Current Technologies for Device Independence

Mark H. Butler
Publishing Systems and Solutions Laboratory
HP Laboratories Bristol
HPL-2001-83
April 4th , 2001*

world-wide web, mobile, XML, device independent, adaptation

Increasing numbers of users want to access web content from devices such as WAP phones or PDA's. Creating content for each device would be expensive and time consuming. This report provides a survey of current technologies related to the creation of device independent web content and web applications.

* Internal Accession Date Only Approved for External Publication

1 Introduction

Due to device proliferation, content providers can no longer deliver one version of their content to the user as they need to deliver an appropriate form of content depending on the capabilities of the viewing device. Re-authoring content, in order to support different markup languages or the different capabilities of each device, is clearly impractical whereas providing content for a single device or browser excludes large numbers of users. This report investigates how we can address these problems by providing device independent content. It provides a comprehensive survey of existing technologies organized in the Framework shown in Figure 1.

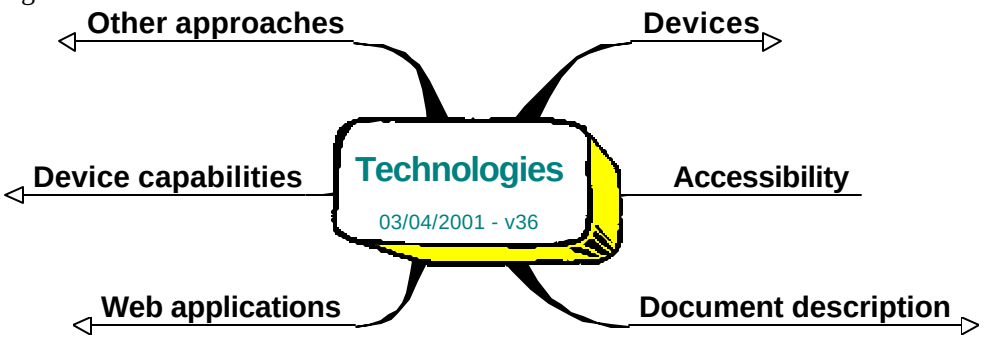

Figure 1 - Existing technology for device independence

2 Devices

Users want to view Internet content and use web applications on a variety of devices including PCs, electronic book readers, PDAs, phones, interactive TVs, voice browsers, printers and embedded devices such as cameras. A useful summary of typical variations in device capabilities is shown in Figure 2.

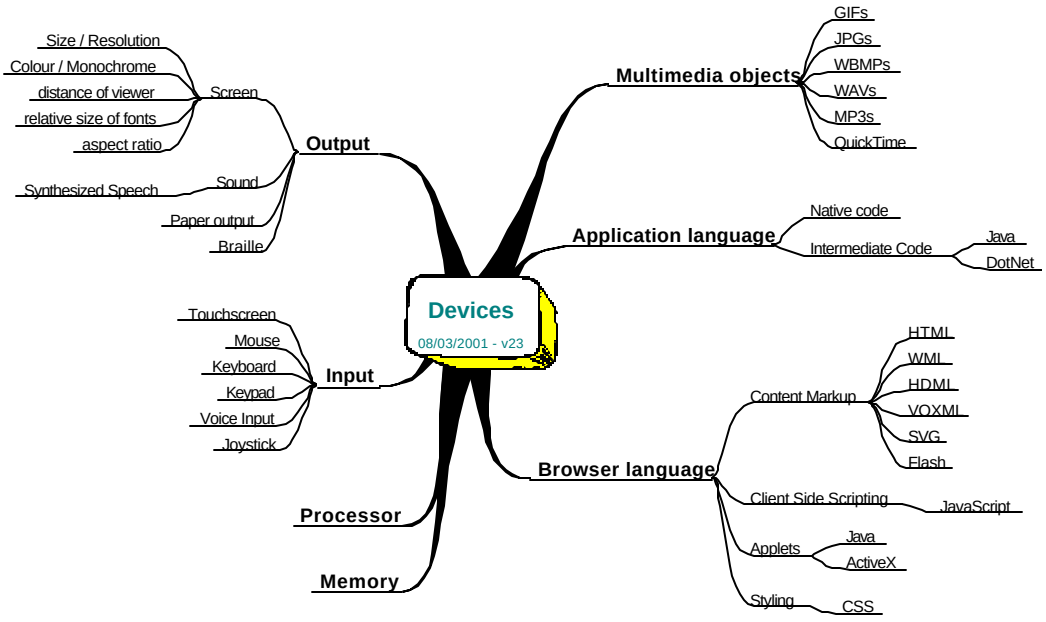

Figure 2 - Variations in Device Capabilities

When the device uses content it receives it in the form of multimedia objects, application languages or browser languages (shown on the right hand of Figure 2). Current devices support a variety of different content types partly determined by their underlying hardware

capabilities (shown on the left hand side of Figure 2). In order to support device independence we must be able to deliver content in a format compatible with a device. For example if a handheld device can read GIF images but not JPEG images it is necessary to convert one format to another. In addition the content must reflect the underlying hardware capabilities of the device so we may need to do some additional image processing if the target device can only display four level grey scale output.

3 Specifying Device Capabilities

Currently there are three different places adaptation can take place as shown in Figure 3: the server, the proxy and the client browser. There are examples of all three architectures: server based (Cocoon, Axkit), proxy based (AvantGo, Palm Web Clipping) and client based (XHTML / CSS). These examples will be discussed in later sections. If adaptation occurs at the server or the proxy, these entities will need to know something about the capabilities of the client. They will either need a unique identifier for the client device so they can retrieve a capability specification from a repository or they will need the capability specification itself.

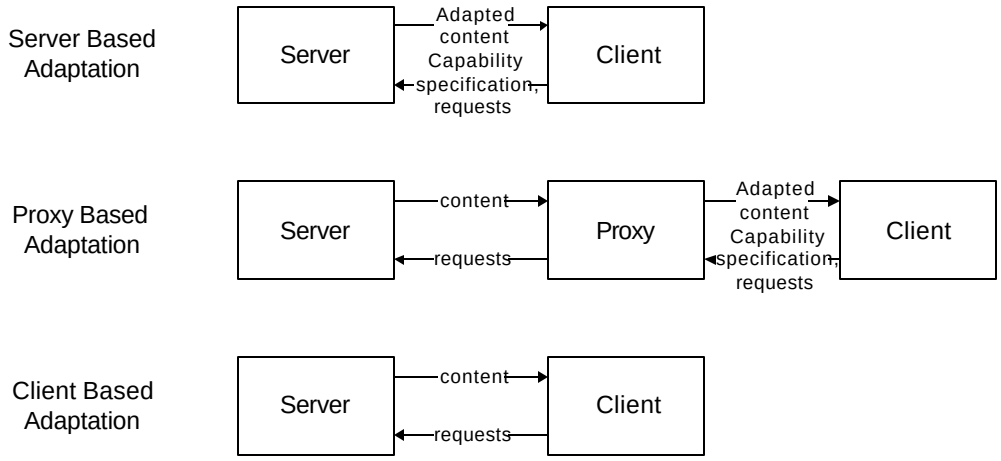

Figure 3 - Adaptation Types

Currently, servers and proxies can determine the identity of a particular device using the request header field in the HTTP protocol. In addition there are four alternative proposed capability specification schemes: the W3C composite capability / preferences profile (CC/PP), the WAP User Agent Profile (UAPROF) standard, the SyncML Device Information standard (DevInf) and the Universal Plug and Play Standard (UPnP).

3.1 HTTP Request Header Fields

User agents (web clients) identify themselves when they send requests to web servers[1]. This is done primarily for statistical purposes and the tracing of protocol violations but does support the automated recognition of user agents. For example early Netscape products generate User-Agent strings that look like this:

```
Mozilla/4.04 (X11; I; SunOS 5.4 sun4m)
```

Where the user agent string has the following syntax:

```
Browser / version( platform ; security-level; OS-or-CPU description)
```

Ideally, devices should reveal more information about their capabilities and preferences than this. There have been attempts to extend the request header format but this has not occurred in

a standardized way. For example Netscape request headers contain a proprietary language specification; certain proxies append proprietary information to the request header.

Using the user-agent string to perform content adaptation has caused problems. Early on in the development of the web, webmasters used the user-agent string to determine whether to send frames to a browser. These sites would only send frames to browsers that identified themselves as "Mozilla" as this was the first browser with frames support. Consequently Microsoft made their browser claim to be Mozilla because that was the only way to let their users view framed web pages. Clearly if we do provide specific content for a device, we need to make sure the device or browser can override that if necessary.

3.2 CC/PP Composite Capability Preferences Profile

The Composite Capability / Preferences Profile (CC/PP) standard[2] is being created by the W3C. CC/PP is a comprehensive method for communicating the capabilities of devices such as clients, proxies, gateways and caches as well as resources such as documents. CC/PP is based on the Resource Description Framework (RDF)[3], another standard created by the W3C. RDF was originally intended for describing web pages so that they are more easily indexed and understood by search engines. In RDF all entities are described as resources and consist of a resource name, a resource property and an attribute. These resources are organized in components to form a schema. CC/PP is not a specific vocabulary for specifying device capabilities. Rather it is a generic language for constructing such vocabularies. Example vocabularies have been demonstrated for:

- *Print and display devices* describing the device identifier, the screen size in characters and pixels, the MIME types, the character set and the colour palette available.
- *User agent headers* describing, the terminal hardware e.g. CPU, screen size, the terminal software e.g. OS Name, OS Version, OS Vendor and the browser e.g. type, name, version and the MIME types it accepts.
- *Proxy servers* describing the MIME types it accepts on behalf of the client, the MIME types it transforms on behalf of the client and the MIME types it rejects on behalf of the client.

