ESPOO 2003 VTT PUBLICATIONS 515

Marko Palviainen & Timo Laakko

mPlaton

Browsing and development platform of mobile applications

VTT PUBLICATIONS 515

mPlaton
Browsing and development platform of mobile applications

Marko Palviainen & Timo Laakko

VTT Information Technology

ISBN 951–38–6265–8 (soft back ed.)
ISSN 1235–0621 (soft back ed.)

ISBN 951–38–6266–6 (URL: http://www.vtt.fi/inf/pdf/)
ISSN 1455–0849 (URL: http://www.vtt.fi/inf/pdf/)

JULKAISIJA – UTGIVARE – PUBLISHER

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

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

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

VTT Tietotekniikka, Sinitaival 6, PL 1206, 33101 TAMPERE
puh. vaihde (03) 316 3356, faksi (03) 316 3380, (03) 317 4102,

VTT Informationsteknik, Sinitaival 6, PB 1206, 33101 TAMMERFORS
tel. växel (03) 316 3356, fax (03) 316 3380, (03) 317 4102

VTT Information Technology, Sinitaival 6, P.O.Box 1206, FIN–33101 TAMPERE, Finland
phone internat. + 358 3 316 3356, fax + 358 3 316 3380, + 358 3 317 4102

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

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

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

Technical editing Marja Kettunen

Otamedia Oy, Espoo 2004

Palviainen, Marko & Laakko, Timo. mPlaton - Browsing and development platform of mobile applications. Espoo 2003. VTT Publications 515. 98 p.

Keywords mobile application development, mobile Internet, WAP, XML, XHTML, browsing, mobile platform

Abstract

VTT Information Technology has created a comprehensive mobile Internet research and development environment, the components of which enable experimentation of new multimedia technologies and ideas, and their implementations in the mobile user's Internet environment before the method is standardized and incorporated into commercial products. This reports describes the mPlaton browsing and development platform of mobile applications, which forms a part of the VTT's mobile Internet research environment.

mPlaton supports the development of client-side mobile browsing applications, so that implemented parts are consistent, generic and reusable. It provides frameworks and solutions how to browse the content in mobile environment, present the content with profile information in separate user agent implementations with multimodal controls available and create and edit the browsing content.

The core of mPlaton include frameworks for browsing (AGB), for mobile user agents, content types and multimodal controls (MIMEFrame), and for XML editors (FEdXML). The frameworks are designed for mobile environments so that they are very light and scalable, based on generic standards, and are transferable for separate platforms and can be implemented by using different kinds of object-oriented languages (e.g., Java and C++). The solutions introduced in this work are implemented in Java SE, Java Personal Profile and Java MIDP platforms. The solutions are based strongly on standards (e.g., by World Wide Web Consortium (W3C) and Open Mobile Alliance (OMA)) and generic object-oriented technologies, and they are designed to be generic enough to enable the transferability to other platforms.

Preface

The work related to the mPlaton platform described in this report has been carried out within several research projects at VTT Information Technology. The early groundwork (years 1999 – 2000) was related to the development of the VisualWML system (a development tool for WAP applications) - a predecessor of the mPlaton toolkit. VisualWML was developed within the research projects "WML-Browser" (year 1999) and "WAP-Multimedia" (2000). After, many parts of mPlaton toolkit were introduced during the "AMMME" project (2001). Finally, the core frameworks of mPlaton were defined and implemented during the current research project "Adaptive context-aware mobile and multimedia applications and their development environment - ALLLAS" (http://www.vtt.fi/tte/proj/alllas).

One of the initial aims of this work has been to support other mobile Internet and applications projects at VTT by providing a browsing and development platform for mobile applications. The target is to enable the experimentation with new technologies and ideas, and their implementation in mobile user's Internet environment before the method is standardised and incorporated in commercial products.

The solutions provided by mPlaton are based strongly on the evolving Mobile Internet standards (e.g., by OMA and W3C) as well as generic object-oriented technologies, and they are designed to be generic enough to enable the transferability to other platforms.

Contents

List of symbols

API	Application Programming Interface
CC/PP	Composite Capability/Preference Profiles
CSS	Cascading Style Sheets
DOM	Document Object Model
DTD	Document Type Definition
GPRS	General Packet Radio Service
GSM	Global System for Mobile communications
HSCSD	High Speed Circuit Switched Data
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IMPS	Instant Messaging and Presence Service
OMA	Open Mobile Alliance
MIDP	Mobile Information Device Profile
MMS	Multimedia Messaging Service
RFC	Request For Comments
RDF	Resource Description Framework
RTP	Real-time Transport Protocol

SMIL	Synchronized Multimedia Integration Language
SVG	Scalable Vector Graphics
UAProf	User Agent Profile
UML	Unified Modelling Language
UMTS	Universal Mobile Telecommunications System
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WAE	Wireless Application Environment
WAP	Wireless Application Protocol
WCSS	WAP Cascading Style Sheets
WLAN	Wireless Local-Area Network
WML	Wireless Markup Language
WSP	Wireless Session Protocol
XHTML	eXtensible HyperText Markup Language
XML	eXtensible Markup Language
XPath	XML Path Language
XSL	eXtensible Stylesheet Language
XSLT	eXtensible Stylesheet Language Transformations
XSL-FO	eXtensible Stylesheet Language Formatting Objects

1. Introduction

The use of mobile applications is becoming a part of everyday life in consequence of the development of mobile devices and network solutions. At the same time, the diversity of the mobile environment is increasing. For instance, mobile devices will have multiple network connections and the available bandwidth may vary a lot. There will be several kinds of new mobile applications. In particular, context and location awareness will be properties of many mobile applications. The Semantic Web will enable intelligent services, like information brokers, filters and search agents, which can offer greater functionality and interoperability than current stand-alone services. Multimodality improves the usability by combining several interaction ways like visuals, voice and touch.

The characteristics of mobile environment set many requirements for mobile applications. Mobile devices memory, processing power as well as other capabilities are rather limited compared to desktop PCs. In particular, mobile applications should satisfy the needs of different kinds of mobile users as well as be very light, reusable and transferable to various kinds of mobile devices. This all poses several challenges for mobile application development and tools.

1.1 VTT's Mobile Internet research and development platform

VTT Information Technology has created a comprehensive mobile Internet research and development environment. It enables experimentation of new multimedia technologies and ideas, and their implementations in the mobile user's Internet environment before the technologies are standardized and incorporated into commercial products (Figure 1.1). The system covers the entire chain from content servers through presentation format and data transfer protocol modifications to the user's browser and terminal. It includes the key components needed for the development and testing of new services as well as development tools for applications developers. For instance, specifications of Open Mobile Alliance (OMA) are used as a part of the implementation of the platform. The platform has been used in several national and EU projects. The open, modular architecture of the system allows its modification as the related specifications are continuously developed and the exploitation of the components and their porting into real environment.

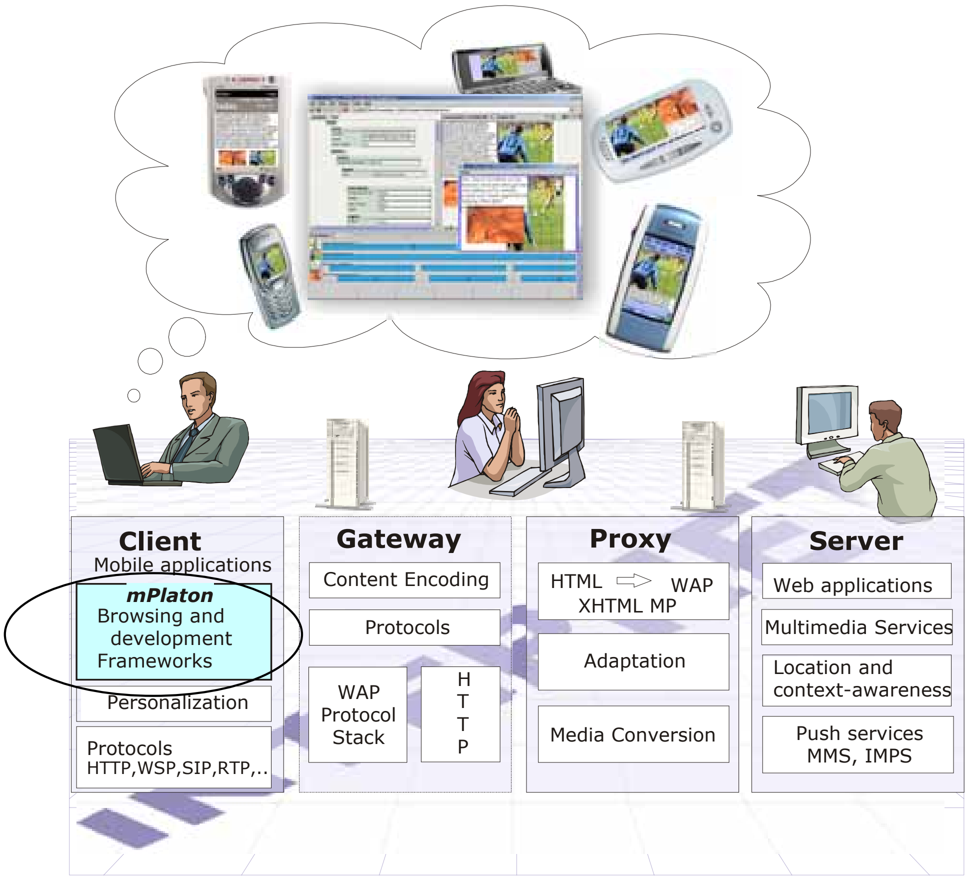


Figure 1.1. Mobile Internet research environment.

1.2 mPlaton – a browsing and development platform

This report describes the "mPlaton" browsing and development platform for mobile applications. mPlaton forms an essential part of the mobile Internet research environment (see Figure 1.1), and it has been developed since the year 1999 within several projects of VTT Information Technology. Key research topics of them related to mPlaton included how to

- browse the content in mobile environment,

- present the content with profile information in separate user agent implementations with multimodal controls available, and

- construct the browsing content to mobile environment.

As results, new frameworks have been created for

- browsing (AGB);

- mobile user agents, content types and multimodal controls (MIMEFrame); and

- XML editors (FEdXML).

The core frameworks (AGB, MIMEFrame and FEdXML) are designed for mobile environments so that they are very light and scalable, based on generic standards, and are transferable to separate platforms and can be implemented by using different kinds of object-oriented languages (e.g., Java and C++). The solutions introduced in this work are implemented by using Java 2 SE, Personal Java and Java MIDP platforms. The core solutions described strongly rely on standards (e.g., by OMA and W3C) and generic object-oriented technologies. They are designed to be generic enough that transferability for other platforms is enabled.

Basic concepts and properties of Mobile Internet browsing applications are outlined in Chapter 2. Then, detailed descriptions of the core frameworks of mPlaton are provided in Chapters 3, 4, 5 and 6. By using the frameworks several new applications have been designed and implemented: mPlaton Toolkit – a development toolkit for mobile applications, µBrowser - a micro browser implementation for PDA devices, and MIDBrowser - a browser implementation for Java MIDP enabled mobile phones; the applications are described in Chapter 7. Finally, the results are discussed and conclusions are drawn in Chapters 8 and 9.

2. Mobile Internet and browsing applications

Mobile Internet is growing rapidly. New specifications and definitions are being prepared in several working groups and organisations. Two central organisations amongst others are Open Mobile Alliance (OMA) and World Wide Web Consortium (W3C), which have prepared several new specifications and defacto standards. Mobile Internet makes it possible to construct new kinds of services for the users, such as

- adaptive and intelligent mobile browsing services, and
- context- and location-aware services,
- push, multimedia messaging, mobile instant messaging and presence services.

The mobile contexts of use as well as the characteristics of mobile devices and networks may vary a lot [ArD02]. Thus, there is a great need for adaptive mobile browsing applications, which are capable to be profiled for the user, device and available network connection.

Personalised, context-aware and location-based mobile computing is excepted to be one of the major ingredients of the future wireless Internet services [MiP01][MK+02]. Many information and services are relevant only in limited context [CoK99]. Information about the user's environment makes it possible to implement time-aware, location-aware, device-aware and personalised applications [HS+03]. Context-aware applications can offer contextual sensing, adaptation, resource discovery and augmentation methods to interact with the environment [Pas98]. The context information can be used in adapting the user interface and providing relevant services to the user [DS+99].

The new developing mobile platforms (e.g., Java MIDP, Symbian and Nokia Series 60) give increasing opportunities to mobile software developers. It is important that same applications can be reused and transferred to the different mobile devices.

Mobile browsing standards, intelligent browsing services, adaptive mobile browsing services, multimodality and mobile platforms are described in the following subsections.

2.1 Standardization

The Wireless Application Environment [WAE] specified by OMA provides an extensible environment for application development. The most important content formats of browsing applications include Wireless Markup Language (WML) [WML] and XHTML Mobile Profile [XHTMLMB], which is a part of WAP2 specifications. Both of them are authoring languages for mobile Internet services and applications.

eXtensible Markup Language (XML) [BP+00] provides a generic and compact way to define structured information. Consequently, it is becoming heavily used in several applications, such as, electronic publishing, and, especially, in mobile environments (browsing applications) XML has become the key element.

Both WML and XHTML Mobile Profile are eXtensible Markup Language (XML) applications. XML itself specifies an easy and flexible way to describe and deliver structured information over the World Wide Web (WWW). The modularization work of XHTML [AB+01] supports well adaptation and transformations to other XML applications (such as WML) and between modules (i.e. XHTML Basic [BI+00]). Consequently, browsers have to support several different modules. In particular, XHTML Mobile Profile is a strict superset of the XHTML Basic document type. The WML 2.0 document type extends XHTML Mobile Profile by including WML 1.x compatibility and other extensions. XHTML Mobile Profile is a compact core module, which is to be supported by most of the browsers. The documents written using earlier WML 1.x versions can be transformed into WML 2 format. WAP 2 also includes WAP Cascading Style Sheets (WCSS) [WCSS] targeted for XHTML Mobile Profile and WML 2 documents.

Standardised ways for making descriptions and processing are mandatory in making automatically adapting services. For instance, the Semantic Web [SEMW, BLE01] is one step towards these intelligent browsing services. A key element of data integration is a language for specifying the semantics associated with data content [BeF01]. Metadata can be used for adapting applications in dynamic environments [BC+03]. Ontology provides a common understanding of topics that can be communicated between people and application systems [DM+00]. The Semantic Web will enable intelligent services, like information brokers, filters and search agents, which can offer greater functionality and interoperability than current stand-alone services [DM+00].

2.2 Adaptive and intelligent mobile browsing services

Mobile environment may be divided to layers (cf. Figure 2.1). Services can be transmitted via a wireless network to the terminal of a user. However, mobile terminals have many limitations, which influence the application design. Such are limited input and output capabilities [KHL02], slow network connections, and restricted memory and processing power. Mobile device is to be used "any time and anywhere", and, thus, usage environment is very different (e.g., more distractions) compared to desktop PC. From the service development point of view, the mobile environment sets great challenges while services must fit to the requirements of networks, terminals and users.

Intelligent browsing services may automatically combine content from different sources and filter the content suitable for the user. An intelligent browsing service may utilise, for instance, information about the user (user profile), contexts of use, devices (device profile), network, and the service itself. The services can be automatically constructed and adapted to fit to the current context of use. Adaptation can be done in either the client or server side. On the server side (or in the proxy), the content are to be adapted [BoK01][DeC02][OK+01] to fit to the current context of use so that

- only information needed is transferred over the mobile network,
- the information can be transferred to the device in a reasonable time,
- the device can present the transmitted content, and
- the content corresponds to the contexts of use.

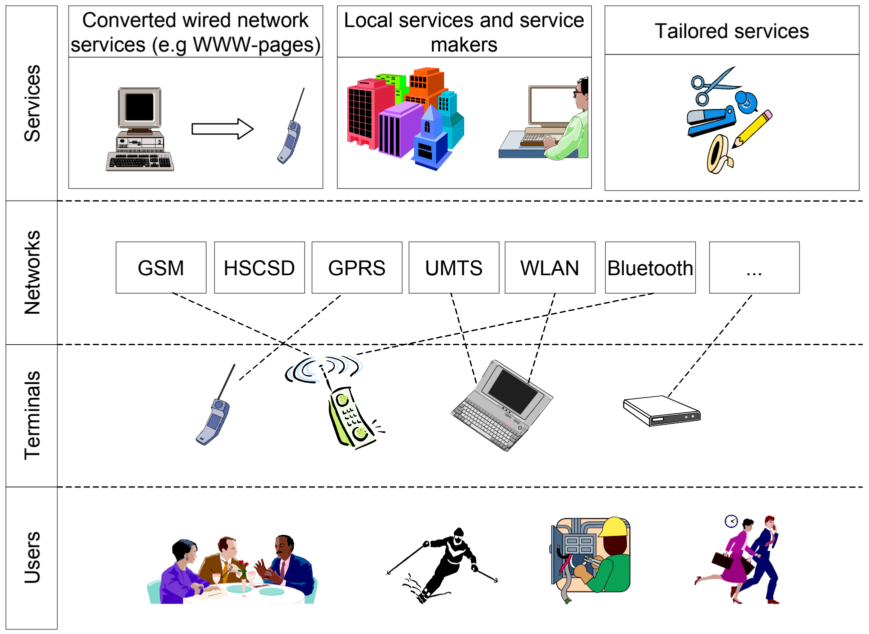

Services
Converted wired network services (e.g WWW-pages)
Local services and service makers
Tailored services
Networks
GSM
HSCSD
GPRS
UMTS
WLAN
Bluetooth
...
Terminals
Users

Figure 2.1. In the future, there will be various kind of mobile applications and services. At the same time, there will be an increasing variety of different kind of mobile user groups, terminals and alternative mobile networks.

After the adaptation, the generated content corresponds to the user preferences, is suited to be transferred via the network connection (in a reasonable time) and the presentation is optimised for the mobile device.

On the client side, profiling can be accomplished by

- requesting only the content, which can be presented in the device (e.g., if the device can not present audio or video, no request for the server-side should be made),

- requesting only the content defined to be browsed in profile information (e.g., the user may not want to see any pictures in order to enable faster operation), and

- profiling the presentation of the content in the device (e.g., by using the style sheets).

Information needed for the adaptation can be obtained from user agent profiles or style sheets, or there could be additional information, for instance, about user preferences.

2.2.1 User Agent Profile

The Resource Description Framework (RDF) [LaS99] provides domain-neutral mechanism to exchange and process metadata [Kle01], while it aims to provide the foundation for metadata interoperability across different resource description communities [DM+00]. RDF is an application of XML and it defines a simple model for describing interrelationships among resources in terms of named properties and values. The Composite Capabilities/Preferences Profile (CC/PP) framework [KR+03, OS+01] creates a structured format for how a client device tells to an origin server about its capabilities and preferences. The CC/PP uses RDF for expressing a user agent's profile. 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 [Hje00].

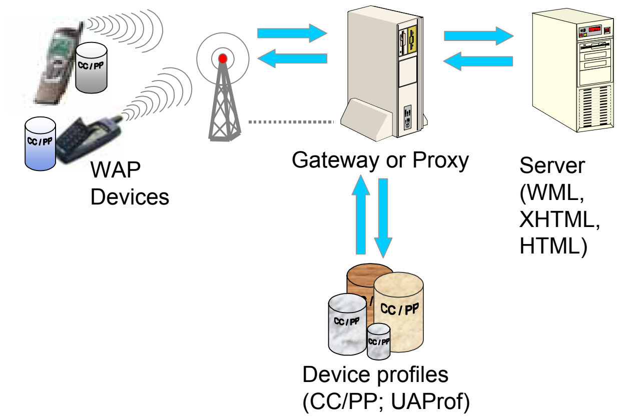
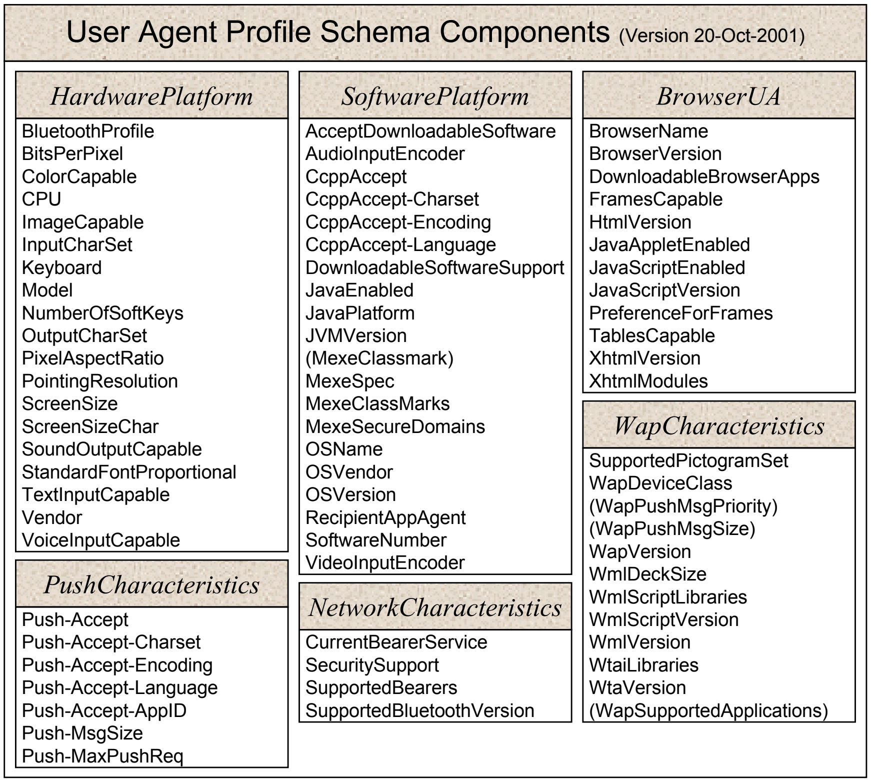


Figure 2.2. A use case of User Agent Profile.

