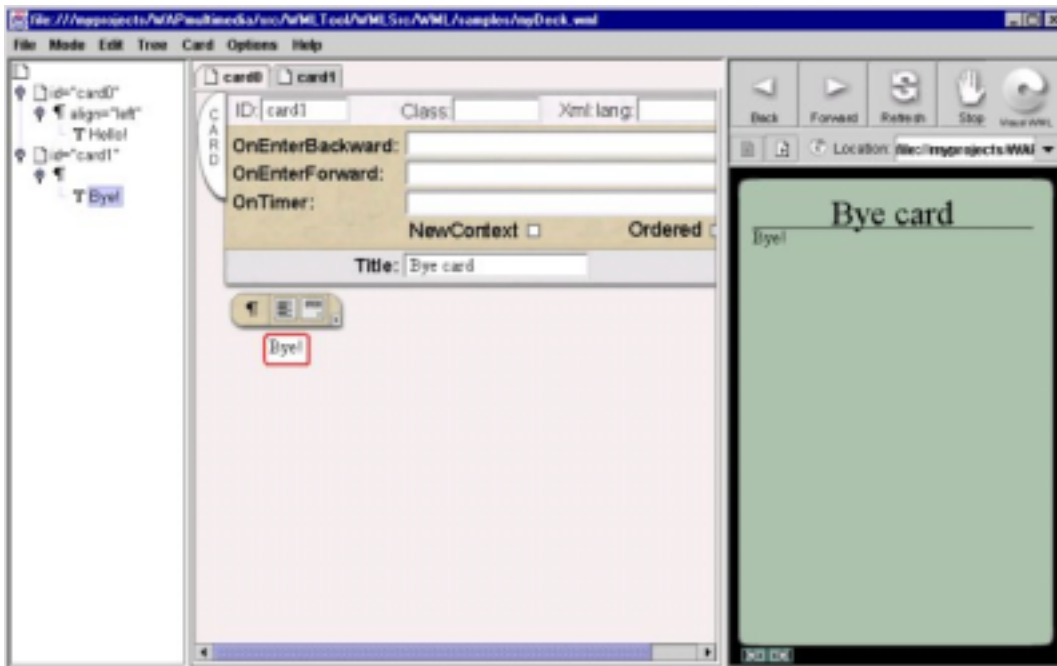


Figure 8.5. All together.

8.4 Adding actions

Let's put some action into our example deck. The *do* element specifies an action e.g. when user click on the button. The *do* element requires a child element e.g. *go* element that is used for indicating a resource. Add *do* element after *text* element in the Hello card (see Figure 8.6).

The program asks you to specify type attribute for the `<do>` element, this is because the type attribute is required. You can get some help about the `<do>` element attributes by clicking on the help? button. Add a new value "accept" and press enter. Then you must choose a child element `<go>` and after that you must specify the href, put value "#card1" and press enter. Now you have made a WML deck with some functionality. You can try how your first deck works in the *WML Browser*. When you are in the first card, click on the arrow and look what happens.