Currently CC/PP fails to address two of the key problems concerning device independence: firstly it does not provide a standard vocabulary for web clients to communicate their capabilities to servers. Secondly it does not describe the type of transformations and customisations that servers are expected to perform on the behalf of devices based on their capabilities. These problems are beyond the scope of the CC/PP working group but they must be addressed in order for CC/PP to be of practical use.

3.3 WAP UAPROF

The WAP User Agent Profile (UAPROF)[4] is a standard being developed by the WAP Forum. It is intended that future WAP devices will use it to communicate their capabilities to a server. It is a CC/PP application so it addresses the first problem highlighted in the previous section i.e. provides a standard vocabulary for WAP clients to communicate their capabilities to servers. However it is not ideal as it would be preferable to have a single vocabulary for all web clients rather than just WAP devices. The current UAPROF specification does not address the second problem highlighted in the previous section mainly how servers or proxies should use the information supplied by clients using the UAPROF.

Interestingly WAP UAPROF considers five different categories of device capability as shown in Figure 3: software, hardware, browser, network and WAP. This means the server can adapt to the capabilities of the network as well as the capabilities of the device

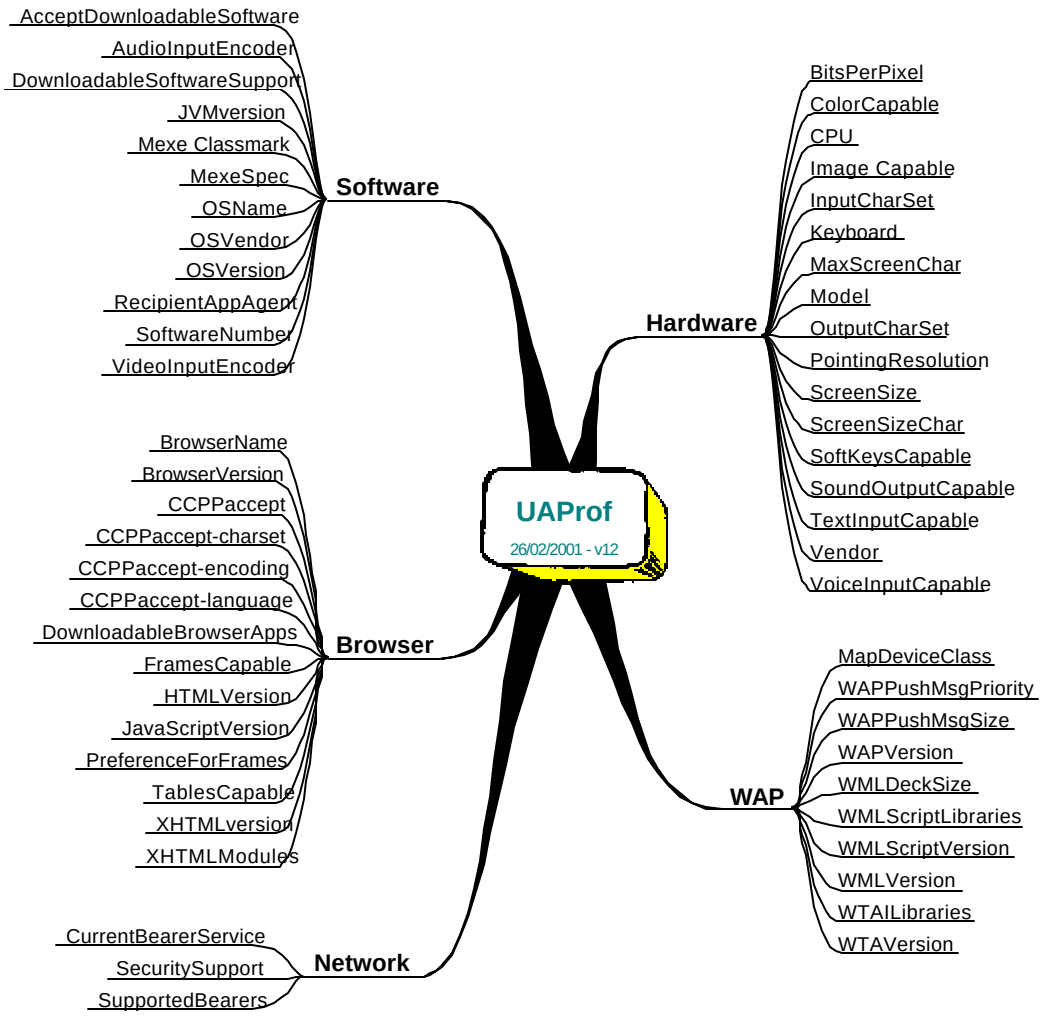

Figure 4 - The WAP UAPROF Specification

3.4 SyncML

The SyncML initiative[5] aims to develop a common synchronisation protocol for data between mobile devices such as phones, PDAs, desktop PCs and servers. Devices such as phones only support a limited number of applications: for example most have an address book and some have diaries. SyncML can be used to synchronise entries used in these applications, but could be used for potentially any file type. The intention is to make SyncML interoperable across a wide variety of transport protocols such as HTTP, WSP (the WAP session protocol) and OBEX (the Bluetooth / IrDA / USB transmission protocol).

When two devices undergo a synchronisation, if they have not synchronised before they have to exchange a description of their capabilities. This is done using the SyncML Device Information (DevInf) standard. Instead of implementing DevInf as a CC/PP vocabulary it has been implemented directly using XML. This choice is unfortunate as CC/PP has been developed expressly to implement standards like DevInf. The DevInf device description comes in a four parts as shown in Figure 5: the device, the content types it can accept, its datastore and any extensions it supports. Like CC/PP, it uses MIME types in order to express the content types that can be transmitted and received by a device.

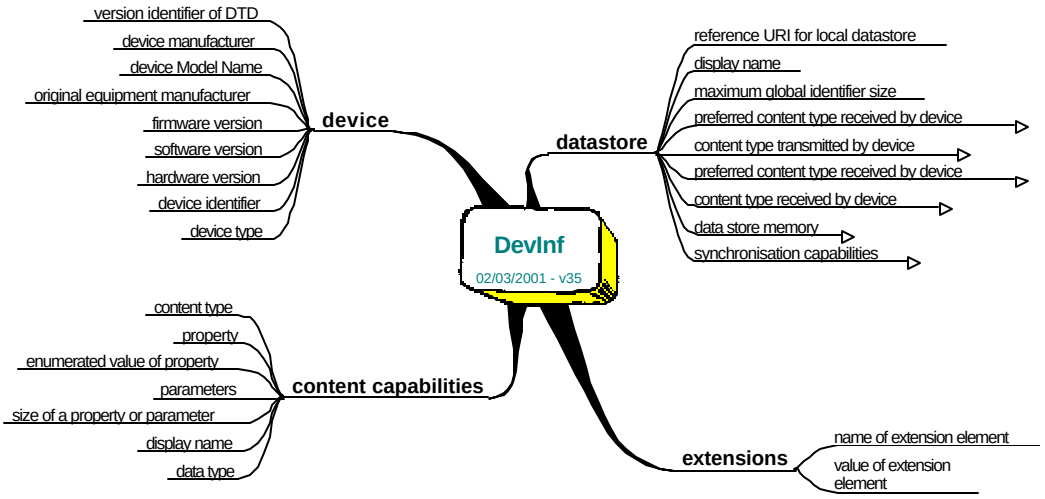

Figure 5 - SyncMLDevInf Specification

DevInf is purely concerned with the logical structure of application data. This is fine for phone book or diary entries, but the SyncML requirements do describe use cases involving the exchange of documents such as presentations. This would require exchange of information about the presentation capabilities of a device.

3.5 Universal Plug and Play

Universal Plug and Play[6] is an interconnectivity standard being proposed by Microsoft. It is aimed at device independent interconnection rather than device independent content but does overlap with CC/PP. Like CC/PP, UPnP uses XML in order to provide a general way of describing device capabilities rather than specifying vocabularies as it is expected that this will be done by device manufacturers. There are a number of example device descriptions such as a CD-Player[7] and a Web-enabled camera[8]. Interestingly, UPnP proposes that XSL as well as XML can be used so that the parent device can manipulate the device description in different ways, for example displaying device capability information as well as controlling device parameters.

4 Document Description

4.1 Printed Media Languages

Document language processors like Troff[9], TeX[10], LaTeX[11], and Lout[12] are designed to produce paginated, hard copy documents. Some of them describe the logical structure of a document e.g. headings, sections, paragraphs, figures etc. In documents the *logical structure* is distinct from the *physical structure* i.e. the size of text, the width of margins. Document processors convert logical structure to physical structure automatically during document processing. For example LaTeX uses algorithms to try to determine the optimum breaking of sentences in order to form paragraphs, the optimum placement and breaking of paragraphs in relation to pages and the optimum placement of floating objects in relation to where they are referenced. Parameters controlling the physical structure of the document can be inherited from templates for document styles or from specific instructions in the document. Document style templates are created for a specific type of media i.e. A4, letter or slides. Therefore in theory reformatting the document for a different media type should just require changing the document style template. However because authors typically add instructions controlling the physical structure directly to the document, changing the media will also generally requires changes to the document. Although these languages try primarily to capture logical structure,

the resulting documents often fail to separate the logical and physical structures of the document so are tied to specific output media.

Documents written in these languages can be distributed electronically by taking the output of the language processor and converting it to an electronic distribution format e.g. PDF or HTML via a different document processor such as Latex2HTML[13]. This takes the logical structure of the document and converts it to HTML so web browsers can read it. Often though HTML converters do not render the document exactly as the author envisioned so it may be necessary to alter either the input document or the output HTML in order to obtain the required results.