The User Agent Profile architecture enables the end-to-end flow of UAProf between the client device, the intermediate network points, and the origin server. Wireless Session Protocol (WSP) clients connect to servers via a WAP gateway and Wireless Profiled HTTP clients may connect to an origin server directly or via proxies. Capability and Preference Information (CPI) is transmitted within the protocol headers, and it consists of information collected from the device hardware, user agent software, and user preferences and network characteristics. The device may not have all this information, but it may indicate the location by a single URI and the WAP gateway or proxy resolves the URI and retrieves the information from the manufacturer host (see Figure 2.2). Then, the gateway or proxy forwards the request to the origin server and includes the profile information in HTTP header.

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 adapts and translates the requested content into the appropriate format using the UAProf. Finally, the gateway or proxy directs the response to a client. The UAProf may be cached in the gateway or proxy, which may also add information (e.g., additional network information) to the profile.


User Agent Profile Schema Components (Version 20-Oct-2001)

HardwarePlatform
BluetoothProfile
BitsPerPixel
ColorCapable
CPU
ImageCapable
InputCharSet
Keyboard
Model
NumberOfSoftKeys
OutputCharSet
PixelAspectRatio
PointingResolution
ScreenSize
ScreenSizeChar
SoundOutputCapable
StandardFontProportional
TextInputCapable
Vendor
VoiceInputCapable

PushCharacteristics
Push-Accept
Push-Accept-Charset
Push-Accept-Encoding
Push-Accept-Language
Push-Accept-AppID
Push-MsgSize
Push-MaxPushReq

SoftwarePlatform
AcceptDownloadableSoftware
AudioInputEncoder
CcppAccept
CcppAccept-Charset
CcppAccept-Encoding
CcppAccept-Language
DownloadableSoftwareSupport
JavaEnabled
JavaPlatform
JVMVersion
(MexeClassmark)
MexeSpec
MexeClassMarks
MexeSecureDomains
OSName
OSVendor
OSVersion
RecipientAppAgent
SoftwareNumber
VideoInputEncoder

NetworkCharacteristics
CurrentBearerService
SecuritySupport
SupportedBearers
SupportedBluetoothVersion

BrowserUA
BrowserName
BrowserVersion
DownloadableBrowserApps
FramesCapable
HtmlVersion
JavaAppletEnabled
JavaScriptEnabled
JavaScriptVersion
PreferenceForFrames
TablesCapable
XhtmlVersion
XhtmlModules

WapCharacteristics
SupportedPictogramSet
WapDeviceClass
(WapPushMsgPriority)
(WapPushMsgSize)
WapVersion
WmlDeckSize
WmlScriptLibraries
WmlScriptVersion
WmlVersion
WtaiLibraries
WtaVersion
(WapSupportedApplications)

Figure 2.3. The User Agent Profile Components.

The User Agent Profile schema has six key components, which are *HardwarePlatform, SoftwarePlatform, BrowserUA, NetworkCharacteristics, WapCharacteristic*s and *PushCharacteristics* [UAPROF] (Figure 2.3). Each of these resources has a collection of properties that describe the component. The *HardwarePlatform* describes 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 *NetworkCharacteristics* has information about the network's capabilities: for instance, current network bearer service and a supported security. The *WapCharacteristics* has properties to describe WAP capabilities on the device, e.g., *WmlDeckSize* and *WapVersion*. *PushCharacteristics* describes the push connection capabilities of the terminal device. In UAProf case it is possible to add definitions outside of the current UAProf descriptions. It is also possible to define descriptions by telling only the location where the actual description can be found, and, so, it is possible to limit the amount of wireless traffic.

2.2.2 Style sheets

Style sheets are a device independent way to affect the visual appearance of the structured (e.g., XHTML or HTML) documents. Style sheets can be attached to documents and, so, both authors and readers can affect the presentation without changing the original structure of the document. Style sheets can be described by using eXtensible Style Sheet Language (XSL) [AB+01b] and Cascading Style Sheets (CSS) [BW+98] specified by W3C.

XSL is a style sheet description language for XML documents. With XSL it is possible to attach styles to XML documents and to transform XML documents to new ones. The XSL language consists of XSL Transformations (XSLT), XML Path language (XPath) and XSL Formatting Objects (XSL-FO) parts. XSLT is a language for transforming XML documents, XPath is a language for accessing and referring to the parts of the XML document and XSL-FO describes the vocabulary for XML formatting semantics. XSLT describes transformation to another XML document or to another document type. For example, an HTML document can be constructed from XHTML source by using XSLT. XSLT uses the XPath for referring to the parts of an XML document. XSL-FO makes it possible to describe the formatting objects for the XML document and with XSLT these formatting objects can be attached to the target XML document.

CSS is a mechanism to add styles to structured (e.g., HTML and XML) documents. In CSS the style is described by rules, which consist of a selector and a declaration block. The selector defines to what part of the document the rule is connected to and the declaration block tells the style descriptions for that part. Style rules can be defined in several separate CSS descriptions. For documents, CSS descriptions may come from three different origins. These choices are listed below.

1. Author can specify style sheet internally to the document. CSS descriptions can be defined in the document or some external style sheet can be linked to the document.

2. User can specify own style sheet, which can have user preferred style declarations.

3. User Agent can have own CSS description. CSS can be made for some specific user agent so that the layout of the content fits the agent as well as possible.

Figure 2.4 shows an example where the XML document is visualised by using style declarations in separate CSS descriptions. The user agent CSS (e.g., a CSS for mobile device) is on the top level, the user CSS is on the next level and then the author CSS on the last level. Style declarations are selected for XML elements by computing specificity values for each rule defined. In the standard case style declarations with higher specificity value overrides the style declarations with lower specificity value, but CSS offers (e.g., !important) mechanisms to change the order. It is also possible to inherit style declarations from other elements.

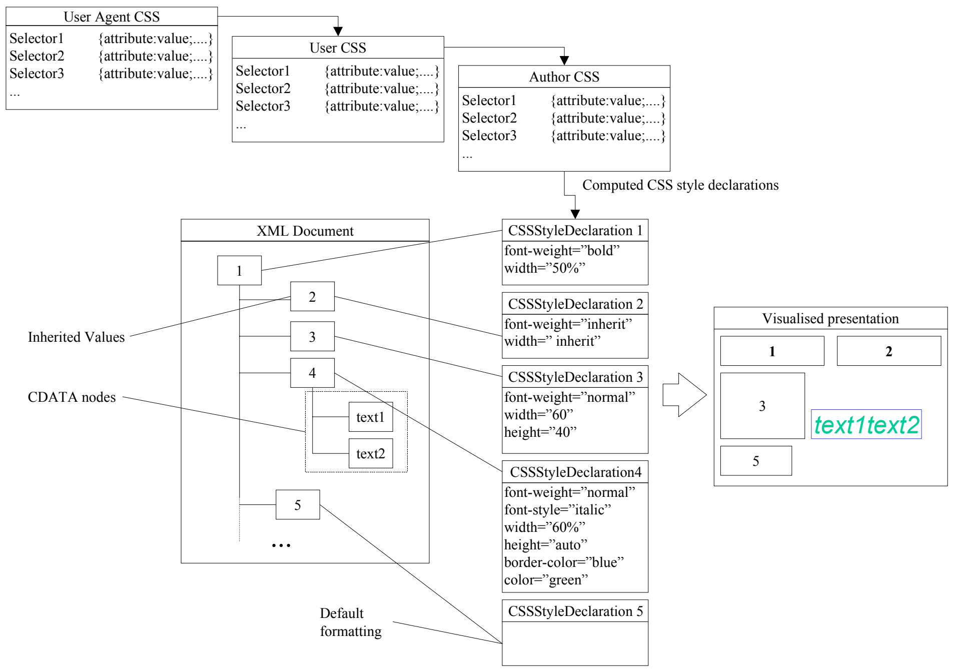


Figure 2.4. Cascading style sheet in XML document.

2.2.3 Personal preferences and other profile information

In order to implement profiled intelligent mobile browsing services, different kinds of information should be available for the software components performing the adaptation [Nob02]. Examples of such information include

- location information,
- user profile (e.g., personal favourites [BS+03]) [GUP], and
- current contexts (e.g., if the user is attending a meeting, the device must be in the silent mode).

For example, the Generic User Profile (GUP) defined by 3GPP is a collection of user related data [GUP]. Some profile information may be collected automatically while the user uses applications.

2.3 Multimodality

Multimodality allows the user to access wireless information in the most natural way, where user can input and receive information with various ways [Kum03]. Mobile users will get the freedom to choose from the multiple modes of interaction (e.g., keyboard, touching screen or voice), which will enable spontaneous and intuitive communication (e.g., forms could be filled with the voice commands). Multimodality enables totally new kinds of (e.g., voice-based) services and it can make the mobile browsing services easier to adopt for new users.

There are needs for clearly defined architectures and standards supporting building of multimodal applications for different types of terminals [NFP01,RS+01]. W3C has a multimodal interaction activity [MMI] for developing markup specifications for synchronisation across multiple modalities and devices with a wide range of capabilities. W3C has published Multimodal Interaction Framework [LR+03], which identifies the major components of multimodal systems. The purpose of the framework is to identify and relate markup languages for multimodal interaction systems. The framework describes the input and output modes widely used today, and it can be extended to include additional modes of user input.

2.4 Mobile application frameworks

For mobile application development there are several core application platforms. Examples of these platforms are J2ME, Symbian OS, Palm OS, Windows CE and Linux. Generally, mobile platforms are to be designed so that they fit in devices with limited memory and processing power. Also, scalability to different device environments is often required. The drawback of the lightness is the fact that only core methods for mobile application construction are supported. Therefore, there is a need for higher-level software frameworks that can extend core platforms and make the implementing of the mobile applications easier. This includes requirements for application components that can easily be embedded in and used by other applications.

A software framework offers a skeleton and guidelines (e.g., well-defined interfaces) for some specific domain application implementations, so that implementations are consistent, modular, and reusable. Generally, as a result of using software frameworks:

- Implementation of new applications becomes easier, because software frameworks offer base skeletons for the implementations.

- Implementations are more understandable, because they follow guidelines (e.g. interfaces) introduced by framework.

- Quality of applications is better, because applications can be implemented of small well-tested reusable modules.

Applications can be easily transferred into new environments by replacing existing modules with new implementations.

mPlaton offers software frameworks for implementing

❑ context-aware mobile pull and push enabled browsers,

❑ mobile user agents and adaptive content types with multimodal controls available, and

❑ editors for constructing mobile XML applications.

3. Overall architecture of mPlaton

The mPlaton platform consists of three main frameworks closely related to each other (see Figure 3.1)

- *AGB* (Architecture for Generic Browser) for mobile browser implementations,
- *MIMEFrame* for presenting content types with profile information in mobile user agent with multimodal controls available, and
- *FEdXML* (Framework for XML editors) for editing the mobile browsing XML applications.

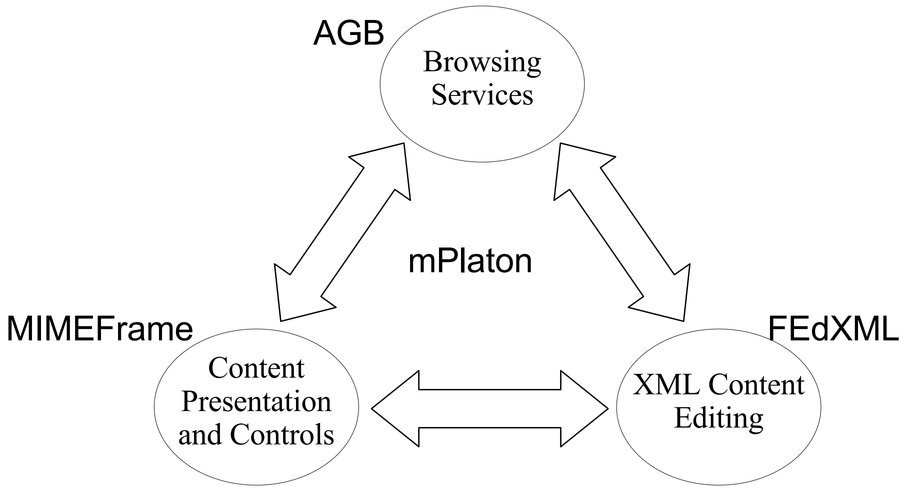


Figure 3.1. Parts of mPlaton architecture.

These frameworks are used as construction blocks of new mobile applications. They can be used separately or combined to work together.

Mobile browsing application typically consists of the server and the client sides. The server side offers the content to browse and the client side presents the browsed content. The browsing can be initiated either by the user (pull services) or initiated outside (push services).

The frameworks enable the construction of mobile pull/push enabled browsing applications. In particular, the frameworks provide means to browse (AGB), present (MIMEFrame) and edit (FEdXML) different content by using the available profile information.

AGB, MIMEFrame and FEdXML can be used together for constructing client side mobile browsing applications. In this, AGB offers the core services for mobile browser implementation. MIMEFrame offers components and interfaces used by the AGB implementation (or other applications) for presenting the content, controls, and user agent properties. FEdXML is a framework for various kinds of XML editors. Particularly, FEdXML enables developers to construct editors with which it is possible to edit the browsed XML content in mobile device.

Further, the mPlaton components can be embedded in other applications, which is one core property of mPlaton. Thus, mPlaton enables the creation of very different kinds of applications. Chapter 7 shows examples of created applications. The examples demonstrates also how to provide implementations of the mPlaton frameworks in different Java platforms (Java SE, Personal Java and Java MIDP). Many of the examples utilise the provided common reference implementations (see Sections 4.3, 5.4 and 6.5) of mPlaton components.

More detailed descriptions of the frameworks are in the Chapters 4, 5 and 6.

4. AGB – Architecture for Generic Browser

Mobile services can be divided into two types: *pull* and *push*. The user requests pull services, whereas push services are invoked externally. All these services are transmitted to the devices via some connection. Also, navigation can be divided into push or pull type operations.

Because of the heterogeneity of the mobile environment, there is a great need for a modular scalable browser architecture that could be configured for different kinds of mobile devices. The architecture should respond to the requirements of the mobile environment. In the following the generic requirements (collected from different sources) are enumerated that we have identified for browser architectures. Firstly, the browser architecture should offer basic navigation operations as commonly found in web browsers, e.g. navigate a new page, navigate in history (back, forward) and abort the navigation [LNR96]. For mobile users, the architecture should offer

- support features of mobile applications, such as push services,
- support for context and location aware browsing services, and
- support for automated initiated browsing (e.g., based on the location or the context of use).

To be suitable for various kinds of mobile devices, the architecture should

- be very light (should work with devices having very limited memory and processing power resources [KuO00]),
- be scalable to different kinds of mobile devices (It should be possible to add and remove push and pull services and connection components from the configuration),
- be flexible and modular so that created components related to push and pull type services could be easily used together in different configurations. It should be possible to easily add components, for example, by modifying the browser configuration descriptions,

- be reusable (e.g., should work in different environments, and the related implementation parts should be as reusable as possible) [GHJV94], and

- minimise the use of batteries (e.g., should support temporal pull and push connections so that the connections are kept open only a certain time).

In addition, the browser architecture should be suitable for the requirements of networks. So, the architecture should

- support different kinds of communication protocols and connections,

- optimise the use of the network capabilities (e.g., by using caching and by adding concurrency [Wat94]),

- support the attachment of profile information to requests (cf. Section 2.2.3), and

- support various kinds of authentication methods.

The developed new browser architecture and framework (AGB), which is designed to fulfill the above requirements, is described in the following Section 4.1. Then, reference implementations for AGB and usage examples are described in Section 4.2.

4.1 Architecture

AGB is an abstract model for mobile browser offering pull (initiated by the user) and push (initiated outside) browsing services (Figure 4.1), where all parts related to browser are added by changing the configuration. The core of the AGB includes interfaces only. The principal aim of AGB is to offer a fully modular structure for mobile browser implementations. Object browser offers primitive navigation operations for pull, push, backward, forward, reload and stop navigation. All the user-initiated browsing is done via the pull method. For navigating in the history there are backward and forward methods, and the reload method can be used to update the browsed objects. For receiving the push messages there is the push method. A key feature of AGB is that all the navigation tasks are finally mapped to push and pull operations.

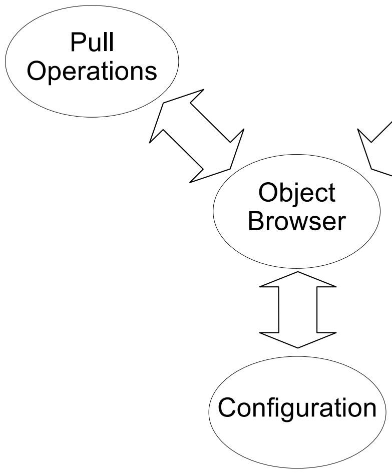


Figure 4.1. Object browser architecture.

AGB connections (pull and push type) can be further classified to persistent, temporarily and disposable connections. For example, an HTTP connection that makes it possible to fetch the content several times by using the same connection is a *persistent connection*. A *temporary connection* is only a certain time available. An HTTP connection can be defined to be valid in certain time by using the cache-control header. A *disposable connection* is, e.g., a reader type connection, which makes it possible to read the content only once from the given source.

4.1.1 Object pull

AGB supports synchronous and asynchronous browsing. In synchronous browsing, connections are opened and content is fetched in the main process. The synchronous browsing blocks the main process until the browsing is completed. This means that the browser can not be used during the synchronous browsing operation. Synchronous browsing has to be used in devices, where there are no support for thread-based processing or not enough memory or processing power for asynchronous browsing.

In order to prevent the browser from blocking during the loading of the content and to enable multiple simultaneous connections, browsing can be performed asynchronously in separate threads [CDK01]. This way the object loading time will be the sum of time spent on connection opening, request sending and response receiving. For instance, an HTML page may include several pull sources such as images, texts and videos. If these load operations are done at the same time, the perceived load time is the maximum of the load times of the objects [Luo98] (Figure 4.2).

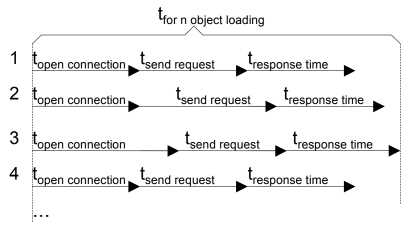

$t_{\text{for n object loading}}$
1 $t_{\text{open connection}}$ $t_{\text{send request}}$ $t_{\text{response time}}$
2 $t_{\text{open connection}}$ $t_{\text{send request}}$ $t_{\text{response time}}$
3 $t_{\text{open connection}}$ $t_{\text{send request}}$ $t_{\text{response time}}$
4 $t_{\text{open connection}}$ $t_{\text{send request}}$ $t_{\text{response time}}$
...

Figure 4.2. Perceived time spent on loading multiple objects with asynchronous load operations.

To keep AGB highly pluggable the connections and objects are constructed in connection and object factories (Figure 4.3). All supported connections and objects plugins can be configured to these factories. New connections and objects are constructed by using an *ObjectInitiator,* which tells the name and initialisation attributes for the object to construct. If a new object is constructed successfully it is returned, otherwise an exception is thrown.

In AGB, pull operations can be initiated by using a connection initiator or a connection object. This makes it is possible to construct (e.g., HTTP, bluetooth and RTP) connections in initialisation threads. The *ConnectionInitiator* interface offers methods for

- getting an *ObjectInitiator* object, for the connection to be constructed in connection factory,
- querying if the connection initiator valid is to keep in cache, and
- identifying if an equal *ConnectionInitiator* is already in the cache.