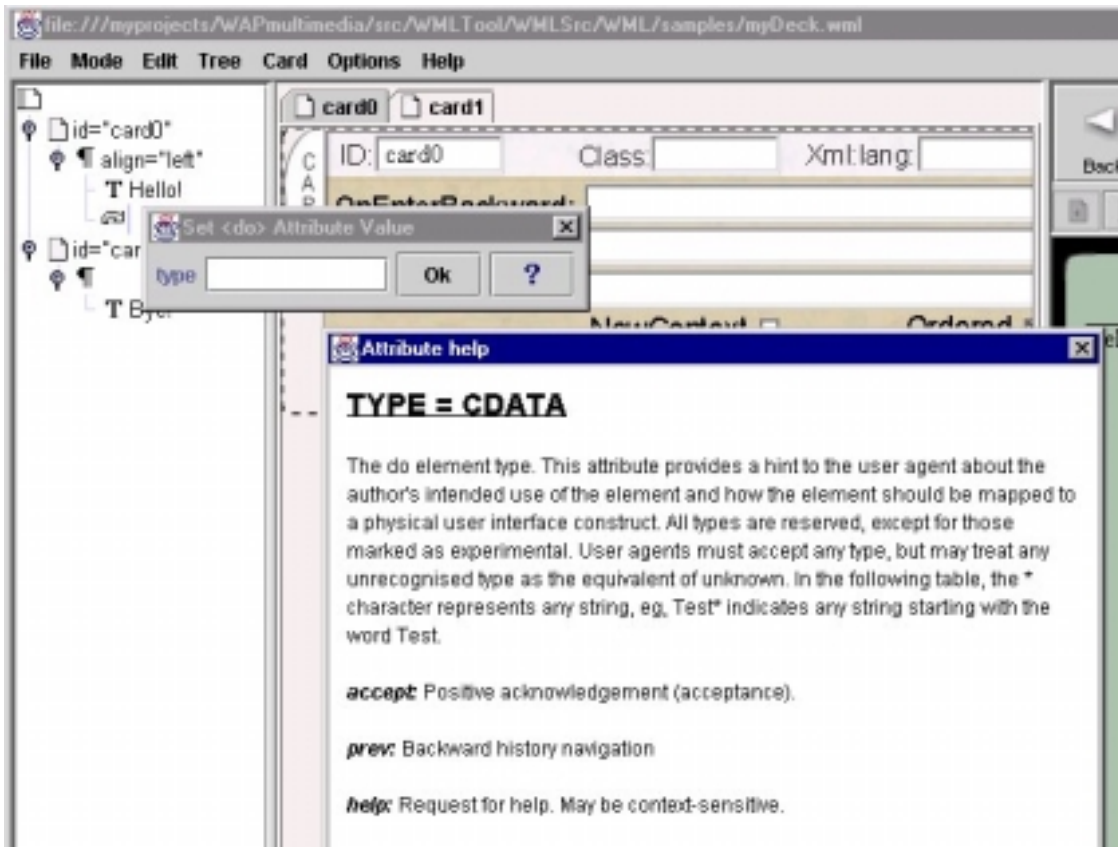


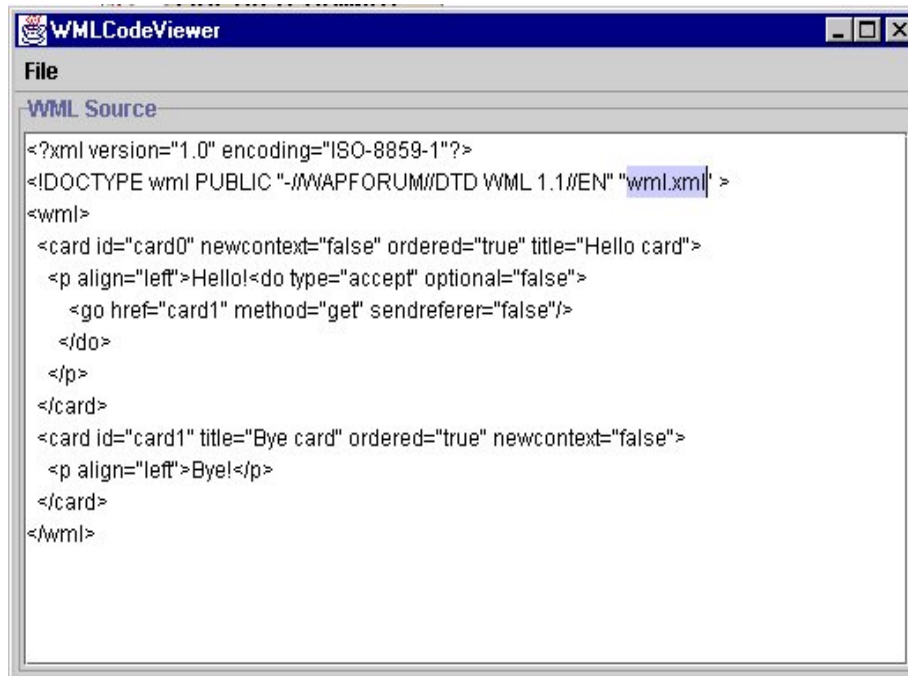
Figure 8.6. An required attribute.