4.2 WYSIWYG Document Preparation

As computers have increased in performance, users have moved away from using document description languages to using WYSIWYG (What You See Is What You Get) document preparation systems such as Microsoft Word. These systems are simpler for naive users as they do not require the user to learn the syntax of the language. Also, the user can see what the final document will look like as they edit it. Unfortunately they allow users to edit both the logical and physical structure of the document. As users may be unsure of the difference, they often add physical structure information when really they mean to add logical structure information e.g. changing a font size as opposed to selecting some text as being a header. This means that compared to document description languages, documents are hard to reformat for other media types. Converters do exist for turning documents to PDF or HTML, although the HTML conversion is less than ideal.

4.3 Web Oriented Languages

The World Wide Web was originally envisioned as being a means of allowing multiple, heterogeneous clients to retrieve and display documents in a common language. HTML concentrates on logical document layout e.g. headers, titles and paragraphs for this reason. Increasingly though, many HTML pages contain physical document layout information meaning that pages are not compatible with all web clients. This has occurred for a number of reasons. Firstly designers who use the web were frustrated with the amount of physical document layout control available in HTML so they found ways of "subverting" HTML elements in order to perform tasks they were never intended to. For example tables are often used to achieve effects such as multiple column page layout. Secondly HTML browsers added support for physical layout tags such as the font tag to HTML in order to increase the variety of layout available on their browser. HTML authors have then adopted these tags and HTML has been forced to adopt physical layout as well as logical layout tags[14]. Thirdly WYSIWYG editors have become available for HTML which allow HTML authors to make the same mistake as with WYSIWYG documents i.e. specifying physical document layout when they should be specifying logical document layout and vice-versa. In some cases the editing tools themselves use physical layout in preference to logical layout. Some pages are so browser dependent it is not uncommon to see them displaying a "best viewed in browser x" logo or "best displayed at resolution x" logo[15].

The W3C is aware of these problems so has several activities aimed at resolving them. Firstly they have proposed that the physical structure description of pages should be separated from the page and placed in a stylesheet[16]. In this way different devices and media can be accommodated using different stylesheets, in a similar way to different document templates for printed media languages. The W3C also has an accessibility activity[17] which recognizes that content authors, browser creators and editing tool creators all have a part to play in ensuring the web is device independent. This accessibility activity will be discussed in more depth in a later section. Finally they have created a revised version of HTML called XHTML[18].

4.4 Electronic Publications

Currently there are three major standard formats for electronic publications: Docbook, PDF and the Open E-Book (OEB) Forum specification.

4.4.1 Docbook

Although it is possible to convert document layout languages or WYSIWYG documents in to HTML, or HTML documents into a form suitable for printing, the converted document is never as good as if the document had been produced for that media from scratch. It is possible to modify the results of translation programs by hand but this means the changes have to be applied every time the document is updated in the master format. Therefore there is a need for a document layout language aimed at text only electronic, web and hard copy media. Docbook[19] is a popular set of tags for describing books, articles and other prose documents, particularly technical documentation. It can annotate documents in quite a rich way; for example it can distinguish between screen shots and diagrams. It is defined using SGML and XML and has been used in the Linux Documentation project[20]. There are transformations for producing HTML or PDF from the XML version of Docbook using XSLT.

The original LinuxDoc format and another language called Latte[21] also allows the creation of documents targeted at text, HTML and PDF output. Docbook has now replaced LinuxDoc. Latte is less well known than Docbook and has a TeX like syntax rather than an XML syntax.

4.4.2 PDF

PDF[22] is an electronic document distribution format developed by Adobe to replace Postscript. Whereas Postscript was a full programming language, PDF is a page description format. This means PDF code is generally simpler, more consistent and predictable than Postscript. PDF also has the advantage that it is page independent meaning individual pages can be rendered without having to interpret the whole document specification. It uses vectors as opposed to bit maps, which means it scales well and can render high quality documents.

PDF is aimed primarily at capturing the documents appearance rather that its structure. This has a number of advantages. PDF is easy to create. It is better at typography and presentation than OEB, as it can render documents so they are identical to printed media. It is effective for technical publishing and foreign languages as it is possible to include non-standard characters in the document, therefore ensuring the reader can render them correctly. PDF also has a number of disadvantages. The format is controlled by Adobe and is proprietary. It only works well if the document is formatted for the device it is displayed on so if you have a PDA device, you are going to have difficulty reading documents formatted for A4 without excessive scrolling and zooming. It is not possible to extract the document structure and text from the PDF document, which means certain processing and searching operations are difficult. Finally PDF files also contain little structural information that can make searching or navigating them difficult.

4.4.3 Open Electronic Book Forum

The Open Electronic Book Forum[23] tries to set devices for electronic book readers and electronic book content. The OEB standard tries to capture document structure rather than appearance. OEB is based on XML and comes in two flavours: basic OEB is based on an XML compliant version of HTML 4.0 and can use CSS stylesheets (with some restrictions). Extended OEB is based on XML and requires the use of CSS in order to render unfamiliar mark-up. OEB also supports JPEG and PNG images as core media types and can optionally support other media types using a "fallback" file in case readers cannot process them.

OEB has some advantages compared to PDF. Firstly it should be better at coping with a wide range of devices than PDF as it separates the logical and physical structures of documents.

However this will depend on how authors mark up their publications just as it does in HTML. OEB should provide better backward compatibility than PDF does, and there is always the possibility of upgrading files by adding additional mark-up. It should also be possible to provide alternative renderings of OEB documents - for example Braille or audio - which is not possible with PDF. Finally, unlike HTML, there are DTD's for the basic and extended OEB formats enabling documents to be validated.

OEB does have some disadvantages. Firstly many publishers use proprietary formats like Quark for storing publications. Converting from these proprietary formats to XML is hard although tools are becoming available to help with the process. However, as noted in the section on WYSIWYG editors, tools can be used incorrectly so authors or editors must properly understand logical and physical document structure and the need to separate the two. Secondly although OEB is based on Unicode, there is no guarantee that readers will have the correct character set for displaying symbols or international Unicode characters. Thirdly there is no guarantee of the graphics capability of readers and as OEB uses a bitmap graphics format it degrades badly on less capable devices.

As PDF and OEB both have disadvantages and advantages, it is expected that publishers will need to use both PDF and OEB publication formats[24]. PDF is aimed at distributing documents electronically - for example journal papers - so they may be printed out or read on the screen of a PC. OEB on the other hand is targeted at electronic book readers that are aimed at replacing conventional books by allowing users to download E-publications electronically to the reader. In addition to dedicated E-Book readers, OEB reader software is also available for PDA's. For example MobiPocket[25] has produced reader software for Palm OS, EPOC32 and Windows CE devices.

4.5 Constraint-Based Layout

We have discussed how documents may be stored in document description languages, proprietary formats aimed at editing packages, web-oriented languages and electronic document formats. All these formats are currently in use today by different publishing and reading communities. In addition to these descriptions, there has also been research into an alternative way of specifying document physical structure so that it adapts more easily to different devices. This method is called *constraint-based layout*[26] and is related to work on constraint-based user interfaces that has been conducted since the early eighties. Here the document author describes aspects of the physical structure of the document as a series of mathematical constraints e.g.

```
page_height=header_height+text_area_height+footer_height
```

It is then the responsibility of software that is rendering the document to try to come up with a feasible solution that meets the document layout constraints, the constraints inherent in the device (e.g. screen size) and those specified by the user (e.g. preferred font size). Document layout languages such as LaTeX use similar optimisation methods in order to decide on the optimum breaking of lines or the optimum positioning of paragraphs on pages. However in LaTeX the optimisation algorithms are hard-coded into the document processor and use heuristics to solve the constraints efficiently.

A prototype web page browser for viewing documents using constraint-based layout is available on-line[27] and typical output from the browser for different screen configurations are shown in Figure 6, Figure 7, Figure 8 and Figure 9. The solvers used in this prototype use both local propagation methods (for continuous constraints) and finite domain solvers (for discrete constraints such as font size). Constraint satisfaction is computationally expensive compared to rendering a HTML document but can be achieved at a reasonable speed on a PC.

There are a number of problems with constraint-based layout. Firstly it is necessary for designers or editors to specify document layout as mathematical formulae that may be difficult. Secondly it is necessary for the browser to support constraint-based layout. Thirdly typically these methods use hierarchical constraints meaning different constraints have different "weights" of importance. This can lead to unexpected results from the solver. Therefore for the moment it is not clear whether constraint based layout offers a viable solution to the problem of device independence.

Figure 6 - Alternative Layout 1

Figure 7 - Alternative Layout 2

Figure 8 - Alternative Layout 3

Figure 9 - Alternative Layout 4

5 Accessibility

As previously noted the original intention of HTML was to make documents available on a number of devices. Due to the way HTML has been used and the way the language has developed HTML has not really lived up to this original design goal. Therefore the W3C has an accessibility activity[17] that aims to ensure that content, browsers and authoring tools all make the web accessible as possible to users with different needs and different browsing clients. The HTML Writer's Guild is also trying to promote web accessibility[28]. The term accessibility is often used to refer to users with special needs e.g. those requiring Braille or speech output but the methods do support device independence. However it is worth stressing that accessibility in reality covers any web browser other than the one used by the designer when they designed the page. The accessibility activity has produced a number of guidelines and these are summarised in Figure 10.

The guidelines are based on the idea that the only truly device independent form of content is text. Therefore any element essential to a page that is not text must have a text equivalent. Careful thought needs to be given to providing an equivalent with the same meaning for all

users[29]. Secondly they recognise that the physical structure of documents is device specific so should be separated from the document using a stylesheet. This allows the document to be rendered on multiple devices using different stylesheets. Finally they emphasise that content, authoring tools and browsing software must all obey the W3C standards in order to achieve accessibility.