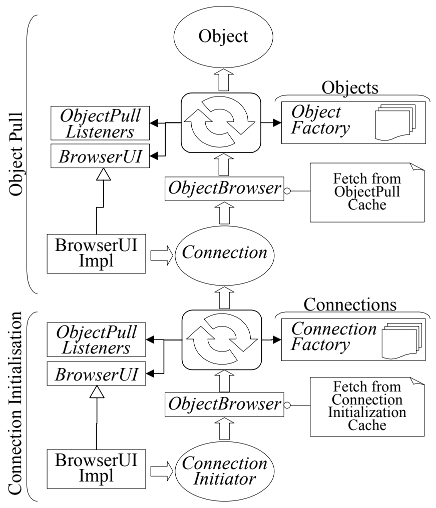


Figure 4.3. The Object pull operation can be executed with connection initiator or with connection object.

New objects are constructed in the object pull thread by using connections. The *Connection* interface offers methods for

- opening the connection, which returns an *ObjectInitiator* for the object to construct in object factory,
- querying if the connection is valid to keep in cache,
- identifying if there is an equal *Connection* in the cache, and
- closing the connection.

Authentication within the connections can be implemented so that an authentication request is thrown when a connection is to be opened. The thrown authentication request is caught in object pull thread and the request is sent for the *BrowserUI*. The *AuthenticationRequest* interface offers methods for getting the connection to authenticate and for retrying the object pull with authenticated connection.

The two-level object pull mechanism makes it possible to asynchronously fetch content with opened connections or to asynchronously initialise new connections for content fetching. This means that both opening the connections and fetching the objects can be done in the background threads so that these operations do not interrupt the main process of the browser. Connection initialisation and object pull operations have one of the following states:

1. Operation completed

2. Operation not completed

3. Operation failed

For minimising requests to mobile network, AGB has caches for connection initialisation and for object pull operations. The size of the cache depends on the memory size, and it can be configured in AGB. In an object pull operation with a connection initiator a look-up operation to the connection initialisation cache is carried out first. If the cache contains an initialised connection, the object pull operation is done with the initialised connection. If the cache contains an incomplete connection, the requestor is registered as a pull listener of the connection initialisation. Later, if the connection is successfully created, a new object pull operation executed with the created connection and the pull listeners are registered as listeners of the new object pull operation. If the cache contains a failed connection initialisation, it is possible to try to reinitialise the connection again by starting a new connection initialisation thread.

In an object pull operation with a connection, a look-up operation to the object pull cache is carried out first. If the cache contains a completed object pull operation, the related object is returned. If an incomplete object pull operation is in the cache, the requestor is registered as a listener of the object pull. When the object pull is completed, the listeners are notified of the pulled object. If the cache contains a failed object pull operation, starting a new object pull thread performs the reload operation.

4.1.2 Object push

New push (e.g., WAP push, MMS or SMS) connections can be added to the AGB by using the push adapters (Figure 4.4). A push adapter receives the messages coming from the push connection and maps received messages to pull operations or push indications of AGB.

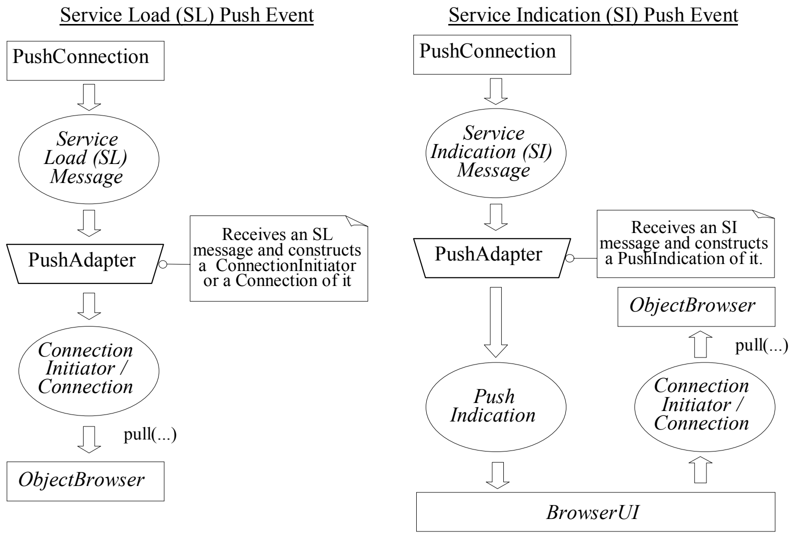

Service Load (SL) Push Event
PushConnection
Service Load (SL) Message
PushAdapter
Receives an SL message and constructs a ConnectionInitiator or a Connection of it
Connection Initiator / Connection
pull(...)
ObjectBrowser
Service Indication (SI) Push Event
PushConnection
Service Indication (SI) Message
PushAdapter
Receives an SI message and constructs a PushIndication of it.
ObjectBrowser
pull(...)
Push Indication
Connection Initiator / Connection
BrowserUI

Figure 4.4. Push adapters for SL and SI messages.

Push connections are normally implemented so that a listener is registered for a defined port. Listeners are notified of new push messages and, then, depending of the protocol implementation, the message can be handled either immediately or later, e.g., by using a thread based processing. Normally, the mobile device is not always connected to the network, but time-slots are used to reduce the battery usage (e.g., in Flex protocol). In AGB, it is possible to dynamically change the push adapter configuration. If a push connection is not available, the push adapter can be removed from the configuration and if the push connection will become available again the push adapter can be added to the configuration. When a push adapter is removed, all resources related to the adapter should be released (in the close method of the push adapter interface).

In AGB, there are two types of push events:

1. Service Load (push message is loaded immediately)

2. Service Indication (push message gives the location of the actual push object and, then, the user can navigate to the service)

In service load, events should adapt to the AGB pull operations. This means that push adapters adapt push messages to *ConnectionInitiator* or *Connection* objects, which can be used in AGB pull operations. Service indication messages are passed for the *BrowserUI* as *PushIndication* objects. The *PushIndication* interface includes methods for getting the original push indication message and the push connection where the push message is received.

4.1.3 AGB configuration

Because of the heterogeneity of the mobile devices, the mobile browser architecture should be highly modular and configurable. The flexibility and modularity of AGB is achieved by the interfaces so that all the core (parts offered by the reference implementation) and external parts can be configured to AGB as components. The *BrowserConfiguration* interface makes it possible to replace parts of the browser with new implementations (Figure 4.5).

The *BrowserUI* interface is used to configure user interfaces. It offers methods for receiving push indications, for handling authentication requests and for handling errors during the browsing. *BrowserUI* has also methods for notifying of object browsing started or stopped events. Browsed content can be received by adding object pull listeners to the browser configuration.

AGB object factories are used to handle connections and object plugins. The *ObjectFactory* interface offers methods for changing the configuration of plugins in runtime. Additionally, the connection and object factories can be replaced. AGB initialises the push adapters defined in the configuration. The configuration of push adapters can be changed in runtime by using the methods of the *PushAdapters* interface.

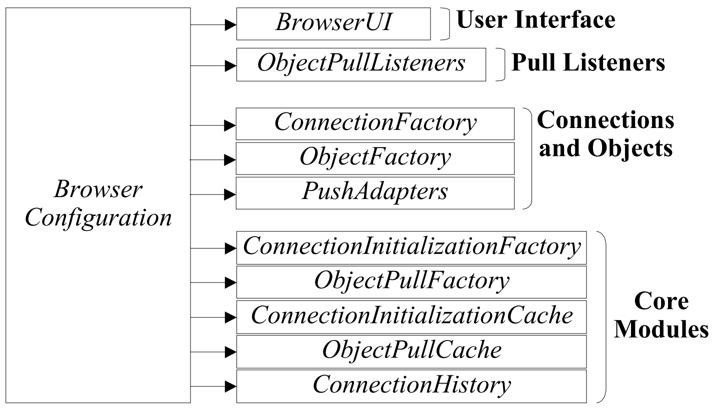


Figure 4.5. AGB configuration.

For making the AGB a highly transferable system, all the core parts of AGB can be replaced. This is important because there are great differences between platforms (e.g., Java MIDP vs. Java SE 2), and, so, it is important that also the core parts of the object browser can be replaced with solutions suitable for the environment in question. Connection initialization and object pull implementations can be used to replace reference *ConnectionInitializationFactory* and *ObjectPullFactory* implementations. The reference cache and connection history implementations can be replaced by introducing new implementations for the *ConnectionInitializationCache*, *ObjectPullCache* and *ConnectionHistory* interfaces.

4.1.4 Triggered browsing

In order to construct location- and context-aware browsing services, we have built a Triggered Browsing extension module (called ATB) for AGB. ATB extends the object browser so that pull and push browsing requests can be done with triggers (Figure 4.6). The triggered browser saves browsing requests and tries to execute the saved requests with specified rate (e.g., once in 3 seconds) in execution thread.

Browsing triggers makes it possible to implement various kinds of (e.g., location and context-based) triggering conditions. The *BrowsingTrigger* interface offers

- *isValidToExecute()*, and
- *isValid()* methods.

A browsing request is executed if the trigger notifies (the *isValidToExecute* method returns true) that browsing is valid to execute. If browsing request is not valid (e.g., the timestamp of the trigger is outdated), the request is removed from the triggered browser.

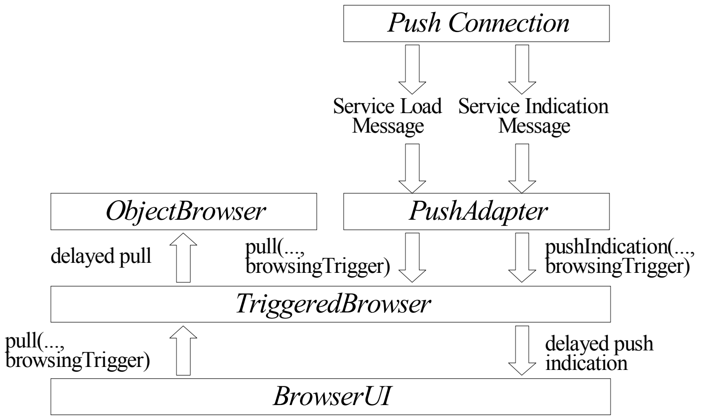
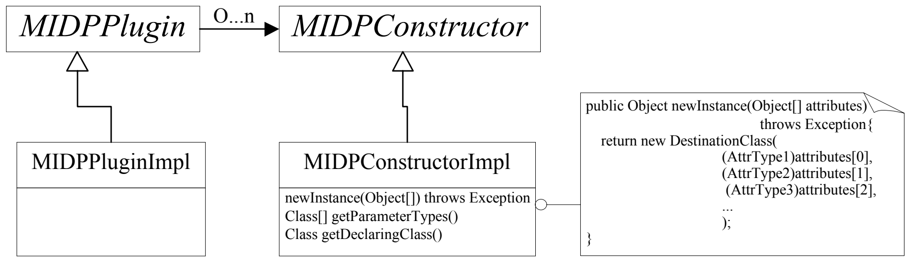


Figure 4.6. Triggered browser extends object browser with triggered browsing services.

Push adapters can map Service Load (SL) messages to connection or connection initiator objects, and, then, pass them for the triggered browser with specified triggers. The pull operation will be done when the browsing is triggered. Service Indication (SI) messages can be mapped to push indication objects and, then the push indications can passed for the triggered browser with specified triggers. The triggered push indication will be passed for the *BrowserUI*.

4.2 Reference implementations

In order to evaluate the AGB interfaces, we have done reference implementations of the core parts of AGB for Java 2 SE and Java MIDP environments. However, the environments set several restrictions for *ConnectionFactory* and *ObjectFactory* implementations. Java 2 SE and Java Personal Profile environments offer methods for creating new instances of Java classes with arguments. In object factory there could be a mechanism for selecting the correct constructor for given arguments. After the constructor is selected a new instance of class can be constructed. In Java 2 SE and Java Personal Profile environments object factory can be implemented so that new instances of classes are implemented by using constructors offered by Java (*java.lang.reflect.Constructor*).



MIDPPlugin
O...n
MIDPConstructor
MIDPPluginImpl
MIDPConstructorImpl
newInstance(Object[]) throws Exception
Class[] getParameterTypes()
Class getDeclaringClass()

```
public Object newInstance(Object[] attributes)
                               throws Exception{
   return new DestinationClass(
                       (AttrType1)attributes[0],
                       (AttrType2)attributes[1],
                       (AttrType3)attributes[2],
                       ...
                       );
}
```

Figure 4.7. The implementation of Java MIDP plugin.

However, Java MIDP offers no support for such constructors. It offers only a method for creating instances of classes without arguments. In Java MIDP environment, an object factory can be implemented by using *MIDPPlugin* and *MIDPConstructor* interfaces (Figure 4.7). The *MIDPPlugin* interface offers a method for getting available *MIDPConstructor* objects for the plugin. The *MIDPConstructor* interface offers methods for getting

- parameter classes for the constructor,
- the new instance of the destination class with attributes, and
- the destination class of the constructor.

The *MIDPPlugin* implementation is needed for all the plugin types. Object factory finds the suitable constructor for the given attribute types, and then uses the *MIDPConstructor* for constructing a new instance of the destination class.

In AGB, the content is browsed by object pull and push operations. In order to use the pull mechanism, available connections and object plugins has to be configured to the system by using the methods of connection and object factory interfaces. Names and class paths for the supported plugins can be defined in configuration files. When the AGB is started, the defined plugins are inserted to the active browser configuration. Figure 4.8 illustrates how plug-in components for the pull operation can be implemented and configured. A connection object gives the plug-in name to be constructed and the parameters for the constructor.

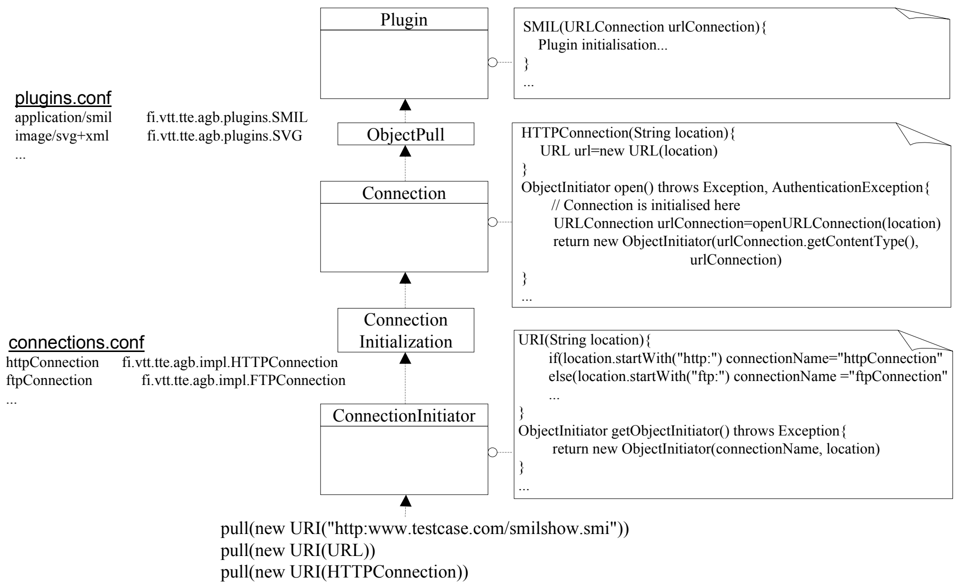

Plugin
SMIL(URLConnection urlConnection){
Plugin initialisation...
}
...
plugins.conf
application/smil fi.vtt.tte.agb.plugins.SMIL
image/svg+xml fi.vtt.tte.agb.plugins.SVG
...
ObjectPull
HTTPConnection(String location){
URL url=new URL(location)
}
ObjectInitiator open() throws Exception, AuthenticationException{
// Connection is initialised here
URLConnection urlConnection=openURLConnection(location)
return new ObjectInitiator(urlConnection.getContentType(), urlConnection)
}
...
Connection
Connection Initialization
connections.conf
httpConnection fi.vtt.tte.agb.impl.HTTPConnection
ftpConnection fi.vtt.tte.agb.impl.FTPConnection
...
URI(String location){
if(location.startWith("http:") connectionName="httpConnection"
else(location.startWith("ftp:") connectionName ="ftpConnection"
...
}
ObjectInitiator getObjectInitiator() throws Exception{
return new ObjectInitiator(connectionName, location)
}
...
ConnectionInitiator
pull(new URI("http:www.testcase.com/smilshow.smi"))
pull(new URI(URL))
pull(new URI(HTTPConnection))

Figure 4.8. Implementation of a connection initiator, connection and object plug-ins.

In the pull operation with *ConnectionInitiator*, the connections are constructed in *ConnectionInitialisation* threads (Figure 4.9). After connection is constructed object pull operation with the connection can be started. A temporary connection can be implemented so that in the connection implementation there is a timer, which closes the connection after a certain time interval and assigns the connection to be not valid to kept in cache. Opened connection can also be closed and removed by using the methods of the connection initialization cache.

If authentication is required (for the connection to be opened), an authentication request is thrown, and AGB passes the request for the *BrowserUI*. After the connection is authenticated, object pull operation can be completed. Next, when object pull is completed, object pull listeners are notified of the new object.

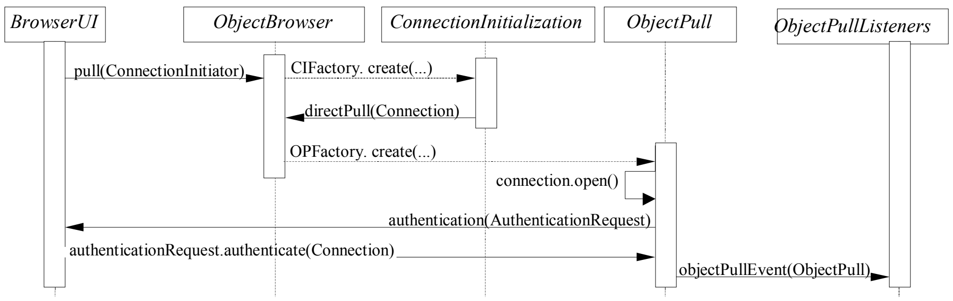
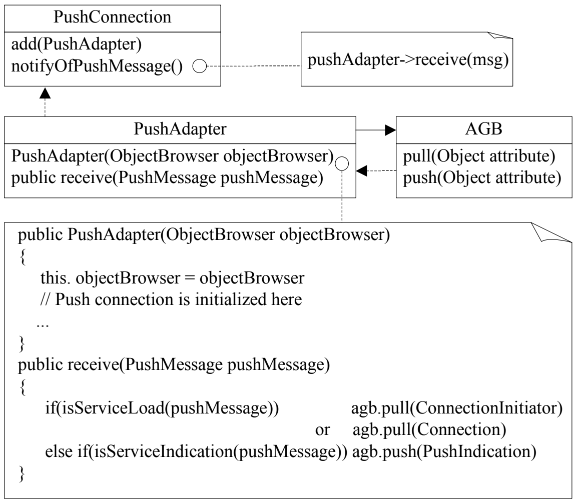


Figure 4.9. UML sequence diagram for object pull operation with connection initiator.

Figure 4.10 shows how a push adapter can be implemented.



```
public PushAdapter(ObjectBrowser objectBrowser)
{
    this. objectBrowser = objectBrowser
    // Push connection is initialized here
    ...
}
public receive(PushMessage pushMessage)
{
     if(isServiceLoad(pushMessage))            agb.pull(ConnectionInitiator)
                                           or  agb.pull(Connection)
     else if(isServiceIndication(pushMessage)) agb.push(PushIndication)
}
```

Figure 4.10. A push connection is added to AGB by using a push adapter structure.

Push messages can be received by implementing an adapter, which maps received messages to object pull or push indication operations (Figure 4.11). WAP push connection supports service load (SL) and service indication (SI) type-of messages. The SL message push adapter can be used to construct either a *ConnectionInitiator* or *Connection* type-of object and then starts an object pull operation with the constructed object. The SI message push adapter can be used to construct a *PushIndication* object, which can be sent for *BrowserUI*. The user gets an indication of the available push service and can browse to the content referred in the message. The push adapters can also be used for implementing instant messaging mechanisms.