8.5 Saving

Before saving you must note that the current version of the program supports the WML version 1.1 and the DTD it uses is located at the WAPForum. If you want to use a local DTD, open the source editor by choosing Options->View WML Code (see Figure 8.7).

Replace the text "http://www.wapforum.org/DTD/wml_1.1.xml" with "wml.xml". Then choose File->save and Recompile Deck.

The normal saving operation can be done by choosing File->saveDeck in the main program. This example is in the samples directory, named "myDeck".

A screenshot of a software window titled "WMLCodeViewer". The window has a menu bar with "File" and a toolbar with standard window controls. The main area is labeled "WML Source" and contains the following XML code:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM/DTD WML 1.1/EN" "wml.xml" >
<wml>
  <card id="card0" newcontext="false" ordered="true" title="Hello card">
    <p align="left">Hello!<do type="accept" optional="false">
      <go href="card1" method="get" sendreferer="false"/>
    </do>
  </p>
</card>
<card id="card1" title="Bye card" ordered="true" newcontext="false">
  <p align="left">Bye!</p>
</card>
</wml>
```

Figure 8.7. Code View.

9. User requirements and design process

The goal of the usability evaluation was to identify the usability problems of the Visual WML system of the WML Browser project.

Our usability framework included effectiveness, efficiency and satisfaction as defined in ISO standard ISO 9241-11 [ISO98].

Effectiveness is the accuracy and completeness with which users achieve specified goals. Thus effectiveness defines if the system includes the right functions. The effectiveness of Visual WML defines how well the user can edit and preview WML files.

Efficiency defines the resources expended in relation to the accuracy and completeness with which users achieve goals. Thus efficiency defines how fluently the user can use the functions of the system. The efficiency of Visual WML defines *how fast* the user can create and edit WML code and *how much effort* is required of the user to do it.

Satisfaction is the extent to which users are free from discomfort, and attitudes towards the use of the product.

In our project we have evaluated also the *usefulness* of the program. Usefulness and usability predict the acceptance of the services.

9.1 Methods

9.1.1 Design process

Our human-centred design process consisted of four kinds of activities as described in ISO standard 13407 “Human-centred design processes for interactive systems” [ISO99]:

- understand and specify the context of use
- specify the user and organisational requirements
- produce design solutions
- evaluate designs against requirements

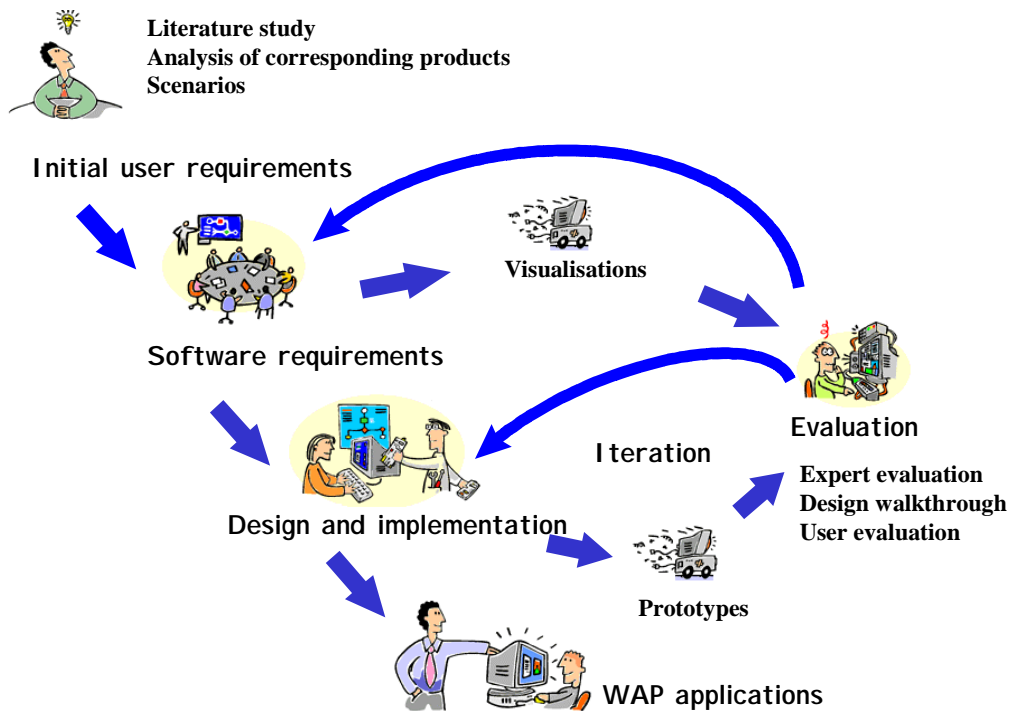


Figure 9.1. The design process of WML Browser project.

Figure 9.1 describes the design process of WML Browser project. Initial user requirements and context of use were based on a literature survey of current research results, analysis of corresponding products and scenarios of typical use cases.

We could only define draft user requirements in the beginning of the design process. Our design process allowed us to identify new user requirements and feed them back into the process throughout the whole life cycle of the project. Design rationale has been very essential in this kind of design process because the WML specification work is still going on and new WAP devices and software development environments are constantly released. We have to keep track of the changes in user requirements and how we were able to respond to the requirements.