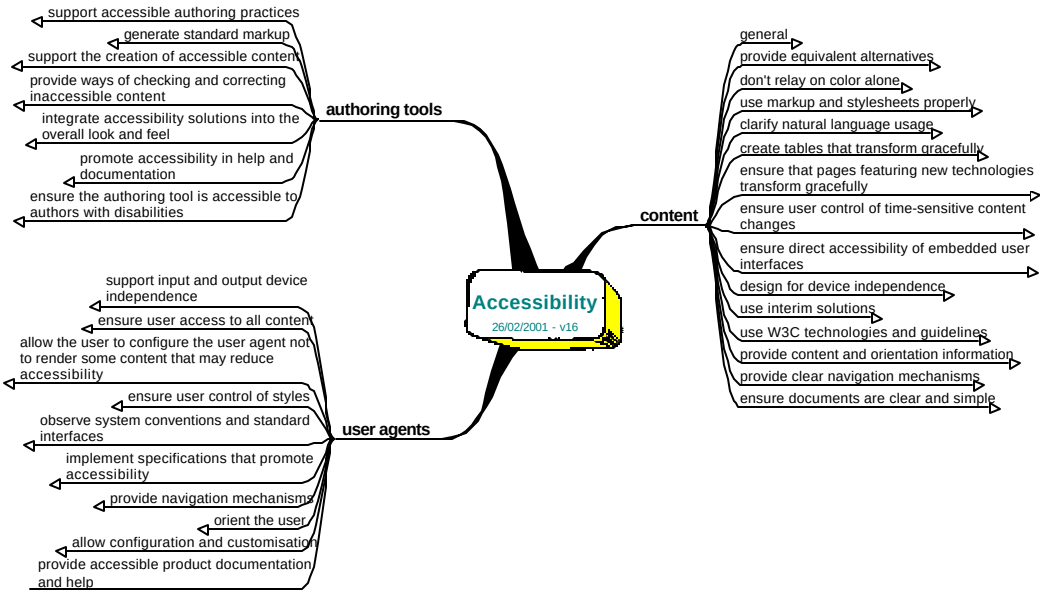

Figure 10 - W3C Guidelines for accessibility

Early on in the history of the web, improving web site accessibility would have solved the problem of device independence. Unfortunately as new web markup languages such as WML and HDML have arrived, this is no longer the case. One way around this may be to replace WML and HTML with a single language such as XHTML and this will be discussed later.

6 Web Applications

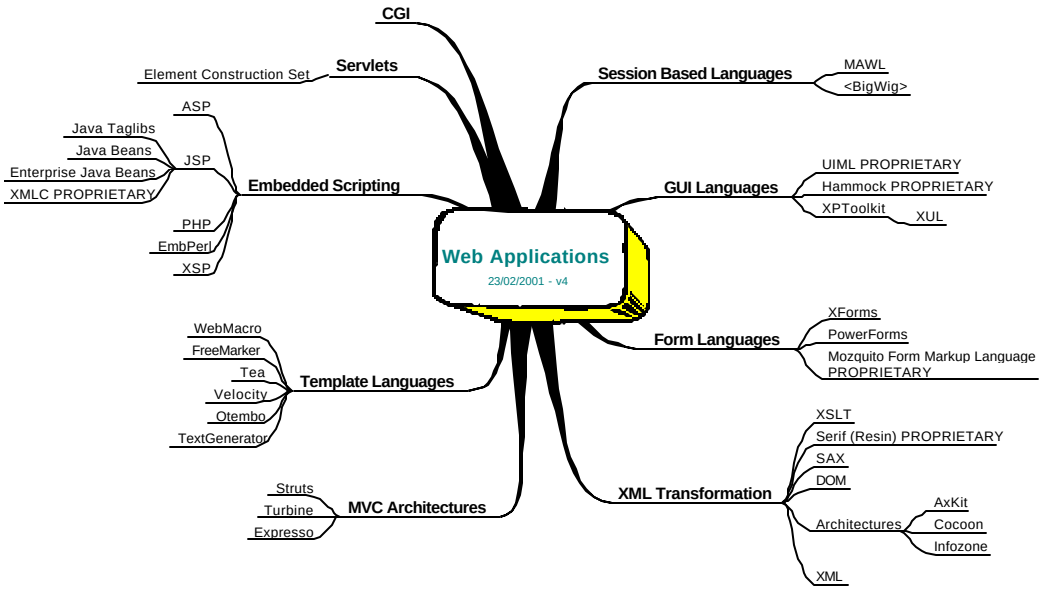

Figure 11 - Existing technology for web applications

Although the World-Wide Web was first created as a means for accessing documents, increasingly the web is used to access applications. Even when users are using documents they often obtain those documents via applications known as portals. Therefore in order to understand how to achieve device independence we also need to consider how applications are built as well as how documents are written. Figure 11 shows a framework for categorising different technologies involved in the creation of web applications. These technologies will be discussed in subsequent sections but there are also a number of surveys available on the web such as [30], [31] and [32]. One problem common in web application technologies is how to separate application logic from presentation. This is a very similar problem to the device independence problem of separating document content from document presentation.

6.1 CGI

CGI, the common gateway interface, was one of the first methods for creating web applications. It defines a communication standard between a CGI application and the web server. As it is a communication protocol rather than a language, CGI applications can be written with a variety of languages including C, C++, Java, Perl or Visual Basic. Perl in particular is a popular choice as it is simple, high-level and has many freely available extension libraries.

In its most basic form, CGI is inefficient at handling concurrent client requests. A CGI program needs to create a separate process for each user request. Each process terminates as soon as the data transfer is complete. Spawning a separate program instance for each client request takes extra time so CGI programs are not really suitable for high-traffic applications that need to process a large number of client requests. Recently FastCGI[33] and the Apache mod_perl module have addressed these inefficiencies.

CGI, along with many other approaches, also has two other problems: the difficulty of separating presentation from logic and the way that HTTP, which is a stateless protocol, makes it difficult to control the sequence of pages in a session. It is hard to separate presentation from logic because CGI code often contains embedded HTML markup. This is not ideal as changes to the presentation of the application, ideally done by a designer, requires changes to code which must be done by a programmer.

In order to understand the problem about controlling the sequence of pages in a session, consider an example where we may need to make sure that a user has supplied their details before they are allowed to use a web application. In a conventional program this could be achieved simply by flow of control. In a web application the flow of control is determined by the order the user retrieves pages so it is necessary to use other approaches. Netscape realised this problem early on and proposed cookies as a mechanism to give the client a state that could be set by the server. Typically programmers solve this problem using a mixture of hidden form input values, query parameters and cookies to store the state of the web client. This is necessary because not all browsers support cookies or users may turn them off. In order to provide security or control of session sequence, pages have to call code to interrogate the client's state before it is allowed to access that page as well as generate the page accordingly. This is not an ideal solution as it leads to unnecessary code complication or if it is neglected can lead to possible access control issues.

6.2 Servlets

Servlets[34] were developed by Sun to address the efficiency problems of CGI. A servlet is a piece of server side Java code run in a server application to answer client requests normally using HTTP. When a servlet is called for the first time, it is loaded into memory. After the request is processed, the servlet remains in memory and will not be unloaded until the server is shut down. This means that Servlets can be called efficiently many times without the

overheads of creating new processes as in CGI. The Java servlet API is threaded so that each instance of a servlet can handle multiple requests simultaneously.

However, just like CGI, Servlets typically contain embedded HTML. They have request and response API's for dealing with form variables, query strings or cookies but as in CGI programmers have to work at quite a low level in order to control the session sequence of a web application.

Servlets are not necessarily page-centric i.e. there does not have to be a one-to-one correspondence between pages and Servlets. This means that pages can potentially share common code or resources, such as keeping database connections persistent via a connection pool.

6.3 Embedded Scripting Languages

As well as CGI and Servlets, a number of embedded scripting languages have been developed for creating web applications. Embedded scripting languages allow you to put snippets of code inside HTML. This was originally done to avoid the problem in CGI and Servlets that the application code often contains HTML code. However in an embedded scripting language the HTML code contains application code producing the same problem but the resulting pages are generally shorter than the script versions, so these languages do lend themselves to faster site development. In addition, early embedded scripting languages suffered from being page-centric. Because code is embedded on a page, it is hard to share code or resources between pages. For example in ASP, it is common to have to open a database connection on a per-page basis that is inefficient. Newer versions of these languages have recognized this problem and provide workarounds.

Examples of embedded scripting languages include Java Server Pages, Active Server Pages, Professional Home Pages and EmbPerl. EXtensible Server Pages, to be discussed later, is an XML variation on Java Server Pages.

- *Java Server Pages (JSP)*[35] are based on Java. Pages are transformed into Java Servlets and compiled which makes them very efficient but also means they generate rather confusing error messages. Recent versions of JSP allow JSP pages to call objects known as Java Beans, which provides better support for separating presentation from logic.
- *Active Server Pages (ASP)*[36] has been developed by Microsoft and can use JScript, Perl or VBScript as a scripting language. On the Windows Platforms ASP is based on COM, Microsoft's Component Object Model, so scripts can call COM components in a similar way to JSP calling Java Beans. The COM architecture is acknowledged as being rather complicated and slow. ASP also supports easy construction of database applications via Microsoft's Active Data Objects and SQL.
- *Professional Home Pages (PHP)*[37] is an open-source embedded scripting language based on C. Like ASP it provides support for SQL so it can be used with databases. Newer releases of PHP provide functionality like connection pooling in order to try to optimise database access.
- *EmbPerl*[38] is an open-source embedded scripting language based on Perl. It is newer than the others so has less support. Compared to the other languages it has an idiosyncratic syntax.