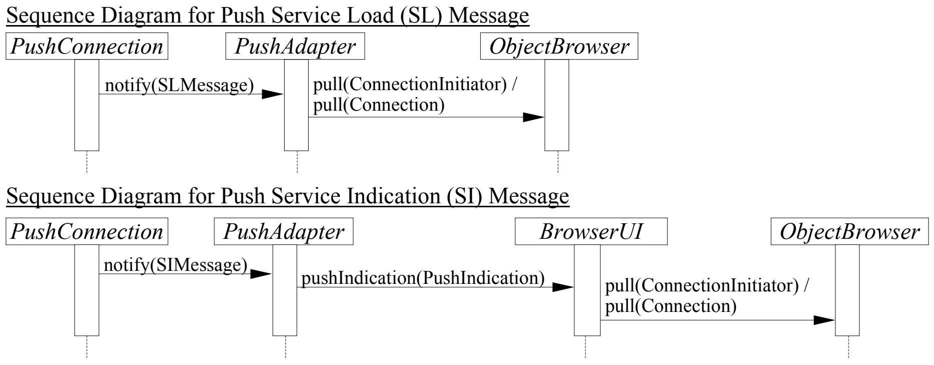

Sequence Diagram for Push Service Load (SL) Message
PushConnection
PushAdapter
ObjectBrowser
notify(SLMessage)
pull(ConnectionInitiator) / pull(Connection)
Sequence Diagram for Push Service Indication (SI) Message
PushConnection
PushAdapter
BrowserUI
ObjectBrowser
notify(SIMessage)
pushIndication(PushIndication)
pull(ConnectionInitiator) / pull(Connection)

Figure 4.11. UML sequence diagram for object push events.

In order to test the ATB, we implemented an information service where the browser automatically browses content from various kinds of web sources for the mobile user. We collected first a sequence of web addresses of (e.g., news, stock rates, and weather forecasts) services and, then, constructed a trigger, which triggers browsing in certain times. Subsequently, we registered those addresses and triggers to the triggered browser and launched the information browsing service.

We tested the triggered push with WAP push mechanism. In that the push indications of new browsing content where delayed so that the indications where passed for the browser UI after some certain triggering condition was fulfilled. A triggering condition can be based on the context of use; for example, if the user is at work (context), the browsing context can be delayed, but after user comes to home (context) the content is shown to the user.

We have used the reference implementations as a base for mobile browser implementations in Java SE 2, Java Personal Profile, and Java MIDP environments (Figure 4.12). We have implemented different kinds of pull (e.g., HTTP, RTP, Reader) and (e.g., WAP) push connections and (e.g., text, image, audio, video and WML, SMIL, XHTML) content type plugins for AGB based browsing. Toolkit, µBrowser and MIDBrowser implementations are introduced in Chapter 7. However, because AGB provides a generic framework based on interfaces, practically any connection and content type implementations can be introduced by providing implementations of required AGB interfaces.

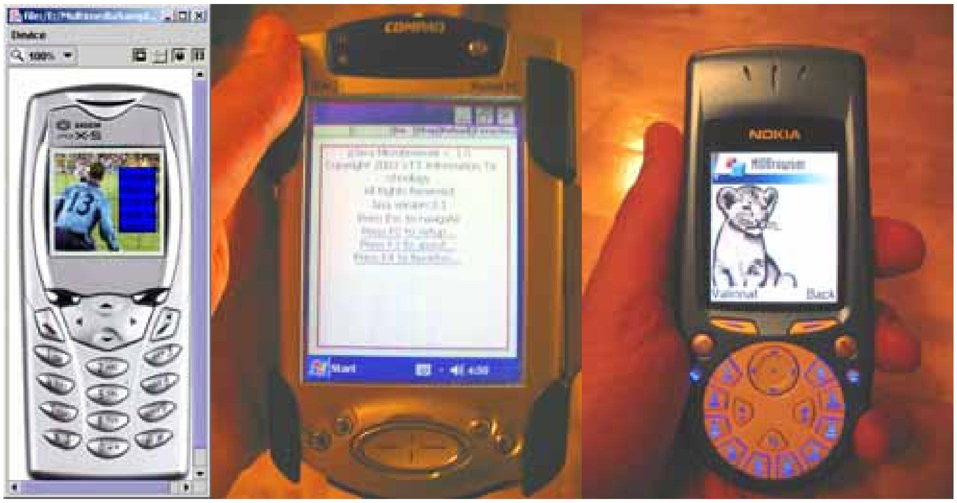


Figure 4.12. AGB in Java SE 2, Java Personal Profile and Java MIDP environments.

5. MIMEFrame - Framework for mobile user agents, content types and multimodal controls

In forthcoming mobile browsing applications there are different kinds of contents to browse, different kinds of user agents presenting that content, and multimodal controls available. So, there are requirements for frameworks, which support the construction of the client-side of mobile browsing applications. A framework should introduce general guidelines for client-side implementations so that created parts are consistent, reusable and can co-operate. To be suitable for mobile environment, we have identified several requirements. The framework should support

- different kinds of user agents and implementation platforms,
- different kinds of content types,
- combining different kinds of content to work together (e.g., VoiceXML and XHTML),
- using of browsing and content fetching services,
- using of the profiling information (e.g., UAProf, Style sheets and user preferences),
- multimodal controls, and
- extensions (so that any kind of new services and resources can be added to the framework in the future).

Further, for making the framework suitable to various kinds of mobile platforms, it should be

- very light,
- highly transferable so that it can work in different platforms, and

- generic and extendable to fulfil the future needs.

To our knowledge, there are no previous publications of frameworks fulfilling the described requirements. We have developed a framework, called MIMEFrame, which aims to fulfil the above requirements.

This chapter is organised as follows. First, an architecture for client-side of mobile browsing application is introduced in Section 5.1, then the MIMEFrame is described in Section 5.2 and a reference implementation and use cases for MIMEFrame are introduced in Section 5.3.

5.1 Architecture

The client-side of mobile browsing application can be divided to *user agent*, *content* and *controls* (Figure 5.1). User agent describes the use environment and presents the content and controls are a way to control the mobile browsing application. In this work these co-operating parts combine the base architecture for the client-side of mobile browsing applications. The requirements what we have identified for the user agent, content and controls parts are described in the following subsections.

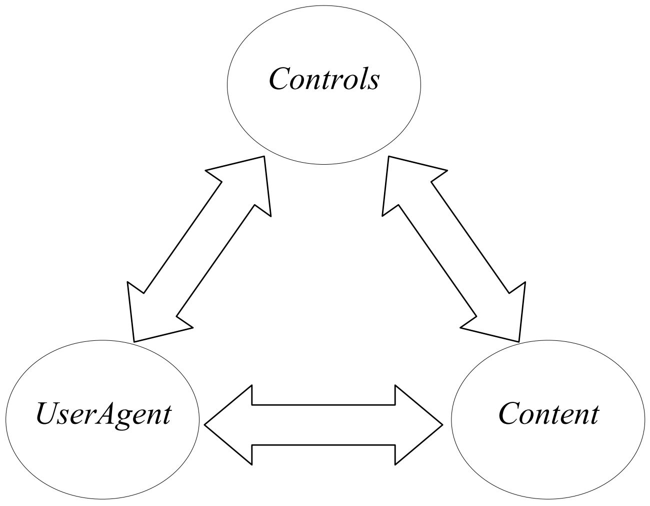


Figure 5.1. Base components of the MIMEFrame architecture.

5.1.1 User Agent

In this work, the user agent is defined as an abstract model for the mobile device and the user. To be suitable for mobile browsing applications, the user agent should offer

- access to browsing services,
- extensible resources and services,
- support for using various kinds of user interfaces in presenting the content.

The user agent is capable to present content and to offer services for the other parts (e.g., for the content types) of the mobile browsing application. Access to the browsing services should offer methods for asynchronous browsing, for object fetching, for navigating in the browsing history, and for reloading the downloaded content. Asynchronous non-blocking content fetching is required, because both opening the connections and downloading the content are normally much slower in mobile Internet than in wired Internet and so content fetching is not allowed to block the UI of the browser. To be suitable for mobile use the push type-of browsing should also be supported.

The context of use and characteristics of mobile environment (e.g., device and networks) should be taken into account in mobile browsing applications. The user agent should offer various kinds of (e.g., user agent profiles (Section 2.3.1), user preferences, style sheets) resources for the server side (e.g., the size of the browsing application can be adapted to fit for the current network and device) and for the client side (e.g., style sheets) adaptation. User agent should also offer methods for content types co-operation. User agent should enable content types to share and use various kinds of resources and services offered by other content types.

The user agent should support use of various kinds of UI components in presenting the content. There should be methods for updating the user interface, getting the actual UI component, and handling the errors. In some cases, access to the actual UI component is needed for using the methods of the used UI implementation. Also, methods for handling errors raised during using the mobile browsing application are needed.

5.1.2 Content

The content is defined as the actual content, which is presented in the user agent. For allowing the presentation and co-operation of different kinds of content types in mobile devices, content types should offer methods for

- creating new instances of the content to be presented in user agent,
- rendering the content and for updating the content rendering components,
- querying the content type,
- offering various kinds of controls for the content,
- receiving (e.g., controller) events from various sources, and
- closing the browsed content so that all the resources related to the closed content are released.

A new content type instance should be constructed by using resources offered by the user agent (e.g., user agent profile, user preferences and style sheets). If resources are changed the content type instance should be updated. Content type should offer methods for rendering the content and for updating the components related to it. For minimising the use of memory resources, content type should release all the resources related it when it is closed (e.g., when a new content is browsed).

A flexible mechanism for controlling different kinds of content types is needed. Content type should support the use of various kinds of controls so that it is possible to add and remove controls in run-time. Content types should also offer methods for receiving (e.g., controller) events from separate sources.

5.1.3 Controls

The mobile browsing applications can be controlled by using various kinds of controls (e.g., keyboard, touching screen or voice). In this work, controls are divided to modal and non-modal controls. For modal controls there should be methods for

- handling modal input events, which come from various input controllers,
- adding and removing new event listeners in run-time so that, e.g., user agent and content types can handle events came from various sources,
- defining the active modal event listener (focus owner), and
- moving the focus.

A modal event is handled so that only the current focus owner handles the event. There could be multiple modal event listeners available (e.g., offered by various kinds of content types). For instance, in a SMIL demonstration, there can be several content types having own modal event listeners. There should be methods for moving (e.g., by using tabulator in keyboard or by using some voice commands) the focus in modal listeners hierarchy. Modal listeners can be combined to the tree structure (Figure 5.2), which can be used, e.g., with the container content types. The focus can be moved to the correct modal event listener by using the tree structure.

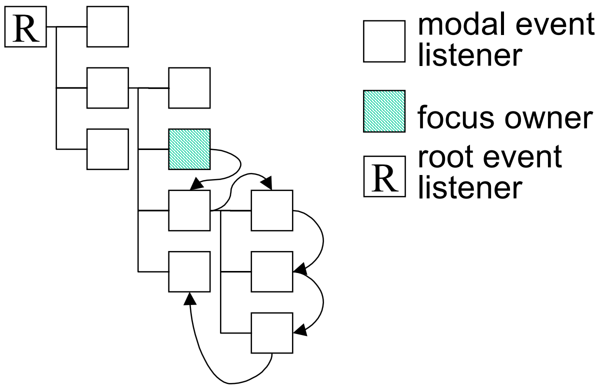


Figure 5.2. Modal event listeners in tree hierarchy.

In mobile browsing applications, there can also be non-modal controls. Non-modal events are arisen outside, e.g., there could be time and location-based events and mechanisms by which the other users can control the browsing application. In order to handle the non-modal events, there should be methods for receiving non-modal events coming from various sources as well as mechanisms for notifying listeners of the non-modal events.

5.2 MIMEFrame framework

The MIMEFrame framework, based on the client-side architecture introduced in Section 3, fulfils the requirements described for the user agent, content and controls components. MIMEFrame supports the creating of different kinds of content types for different mobile user agent implementations with modal and non-modal controls available. MIMEFrame contains three main interfaces: *UserAgent*, *MIME* and *Controls* (Figure 5.3).

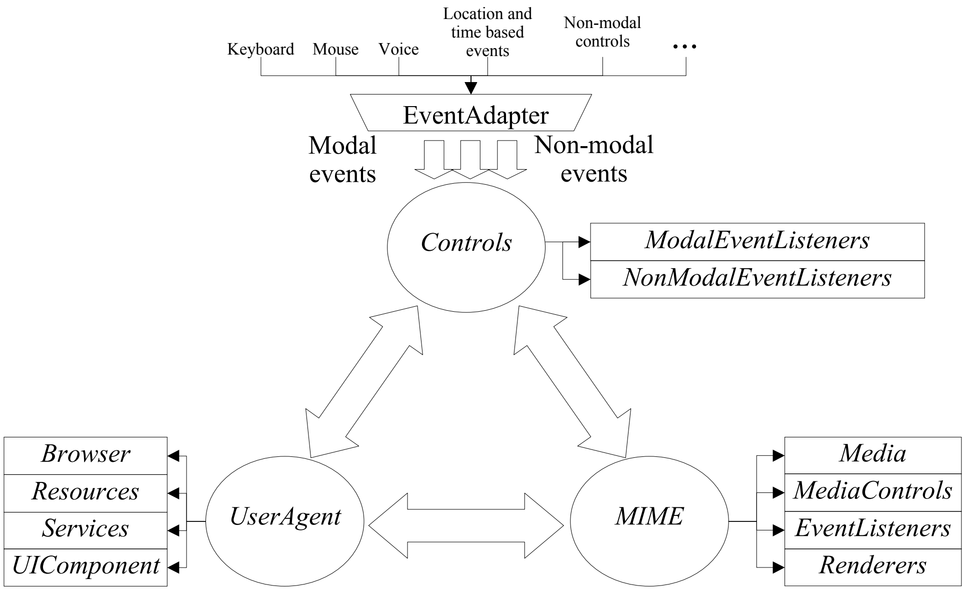


Figure 5.3. MIMEFrame is a framework for user agents and content types with multimodal controls available.

5.2.1 User agent

UserAgent is an interface for user agent implementations to present different kinds of content types (*MIMEs*). The *UserAgent* interface offers access to

- browsing and fetching services (*Browser*),
- a user interface component (*UIComponent*),
- extensible resources (*Resources*), and services (*Services*).

The *Browser* interface offers access to browsing and object fetching services. It offers methods for

- asynchronous browsing and object fetching (browsing can be done in the thread),
- moving in the navigation history (back and forward), stop (browsing in the background is stopped), reload (last browsed object is loaded again), and
- object push.

The *Browser* interface is designed so that the browsing can made asynchronously in an own thread so that both opening the connections and fetching the objects can be done in the background. For thread-based object fetching the *ObjectFetchingAttributes* interface offers methods for getting the name and attributes of the connection. The fetched object can be returned (e.g., from the thread) to the caller by using the *objectFecthed(Object)* method in the *ObjectFetchingAttributes* interface.

The *UIComponent* interface offers methods for getting the actual UI component, for repainting the user interface, and for handling errors. *UIComponent* provides access to the actual UI component, which can be used for utilising the methods of the UI implementation. UI component can be used for presenting all kinds of errors, which are raised during the use of the browsing application.

User agent and various kinds of content types can share resources by using methods of the *Resources* interface. Resources can contain (e.g., user agent profile, style sheet, user preferences and the user agent configuration) information, which can be used in presenting and profiling the content. The *Resources* interface offers methods for

- adding and removing resources,
- getting resources by name and getting enumeration of the available resources, and
- notifying the registered resource listeners of the changes in the resources.

New services (e.g., location or context-aware services) can be offered for the user agent and content types by using methods of the *Services* interface. The *Services* interface has methods for

- executing available services,
- adding, and removing services, and
- querying the names of the available services.

Extensible services enable co-operation of different kinds of content types. It is a way for content types to use services offered by other content types. Various kinds of services can be implemented by using a *Service* interface, which offers a method: *Object executeService(Object[ ] attributes) throws Exception*.

Services can be executed by giving the name and the attributes for the service. If the service for given name is found, the service is executed. The result of the service is returned as an object. If errors are raised during the service execution, an exception can be thrown.

5.2.2 MIME

The *MIME* interface enables the implementation of different kinds of content types. The *MIME* instances can co-operate (e.g., in the container content types like in XHTML or SMIL) and be presented in user agent. The *MIME* interface include methods for

- getting new instances of the *MIME*,
- rendering the content in *MIME*,
- updating the *MIME*, and
- closing the *MIME* type.

When a *MIME* object is downloaded, a new instance of the *MIME* type can be obtained by calling the ***getInstance(UserAgent)*** method (Figure 5.4) and using (e.g., profiling information) resources registered to the user agent. The created *MIME* instance can register the services, which it can offer, to the user agent services. *UserAgent* offers access to the browsing services, and container (e.g., XHTML or SMIL) *MIME* types can use these services for fetching the (e.g., video or image) content in presentations.

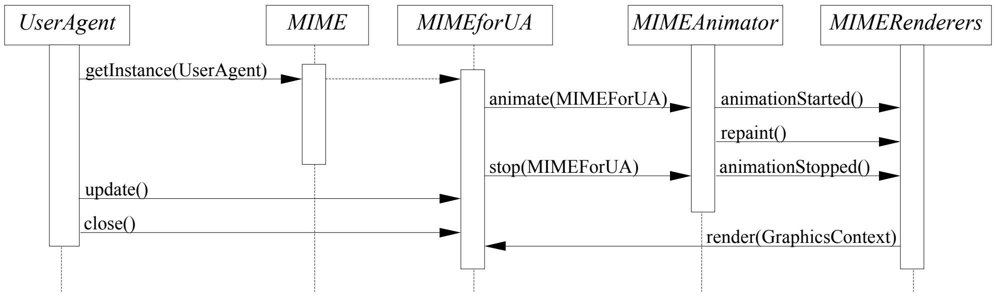


Figure 5.4. A UML sequence diagram of using MIME.

The content in *MIME* can be visualised by using a render method of the *MIME* interface. The *GraphicsContext* implementation depends of the environment where the *MIME* is rendered, e.g., graphics methods are very different in Java 2 SE from those in Java MIDP environment. *MIME* can use methods of current graphics context by casting the given graphics context to the type known by the *MIME*. If the available resources or services are changed, the update method can be used for refreshing the *MIME* type. When the *MIME* type is not any more used, the memory resources related to the *MIME* type can be released by the close method.

A *MIMERenderer* is an interface for *MIME* rendering components. For handling *MIME* rendering components, the *Renderers* interface offers methods for

- adding and removing rendering components,
- repainting the registered rendering components, and
- notifying the registered rendering components when *MIME* playing is started or stopped.

The *MIMEAnimator* interface offers methods for animated content types. *MIMEAnimator* works so that it repaints the rendering components of animated *MIMEs* with specified frequency. When animating is started or stopped the rendering components of animated *MIME* type are notified.

The *MIME* interface offers access to the *Media*, *MediaControls*, and *EventListeners* interfaces. The *Media* interface offers methods for getting the original media object, the top-level (e.g., Text, Audio, Video and Multipart) and full media type (e.g., text/html), and the location of the content (e.g., http://test/image.jpg). The top-level and full media types (of the *Media* interface) conform to the RFC 2045 and 2046 standards. Standards RFC 2045 and 2046 specify the format for transmitting different kind of content in messages. For Multipurpose Internet Mail Extensions (MIME), RFC 2046 define five discrete top-level media types: text, image, audio, video and application and two composite top-level media types: message and multipart.

The *MediaControls* interface enables *MIMEs* to offer various kinds of media controls. The *MediaControls* interface offers methods for

- adding and removing media controls, and
- getting the enumeration of the available media controls.

Media controls enables one to control any kind of media content; the only requirement is that the controlling component knows the types of the available media controls.

MIME types can be notified of events by using event listeners. The *EventListeners* interface offers methods for adding and removing available event listeners, and for getting the enumeration of available event listeners. A *MIME* type can offer various kinds of event listeners, which can be used, e.g., for notifying of modal or non-modal controller events.

5.2.3 Controls

Controller events coming from various sources can be handled by using the methods of the *Controls* interface. Controller events can be handled by implementing an input adapter. An input adapter should put the incoming events, first, into a event queue (in temporal order), and, then, classify the (e.g., keyboard, mouse or voice controller) events to modal or (e.g., time based events) non-modal events, and, finally, pass the events to *Controls*. The *Controls* interface offers methods for receiving modal and non-modal events. Modal events are handled exclusively so that only one modal event listener (focus owner) handles the event. Three phases are described below.

1. The event is first sent for the *RootEventListener*. If the event is handled, more steps are not made. Otherwise, there are continued to the phase two.

2. In the second phase, the focus is requested from *RootEventListener* (a modal event is given as an input). If the focus is changed (focus is gained or lost), no more steps are made. Otherwise, handling proceeds to the phase three.

3. In the third phase, focus owner (if available) handles the modal event.

The hierarchical structure of modal event listeners can be constructed by using the *ModalEventListenerHierarchy* interface, which offers methods for moving the focus correctly in the modal event listeners hierarchy. The *ModalEventListener* interface offers methods for

- getting the focus (it gets the modal event as an input and returns the focus),
- handling the modal event, and
- getting the hierarchy of modal event listeners.

