

Approaches to develop services for wired and wireless networks.

By Christian Feichtner, Siemens Austria, PSE MCS CN4

The number of people who are using the Internet and the World Wide Web has grown dramatically during the past few years. Companies have made considerable efforts to create content and services for the web. However, since the advent of WAP, many of them face the problem of porting services to the WAP platform and will continue to do so with the new generations of mobile wireless devices. This paper focuses on the main problem, to develop services for a wide range of devices having different characteristics like screen size or navigation options and discusses two approaches on how to solve those problems.

1.1 The Dilemma

Developers creating services for the wired and wireless world have to deal with the problem of small displays, limited memory and computing power on the wireless platform versus high resolution displays, huge memory and efficient computing power on wired desktop computers just to name a few. If a service should be available on both platforms it has to be adapted for each possible device it will be used on. This requires developers to create several versions of a service, compatible with the device capabilities of each class of device or even each single device in the worst case. At present, there is no real solution that will enable developers to create a service for the wired and wireless world with a minimum of effort without rewriting the entire service for each platform or device.

1.1.1 HTML vs. WML

For a long time, HTML [1] has been the language which was used to generate services and to design navigation for web content. HTML is a standardized hypertext markup language, derived from SGML and is used to create the content and layout for a web page. Navigation is implemented by providing hyperlinks to the users. A browser takes the HTML file as input and will render the final layout on the screen based on the content and layout information contained in the document. Users then are able to navigate by using a graphical pointing device to click on hyperlinks. Services based on that, are made for rather big screens on powerful computers. One disadvantage of HTML is, that it is not XML compliant, which is becoming more and more important today as XML is standardized and allows a variety of automatic processing options.

Totally contrary to the web, services for mobile devices have to be small and must run on devices with limited screen capabilities and power supply, offering only a few keys for input. The WAP Forum [3] is responsible for developing the WAP architecture along with the markup language used to design services. This markup language, called wireless markup language (WML), though it offers quite similar elements to HTML, is incompatible with HTML. WML is fully XML [2] compliant and offers additional tags especially designed to ease the layout and the navigation on mobile devices.

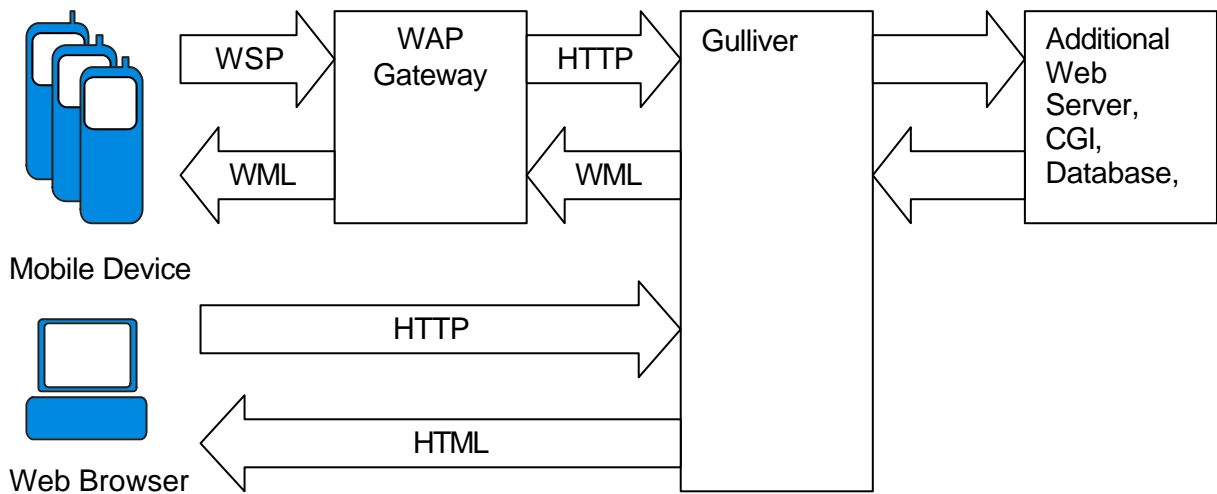
The problem faced by service developers is how to reuse existing services and content in both worlds, as the wireless world undoubtedly will become more and more important. Currently developers have to re-write services in order to work on small, wireless devices. It is also not recommendable to match navigation patterns from the wired world like the world wide web to the wireless WAP world. So a solution is needed, which will allow developers to create services for both worlds with a minimum of effort.