Evaluation has been a continuous activity during the requirements definition and design phases of the project (Figure 9.2). The aim of the evaluation was to find usability problems, to understand the reasons for the problems and to give feedback and new ideas to the design. In this way we could assure that the usability problems were identified and fixed as early as possible.



Figure 9.2. Feedback from users was taken into account during the whole development process.

One usability expert was present throughout the development of the system. He gave input to UI design decisions whenever it was needed. Additional design walkthroughs with three usability experts were also held. This means meetings where the users, application field experts, designers and usability experts together evaluate design solutions to get feedback and to generate new ideas.

A small-scale user trial with 3 users was held during the project. At later stages of the software, feedback from actual users were used in developing the system.

10. Conclusions

10.1 Overall evaluation results

Our test users found the program useful, especially in comparison with text-based editors. The visual representation clarified the hierarchical structure of the document. Support for multiple phone simulators was also a popular feature.

The main problems in Visual WML were not necessarily the usability of existing functions, but rather lack of support for some basic functions. The Undo function seemed to be one of the main concerns and it is currently under development.

Interaction with the program was somewhat hindered by delays in system feedback. After the user clicked a function there was often a short pause before the function took place. The Java platform is the reason for some of the hitches. In the current version the response times have improved.

Some of the user feedback dealt with organisation of functions in the menu bar. It became apparent that actual use of the editor required a different organisation of functions than what seemed appropriate during development. For instance, more menu items were added under "View" option and some of the terms were changed. Keyboard shortcuts were added for each menu item. In general, user wished for quite extensive keyboard support.

The rough preview of how the document looks in several different phone models was appealing to the users. However, the interaction logic of the mobile browsers varies a lot and this makes it difficult to design a user interface that would work well on different platforms.

Visual WML has been updated and developed during the project according to feedback from the users and the comments from usability experts.

10.2 Technical results

Visual WML is a good example of how the utilizing of (new and/or just developing) open, standard technologies (such as Java, XML, DOM, CC/PP) enable an easy and fast way to make such an efficient tool. The Java architecture enables an easy object management and provides a large amount of classes for building graphical user interfaces, network programming and data processing. Java enables very efficient way to process XML.