6.4 Template Languages

Template languages, like embedded scripting languages, take a page-centric approach, allowing the developer / page designer to access logic within HTML pages. In contrast to embedded scripting languages they remove the ability to put raw Java code in a page, supposedly to improve the separation between presentation and logic. They also provide support for elements like common headers and footers or repeated elements. They are less widely used than embedding scripting languages:

- *WebMacro*[39] is an open source scripting language that works with HTML and XML. It is based on Perl so is powerful but uses a class analyser that has quite large runtime overheads. The templates and the Java objects communicate using a shared area of code / data known as a context.
- *Freemarker*[40] is an open-source scripting language. In Freemarker data objects are passed to the template in tree form. This means that the logic part of a Freemarker application is generally more complicated than WebMacro as data needs to be converted into a special format but the resulting templates contain fewer programming constructs.
- *Tea*[41] is a template language developed by Disney's web division and used in a number of serious commercial portals. It deliberately adopts a very restricted set of programming constructs in order to avoid errors and allow designers to edit pages. Tea also has an integrated development environment called Kettle. Because Tea has been used for serious commercial development it is stable and has good documentation.
- *Otembo*[42] has some of the features of an embedded scripting language as you can place SQL queries directly into templates. Otembo has very little documentation compared to the other template languages.
- *Velocity (VTL)*[43] is an open source template language developed by Apache based on Java. VTL can be used both as a standalone utility for generating text or mark-up as well as an integrated component of a web application. It provides a wide range of common programming language and macro constructs.
- *TextGenerator*[44] is a general language for generating text or mark-up. The output language can be HTML, WML, XML or Latex. Like Velocity, it has a wide range of common programming language constructs. Unfortunately there is little example code.
- *TRiX*[45], [46], [47] is a template language developed by Hewlett Packard Laboratories.

6.5 Session Based Languages

CGI and Servlets take a script-centric approach to developing web applications. Embedded scripting languages and template languages on the other hand use a page-centric approach. As we have already noted, session sequence control can be difficult with page-centric approaches. There are currently two examples of session-centric languages that have been developed to try to overcome these problems. In these languages, the application logic controls the display of presentation rather than the client requesting a page. This can simplify the construction of web applications:

- *MAWL*[48] stands for "the Mother of all web languages". It is a research language developed by ATT around 1995. MAWL programs are organized as a set of sessions that define entry points to a service. The sessions represent the logic of the application, and control the rendering of templates that form the presentation layer of the application. The sessions are described in a programming language similar to C and presentation is described in a language derived from HTML called MHTML. Separating logic and presentation in this way allows document templates to be syntax-checked which helps ensure content can be rendered correctly. It also helps support device independence and early MAWL demonstration applications were implemented for both graphical and voice browsers.
- *<Bigwig>*[49] is an academic project that takes inspiration from MAWL based at the University of Aarhus, Denmark. As well as being a session-centric language, <Bigwig> incorporates a number of novel ideas including the idea of client-side form validation using a language called PowerForms.

6.6 Form Based Languages

Another area of complexity in web applications is the way that data is submitted to the server via forms. The validity of the form must be checked on the server; this can be slow but also adds complexity to the application. If the information entered was invalid somehow the form needs to be returned to the client or the user will have to type everything in again. This has to

be done via query strings or hidden form variables that adds complexity as they were not really designed for this purpose. Checking form validity can be solved using client-side JavaScript. However JavaScript was not designed with this application in mind so in some ways it is too low-level for this kind of task. Some early suggestions for a XML form language are described in [50]. Recently a number of specialist languages have been developed:

- *PowerForms*[51] is an XML form description language developed as part of <BigWig>. Powerforms provides support for validating user input but not device independence. A form can specify data validation rules both for individual input fields and for interdependencies between those fields allowing very precise specifications to be created. PowerForms descriptions are compiled to produce HTML / JavaScript code that implements the form.
- *Form Markup-Language (FML)*[52] is a commercial language that has been developed by Mozquito Technologies as a precursor to XForms. A form in FML can consist of a number of pages in a single file, just as a deck in WML can consist of a number of cards. FML provides more limited input validation than PowerForms via several predefined input types such as number, date, URL, domain, email address and expire data. It also provides support for the calculation and display of field values at run-time, for example providing a total in an expense application. It provides constructs called *toggles* that allow the programmer to activate and deactivate objects within a document. This can be used to hide fields that are not relevant. It also provides a mechanism to declare event handlers to process specific form events.
- *XForms*[53] is a standard being developed by the W3C that aims to replace the current XHTML form controls. An XForms description consists of a device independent model, device independent user interface details and device specific user interface details. It also defines limits and restrictions that apply to the model items when the form is being filled and relationships and dependencies, to support input validation in a similar way to PowerForms and FML. It is anticipated that XForms will become dominant if eventually it becomes supported natively in web browsers.

6.7 User Interface Based Languages

As well as forms, there are also a number of languages for creating more complex event based user interfaces in web browsers:

- *UIML*[54] is an XML compliant declarative language that provides cross platform user interfaces. It is a commercial product developed by Harmonia[55]. It tries to achieve platform independence by separating user-interface code from non-user interface code so that common features of interface code for different devices can be factored out. This is done by splitting the user interface description into five sections: description, structure, data, style and events. In addition, it supports local events as well as external events that propagate outside the interface description. Internal events can be used to perform tasks like real-time updating found in FML. UIML descriptions can either be written in general UIML or device-specific UIML where the vocabulary is closely aligned with a specific device. Device-specific UIML can then be mapped to general UIML using a transform language. At the moment only beta versions of a Java and WML renderer exist.
- *Hammock*[56] is a commercial product developed by a company called OOP. It tries to tackle the problem of cross-platform UI's like UIML, but instead of using a declarative language it provides a Java library that is very similar in structure to Swing. At runtime, the library calls a renderer that converts output into a target form such as HTML. At the moment only a HTML renderer exists although the intention is to support a WML renderer as well. Hammock uses four types of object: an application object, which corresponds to a Java servlet, a page object, which corresponds to a viewport or a deck of pages, a form object, which is where the bulk of the coding is done and GUI objects that sit in form objects.
- *XPToolkit*[57] has been developed as part of the Mozilla browser as a collection of loosely related facilities which aim to provide a platform independent API for user interfaces. The

XPToolkit specification describes various objects that are wrapped in a package described by an XML compliant language called XUL[58]. For example the Mozilla browser is a package described in XUL hence it is possible to customize the browser by editing the XUL description. Mozilla take the view that a XUL description will not be truly cross-platform. Currently XPToolkit is aimed at PC's and there are no plans to extend it for phones or PDA's.

6.8 Componentisation

As noted in a previous section, one problem with JSP is the difficulty in separating logic from presentation. Two technologies, custom tag libraries and using JSP's with Java Beans have been developed to address this problem:

- *Java Taglibs*[59] are collections of custom tags. The tags can be used to replace common items of logic in JSP pages. A custom tag can call elements called actions that can create and access programming language objects and change the output stream. There are a number of pre-existing tag libraries such as the Jakarta Taglib[60] containing a number of tags for processing XML input and applying XSLT transformations as described later.
- *Java Beans*[61] are simply Java classes that adhere to the naming and design conventions laid down in the Java Beans API. A Java Bean has a number of properties, which are member variables accessed using get and set methods. The idea of defining objects in this way is they then naturally form reusable components with a very clean and well-defined interface. JSP pages can be simplified by inserting references to Java Beans rather than raw Java using special tags.

There have also been two technologies developed aimed at separating presentation from logic in Servlets:

- *XMLC (XML compiler)*[62,63] is a part of a commercial application server called Enhydra[64] produced by Lutris. It is an application that compiles HTML or XML documents into Java objects using DOM. The aim of this is to overcome the problem of having to include embedded HTML or XML in servlet code. Instead the servlet calls to the object created by XMLC. As with other auto-translation utilities, problems can occur if it is necessary to make alterations to the XMLC output as those changes need to be made every time the object is recompiled.
- The *Element Construction Set (ECS)*[65] is a Java library produced by Jakarta that provides an interface for generating HTML or XML from a servlet without having to resort to println commands. It can be used to export mark-up for a number of target languages. It would be possible to use an approach similar to ECS to implement programmatic device independent interfaces, as used in HawHaw or DotNet that will be discussed later.

6.9 Architectures

Many applications are based on the three tier (also known as Model 1) pattern which divides the application into a data source (bottom), middle and client tier[66]. There is an alternative pattern though called the Model-View-Controller[67] (also known as Model 2) for separating application presentation (the view) from application logic (the model) has been in use since the early 1980's. It has recently been identified as a key pattern for developing web applications. In the MVC architecture housekeeping activities, such as maintaining database connections, are typically performed by the controller. A simplified model of the MVC architecture is shown in Figure 12. MVC architectures are primarily associated with the Java 2 Enterprise Edition (J2EE)[68] and hence JSP, Servlets and Java Beans although it is possible to implement them using ASP or PHP. In order to simplify the process, a number of open-source MVC architectures have been developed:

- *Struts*[69] is a JSP implementation of an MVC architecture. It provides flow control for JSP based web applications, a controller servlet, JSP custom tag libraries, and various utilities to support XML and internationalisation. In Struts, the controller consists of a

number of Servlets that are configured by defining a set of mappings against a request URI. The Servlet is responsible for performing the desired business logic and then dispatching the control to the appropriate view component to create the response. Struts provides support for different presentation techniques including application specific custom tags, page composition with includes and image-rendering components for dynamically generated images.