1.2 An approach for device-independent service provisioning: The Gulliver Project

"Gulliver" is a research project cooperation, currently being in it's final phase, between Siemens PSE and the University of Linz, Austria. Gulliver, which is an open platform addresses the following issues:

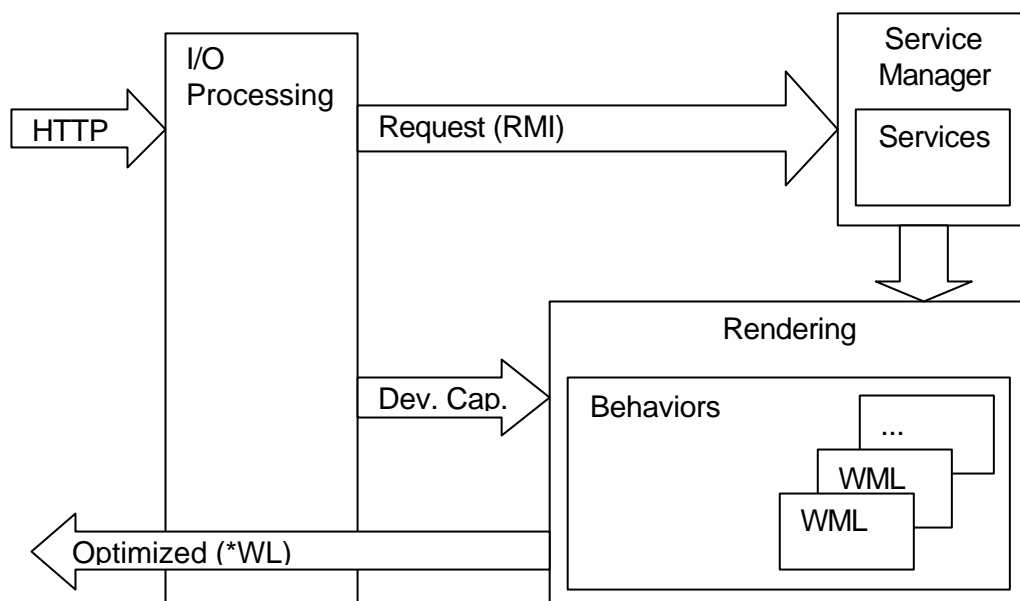
- Man-Machine-Interaction is device specific
 - Hardware (Graphical, Keys only, ...), Software (Browser Version, ...)
- Context aware content (time and location)
- User Profiles
 - Personal Interests, Ad-Hoc Meetings, Age, Language
- The data transmission medium is taken into account
- QoS Parameters

The Gulliver platform allows to design and to integrate services using a common language. The following graphics shows the generic Gulliver architecture.



Picture 1: Gulliver overall architecture

Devices using a Gulliver service, communicate with the Gulliver platform the standardized HTTP protocol. The output being sent to the device is dependent on the device characteristics, which have been determined. The following picture is a closer look into the Gulliver box from the above picture.



A HTTP request for a service is received by the I/O processing unit. This unit forwards the request via RMI to the service manager and the service. The service generates output which is passed to the rendering unit, which itself contains behaviors for several devices. Using these, the renderer generates optimized *ML (WML, HTML, XHTML, ...) output, which is then sent to the device. The I/O processing part passes the optimized *ML response through to the client.

Gulliver has successfully been presented using a demo service called "Vienna Guided Tour", which also demonstrated a suspend/resume feature allowing a user to suspend the tour at will and to resume it at a later time and incorporates user preferences, which allow the user to specify if he wants to see museums or churches for example.

1.3 Separating Content, Layout and Navigation

A service designed to run with a web browser will probably run, but be difficult to use on a wireless device. To solve this problem, developers should clearly separate a service's content, layout and navigation. This can be achieved by utilizing standards like XML and XML style sheets (XSL) [8]. The content only incorporates the structure, making no implications or definitions on how it should be displayed on a device. XML is utilized to structure the content. The layout defines how the content should be displayed, which may be different for each class of devices like powerful desktop computers or thin wireless devices. A combination of content and layout is what the user actually sees on the screen. The navigation is implemented using hyperlinks, but is also to be regarded separately from the content and layout, which makes it possible to define different navigation patterns for different devices. By combining the content, layout and navigation in a way suitable for a specific device a service becomes truly device independent.