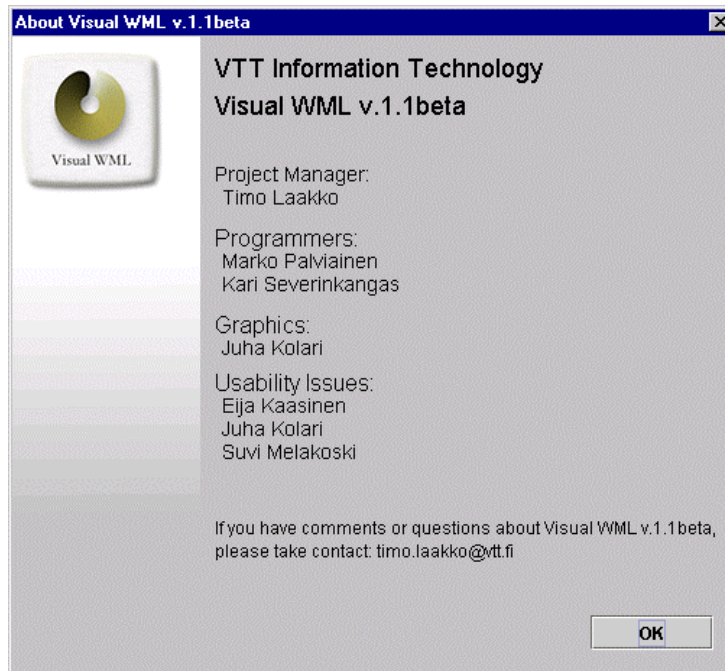


Figure 10.1. Visual WML logo and the project team.

XML has proved its functionality and flexibility for storing and delivering information in Web. WML is one prove of the extensibility of XML. The most important feature of WML is that it is not dependent on a specific user interface, like HTML, which enables the use of WML in various different devices with different input and output capabilities.

During the project, the WAP specifications have developed significantly. For instance, there exists now a WAP UAPROF specification as a part of WAP1.2.

The project team members (Figure 10.1) had actively followed the specification work done within WAP-forum, and also attended the working group meetings of some key areas. The active following of the specification work provided a good basis for new ideas, and their timely implementation before the actual specifications are publicly available. A good example of that was the very early utilization of the WAP UAPROF and W3C CC/PP specification work.

10.3 Future work

Our future plans include, for instance, to further study the adaptivity and personalization of services. For that UAPROF [UAPROF] is a very promising way to describe terminal and network properties.

We are planning to concentrate on future personal multimedia aspects in WAP. In that, we would like to realize support to new content types (such as audio and video, MPEG-4, MPEG-7) on our WAP development environment, and to elaborate new multimedia applications. The development of new wireless network solutions (such as GPRS, UMTS, Wireless LAN) will directly support WAP multimedia applications.

We would also like to extend the operating environment. This work could possible include porting and optimising the code to other (mobile) operating systems like EPOC, Windows CE and PalmOS.

Also, we are especially interested in mobile usability aspects, and the system will be used for different kinds of usability tests.

11. Summary

Visual WML is a tool targeted for creating, editing and browsing WML (Wireless Markup Language) documents. It is also possible to emulate different kinds of WAP (Wireless Application Protocol) devices by utilizing User Agent Profiles.

The WAP architecture provides an extensible environment for application development for mobile communication devices. WAP WML is an XML (Extensible Markup Language) application. XML provides an easy and flexible way to describe and deliver structured information.

The increasing terminal diversity and the increasing population of the mobile users will add the demand for more and more personal usage profiles. At the same time, the forthcoming mobile communication channels (GPRS, UMTS) will increase the transmission capacity, and wireless local area networks (Wireless LAN) will offer fast and easy access to wired internets and services. All this poses challenges to application development, and hence, new tools for WAP application development are required.

Acknowledgements

The work described in this document has been done within the WML-Browser project of VTT Information Technology during March 1999 – April 2000.