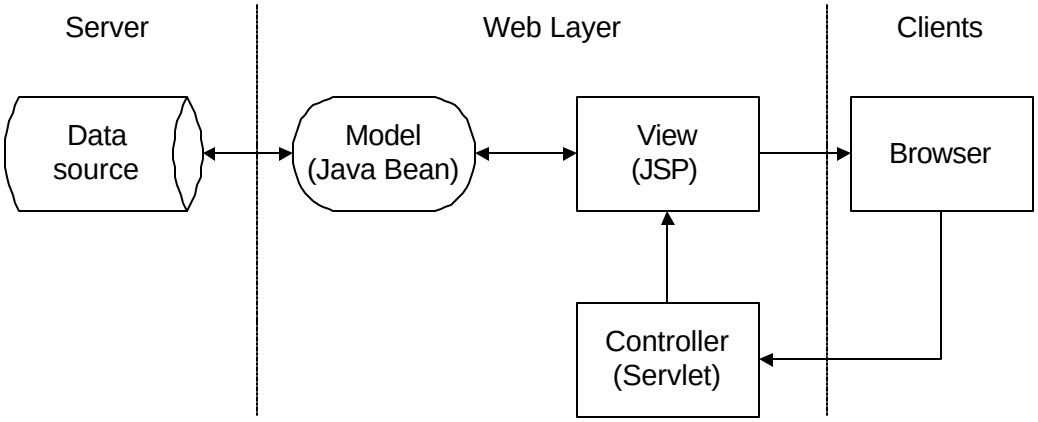

Figure 12 - Model View Controller Architecture

- *Turbine*[70] is similar to Struts but offers a richer feature set. It is intentionally not tied to JSP but can also be used with Cocoon, ECS, WebMacro, Freemarker or Velocity. It contains a collection of reusable components aimed at easing server side development. Turbine provides MVC flow control, integration with Object Relational Mapping tools, a job scheduler, localization services, caching services, security and many other features. It also has support for XML. Typical applications of Turbine include shopping carts, link directories, project bug tracking systems and portals like JetSpeed.
- *Expresso*[71] is a layered controller architecture. It separates the controller into a database layer, a framework layer and an application components layer. The framework layer provides security, job control, logging, database connection pooling and caching, event notification, configuration values and database objects. The application components layer provides functions such as ePortal content management, discussion forums, search engines, data warehousing toolkit and online forms. Expresso is open source although it is commercially funded. There is an Expresso-XML object that provides support for XML / XSLT.

6.10 XML Transformation

XML, the EXtensible Markup Language has evolved from both SGML (Standard Generalized Markup Language) and HTML (HyperText Mark-up Language). XML is a data definition language designed to structure data and describe information. The author of an XML document is able to create their own tags and is therefore freed from constraints of the predefined tags found in HTML. A DTD (Document Type Definition) is used to define the vocabulary and syntax of the user-defined tags. There are a number of device independent solutions based on the idea that a page can be marked up using XML and then transformed to a number of different output formats. There are several tools available to perform this transformation:

- *XSLT*[72], the *EXtensible Stylesheet Language Transformations* is an XML compliant language created to solve the problems of how to transform XML documents into another type of document. XSLT is a recursive declarative language that describes rules that can transform XML documents. An XSLT processor reads in the XML and XSLT stylesheet and based on the instructions in the stylesheet outputs a new document (e.g. in XML,

HTML or WML). XSLT is a powerful and fast way of manipulating data. However there is a general feeling that it is a complicated language[73] to use so in particular designers may not want to write XSLT unless it is done automatically in a manner similar to WYSIWYG HTML editors. XSLT and XML can support device independence allowing the creation of a site where each page of content is described by a single source document in XML. XSLT stylesheets are then used to transform this document for multiple target devices. In practice though, defining the tags and transforms in such a way that there is a single stylesheet per device has been found to be difficult. Prototype sites often end up having a stylesheet per device per page that is clearly not ideal.

- *SAX (Simple API for XML)* is a way of making XML data accessible in Java. It is an event based XML parser that processes the document sequentially. This is very memory efficient, as the parser does not need to keep a copy of the complete document in memory. SAX is useful because it may be simpler to specify certain types of transformations in a procedural language like Java than in a declarative language like XSLT.
- *DOM (Document Object Model)* is a way of accessing XML and HTML documents. A DOM parser uses a tree-based representation of an XML document and can interface to a number of languages such as Java or C++. Unlike SAX, it usually keeps the entire document in memory but it can support more complex transformations. DOM can be very expensive in terms of the memory required to hold the documents and hence the speed it can then process those documents.
- There are two Perl-like languages for XML transformation that use XPath. *XMLScript*[74] has been developed by a company called DecisionSoft. *XPathScript*[75] has been developed as part of AxKit. Both these languages are non-side effect free, unlike XSLT. This has advantages and disadvantages but this allows the languages to use conventional variables and functions.

6.11 XML / XSLT Architectures

Sun has suggested there are three approaches for using JSP with XML / XSLT in web applications: *single pipeline, multiple pipeline* and *combination*[76]. In the single pipeline approach shown in Figure 13, the JSP page generates XML that is transformed to multiple output languages via multiple XSLT stylesheets. By contrast in the multiple pipeline approach shown in Figure 14, there is a different JSP page per device but backend logic can be shared between pages using Java Beans and custom tag libraries.

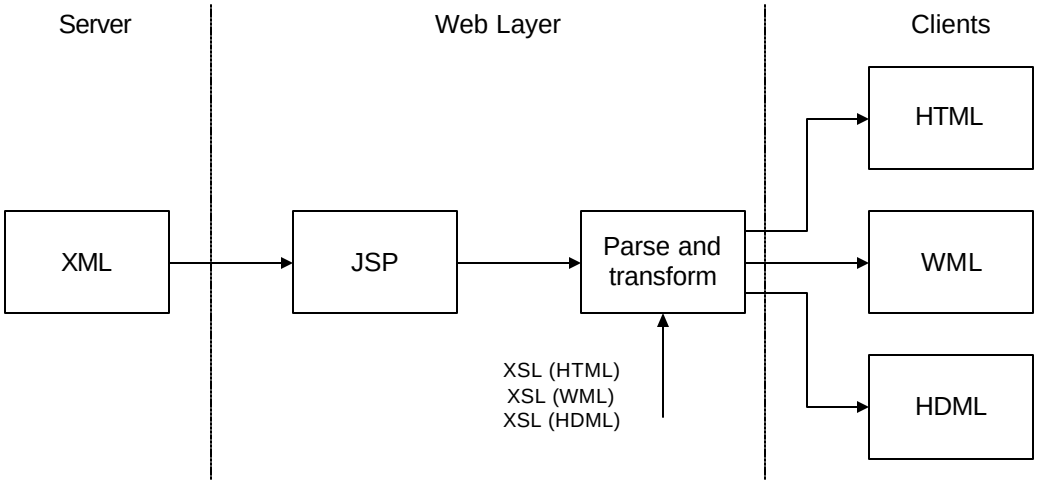

Figure 13 - Single Pipeline

In a combination architecture, you can use a mixture of these approaches e.g. one pipeline per language but styling for different dialects of languages. Clearly the multiple pipeline approach

requires a lot of re-authoring, but if we have separated logic from presentation effectively it may be less complicated than the single pipeline case if we have one stylesheet per page per device. In addition the multiple pipeline approach may be more performance efficient particularly if the pages being generated are highly dynamic so there is no advantage in caching them - for example displaying users bank accounts.

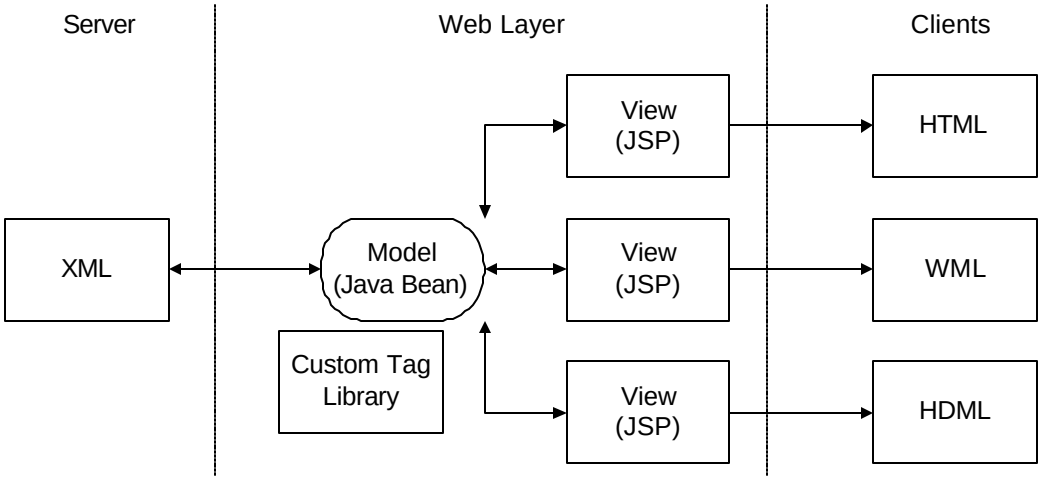

Figure 14 - Multiple Pipeline

Three architectures, Cocoon, AxKit and Infozone have been developed as implementations of the single pipeline approach.