Non-modal events are passed to the all non-modal event listeners. Non-modal event listeners can be added or removed from *Controls* at run-time. Non-modal events are handled so that

1. non-modal event is first sent for *RootEventListener*, and
2. after that the event is sent to the other non-modal event listeners.

Each *NonModalEventListener* decides how to react to the event. If a listener can cast the non-modal event to the known (e.g. time-based) event type, it can handle the event, otherwise the event is skipped.

5.3 Reference implementations

In order to evaluate MIMEFrame, we have done reference implementations for the *UserAgent*, *Control* and *MIME* components of MIMEFrame with Java, which we have tested in Java 2 SE, Java Personal Profile and Java MIDP environments. Reference implementation offers interfaces for different kind of resources. The *UAProf* interface offers methods for reading and setting values from/to the user agent profile. It supports the attribute value types defined in the UAProf specification [UAPROF] (e.g., Literal, Dimension, Number and Boolean) (cf. Section 2.2.1). The *UserPreferences* interface is for reading and setting values from/to user preferences. User preferences contain information related to the current contexts of use, e.g., there can be set values for keys like "Show subtitles", "Use large fonts" and "Silent Mode". The *UAStyleSheet* interface offers methods for getting the actual style sheet (e.g., based on interfaces defined by W3C). The *UserAgentConfiguration* interface offers access to the textual key value pairs, which can be used for configuring the user agent. User agent configuration can contain, e.g., information related to the user interface (e.g., information of the layout), path settings, and other values, that the user agent can use.

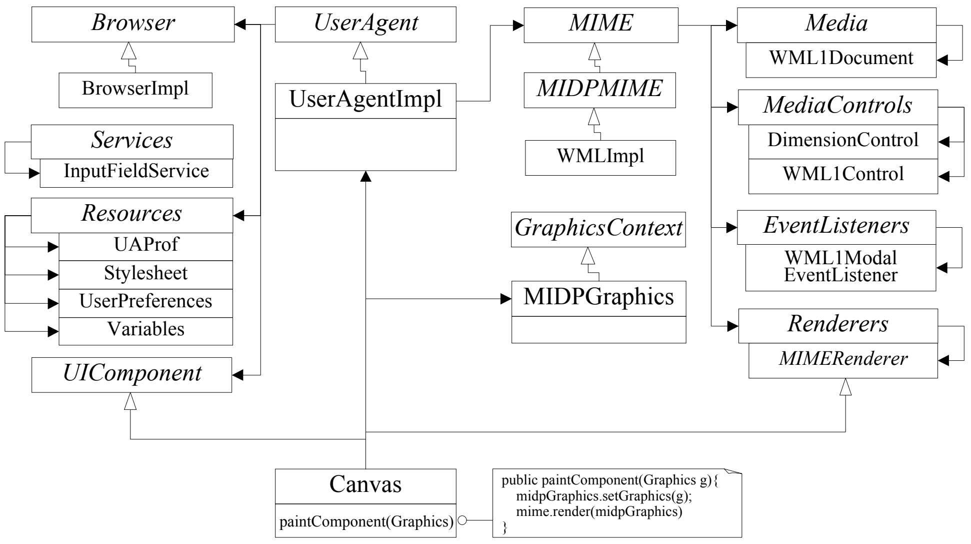
The reference implementation offers the following interfaces to control media types:

- *PlayControl* (for controlling video and audio types),
- *MediaTimeControl* (for real-time content types),
- *DimensionControl* (for getting the visible dimension of the media), and
- *TextControl* (for getting the textual presentation for the media).

We have implemented text, image (e.g., jpeg, gif, png), WML, XHTML, SMIL, VoiceXML and real-time audio and video content types. The following subsections show how the reference implementation of MIMEFrame is used in Java MIDP environment, and there are examples of presenting mobile browsing content with resources and multimodal controls.

5.3.1 MIMEFrame in Java MIDP environment

We used AGB (Chapter 4) to implement the browsing features in Java MIDP environment. User agent profile, style sheet, user preferences and variables are available in the user agent resources. The user agent offers an input field service for the content types. The service opens an input field and when the editing is done the editing results are returned to the caller. WML plugin registers *Variables* to the resources of user agent. This means that other content types can modify attribute values and, so, they can effect to the WML presentation. E.g., a VoiceXML content type plugin could be used to modify these attribute values. The canvas is a UI component for presenting various kinds of content types.



```
public paintComponent(Graphics g){
    midpGraphics.setGraphics(g);
    mime.render(midpGraphics)
}
```

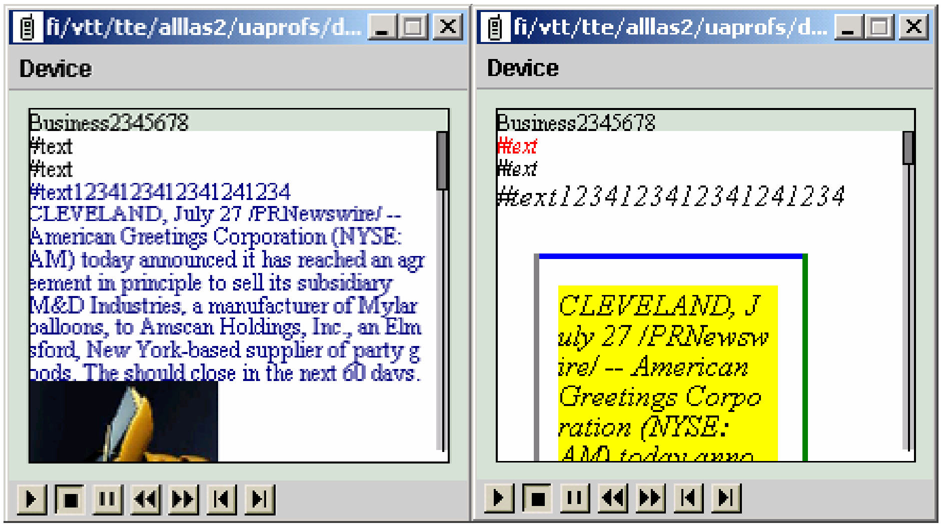
Figure 5.5. MIMEFrame in Java MIDP environment.

Figure 5.5 shows how MIMEFrame is used in Java MIDP environment. *MIDPMIME* is an extension for *MIME* types in the MIDP environment. *MIME* types are rendered so that the render method of *MIME* is called in the paint method of the canvas, and the graphics context (*MIDPGraphics*) is given as an attribute. The *MIME* type can render the content in it to the given graphics context. The graphics context can offer access to the actual graphics context and then it can offer additional information needed (e.g., background and foreground colours and fonts) for rendering. The graphics context can define the bounds for the rendering area so that a content type (e.g., image) in a (e.g., SMIL) container content type can render the content to the given bounds.

The Java MIDP user agent is controlled by a keyboard. Keyboard events are mapped to modal key events and then passed to *Controls*. User agent has *RootModalEventListener* for the controller events. If the user agent does not handle the event, the event is sent for the focus owner. *Controls* can notify the WML plugin of modal key events by using the modal event listener (*WMLModalEventListener*) offered by WML plugin. With keyboard it is possible, e.g., to navigate in WML documents. *WMLPlugin* can be controlled by dimension (*DimensionControl*) and by using WML plugin specific controls (*WMLControls*).

5.3.2 Presenting mobile content with resources and multimodal controls

Figure 5.6 shows an example of using resources in presenting the content types. In the example, UAProf and style sheet resources are applied in XHTML presentation.


fi/vtt/tte/alllas2/uaprofs/d...

Device

Business2345678
#text
#text
#text1234123412341241234
CLEVELAND, July 27 /PRNewswire/ --
American Greetings Corporation (NYSE:
AM) today announced it has reached an agr
eement in principle to sell its subsidiary
M&D Industries, a manufacturer of Mylar
balloons, to Amscan Holdings, Inc., an Elm
sford, New York-based supplier of party g
oods. The should close in the next 60 days.

fi/vtt/tte/alllas2/uaprofs/d...

Device

Business2345678
#text
#text
#text1234123412341241234
CLEVELAND, J
uly 27 /PRNewsw
ire/ -- American
Greetings Corpo
ration (NYSE:

Figure 5.6. An XHTML document presented via two different style sheets.

We have constructed a testing environment for multimodal controls with multiple available content types. In the implementation, input commands can be given by using keyboard, mouse or by a voice command prompter. Figure 5.7 shows a combined VoiceXML and WML presentation where a voice command prompter can be used for moving the focus, for selecting the objects and for giving various kinds of input commands.

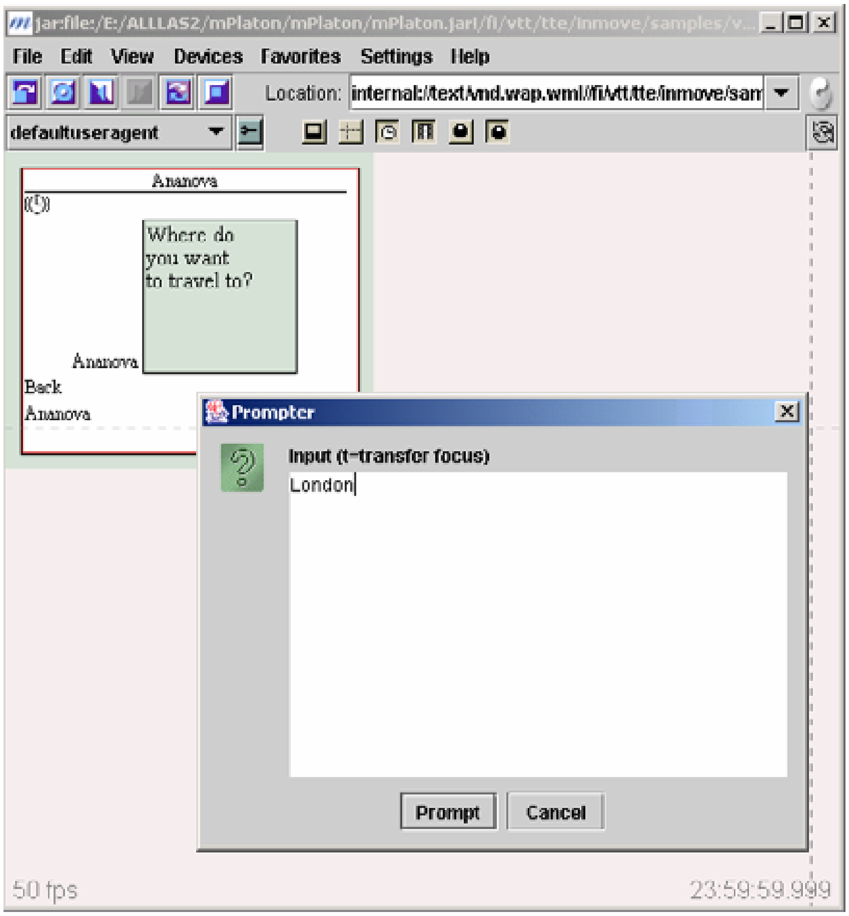

jar:file:/E:/ALLLAS2/mPlaton/mPlaton/mPlaton.jar!/fi/vtt/tte/inmove/samples/v...
File Edit View Devices Favorites Settings Help
Location: internal:/text/vnd.wap.wml//fi/vtt/tte/inmove/sam
defaultuseragent
Ananova
Where do you want to travel to?
Ananova
Back
Ananova
Prompter
Input (t=transfer focus)
London
Prompt
Cancel
50 fps
23:59:59.999

Figure 5.7. VoiceXML combined to WML presentation.

6. FEdXML – Framework for XML editors

In the future, consumers will produce content in mobile devices in real time, like they now do in the wired Internet [DiB01]. Consequently, there is a great need for content producing tools. This chapter concentrates on XML editors and introduces a new generic framework for component-based XML editors (called FEdXML), which can be used as the base of XML editor implementations. First, in this chapter, we provide a classification of different kinds of XML editors and outline requirements for a modular architecture of XML editors in Section 6.1. Next, in Sections 6.2 and 6.3, the FEdXML architecture and framework are described in detail. Reference implementation for FEdXML and practical use cases for the framework are presented in Section 6.4.

In order to enable non-experts to construct XML (e.g., SMIL, SVG and VoiceXML) applications, the usability of XML editing tools should be noticed [Arn99, Jok03]. In general, an XML editor should

- ❑ show the XML content as illustrative as possible,
- ❑ offer effective tools for XML handling,
- ❑ make sure that the edited content follow the language specification, and
- ❑ guide and instruct the user while editing the XML-content.

In this work, we divide XML editors into four main categories:

1. Textual editors,
2. Generic XML editors,
3. Configurable XML editors and
4. Language specific XML editors.

We mainly consider the creation of XML editors that are capable of editing the XML model directly and, in particular, keeping the edited content valid. A textual source is understood to be an XML document if it complies with the XML well-formedness constraints defined in the XML definition [BP+00]. An XML document is valid if it is well-formed and follows the rules, and the model defined in an associated Document Type Definition (DTD), which is a description, which defines the elements, attributes and structure of the target XML language. Schema languages like XML Schema [Fal01] by W3C and RELAX NG by OASIS and ISO/IEC JTC1, support namespaces and data types (e.g., numeric data types). Schema provides a markup vocabulary. It describes formally permissible names for tags and attributes and permissible structural relationships between such tags and attributes [MT+02].

A generic XML editor

- is applicable for any XML-based language,
- uses DOM as a internal representation of XML,
- offers controls for XML editing and navigating in the document structure, and
- includes a visualisation view to each document being edited.

The benefit of a generic XML editor is that it can be used with any XML-based language. However, the problem of the general approach is often the poor usability. Several usability problems may arise if the model does not sufficiently notify the characteristics of the target XML language, if the views are not designed to show the content of the target XML language, and if there are only generic editing controls (not language-specific controls like wizards or templates).

The usability of a generic XML editor can be improved by implementing configuration mechanisms that can be used to adapt the editor for a specific XML-based language (Figure 6.1). Configurable XML editors have generic model, view and controller components, which can be configured by using some external descriptions. However, in this solution, the views and controllers should be general enough to suit different XML languages and, also, capable of adapting to the characteristics of specific XML languages.

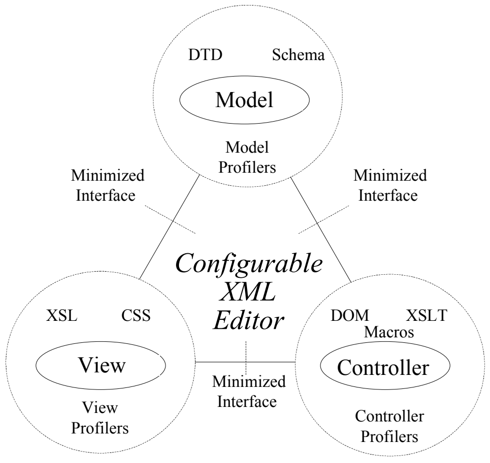


Figure 6.1. Configurable XML editor.

The characteristics of XML languages vary a lot. For example, the creation of SMIL presentations differs greatly from the construction of VoiceXML applications. So, there is a need for language-specific XML editor features that may be implemented by adding language-specific MVC components to the generic XML editor. The language-specific components should be designed to work perfectly with the XML-based target language. A model for a specific XML language should follow the language rules exactly so that all the editing complies with the rules defined in the language description. Information and functions relevant to the current editing job should be easily available to the user. For editing there should be language-specific controls like special commands, wizards and templates to make the editing as easy and efficient as possible.

6.1 Requirements for XML editor framework

In order to minimise the effort to construct XML editors for different kinds of environments (desktop and mobile), an XML editor framework should (at least) introduce the common core modules of the XML editor. However, the mobile environment sets many requirements for application development. In mobile devices the memory and processing power and input and output capabilities are normally very limited. Also, the capabilities of mobile devices vary a lot. Thus, the framework should be

- very light and scalable, so that it works on desktop and mobile environment;
- generic, so that it is not dependent on any specific (e.g., a parser) implementation;
- transferable, so that framework and implementations related to it can be transferred to very different kinds of devices and environments; and
- reusable, so that the framework and related parts can be used in various kinds of XML editor configurations.

Because of the great diversity of mobile environments, the framework should offer a modular approach to construction of XML editors. Modular approach enables one to

- construct XML editors of small well-tested reusable modules, which implement interfaces defined by the framework, and
- replace specific parts of XML editor with new implementations so that XML editor can work in various kinds of environments.

To keep the framework reusable and transferable to very different kinds of environments the core of the framework should include only interfaces. The core interfaces should

- be possible to be implemented in various kinds of environments (e.g., J2ME and Symbian) with various kinds object-oriented languages available (e.g., with the Java and C++),
- offer base for implementing the modules of XML editor, and
- be based on standard interfaces, so that editor components are as generic as possible.

This report describes a new generic framework for XML editors, FEdXML, which seeks to address all the above requirements. FEdXML introduces a generic architecture for modular XML editors, where the XML editor is divided by using the Model-View-Controller structure (cf. Section 6.2). FEdXML is a light, scalable, and highly transferable architecture (Section 6.3). The core of FEdXML contains only interfaces and it makes it possible to construct XML editors for various kinds of environments. FEdXML relies strongly on standard XML interfaces introduced by W3C.

In order to evaluate the FEdXML, we have constructed reference Java implementations (cf. Section 6.4) for the core parts of the framework. These reference implementations can be used as a base for generic, configurable and language-specific XML editor implementations.

6.2 Architecture for modular XML editors

We have identified the following requirements for the *Model-View-Controller* (MVC) [BM+96] parts of XML editor. The *XML model* should offer methods for

- querying what it is possible to do for the model according to the underlying grammar,
- executing various kinds of editing operations,
- validating the edited model, and
- updating changes in XML model to the other parts of the editor.

XML view should visualise the content in XML document and *XML controller* should offer methods for XML content editing. In addition, in order to construct configurable XML editors there is a need for methods for configuring the XML

- model (e.g., by enhancing the editing and validation methods),
- views (e.g., by configuring the visual appearance of XML editor), and
- controllers (e.g., by defining new XML editing controls like macros and wizards).

In Figure 6.2, the base architecture model notifying the above requirements is shown. The modular architecture is combined of XML model, view, controller and configuration methods parts. These core modules co-operate and form the skeleton for the XML editor implementations.

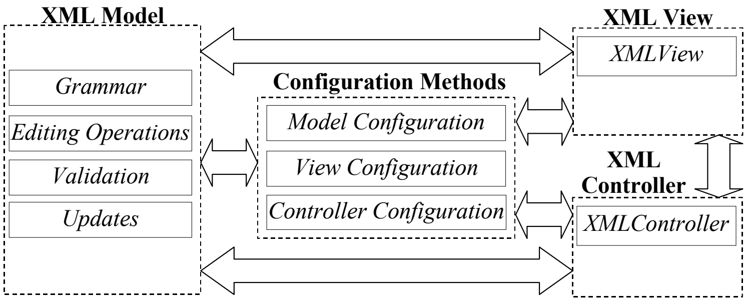


Figure 6.2. Architecture for modular XML editors.

6.3 FEdXML framework

The FEdXML framework is based on the architecture model defined above in Section 6.2. FEdXML divides the XML editor by using the *Model-View-Controller* (MVC) structure and offers methods for configuring the MVC parts of XML editor (Figure 6.3).

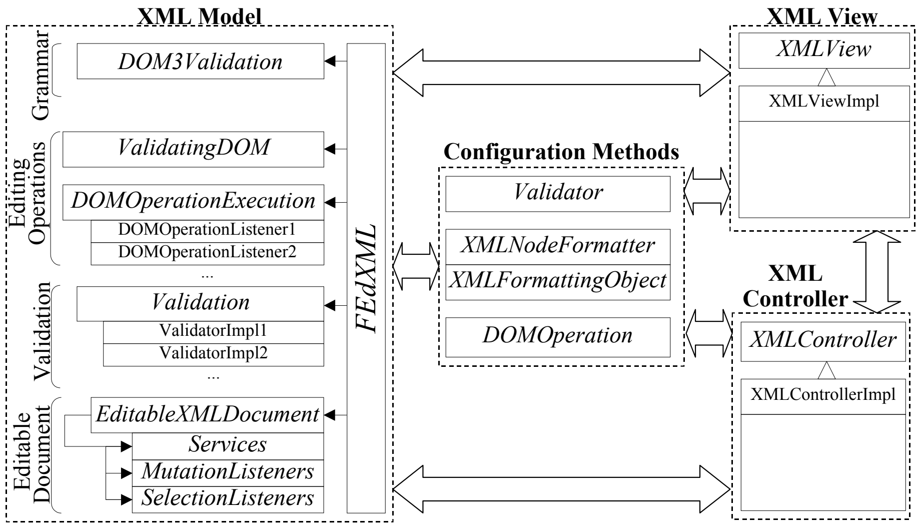


Figure 6.3. FEdXML architecture.