One of the initial aims of the WML Browser project was to support other WAP research projects of VTT. Because there were no suitable WAP browser tools available when the early WAP projects (WAP-Proxy, WAP-stack, and APRO) of VTT were in progress, there was a need to start a new project that concentrates on the WAP client-side browser tools. The VisualWML team members are enumerated in Figure 10.1.

Visual WML has benefited from the technical help and advice of several persons from VTT, and the other WAP projects (<http://www.vtt.fi/tte/projects/WAP/>). Kari Severinkangas has implemented many parts of the VisualWML, and his Engineer's Study [SEVE00] provided a good source for parts of this document. Also, the technical advice and comments from the companies participated the above mentioned WAP projects have been very helpful.

References

- [CC/PP] Composite Capability/Preference Profiles (CC/PP): A user side framework for content negotiation
<http://www.w3.org/TR/NOTE-CCPP/>
- [CC/PP_P] CC/PP exchange protocol based on HTTP Extension Framework.
<http://www.w3.org/TR/NOTE-CCPPexchange>
- [CC/PP_R] Composite Capabilities/Preference Profiles: Requirements and Architecture.
<http://www.w3.org/TR/2000/WD-CCPP-ra-20000228/>
- [DOM] Document object model (DOM).
<http://www.w3c.org/DOM/>
- [DTD] Document Type Definition. The design of the XML specification DTD available at <http://www.w3.org/XML/1998/06/xmlspec-report-19990205.htm> - AEN39
- [HCI99] Kaasinen, E., Aaltonen, M. and Laakko, T. Defining User Requirements for WAP Services. HCI'99 International, 8th International Conference on Human-Computer Interaction, August 22–27, 1999, Munich, Germany.
- [ISO98] ISO 9241-11. International standard. Ergonomic requirements for office work with visual display terminals – Guidance on Usability. 1998.
- [ISO99] ISO 13407. International standard. Human-centred design processes for interactive systems. 1999.
- [LEP99] Leppänen, J., Laakko, T. and Kylänpää, M. Prototype development for the WAP application. In ACTS Mobile Communications Summit, Sorrento 8th – 11th June 1999.
- [LEV98] Leventhal, M., Lewis, D. and Fuchs, M. Designing XML Internet Applications. Prentice Hall, 1998.
- [OMGI] OMG IDL Syntax and Semantics.
<http://www.ti5.tu-harburg.de/Manual/OMG/CORBA/index.htm>

- [RDF] Resource Description Framework (RDF) Schema Specification
<http://www.w3.org/TR/1999/PR-rdf-schema-19990303/>
- [RDF_M] Resource Description Framework (RDF) Model and Syntax Specification
<http://www.w3.org/TR/PR-rdf-syntax/>
- [RFC2068] Fielding, R. et al. Hypertext Transfer Protocol – HTTP/1.1. January 1997. <ftp://ftp.isi.edu/in-notes/rfc2068.txt>
- [SAX] Simple API for XML <http://www.megginson.com/SAX/index.html>
- [SEVE00] Severinkangas, Kari. Visual WML – An XML tool for WAP. Thesis, Häme polytechnic, Riihimäki, 2000.
- [SGML] Information Processing – Text and Office Systems – Standard generalized Markup Language (SGML). ISO 8879:1986.