- *Cocoon*[77] was developed by Apache for publishing XML to multiple target devices. It provides caching in order to speed up document delivery. It uses XSP, the EXtensible Server Pages Language, an XML compliant version of Java Server Pages in order to dynamically generate XML on the fly. XSP allows static XML data with Java fragments. Like other embedded scripting languages, XSP suffers as content and logic are placed in the same file. Cocoon users have suggested that stylesheets could be used to separate data from logic in XSP pages but this seems unnecessarily complicated.
- *Axkit*[78] is similar to Cocoon but written in Perl rather than Java. It uses Perl and has support for XML, XSLT and XSP as well as a new language called XPathScript[79] that is a XML transformation language based on Perl and XPath. The Axkit project also intends to develop stylesheet languages based on SAX rather than XSLT. Such languages would be very efficient in terms of processing and memory usage. This means they would have potential for being used for client-side styling on mobile devices as well as on servers.
- *Infozone*[80] is similar to Cocoon. It provides some additional features compared to Cocoon and Axkit such as a content management system that claims to provide hierarchical document storage, document versioning support and user management and workflow. As well as XSLT Infozone also provides Lexus, an implementation of the XML query-update language. From the documentation it seems the XML query-update language is very similar to XSLT except that you do not have to copy the source document; instead you either add, alter or delete nodes from the source document. Currently the Infozone project is much less advanced than Cocoon or Axkit.
- *Serif*[81] is part of a commercial application server called Resin produced by a company Caucho. Like Cocoon it allows pages to call an XSLT processor but uses a proprietary language called XTP rather than XSP.

Using XML and XSLT for device specific delivery has some advantages. Firstly, different WAP devices vary in the way they implement the WAP standard. For example Nokia and Phone.Com WAP browsers implement selection lists in different ways. Using XML / XSLT,

it is possible to style documents for different types of phones. However targeting content at specific phones increases the number of transforms required, and the process is difficult because there are no formal specifications of how individual phones interpret the WML standard. Another advantage of XML / XSLT is they allow the amount of text delivered to different devices to be controlled, assuming the original content has been marked up correctly. For example mobile devices may just want to receive the abstract of a paper whereas PC clients might want to receive the whole paper. Similar mechanisms can be used to specify which images should be displayed on which devices.

However there are also possible disadvantages with XML / XSLT. Firstly when creating content it is possible that people will just sit down and start tagging up arbitrary XML and writing stylesheets to style XML for different devices. This leads to a situation where there are individual stylesheets for each device viewing each page. Of course then it is possible for authors to go through the stylesheets and try to find common transforms and then place them in central include files. Ideally though, the starting point would be defining an XML compliant device independent authoring language. If this language was designed using a top down approach (rather than the bottom-up approach just described), it would minimize the number of transforms needed it would be possible to validate the content of documents. This is the same approach as has been taken with Docbook. In addition XSLT might find more acceptance if there was some kind of WYSIWYG authoring tool for transforms although how this would work is not clear.

7 Other Approaches

7.1 Transcoding

Currently one common approach to the problem of displaying Internet content on multiple devices is transcoding[82]. Here servers or proxies retrieve HTML content from the Internet and convert it to a form suitable for the device e.g. WML. Some companies have made these transcoding portals available to users over the Internet[83] and produce results as shown in Figure 15 and Figure 16. Transcoding has the advantage that there is no need to re-author any content. However it tends to be an unreliable solution as there is no guarantee the transcoded content will be usable on the target device. This is due to a number of reasons. Firstly the original HTML may not meet accessibility guidelines so for example site navigation may be impossible without displaying images. Alternatively the site may be dependent on frames or the site may use tables to organize its physical layout. Secondly web applications are unlikely to work as most WAP gateways do not support the use of cookies and most transcoding proxies cannot cope with form interaction. Thirdly there is a 1.4Kbyte limit on the size of compressed pages on WAP phones. This means the transcoding proxy must split up pages so they meet this criterion. As this is done automatically, this may degrade the sites navigability or understandability.

Unfortunately there are no open source implementations of transcoding proxies but some implementations of intelligent proxies such as Muffin[84] or Proxomitron[85] do exist. It might be possible to implement a transcoding proxy based on one of these architectures. There are a number of commercial transcoding proxies. IBM has done research on an architecture called Web Based Intermediaries[86] and produced a product called WebSphere transcoding publisher[87]. This provides standard HTML to WML transcoding as well as XML with XSLT styling. The IBM product can also style pages based on external annotations[88].

Figure 15 - XIFT transcoding applied to HP.com

Figure 16 - DeckIt transcoding applied to invent.hpl.hp.com

Another company called PyWeb[89] have taken an alternative approach in their DeckIt[90] transcoding proxy. PyWeb provide a transcoding service with a difference: the proxy can style the content for WML devices using special tags that are embedded as HTML comments. Using DeckIt, website authors can create WML pages by adding some extra tags to existing web pages. This is much simpler than learning new complex languages such as XML and XSLT but this solution does not scale well for multiple device types so whereas this may be okay just to provide WML support that still means the website does not support interactive TV, E-Book readers, etc.

7.2 AvantGo

AvantGo[91] is a very popular solution for making Web content available on Palm OS, Windows CE and WAP devices. A sample AvantGo page is shown in Figure 17. Currently there are over 2 million users registered AvantGo users in 8 different countries with access to over 600 different content providers. AvantGo uses a channel model where the user subscribes to a number of web sites. Pages from those sites are then retrieved when the user performs a synchronisation operation between the handheld device and their PC. Retrieval occurs via the AvantGo proxy that performs a transcoding operation stripping out any unnecessary HTML elements. The web pages are then cached on the handheld, avoiding the latency problem inherent in wireless communication. Instead of using a new mark-up language like WAP, AvantGo uses a subset of HTML i.e. no frames, no layout using tables and no JavaScript. This means that it has been very easy for existing web sites to provide AvantGo content. In addition because AvantGo caches pages it does not have any of the delays that WAP has. Solving the latency problem in this way has also increased compatibility with existing web content. The WAP version of AvantGo is less successful than the PDA version as it suffers from the same delay in content retrieval inherent in WAP devices.

AvantGo also provides support for forms although they are only submitted to the server on synchronisation. Despite this limitation it is still possible to create some useful web services. AvantGo can provide automatic form completion for personal details in a secure manner. Although there is variation between the capabilities of Palm devices and Windows CE devices, the AvantGo style guide[92] suggests that content authors aim content at the lowest device in order to make sure content is available on all devices.

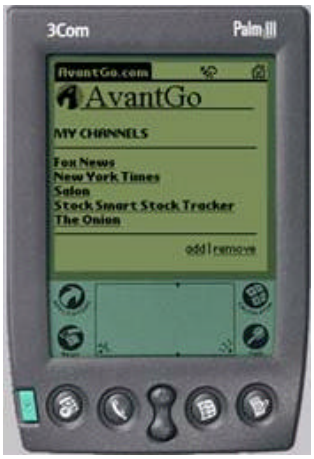

Figure 17 - AvantGo in use on a Palm OS handheld

7.3 Web Clipping

Web clipping[93] is Palm Computing's own proprietary mobile access technology. Palm was one of the first companies to release a wireless enabled handheld in the form of the Palm VII and web clipping was initially developed for this device. To build a web clipping application authors create a PQA file in a language similar to HTML that specifies how to extract information from an existing page on the Internet and how to create a new page based on that information. The page is then compiled to produce an application that can be used on the Palm device. When the user invokes the application, the device sends out a request to a proxy server. The proxy server then retrieves the requested web page from the Internet and discards as much unnecessary information as possible. This is because charges on the Palm VII packet wireless network are on a packet usage basis, so it is desirable to try to minimize the amount of information sent to the client. Compared to AvantGo, this approach seems slightly more complicated but it does allow the customisation of wireless applications and is particularly suited to dynamic content applications e.g. retrieving stock quotes, maps, news etc as shown in Figure 18. As Palm VII's are only available in the US, web clipping does not yet have much regionalized content.

Figure 18 - Palm Web Clipping Application

Oracle has developed a similar approach to web clipping for its Portal-To-Go and Application Server Wireless edition platforms[94]. Here when a client requests a document, the server produces an XML document containing query descriptions. The server takes the queries and uses them to retrieve web pages. The web pages are converted to XML, and then integrated into the original document. This document is then styled using XSLT for the client. There is also an open-source Perl Library, News Clipper[95], which can perform similar conversions although not aimed specifically at mobile clients.

7.4 ASP+ DotNet

DotNet[96] is the name given to Microsoft's new E-Services initiative. Microsoft is proposing two methods for supporting mobile devices in DotNet[97]. Firstly they are going to provide an implementation of XSLT that can be called from ASP called XSLISAPI[98]. This is aimed primarily at static content. As well as XSLT, XSLISAPI also provides support for caching the converted pages and identifying the capabilities of the browser. Secondly DotNet will provide a new version of ASP called ASP+. ASP+ provides a new, event driven approach to writing web applications called WebForms[99]. This means ASP+ has some similarities to Hammock and UIML, as well as providing a level of abstraction to hide some of the details of managing sessions. There is a mobile device variant of WebForms called MobileForms[100] aimed specifically at PDA and WAP devices. Developers wanting to target both PCs and mobile devices will need to produce both WebForm and MobileForm views, but it should be possible to share the logic component of these applications.

7.5 PHP HawHaw Library

The HawHaw [101] library is an open-source library for developing websites in PHP that can be viewed on WAP phones, iMode phones, HDML phones, Palm and Windows CE PC's running AvantGo and PC's using standard web browsers. It works by supplying a set of methods to create content. When a device requests a page, HawHaw looks at the User Agent string and converts the methods into the correct tags for the device. The resulting sites do not look ideal on a PC as they look styled for WAP as shown in Figure 19. However the library code is short and concise and it supports a lot of devices without the need to write any additional stylesheets. BT Ignite has used Hawhaw to create a network monitoring application. HawHaw uses a programming approach so it is not really suitable for web designers, but it is simpler than other programming solutions such as XML / XSLT or J2EE. An example HawHaw page is shown on a PC browser in Figure 19 and on a WAP browser are shown in Figure 20.

Figure 19 - PC Client viewing HawHaw page

Figure 20 - WAP Client viewing HawHaw page

There is a commercial product similar to HawHaw but based on Java called AVIDRapidTools[102]. This supports a wide range of devices such as WAP, iMode, Palm, Windows CE and PCs.