6.3.1 XML model

FEdXML provides interfaces for accessing the underlying grammar (*DOM3Validation*), for executing editing operations (*ValidatingDOM* and *DOMOperationExecution*), for validating the content (*Validation*), and for accessing the editable XML document (*EditableXMLDocument*). These core interfaces of FEdXML are described in the following subsections.

6.3.1.1 Grammar

W3C has published (currently a candidate recommendation) validation interfaces for DOM (level 3) [CKR03] for making the queries to the underlying grammar. The interfaces offer methods for making queries like where the nodes or attributes can be added to or removed from. They can be accessed by using the methods of the *DOM3Validation* interface of FEdXML.

6.3.1.2 Editing operations

FEdXML offers DOM operations for implementing primitive or more advanced XML editing operations like macros and wizards. The *DOMOperation* interface offers methods for

- executing the XML editing operation,
- getting a mutation event for the operation, and
- getting an inverse operation for cancelling the operation.

When a DOM operation is executed the mutation listeners of XML document and the DOM operation listeners are notified. For example, an undo manager implementation can be a DOM operation listener. The *DOMOperation* interface offers methods to implement undo- and redo-operations for DOM editing operations.

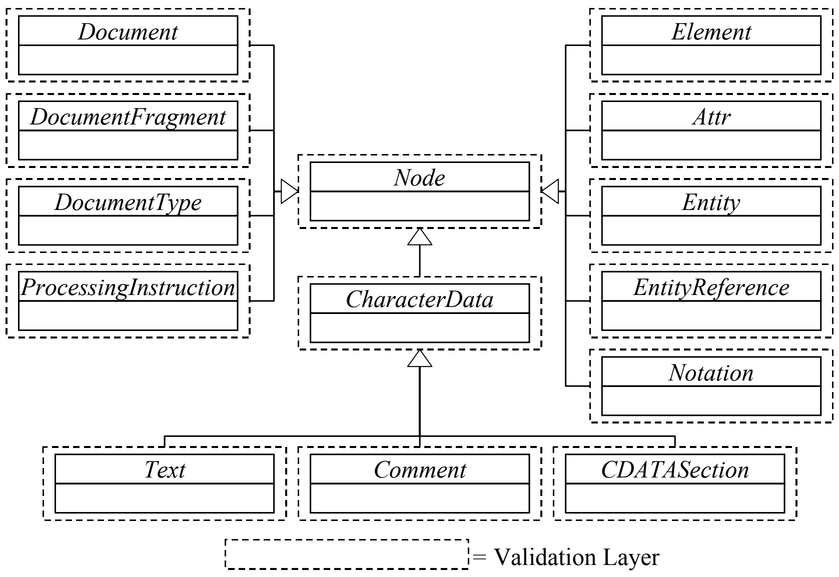


Figure 6.4. FEdXML adds validation to the DOM data types.

XML content can be edited via data type interfaces (e.g., *Node, Element, Attr, Text* etc.) of DOM. FEdXML adds a validation mechanism to all the basic data types of the DOM (Figure 6.4). The validating versions for those data types can be obtained by using the methods of the *ValidatingDOM* interface. The editing methods of the validating data types can be implemented as DOM operations so that the editing is validated against the defined validators. If a validation error is raised, the editing is cancelled and a validation exception is thrown.

6.3.1.3 Validations

In FEdXML, the XML content is validated by using validator components. It is possible to use several (e.g., DTD or XML Schema based and language specific) validator components at the same time.

A single validator component can be straightforwardly implemented, because only the *validate(Node)* method has to be implemented. A validator validates a single node in XML structure and, if needed, a validation exception can be thrown. In FEdXML, validation results are classified into four categories

- *fatal errors* (e.g., element is in wrong place or wrong attribute in the element),
- *errors* (e.g., wrong attribute value or required attributes or child elements are not defined for the element),
- *warnings* (e.g., there can be a implementation that does not support all the features of the edited language, e.g., attribute values of elements), and
- *accepted* (no validation errors in the XML-content).

Validation exceptions can be implemented by using *FatalValidationError*, *ValidationError* and *ValidationWarning* interfaces. If during editing a validation error is thrown, the editing operation is not executed and the error can be shown to the user. If a warning is thrown, the operation is executed and warning can be shown for the user (the *ValidationWarningListener* component is called). If validation errors are not thrown, the editing operation is executed.

Validators can be used for directing the editing process. For example, if a required child element is not defined, a validator can throw an exception. The exception can be caught, and, then, a UI component for defining the child element can be suggested. After the child element is defined, the same editing operation can be tried again and if exceptions are not thrown, the editing is updated to the XML document. A validator can be used for tutoring the user (e.g., of the valid attribute values). A validator can also warn the user of the features, which are not supported by the current (e.g., browser) implementation.

6.3.1.4 Change-propagation mechanisms

The *EditableXMLDocument* interface offers access to XML document services and selection and mutation listeners. *EditableXMLDocument* can offer various kinds of services (e.g., an access to underlying grammar of XML document). The *SelectionListeners* interface includes methods for notifying the defined selection listeners of *Node* and *Range* selections in the edited XML document. The *MutationListeners* interface includes methods for notifying the defined mutation listeners of mutation events in the edited XML document. The *MutationListener* interface is based on the standard *MutationEvent* interface defined by W3C.

6.3.2 XML view

The XML view is used to visualise the visible nodes of DOM. There are methods for

- implementing XML views (*XMLView* and *XMLVisualElement* interfaces), and

- notifying XML views of the selections and mutations in XML model.

The *XMLView* interface gives an abstraction for XML view. The *XMLVisualElement* interface includes methods for getting the node and the bounds of the visual element of the XML view. *XMLView* offers methods for getting the visual XML element that is located in the given coordinates and for getting the visual XML element for the node of the edited DOM. These methods can be used in XML editing component (e.g., a popup menu or attribute input field) for obtaining (e.g., the bounds of) the visual XML element for the node to edit. XML views can be configured to be selection and mutation listeners of the XML document. When nodes in XML document are selected or edited, XML views are notified and updated.

6.3.3 XML controller

The *XMLController* interface enables the use of various kinds of the input mechanisms in XML editing (Figure 6.5). XML views can include listeners for input events, e.g., for mouse and keyboard. When the event listener is notified of an input event the event can be passed for the XML controller, which can use the input events for directing various kinds of XML editing (e.g., popup menus, text and attribute dialogs) components. In other words, *XMLController* separates XML editing components from XML view so that various kinds of XML editing components can be used in various kinds of XML views.

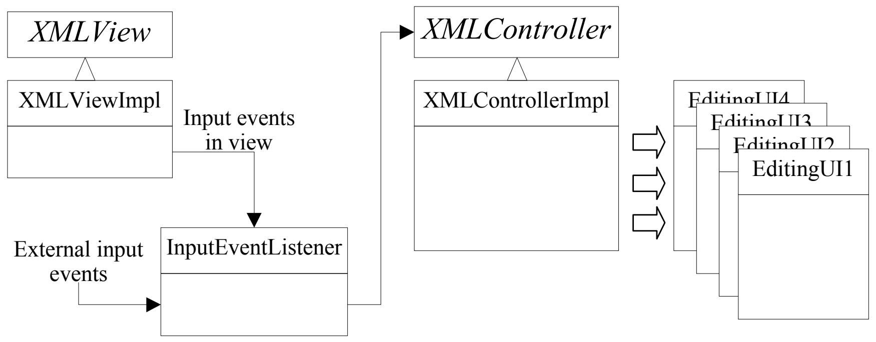


Figure 6.5. XML controller implementation.

6.3.4 Constructing configurable XML editors

There are methods for configuring

- the model (cf. the *Validation* interface),
- the visual appearance (*XMLNodeFormatter* and *XMLFormattingObject* interfaces) of XML editor, and
- the editing operations (the *DOMOperation* interface) of XML editor.

The XML model can be configured to fit for a specific XML language by using the validators. Validators enable implementations of external language-specific features for validation like dynamic checks (e.g., for checking some attribute values in SMIL demonstration) and methods for passing tutoring and warning messages to the user of XML editor.

The user interface of the XML editor can be configured by formatting the visual appearance of the XML views and controls. *XMLNodeFormatter* enables to

- hide,
- arrange, and
- attach formatting objects to the nodes of XML document.

The possibility to format objects enables the attachment of style declarations (e.g., defined in style sheet) to the nodes of XML document and to define the layout (e.g., fonts, colours, icons and visible parts) for the XML content. *XMLNodeFormatterManager* manages the various kinds of XML node formatter implementations. It offers methods for selecting the active node formatter, for adding and removing node formatters, for getting the active and default node formatter name and for getting the enumeration of names of available node formatters (Figure 6.6). When the active XML node formatter is changed, UI components related to it are updated. The UI components can use the *XMLNodeFormatterUser* interface for sharing XML node formatters. For example, an XML editing popup menu can get the active XML node formatter of the view and can use it for hiding and emphasising the content in the menu.

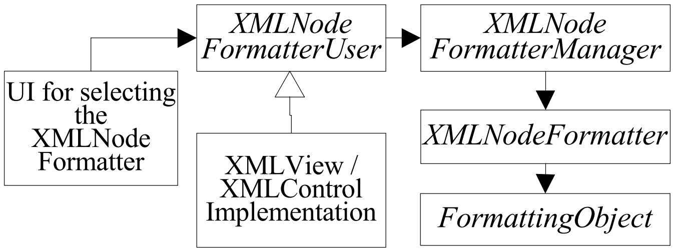


Figure 6.6. XML node formatter can be used with XML views or controls.

XML editing can be further enhanced by constructing macros, wizards and XSLT based editing operations. The *DOMOperation* interface enables to implement different kinds of editing operations for XML editing. XML editing operations are executed in the *DOMOperationExecution* component (as described in Section 6.4.1).

6.4 Reference implementations

We have made reference implementations of FEdXML interfaces, which can be used to

- query what editing operations are allowed for the document according to the grammar,
- execute editing operations,

- validate the XML content against the grammar,
- update changes in XML model to the other parts of XML editor, and
- undo and redo editing operations.

Reference implementations for DOM level 3 validation interfaces use a *Grammar* interface for obtaining validation information, e.g., from DTD or XML Schema (Figure 6.7). We have implemented the *Grammar* interface by using the Xerces parser [Apa]. The *Grammar* interface contains methods for obtaining grammatical rules of the current XML document. It offers methods for obtaining the defined element types and the content types of the elements, methods for obtaining attributes and their default or fixed values, and a method for validating the children of the current node. The Xerces parser includes methods for getting grammatical rules from DTD or XML Schema and, so, it can be used for implementing the *Grammar* interface. When the *XercesImpl* build an XML document it registers the *Grammar* implementation to the XML document services. The grammar service can be used in DOM level 3 validation implementations. In our tests grammatical rules were obtained only from DTDs, because currently supported languages (WML, XHTML, SVG and VoiceXML) are defined by using the DTD notations.

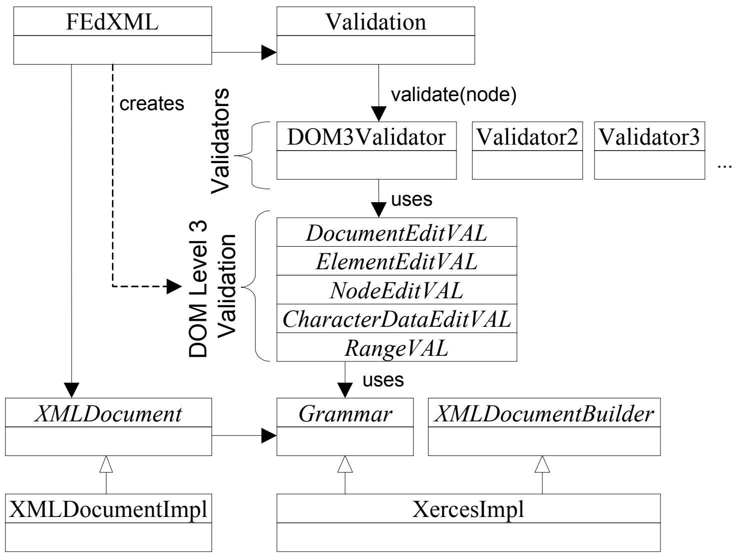
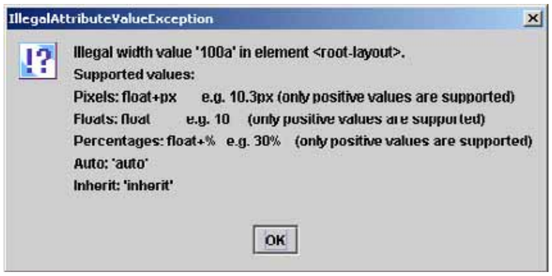


Figure 6.7. Use of FEdXML with the Xerces parser.

FEdXML offers a *DOM3Validator* implementation, which validates the XML content by using the methods offered by DOM level 3 validation. *DOM3Validator* can be used for directing the editing process. E.g., if a required child element is not defined, it throws a define child exception. This exception can be caught and a UI component for defining child element can be shown. Now the same editing operation can be tried to execute again and if exceptions are not thrown the editing is updated to XML document.

For getting language specific validation mechanism, new validators can be defined. We have made validator implementations for WML and SMIL. They validate, e.g., element structure, and attribute values of the edited content. Language specific validators can tutor the user (Figure 6.8). For example, if the user has defined an attribute value, which does not follow the language specification, a validation exception is thrown where there is an additional information message for the user (e.g., "For 'width' only percentages and pixel values are accepted (e.g., 10% or 150px)"). There are also warning mechanisms like "The element 'excl' is not supported by the current implementation, use 'par' or 'seq' elements instead".



```
IllegalAttributeValueException

Illegal width value '100a' in element <root-layout>.
Supported values:
Pixels: float+px      e.g. 10.3px (only positive values are supported)
Floats: float         e.g. 10    (only positive values are supported)
Percentages: float+%  e.g. 30%   (only positive values are supported)
Auto: 'auto'
Inherit: 'inherit'
```

Figure 6.8. Validator based tutoring message.

FEdXML offers methods for configuring the visual appearance of XML document. *XMLNodeFormatter* can be used for hiding, arranging and emphasising visible elements in the XML editor UI. FEdXML offers default implementation for *XMLNodeFormatter*, by which it is possible to define

- named visible nodes,
- attribute groups, and
- formattings for named nodes.

We have implemented *XMLNodeFormatter* for SMIL documents. A generic configurable XML view is shown in Figure 6.9, where the *XMLNodeFormatter* is used for grouping (e.g., layout and timing) attributes, for emphasizing different kinds of attribute groups (are shown with certain color and font) and for hiding the elements and attributes (e.g., so that only timing information is shown).

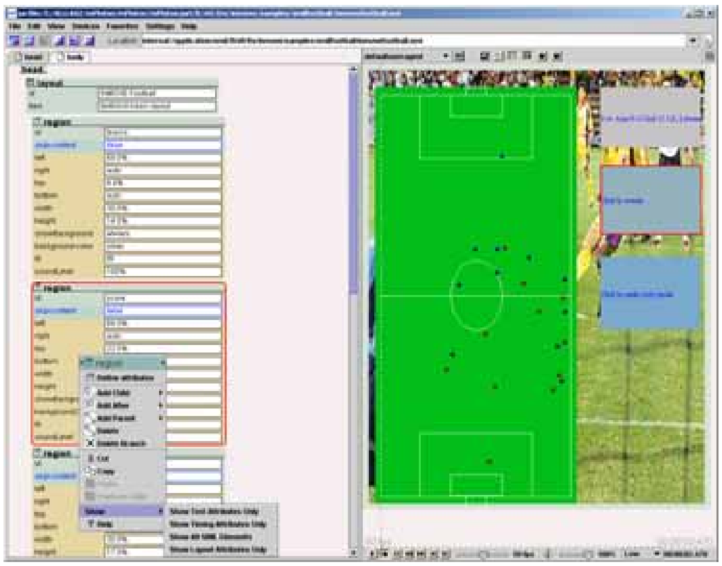


Figure 6.9. XML node formatters for SMIL: A generic configurable XML view on the left and a browser for SMIL presentations on the right.

XML content is edited by using XML editing controllers. They can be constructed by using the validation methods offered by the DOM level 3 validation interfaces. By those, queries can be made to the underlying grammar to check what is possible to do at certain location of XML document. In fact, FEdXML offers validating implementations for all the core interfaces of DOM. Editing controls can use editing commands of those interfaces, and, so, the edited content is validated against defined validators. *DOMOperations* enables the implementation of macros and wizards for XML editing. The operations can be implement undoable. The reference implementation of FEdXML offers a default implementation for undo and redo operations (*UndoManager*).

FEdXML is tested in mobile and desktop environment. In desktop environment, we have constructed a development tool for mobile applications, named "mPlaton Toolkit", which is a tool for editing, creating and browsing different kinds of XML languages in a mobile environment (currently WML, XHTML, SVG and VoiceXML). It can emulate various mobile devices by utilising User Agent Profiles. A more detailed description of the mPlaton Toolkit is provided in Chapter 7.

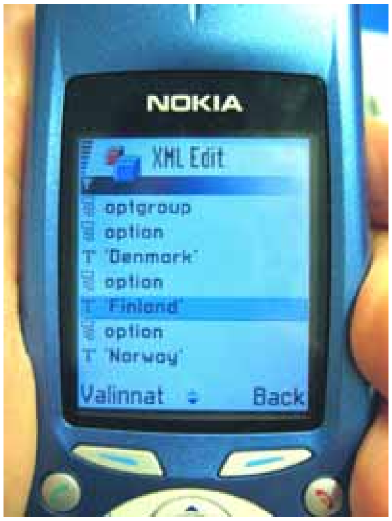


Figure 6.10. XML editor in Java MIDP enabled (Nokia 3650) phone.

We have tested FEdXML in mobile environment by implementing a WML editor for Java MIDP enabled phones (Figure 6.10). The implementation consists of browser and WML editor parts. The WML editor includes methods for navigating in the document structure and for editing the WML content. The visible elements of XML document are visualised in the editor as a list and the XML text and attribute values can be edited by selecting elements in the list. The editor does not currently support editing of the document structure (e.g., elements can not be added), but we plan to enhance the editor with new editing features. The WML editor can be used for template based editing, where the user gets a WML document template and can edit some parts of it (e.g., add own images to the presentation or edit some attributes and texts in the template). The editor uses a *WMLValidator* component (previously implemented for desktop environment [PLK00]) for validating the edited WML content. When editing is done the updated document is presented in WML browser.

7. Applications

mPlaton frameworks have been tested by making separate application implementations for Java 2 SE, Personal Java and Java MIDP environments.

Examples of the applications include

- mPlaton Toolkit - a comprehensive development tool for mobile browsing applications,
- µBrowser - a micro browser implementation for PDA devices, and
- MIDPBrowser - a very light browser implementation for Java MIDP enabled mobile phones.

Descriptions of mPlaton Toolkit, µBrowser and MIDBrowser implementations are in Sections 7.1-7.3.

7.1 mPlaton Toolkit – a development toolkit for mobile browsing applications

mPlaton Toolkit is a comprehensive development toolkit for mobile browsing applications. It is implemented with Java 2 SE, and is used in desktop environments where there are more memory resources and processing power available.

With the toolkit (see Figure 7.1), it is possible to construct different kinds of XML based (e.g., XHTML, WML and SMIL) presentations and to simulate the presentations by using profile information (e.g., User Agent Profiles, style sheets and user preferences). It utilizes all the frameworks of the mPlaton platform.