- [SIRPAC] Simple RDF Parser & Compiler
<http://www.w3.org/RDF/Implementations/SiRPAC/>
- [UAPROF] Wireless Application Group. User Agent Profile Specification. Version 22-June-1999. Available at <http://www.wapforum.org/>
- [WAEO] Wireless Application Protocol. Wireless Application Environment Overview. Version 1.1, 16-June-1999. Available at <http://www.wapforum.org/>
- [WAPA] Wireless Application Protocol. Architecture Specification. version 30-Apr-1998. Available at <http://www.wapforum.org/>
- [WBXML] WAP Binary XML Content Format. Version 1.1, 16-June-1999. Available at <http://www.wapforum.org/>
- [WML] Wireless Application Protocol. Wireless Markup Language Specification, Version 1.1, 16-June-1999. Available at <http://www.wapforum.org/>
- [WMLScript] Wireless Application Protocol. Wireless Markup Language Script. Version 1.1, 17-June-1999. Available at <http://www.wapforum.org/>

- [WSP] Wireless Application Protocol. Wireless Session Protocol Specification. Version 1.1, 28-May-1999. Available at <http://www.wapforum.org/>
- [WWW00] Kaasinen, E., Aaltonen, M., Kolari, J., Melakoski, J. and Laakko, T. Two Approaches to Bringing Internet Services to WAP Devices. The Ninth International World Wide Web Conference (WWW9), May 2000, Amsterdam
- [XLink] XML Linking Language (XLink) Version 1.0
W3C Candidate Recommendation 3 July 2000
<http://www.w3.org/TR/2000/CR-xlink-20000703/>
- [XML] Extensible Markup Language (XML) 1.0. W3C Recommendation 10-February-1998. <http://www.w3.org/TR/1998/REC-xml-19980210>
- [XML4J] A validating XML parser written in Java.
<http://www.alphaworks.ibm.com/tech/xml4j>
- [XPointer] XML Pointer Language (XPointer) Version 1.0
W3C Candidate Recommendation 7 June 2000.
<http://www.w3.org/TR/2000/WD-xptr-20000607>
- [XSL] Extensible Stylesheet Language (XSL) Version 1.0
W3C Working Draft 18 October 2000.
<http://www.w3.org/TR/2000/WD-xsl-20001018/>

Published by



Vuorimiehentie 5, P.O.Box 2000, FIN-02044 VTT, Finland
Phone internat. +358 9 4561
Fax +358 9 456 4374

Series title, number and
report code of publication

VTT Research Notes 2068
VTT-TIED-2068

Author(s) Palviainen, Marko, Laakko, Timo and Kolari, Juha			
Title Visual WML – a development tool for WAP applications			
Abstract <p>This work was carried out in the research project "WML Browser" of VTT Information Technology between March 1999 and April 2000. The main goal of the project was to build a software that is capable of browsing WML (Wireless Markup Language) and emulating different WAP (Wireless Application Protocol) terminals, and allows the WML developer to build and edit WML documents by using visual components. Also, the project group decided to use the existing technologies and well-defined specifications and open standards as much as possible. The developed software was named Visual WML.</p> <p>WAP architecture enables the interconnection of the wireless data network and the wired data network, and WAP technology brings the Internet content and advanced data services to digital cellular phones and other wireless devices. At the same time, the variety of different kinds of mobile terminals and networks are increasing. For instance, the forthcoming mobile communication channels will increase the transmission capacity, and wireless local area networks will offer fast and easy access to wired internet, corporate and home environment LANs and their services. Hence, in particular, tools for WAP application development are required.</p> <p>Visual WML is a tool targeted for editing, creating and browsing WML documents. It is also possible to emulate different kinds of WAP devices by utilizing User Agent Profiles.</p>			
Keywords Wireless Markup Language, XML, Extensive Markup Language, Wireless Application Protocol, WAP application development, mobile internet, Web services, usability			
Activity unit VTT Information Technology, Tekniikantie 4 B, P.O.Box 1203, FIN-02044 VTT, Finland			
ISBN 951-38-5807-3 (soft back edition) 951-38-5808-1 (URL: http://www.inf.vtt.fi/pdf/)		Project number T9SU00001	
Date December 2000	Language English	Pages 55 p.	Price B
Name of project WML Browser		Commissioned by	
Series title and ISSN VTT Tiedotteita – Meddelanden – Research Notes 1235-0605 (soft back edition) 1455-0865 (URL: http://www.inf.vtt.fi/pdf/)		Sold by VTT Information Service P.O.Box 2000, FIN-02044 VTT, Finland Phone internat. +358 9 456 4404 Fax +358 9 456 4374	