7.6 XHTML / CSS

The original solution proposed by the W3C to device independence was for all devices to support XHTML and cascading stylesheets. Then it was realised that XHTML is too complicated for some devices, so the XHTML specification was divided up into modules[103]. Devices must support XHMTL-base but have the option of choosing which other XHTML

modules to support. Using XHTML for delivery to a number of devices has a number of advantages. Firstly it means that all devices are using a common language so content only has to be authored once. Secondly the language is based on HTML, so existing HTML authors will be able to convert to the new language much more quickly (although they need to get rid of some bad habits). Thirdly XHTML and CSS separate presentation from content, so if a page is being displayed on a different device it can either chose to use the default stylesheet, use a stylesheet specifically for that device or use no stylesheet at all. Fourthly unlike XML / XSLT, styling is done at the client rather than the server so there is less loading on the server. Fifthly all clients receive an identical XHTML file, so that files can be exchanged between devices rather than having to re-request files from the server so they can be styled for the target device.

8 Conclusions

Hopefully this report has given an insight into the advantages and disadvantages of present technologies for delivering static and dynamic content to a variety of devices. As a result of this survey it is possible to make a number of observations. Firstly none of the currently available technologies is a perfect solution as they all have advantages and disadvantages. Secondly true device independence will not just require changes by the client device manufacturers but also static content authors, dynamic content programmers, static and dynamic content authoring tool creators as well as the software companies providing the server environment and associated tools. Thirdly technical superiority is not the only measure of a successful solution; for solutions to be successful they must also be widely adopted.

9 Index

10 References

[1] User Agent Strings, http://www.mozilla.org/build/user-agent-strings.html
[2] CC/PP, http://www.w3.org/Mobile/CCPP/
[3] RDF, http://www.w3.org/RDF/
[4] UAProf, http://www.wapforum.com/what/technical.htm
[5] SyncML, http://www.syncml.org
[6] Universal Plug and Play, http://www.upnp.org/
[7] CD-Player sample device template, http://www.upnp.org/download/CDPlayer-1-2001Feb.doc
[8] Axis Web-Enabled Camera, http://www.axis.com/products/documentation/UPnP.doc
[9] Troff, http://www.gnu.org/software/groff/groff.html
[10] TeX, http://www.tug.org/
[11] LaTeX, http://www.latex-project.org/
[12] Lout, http://snark.ptc.spbu.ru/~uwe/lout/
[13] LaTeX2HTML, http://cbl.leeds.ac.uk/nikos/tex2html/doc/latex2html/latex2html.html
[14] Physical versus logical markup, http://www.dantobias.com/webtips/logical.html and http://www.cs.williams.edu/~cs105s00/outlines/CS105_44.html

[15] The Myth of 800x600, http://www.webreview.com/2001/03_16/webauthors/index01.shmtl
[16] Cascading Style Sheets, http://www.w3.org/Style/CSS/
[17] Accessibility, http://www.w3.org/WAI/
[18] XHTML, http://www.w3.org/MarkUp/
[19] DocBook, http://www.docbook.org/
[20] Linux Documentation Project, http://www.linuxdoc.org/
[21] Latte, http://www.latte.org/
[22] PDF, http://www.adobe.com/epaper/main.html
[23] Open Electronic Book Forum, http://www.openebook.org/
[24] XML and PDF, http://www.impressions.com/resources_pgs/SGML_pgs/XML_PDF.pdf
[25] MobiPocket, http://www.mobipocket.com/en/HomePage/default.asp
[26] Constraint Based Layout, http://www.cs.washington.edu/research/constraints/web/mmJournal.html
[27] Constraint Based Web Browser, http://www.csse.monash.edu.au/~rlin/CBN/index.html
[28] HTML Writer's Guild AWARE, http://aware.hwg.org/
[29] Using ALT tags, http://ppewww.ph.gla.ac.uk/~flavell/alt/alt-text.html
[30] CGIs versus Java Servlets, http://rain.vislab.olemiss.edu/~ww1/homepage/project/mypaper.htm
[31] Surveying the landscape, http://xmlc.enhydra.org/software/documentation/competing_frameworks/competing_frameworks.html
[32] Java and XML publishing Frameworks, http://www.runtime-collective.com/static/runtime-new/JavaXML.html
[33] FastCGI, http://www.fastcgi.com/
[34] Servlets, http://java.sun.com/products/servlet/index.html
[35] Java Server Pages, http://java.sun.com/products/jsp/?frontpage-javaplatform
[36] Active Server Pages, http://msdn.microsoft.com/workshop/server/asp/asptutorial.asp
[37] Professional Home Pages, http://www.php.net/
[38] EmbPerl http://perl.apache.org/embperl/
[39] WebMacro, http://webmacro.org/tutorial/intro.html
[40] Freemarker, http://freemarker.sourceforge.net/
[41] Tea, http://opensource.go.com/
[42] Otembo, http://www.meangene.com/otembo/
[43] Velocity, http://jakarta.apache.org/velocity/
[44] TextGenerator, http://www.textgenerator.com
[45] The TRiX Template Resolution Framework, http://www.hpl.hp.com/techreports/98/HPL-98-04.html
[46] Template Resolution in XML / HTML, http://www.hpl.hp.com/techreports/1999/HPL-1999-42.html
[47] TRiX download, http://www-uk.hpl.hp.com/people/sth/trix/
[48] MAWL, http://www.bell-labs.com/project/MAWL/tutorial.html
[49] BigWig http://www.brics.dk/bigwig/
[50] The XML Form language, http://www.hpl.hp.com/techreports/1999/HPL-1999-41.html
[51] PowerForms, http://www.brics.dk/bigwig/powerforms/
[52] Form Markup Language, http://www.mozquito.org/index.html
[53] XForms, http://www.w3.org/MarkUp/Forms/
[54] UIML, http://www.uiml.org/
[55] Harmonia, http://www.harmonia.com/
[56] Hammock, http://www.oop.com/TechnologiesHammock.jsp
[57] Mozilla XPToolkit Architecture, http://www.mozilla.org/xpfe/aom/AOM.html
[58] XUL, http://www.mozilla.org/xpfe/xptoolkit/xulintro.html
[59] Java Taglibs, http://java.sun.com/products/jsp/taglibraries.html
[60] Jakarta Tag Library, http://jakarta.apache.org/taglibs/index.html
[61] Java Beans, http://java.sun.com/products/javabeans/?frontpage-javaplatform
[62] XMLC, http://xmlc.enhydra.org/software/documentation/index.html
[63] Efficient Wireless Design, http://www.webtechniques.com/archives/2001/02/young/
[64] Enhydra, http://www.enhydra.org/
[65] Element Constructor Set, http://jakarta.apache.org/ecs
[66] Understanding Java Server Pages Model 2 architecture, http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html
[67] MVC Architectures in J2EE, http://bishop.ics.hawaii.edu/shenyan/patterns/J2EE_patterns/MVC.html
[68] J2EE, http://java.sun.com/j2ee/
[69] Struts, http://jakarta.apache.org/struts/userGuide/introduction.html
[70] Turbine, http://java.apache.org/turbine

[71] Expresso, http://www.javacorporate.com
[72] XSLT, http://www.w3.org/Style/XSL/
[73] XSLT considered harmful, http://www.xml.com/lpt/a/1999/05/xsl/xslconsidered_1.html
[74] XMLScript, http://www.xmlscript.org/
[75] XPathScript, http://www.axkit.org/docs/xpathscript/guide.dkb
[76] Java, JSP and XML, http://java.sun.com/products/jsp/html/JSPXML.html
[77] Cocoon, http://xml.apache.org/cocoon/
[78] AxKit, http://axkit.org/
[79] XPathScript, http://www.axkit.org/docs/xpathscript/guide.dkb
[80] Infozone, http://www.infozone-group.org/
[81] Serif, http://www.caucho.com/products/resin/ref/xtp/xtp
[82] Internet Transcoding for Universal Access, http://www.research.ibm.com/networked_data_systems/transcoding/
[83] Xift transcoding portal, http://www.xift.com/
[84] Muffin proxy server, http://muffin.doit.org//
[85] Proxomitron proxy server, http://members.tripod.com/Proxomitron/features.html
[86] Web Based Intermedaries, http://www.almaden.ibm.com/cs/wbi/doc/index.html
[87] IBM Transcoding publisher, http://www-4.ibm.com/software/webservers/transcoding/
[88] Annotation-Based Web Content Transcoding, http://www9.org/w9cdrom/169/169.html
[89] PyWeb, http://www.pyweb.com/docs/index.shtml.en
[90] DeckIt, http://www.pyweb.com/php/test_adapt.php3
[91] AvantGo, http://www.avantgo.com/
[92] AvantGo style guide, http://avantgo.com/developer/web/documentation.html
[93] Web Clipping, http://www.palm.com/devzone/webclipping/pqa-talk/pqa-talk.htm
[94] Oracle Wireless Server, http://technet.oracle.com/products/iaswe/
[95] News Clipper, http://newsclipper.sourceforge.net/
[96] .Net, http://msdn.microsoft.com/net/
[97] Microsoft Presentation on Mobile Web Applications, http://www.microsoft.com/winme/00oct/101900fin/FinancialSrvsSumitt_DevToolsForMobileWebApp_SChory/default.htm
[98] XMLISAPI, http://msdn.microsoft.com/xml/general/xslisapifilter.asp
[99] WebForms, http://msdn.microsoft.com/library/dotnet/cpguide/cpconintroductiontowebforms.htm
[100] MobileForms, http://msdn.microsoft.com/vstudio/nextgen/technology/mobilewebforms.asp
[101] HawHaw http://www.hawhaw.de/
[102] AVIDWireless, http://www.avidwireless.com/
[103] XHTML Basic, http://www.w3.org/TR/2000/REC-xhtml-basic-20001219/