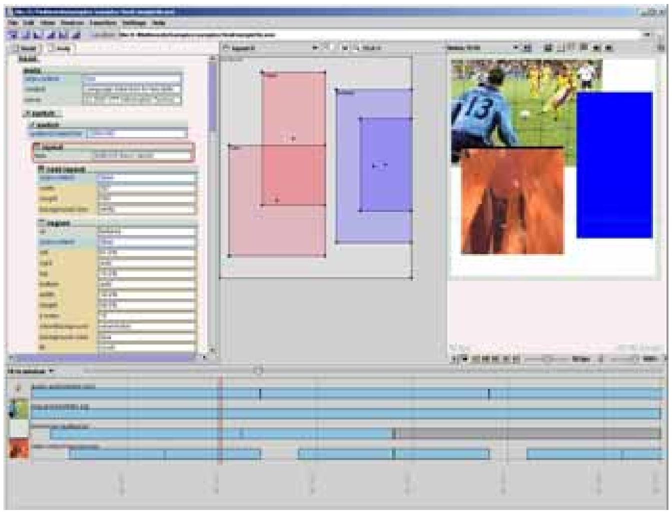
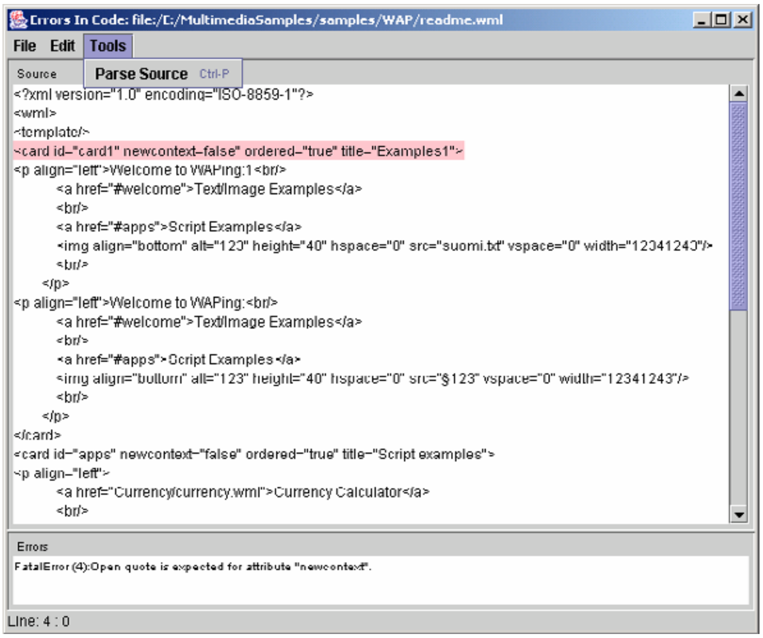


Figure 7.1. The main window of mPlaton Toolkit.

When the XML content is loaded to the editor the XML parser checks the validity of the document. If XML document is not valid the XML source view is shown (Figure 7.2).



Errors In Code: file:/E:/MultimediaSamples/samples/WAP/readme.wml
File Edit Tools
Source
Parse Source Ctrl-P

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<wml>
<template/>
<card id="card1" newcontext=false" ordered="true" title="Examples1">
<p align="left">Welcome to WAPing:1<br/>
        <a href="#welcome">Text/Image Examples</a>
        <br/>
        <a href="#apps">Script Examples</a>
        <img align="bottom" alt="123" height="40" hspace="0" src="suomi.txt" vspace="0" width="12341243"/>
        <br/>
     </p>
<p align="left">Welcome to WAPing:<br/>
        <a href="#welcome">Text/Image Examples</a>
        <br/>
        <a href="#apps">Script Examples</a>
        <img align="bottom" alt="123" height="40" hspace="0" src="$123" vspace="0" width="12341243"/>
        <br/>
     </p>
</card>
<card id="apps" newcontext="false" ordered="true" title="Script examples">
<p align="left">
        <a href="Currency/currency.wml">Currency Calculator</a>
        <br/>
```

Errors