1.3.1 Content: XML

The Extensible Markup Language (XML) [2] is a universal format for structured documents. It allows to structure a document and to store it in a plain text file. This way developers can mark special parts of a document which should be rendered in a special font or color or which should be used for navigation. Using an XML parser, the document can be parsed and the output can be adopted to the capabilities of any device. Several web sites already demonstrate this by having the content stored as XML and layout it differently for screen and print.

1.3.2 Layout: XSL

The Extensible Stylesheet Language (XSL) [8] allows to transform a XML document into an other XML compliant document. XSL hereby contains the definitions on how the transformation has to take place. A XML/XSL processor takes both, the XML and the XSL file as an input and transforms them into a new document, depending on the transformations defined in the XSL.

1.3.3 Navigation

Unfortunately there is no standardized mechanism to store navigation for a service similar to XML and XSL. However, the navigation can be seen as a frame around the service; a frame around each page of a service if we talk about Web and WAP services. . This view would suggest to also store the navigation separately in order to become optimized for several device classes. At present navigation is also defined using XML and XSL is used to transform the navigation into a proper format for a device class.

1.4 Device capabilities

If content, layout and navigation are separated as proposed in the previous chapter, then we also need a way to determine how to combine them according to the capabilities of the

device. Currently a service can determine the device capabilities by a user-agent string which is sent along with each request and typically contains the a user-agent name, version number and manufacturer. But the service has to know or to retrieve the device capabilities from somewhere based on that user-agent string. The browscap method allows a service to use a common file managed by a single institution to determine the characteristics based on the user agent string. The second method is using a CC/PP profile which allows a device to directly inform the service about it's capabilities.

1.4.1 Browscap File

This file, maintained by CyScape [6], is a plain text file, listing the capabilities for almost any known web browser, in a way known from windows initialization (.ini) files. The file is used together with web servers or CGI scripting languages and can be used to determine the capabilities based on the HTTP_USER_AGENT [7] field, which is transmitted with every HTTP GET request – if it is not filtered by a firewall or similar. This approach requires web server administrators or developers to always ensure they have the latest and correct version of this file, or services may not function correctly.

1.4.2 CC/PP

This approach, which is currently being developed by a working group at the world wide web consortium, is XML based and tries to define a RDF [5] based framework for device profile information. The goal of the CC/PP framework [4] is to specify how client devices express their capabilities and preferences (the user agent profile) to the server that originates content (the origin server). The origin server uses the "user agent profile" to produce and deliver content appropriate to the client device. In addition to computer-based client devices, particular attention is being paid to other kinds of devices such as mobile phones. CC/PP is currently a working draft.

1.5 Conclusion

As discussed in this paper, developers have to separate content from layout and navigation to create services device independently for the wired and wireless world. A standardized markup language along with the mechanisms to define a layout already exist which will aid the user in this task. However, in order to reassemble the content, layout ad navigation according to the device's capabilities, a standardized way to determine the device capabilities would be required. The browscap file method is proprietary and CC/PP is currently in the process of being standardized. Future effort should be put into a common, standardized way, which allows a developer to easily determine the capabilities of a device. Along with new technologies like XML and XSL and the proposed separation of content, layout and navigation and finally the mechanisms to provide device capability negotiation it will be easy to develop cross network services in the near future.

2 References

- [1] World Wide Web Consortium: The Markup Language: <http://www.w3.org/MarkUp/>
- [2] World Wide Web Consortium: Extensible Markup Language: <http://www.w3.org/XML/>
- [3] The WAP Forum: <http://www.wapforum.org>
- [4] World Wide Web Consortium: CC/PP Architecture <http://www.w3.org/Mobile/CCPP/>
- [5] World Wide Web Consortium: Resource Description Format: <http://www.w3.org/RDF/>
- [6] Cyscape: Browscap File: <http://www.cyscape.com/browscap/>
- [7] HTTP 1.1: <http://www.rfc-editor.org/rfc/rfc2616.txt>
- [8] World Wide Web Consortium: Extensible Stylesheet Language
<http://www.w3.org/Style/XSL/>

3 Abbreviations

CC/PP	Composite Capabilities/Preference Profiles
CGI	Common Gateway Interface
HTTP	Hypertext Transfer Protocol
RDF	Resource Description Format
SGML	Standard Generalized Markup Language
W3C	World Wide Web Consortium
WAP	Wireless Application Protocol
WWW	World Wide Web
XHTML	Extended Hypertext Markup Language
XML	Extensible Markup Language
XSL	Extensible Style Language