```
FatalError (4):Open quote is expected for attribute "newcontext".
```

Line: 4 : 0

Figure 7.2. XML code view. The XML source is shown on the top and a list of the errors is shown on the bottom.

The source view shows the document source and the error lines. When clicking an item in the error list, the related source code line is emphasized with color. After correcting errors, the XML source can be parsed again. If there are no errors, the parsed XML document can be loaded to visual XML editor. The toolkit offers tree, visual and text view for XML editing (Figure 7.3). Views are generic, which means that they can present any kind of XML languages.

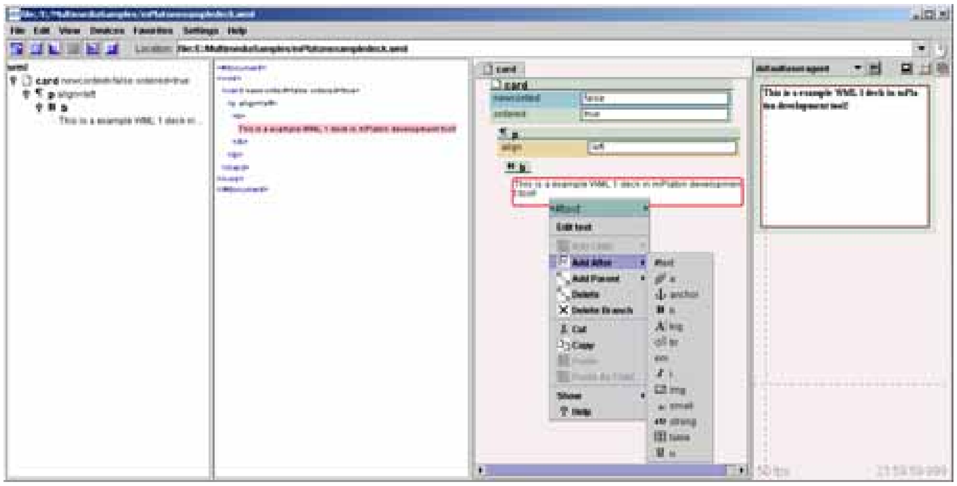


Figure 7.3. Generic views for XML editing. A tree view on the left, an XML text view and an XML visual view on the center and a simulator for visualizing the created XML content on the right.

All the time when the XML content is edited, the created content is shown in the simulator view. In many cases, there is a lot of information in an XML document that makes it hard to find out the essential parts. The usability can be improved by using profiling mechanisms. The generic XML views offered by mPlaton Toolkit can be profiled to work with some specific XML based language. We have implemented profilers for WML and SMIL documents. For example, the profilers enable to hide the content of XML document (e.g., some attributes and elements can be hide), arrange the content (e.g., attributes can be arranged to certain order) and to emphasize the visual elements (e.g., to attach icons and to define colors for XML elements and attributes) in the XML view.

The toolkit offers editor views for WML and SMIL documents (Figure 7.4). In WML view there are graphical elements for WML elements. The layout and timing views are for SMIL editing. In the layout view, it is possible to edit the layout of SMIL presentation, and, in the timing view, it is possible to visualise the timing of the edited SMIL presentation.

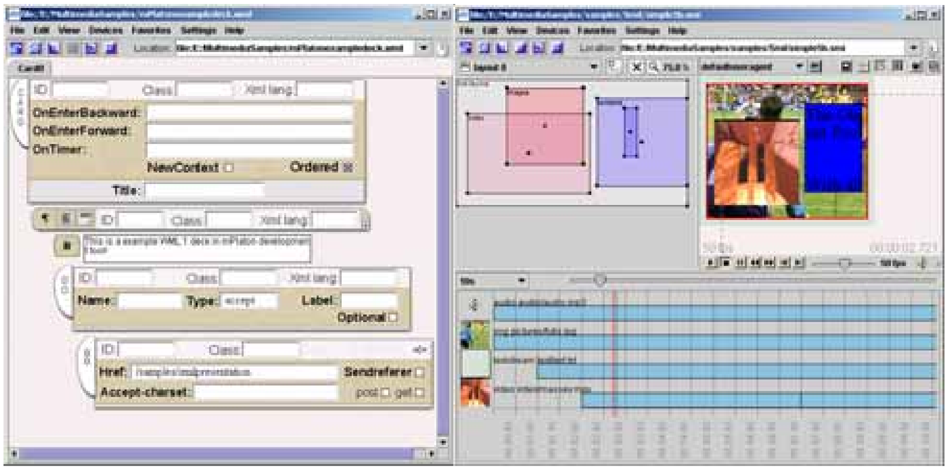
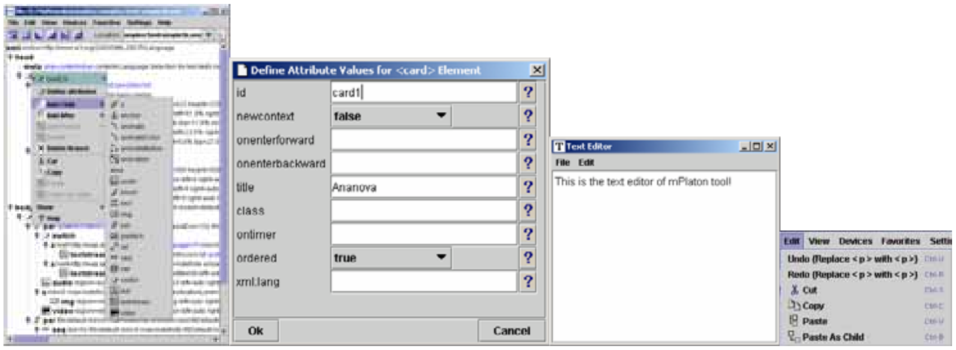


Figure 7.4. mPlaton Toolkit offers specific views for WML and SMIL documents. A visual view for WML documents on the left and layout and timing views for SMIL documents on the right.

With an XML popup menu it is possible to add and remove elements and define attributes and textual values (cf. Figure 7.5). For defining the attribute values, there is an attribute dialog. For textual values, there is a text editor, by which it is possible to edit and load textual content to the XML document. XML editing operations can be cancelled by using the undo and redo controls. The controls of mPlaton Toolkit are generic (can be used for editing any kinds of XML based languages), but they support profiling too. In mPlaton Toolkit, controls can be profiled, e.g., menus can be emphasised (e.g., icons can be attached to the selections) and selections can be reduced (e.g., some selections in XML control menu can be hided).

The XML content is validated by using the information in DTD. In addition, mPlaton Toolkit supports the use of validator components to construct language specific validation mechanisms. For example, for SMIL presentation there is a validation component that gives tutor messages for the user: if attribute values defined by user are not valid the validator throws an exception with the information about the acceptable attribute values. This exception is shown in a tutor message dialog.

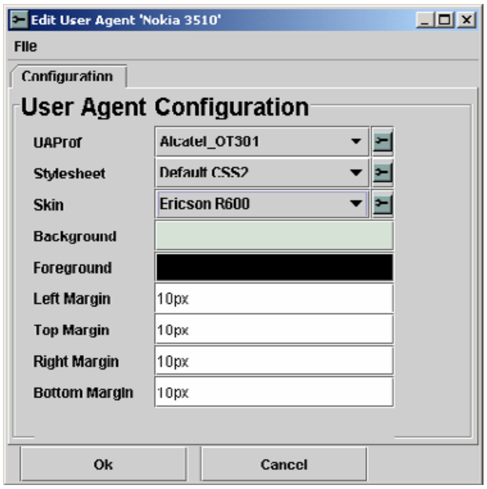

Define Attribute Values for <card> Element
Ok
Cancel
Text Editor
File Edit
Edit View Devices Favorites
Undo (Replace <p> with <p>)
Redo (Replace <p> with <p>)
Cut
Copy
Paste
Paste As Child

id	card1
newcontext	false
onenterforward	
onenterbackward	
title	Ananova
class	
ontimer	
ordered	true
xml:lang	

This is the text editor of mPlaton tool!

Figure 7.5. Controls for XML editing. An XML editing popup on the left, an attribute edit dialog and a text editor on the centre and an XML edit menu on the right.

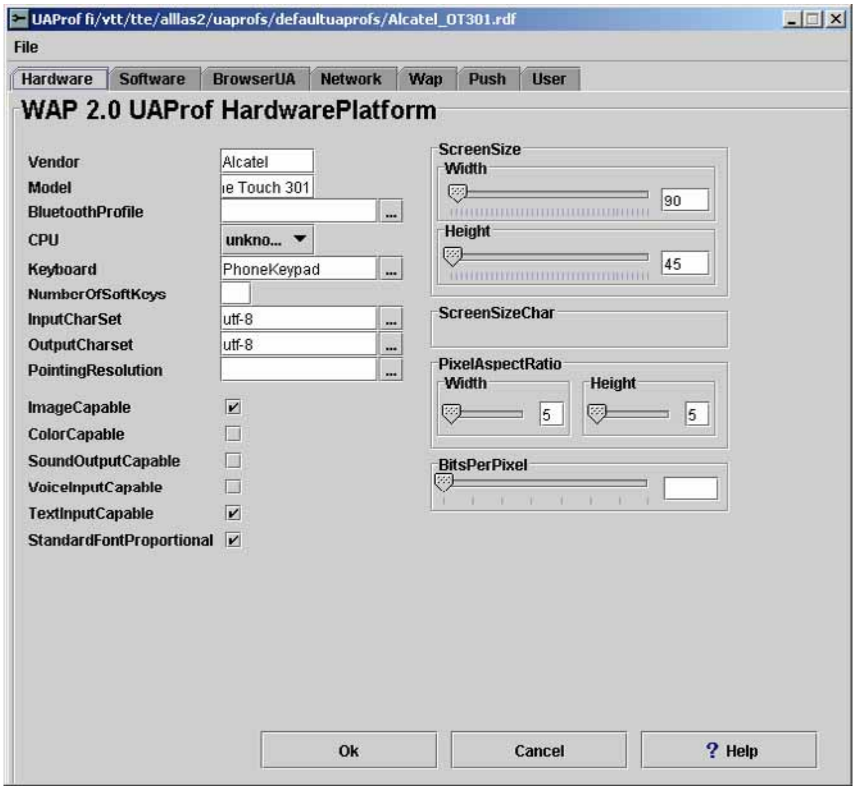
The mPlaton Toolkit supports the presenting of browsing content (e.g., text, images, audio, video and container types like XHTML, WML, SMIL) in simulators. The simulators can utilize profile information like UAProfs, style sheets (CSS2) and user preferences. mPlaton Toolkit includes a special purpose UI for defining user agent configurations (Figure 7.6). User agent configuration is saved in XML format and, thus, the user agent configuration can be easily extended with new definitions. It is possible to define UAProfs, style sheets and skins for mobile browsing application simulation. A user agent configuration defines also information related to the simulator layout like the background and foreground colors of the simulator.



UAProf	Alcatel_OT301
Stylesheet	Default CSS2
Skin	Ericson R600
Background	⬜
Foreground	⬛
Left Margin	10px
Top Margin	10px
Right Margin	10px
Bottom Margin	10px

Figure 7.6. Defining the user agent configuration.

UAProfs can be defined by using the UAProf editor (Figure 7.7). The UAProf editor offers tools for editing the values of UAProf descriptions. After the edition, the modified UAProf can be exported to textual UAProf schema descriptions. Style sheet descriptions can be defined by using a text editor in mPlaton Toolkit.


UAProf fi/vtt/tte/alllas2/uaprofs/defaultuaprofs/Alcatel_OT301.rdf

File

Hardware | Software | BrowserUA | Network | Wap | Push | User

WAP 2.0 UAProf HardwarePlatform

Vendor	Alcatel
Model	ie Touch 301
BluetoothProfile	
CPU	unkno...
Keyboard	PhoneKeypad
NumberOfSoftKeys	
InputCharSet	utf-8
OutputCharset	utf-8
PointingResolution	
ImageCapable	✓
ColorCapable	☐
SoundOutputCapable	☐
VoiceInputCapable	☐
TextInputCapable	✓
StandardFontProportional	✓

ScreenSize: Width 90, Height 45

ScreenSizeChar

PixelAspectRatio: Width 5, Height 5

BitsPerPixel

Figure 7.7. UAProf editor.

mPlaton Toolkit supports the use of images of mobile phones as skins in mobile browsing application simulation (Figure 7.8). With the skin editor it is possible to define layout for the skin image of mobile device. Also, it is possible to define bounds for buttons and for the display. After the skin is defined it can be used in simulating the mobile browsing application. Skin simulator supports controlling the mobile browsing applications with the keyboard in the skin. It is also possible to set multiple values for keys. Then, the alternative values can be obtained by pressing the button multiple times. Skin can be presented in different sizes by using the zoom feature of the Skin editor and simulator.

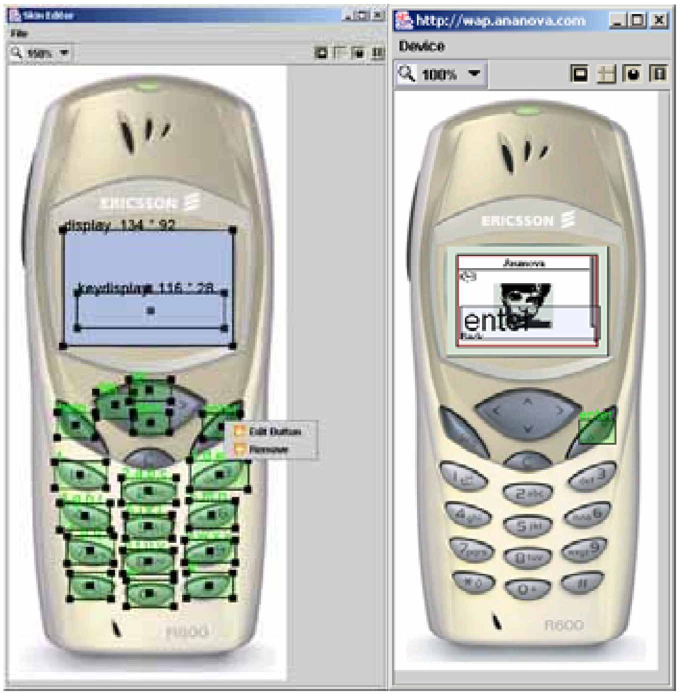


Figure 7.8. Skin editor and simulator.

Whenever defined user agent configurations exist, the edited content can be presented in simulator by using the information in the user agent configuration. Figure 7.9 shows a segment of XHTML content simulated in three different kinds of simulator configurations.

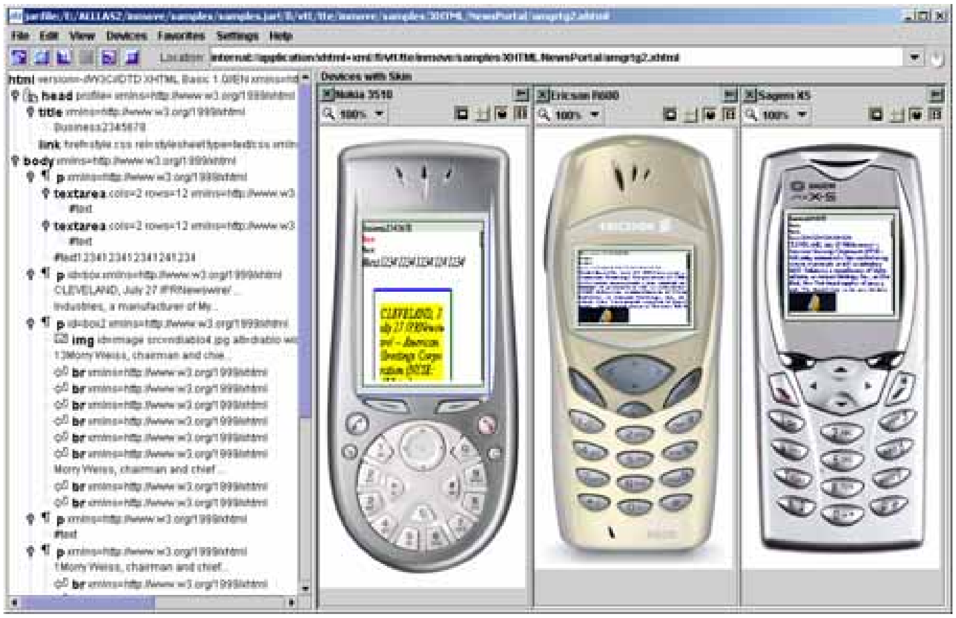


Figure 7.9. XHTML document simulated in various kinds of simulator configurations. UAProfs and style sheets are used in presenting the content.

7.2 µBrowser - a micro browser implementation for PDA devices

For making usable applications for PDA devices, it is essential that used algorithms are efficient and designed to work in the device platform. All kind of processing overheads should be minimised and, e.g., the construction of new objects should be reduced for minimising garbage collection overhead in the program.

µBrowser is implemented using Personal Java (works with JDK 1.1.8; see Figure 7.10). Personal Java supports only a reduced set of Java 2 SE classes and, e.g., it does not directly support Java Swing components. µBrowser supports browsing of various kinds of MIME types (e.g., text, images (jpeg, png, SVG), audio, video, WML, SMIL and XHTML MP) and it has been tested in the Compaq IPAQ PDA device. It includes navigating, browsing history operations (e.g., back, forward, reload) and caching and asynchronous browsing in the fetching of the mobile content.

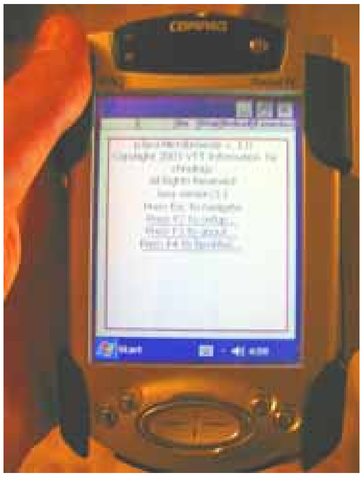


Figure 7.10. µBrowser - a micro browser implementation for PDA devices.

7.3 MIDBrowser - a browser implementation for Java MIDP enabled mobile phones

MIDBrowser is a browser implementation for Java MIDP environment (Figure 7.11). Java MIDP environment differs greatly from Java SE 2 and Personal Java environments. In particular, Java MIDP is designed to work in mobile devices, which have very reduced memory and processing power resources available. Especially, the memory saving is important for Java MIDP enabled mobile phones. The Kilobyte Virtual Machine (KVM) has normally only few hundred kilobytes of memory available and, thus, the memory limitations have to be taken into account in the implementations [NOK02]. Java MIDP does not support floating-point data types, which also has a significant effect on mobile application design. The finalisation is not supported for objects and error handling is limited. Further, Java MIDP does not support Java Native Interface (JNI), user-defined class loaders and Remote Method Invocations (RMI). When writing this, in our knowledge, there are no full DOM parsers for Java MIDP environment. Available XML parser implementations are unconditional implementations and, so, the direct use of them may reduce the reusability of created components. For resolving this problem, we created a reduced implementation for DOM data type interfaces so that the DOM can be constructed by using existing XML parser of Java MIDP environment (e.g., kDOM or Al Sutton parser). Consequently, the XML document can be handled by using the DOM interfaces and the plugin implementations that are not bound to any specific parser implementation.

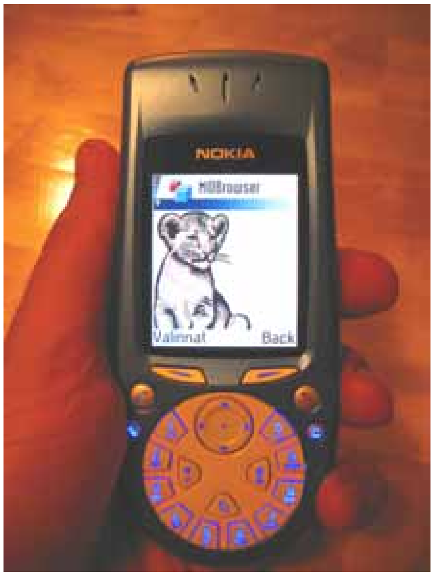


Figure 7.11. Java MIDP browser.

Currently, the MIDPBrowser implementation includes only plugins for WML and SVG, but we aim to implement several new plugins for the Java MIDP environment in the short run.

8. Discussion

This chapter provides evaluations of the mPlaton frameworks based on usage experiences and expert evaluations. The aspects related to AGB, MIMEFrame and FEdXML are in the following Sections 8.1 – 8.3, respectively.

We have constructed reference implementations for the mPlaton frameworks with Java (cf. Sections 4.2, 5.3 and 6.4). We have tested the frameworks in Java 2 SE, Java Personal Profile and Java MIDP environments, but they can also be implemented with other object-oriented languages (like C++). In our tests the frameworks are proved to be suitable for mobile use.

8.1 AGB

AGB contains only the core features of mobile browser. It offers a solution for mobile browsing, but is also suitable for any browsing applications where navigation, caching or push and pull operations are needed. The reference implementation of AGB is very light, and it can be embedded to applications, where there are needs for browsing services.

In contrast to available browsers, the aim of the AGB is not only to offer browser services for the human users, but AGB can be seen as a middleware software offering browsing services for various kinds of mobile applications. With AGB it is possible to implement conventional browsers, but it can also used as a base for dedicated and embedded mobile browser implementations.

AGB shows how various kinds of pull and push connections can be combined to work together in mobile environment. AGB has more generic approach to handle connections than conventional browsers have. In AGB, the connection can be any kind of pull or push typed connection, which makes it possible to construct new objects. A connection is not necessarily only of network connection type, but it can be a resource in mobile device (e.g., location coordinates), which makes it possible to construct asynchronously objects to browse (e.g., a map of the locations). Connections can be persistent, temporal (e.g., HTTP connection with cache-control mechanism) or disposable (e.g., reader typed) connection. Temporal and disposable connections can also be used for saving the memory and the battery of mobile device.

AGB push connections are used to notify of services offered outside. For example, WAP push, SMS, MMS and instant messaging type of services can be introduced as push connections. Further, push connections can be mapped to pull operations, because push messages can offer the location or source data for the content to browse. This information can also be used for fetching and constructing the browsing content in asynchronous pull operation. The used approach separates the receiving of the push messages from fetching the content.

AGB optimises the use of network resources. Both opening the connections and fetching the content can be done asynchronously and, so, there can be several concurrent object pull operations at time. Caches for connections and fetched content minimise the requests sent to mobile networks. It is also possible to create connections with attached profile information that can be utilised in various kinds of adaptation (content, network and terminal adaptation etc).

We have tested AGB in Java 2 SE, Java Personal Profile and Java MIDP environments. The reference implementation offers a well-tested and modular base for mobile pull/push capable mobile browser implementations. We started the development work of AGB in desktop environment, but later we implemented AGB in PDA (Java Personal Profile enabled) devices and, finally, in Java MIDP enabled mobile phones.

In all, AGB saves a lot of implementation effort, since instead of implementing the mobile browser once again, the AGB is used as a base of browser implementations in separate environments. The reference implementation of AGB suits well for the mobile environment, because it

- is very light (size of the compiled Java classes is about 61 Kbytes),
- is scalable and highly configurable (all parts related to AGB are added as components and can be replaced by changing the configuration),
- supports reusability (implemented browser modules can be used in separate browser configurations),

- supports different kinds of communication protocols and connections (can be added as components), and

- optimizes the use of network capabilities (offers methods for asynchronous browsing and for caching).

In practise, implementing a MIDBrowser for Java MIDP enabled phone took only a few days to implement. Although there are great differences between Java MIDP and Java 2 SE environments, the most parts of AGB reference implementation were used directly in Java MIDP environment. The lack of the constructors introduced in Java 2 SE caused some extra work. We implemented few new interfaces to obtain constructors for plugins as well as a new implementation of the object factory mechanism for Java MIDP environment. Most of the effort needed for MIDBrowser implementation caused of the differences between UI components of Java SE and Java MIDP environments.

We converted the WML plugin (previously implemented in Java 2 SE environment) to Java MIDP compliant and the HTTP connection component were rewritten. In all, the AGB saved a huge amount of implementation and testing work and improved the quality of the results, because the most parts of the browser were implemented by using well-tested modules of the reference implementations.

A drawback of AGB is the fact that the extensive use of abstract interfaces spent memory resources (the size of the compiled interfaces of the AGB reference implementation is about 8 KBytes), which can be a problem in mobile devices with very limited memory (e.g., there are Java MIDP enabled phones with only 30 KBytes program memory available). On the other hand, AGB saves a lot of effort and improves the quality of browser products by offering the architecture and reference implementations for the core modules of mobile browser. By using AGB, the developer can put more efforts to implementing more advanced browser and application modules instead of implementing the whole browser from scratch once again.

8.2 MIMEFrame

The reference implementation of MIMEFrame can be embedded to other applications, whenever there are needs for client-side implementations of mobile browsing applications.

We have tested MIMEFrame in different kinds of environments, e.g., in a desktop computer, in a PDA device and in a Java MIDP enabled mobile phone. We have implemented various kinds of content types, which can work together (e.g., text, image, audio, video and SMIL, SVG). Further, we have implemented content types (e.g., XHTML and WML) that applied browsing services and various kinds of resources (e.g., user agent profiles, style sheets) in presenting the content. We have tested multimodal controls by using WML and VoiceXML together where keyboard, mouse and voice controls are available. Generally, the reference implementation of MIMEFrame is scalable and light (the size of the compiled Java classes of the reference implementation is about 59 Kbytes).

MIMEFrame introduces a generic architecture, which describes a skeleton for the client-side of mobile browsing application implementations. Instead of designing the whole client-side architecture for the mobile browsing applications once again the efforts can be targeted on designing modules for mobile browsing applications.

The possible drawback of the modular approach is the computational overhead. The defined interfaces of the framework consume memory (the size of the interfaces of the reference implementation is about 13 Kbytes), which can be a problem in devices with very limited memory resources. On the other hand, in comparison to the monolithic structure, the modular approach enables to construct mobile browsing applications of small replaceable and well-tested modules, which saves a lot of effort and improves the quality of the resulting mobile browsing applications.

8.3 FEdXML

FEdXML introduces a generic architecture for modular XML editors, which defines the core parts of XML editor, and shows how to apply these parts. From the software developer points of view, FEdXML enables the developer to concentrate on developing of new components of XML editor.

The used modular approach makes it easy to replace specific parts of XML editor implementation with new ones. For example, XML editor views and controls should be implemented so that they can be used with small displays and in devices with very limited input methods available (e.g., no pointing device available). In comparison to the monolithic structure, the drawback of the modular approach is the computational overhead. On the other hand, the modular approach enables to construct XML editors of small replaceable and well-tested modules, which saves a lot of effort and improves the quality of the results. An experienced developer can put more efforts for optimising the editor modules suitable for the current usage environment. For new developers, the modular structure is easier to understand. They can become acquainted with the XML editor implementation by elaborating the software components provided by the reference implementations.

In our tests, FEdXML has saved a lot of time and effort. In desktop environment, we have created XML editors for WML, SMIL, VoiceXML and XHTML by using the reference implementation of FEdXML as a base. We used the FEdXML to implement a WML editor for Java MIDP enabled phones. FEdXML saves the coding effort also because existing editor components can be flexible reused (e.g., the existing generic view, control and DTD based validation components). All the core parts of FEdXML are replaceable. For instance, if FEdXML is to be transferred to a new (e.g., mobile device) environment, any part related to the FEdXML can be replaced. Also, the designing of completely new XML editors is easier, because FEdXML offers the base structure of XML editors. Instead of designing the whole editor architecture for the target XML based language the efforts can be targeted on designing language specific (e.g., XML model, views and controllers) components. In general, the use of the existing well-tested editor components will improve the quality of new editor implementations.

9. Conclusions and future work

9.1 Results

The mPlaton browsing and development platform offers frameworks for constructing mobile browsers (AGB), for presenting various kinds of content (MIMEFrame) and for constructing the XML editors (FEdXML). In our tests mPlaton is proved to be a usable base for new implementations (cf. Chapter 8). We have implemented reference implementations for mPlaton frameworks with Java, which can be utilized in new mobile browser and XML editor implementations.

AGB constitutes the core of a modular, reusable browser environment. In particular, AGB offers a solution for mobile browsing, but it is suitable for any browsing applications where navigation, caching or push and pull operations are needed. The key feature of AGB is that it is highly configurable. Furthermore, AGB is very light and scalable. Because the core modules of AGB can be replaced with new implementations, the reference implementation of AGB can be readily improved, e.g., with new optimised cache, history or object pull or connection initialisation modules. If needed, modules of AGB can be changed and replaced even in run-time, e.g., the connections and object plugins can be changed dynamically. All the other parts can be flexibly introduced as modules. Reusability of the implemented plug-in components is good because existing components can be used as parts of different browser configurations and implementations. If environments differ greatly (such as Java MIDP and Java 2 SE), separate versions of the plug-ins may have to be implemented.

The MIMEFrame framework offers generic groundwork for client-side implementations of mobile browsing applications so that implemented parts are consistent, generic, and reusable. The core of the framework offers interfaces for constructing user agents, content types, and control mechanisms. Content type plugins can offer services, which the other parts of mobile client (e.g., content types or user agent) can use. MIMEFrame supports use of different kinds of modal and non-modal controls in mobile browsing applications. Profile information (e.g., UAProfs, style sheets and user preferences) can be used in adjusting the presentation of content objects.

FEdXML introduces a generic architecture for modular XML editors, where the XML editor is divided by using the Model-View-Controller structure. FEdXML is a light, scalable and highly transferable. The core of FEdXML contains only interfaces and it makes it possible to construct XML editors for various kinds of (mobile and desktop) environments. FEdXML relies strongly on standard XML interfaces introduced by W3C.

The frameworks of mPlaton make it possible to construct dedicated browsers and embeddable application components for mobile devices that could provide features not offered by available commercial browsers (e.g., specific UIs and functionality for browsing applications). In particular, the frameworks enable the developers to construct new kind of mobile browsing applications, where in addition to the browsing features there could be external capabilities to present, to edit and to send the browsed data.

In order to bring content to mobile browsers the content producers need development environments. By combining frameworks of mPlaton it is possible to construct development and simulator environments. We have constructed a development toolkit for mobile applications (mPlaton Toolkit) by using the mPlaton frameworks. The toolkit enables developers to test new mobile browsing applications before they are adapted to commercial products. If simulator modules of development environment are suitable for mobile use, they can be installed directly to mobile devices and be used in mobile browsing. The advantage of this is that the simulator and the browser in mobile device can be as similar as possible.

9.2 Future work

Our future research topics include issues related to the browsing and presenting of mobile multimedia content and support for new kinds of services and applications such as context-aware services. Also, the multimodality of applications will bring new research challenges. Further, the construction of XML editors and application tools for the limited mobile devices will offer many interesting research topics in the future.

We will apply the mPlaton frameworks in various kinds of mobile environments. The Java MIDP is an especially interesting platform, because it is becoming so widely supported by various kinds of mobile devices. The mPlaton components can be embedded in other applications. We plan to implement mobile browser applications, which can be integrated into other software products (e.g., in Java MIDP environment). The future research directions related to AGB, MIMEFrame and FEdXML are concluded in the following paragraphs, respectively.

We plan to use AGB as a base of dedicated and embedded browsers. AGB offers capabilities to improve mobile applications with browsing features. AGB makes it possible to test (e.g., instant messaging, location- and context-awareness) features not yet found in commercial browsers. New connection (e.g., instant messaging) and content types (e.g., content related to the location information) will be implemented. AGB will be used as a test bed for new kinds of mobile applications.

Our future work related to MIMEFrame will be to implement application components for new and forthcoming media types, and multimodal controls. The Java MIDP environment offers many interesting possibilities to construct mobile browsing applications. MIMEFrame enables to construct new content types, controls, and user agents, which can present contents. We plan to design location- and context-based content types (e.g., maps), which can be presented in Java MIDP enabled mobile phones.

Our future work related to FEdXML will concern the implementation of new validation, controller and view components. For example, we are planning to implement new XML Schema-based validation components. Future multimedia applications will require sophisticated editor components. For that, a goal is to construct a SMIL editor, which works on Java MIDP enabled mobile phones. New features can be introduced to the mobile SMIL editor by implementing new SMIL editor modules for Java MIPD environment. Also, many modules implemented in desktop environment (e.g., validation) can be used in Java MIDP environment.

Acknowledgements

The work described in this report has been done within several research projects (cf. http://www.vtt.fi/tte/proj/alllas) at VTT Information Technology between March 1999 and September 2003. One of the initial aims of this work has been to support other mobile Internet research projects at VTT by providing a research and development environment for new mobile services and applications. Examples of such projects include "WAPproxy" (see http://ww.vtt.fi/tte/projects/WAP), "Multimedia Services Pilot" (see http://www.vtt.fi/tte/projects/WAMPPI) and "INMOVE" project (see http://inmove.org).

In addition to the authors, several other persons have participated the work documented in this report. Especially, Lauri Mikkola and Juha Sjöholm have implemented several parts, and Ari Ahonen, Juha Kolari and Tiina Kymäläinen have attended in user interface design and evaluations of mPlaton. We wish to thank the evaluators Tarja Systä, Kai Koskimies and Tommi Mikkonen from Tampere University of Technology for many invaluable comments. Also, we wish to thank Ari Ahonen, Eija Kaasinen, Juha Leppänen, Jaakko Lähteenmäki, Vespe Savikko and Timo Tuomisto for the review of the manuscript and useful advice.

This work has benefited from the help and advice of several persons from VTT and persons attended the above mentioned projects. We would like to thank them all.

References

[AB+01] Altheim, M., Boumphrey, F., Dooley, S., McCarron, S., Schnitzenbaumer, S. & Wugofski, T. Modularization of XHTML™. W3C Recommendation, 10 April 2001. http://www.w3.org/TR/xhtml-modularization

[AB+01b] Adler, S., Berglund, A., Caruso, J., Deach, S., Graham, T., Grosso, P., Gutentag, E., Milowski, A., Parnell, S., Richman, J. & Zilles, S. Extensible Stylesheet Language (XSL). W3C Recommendation, 15 October 2001. http://www.w3.org/TR/2001/REC-xsl-20011015/

[Apa] Apache Xerces2 XML Parser for JAVA. http://xml.apache.org/xerces2-j/

[ArD02] *Arreymbi, J. & Dastbaz, M.* Issues in delivering multimedia content to mobile devices. Sixth International Conference on Information Visualisation, 2002. Proceedings, 10-12 July 2002, pp. 622–626.

[Arn99] *Arndt, T.* The evolving role of software engineering in the production of multimedia applications. IEEE International Conference on Multimedia Computing and Systems, 1999. Volume 1, Jul 1999, pp. 79–84.

[BC+03] *Bellavista, P., Corradi, A., Montanari, R., & Stefanelli, C.* Dynamic binding in mobile applications. IEEE Internet Computing, Volume 7, Issue 2, March-April 2003, pp. 34–42.

[BeF01] *Bertino, E. &, Ferrari, E.* XML and data integration. IEEE Internet Computing, Volume 5, Issue 6, Nov/Dec 2001, pp. 75–76.

[BI+00] *Baker, M., Ishikawa, M., Matsui, S., Stark, P., Wugofski, T. & Yamakami, T.* XHTML™ Basic. W3C Recommendation, 19 December 2000. http://www.w3.org/TR/xhtml-basic

[BLE01] *Berners-Lee, T., James Hendler, J. & Lassila, O.* The Semantic Web. Scientific American, May 2001.

[BM+96] *Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P.& Stal, M.* A System of Patterns - Pattern-oriented Software Architecture. Wiley Series in Software Design Patterns (1996).

[BoK01] *Boll, S. & Klas, W.* Z_YX-a multimedia document model for reuse and adaptation of multimedia content. IEEE Transactions on Knowledge and Data Engineering, Volume 13, Issue 3, May-June 2001, pp. 361–382.

[BP+00] *Bray, T., Paoli, J., Sperberg-McQueen, C. M. & Maler E.* Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation, 6 October 2000.http://www.w3c.org/TR/REC-xml/

[BS+03] *van Beek, P., Smith, J.R., Ebrahimi, T., Suzuki, T. & Askelof, J.* Metadata-driven multimedia access. IEEE Signal Processing Magazine, Volume 20, Issue 2, March 2003, pp. 40–52.

[BW+98] *Bos, B., Wium Lie, H., Lilley, C. & Jacobs, I.* Cascading Style Sheets, level 2 (CSS2) Specification. W3C Recommendation 12 May 1998. http://www.w3.org/TR/REC-CSS2/

[CDK01] *Coulouris, G., Dollimore, J. & Kindberg, K.* Distributed Systems - Concepts and design. Addison-Wesley - International Computer Science Series (2001). Pp. 220–231.

[CKR03] *Chang, B., Kesselman, J. & Rahman, R.* Document Object Model (DOM) Level 3 Validation Specification. W3C Candidate Recommendation, 30 July 2003.
http://www.w3.org/TR/2003/CR-DOM-Level-3-Val-20030730/

[CoK99] *Couderc, P. & Kermarrec, A.-M.* Improving level of service for mobile users using context-awareness. Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems, 1999, 19–22 Oct. 1999. Pp. 24–33.

[DS+99] *Dey, A.K., Salber, D., Abowd, G.D. & Futakawa, M.* The Conference Assistant: combining context-awareness with wearable computing. The Third International Symposium on Wearable Computers, 1999. Digest of Papers, 18–19 October 1999. Pp. 21–28.

[DeC02] *Deng, L.Y., Ruei-Xi Chen, Rong-Chi Chang & Teh-Sheng Huang*. Adaptive content model for multimedia presentation. First International Symposium on Cyber Worlds, 2002. Proceedings., 6–8 Nov. 2002. Pp. 209–216.

[DiB01] *Dimitrova, N. & Bove, V.M., Jr.* Connected by media [vision and views]. IEEE Multimedia, Volume 8, Issue 4, Oct.-Dec. 2001, pp. 13–15.

[DM+00] *Decker, S., Melnik, S., van Harmelen, F., Fensel, D., Klein, M., Broekstra, J., Erdmann, M. & Horrocks, I.* The Semantic Web: The roles of XML and RDF. IEEE Internet Computing, Volume 4, Issue 5, Sep/Oct 2000, pp. 63–73.

[Fal01] *Fallside, D. C.* XML Schema Primer. W3C Recommendation, 2 May 2001.
http://www.w3.org/TR/xmlschema-0/

[GHJV94] *Gamma, E., Helm, R., Johnson, R. & Vlissides, J.* Design Patterns - Elements of Reusable Object Oriented Software. Addison-Wesley Professional Computing Series (1994).

[GUP] 3GPP. 3GPP Generic User Profile – Data Description Method. Technical Report TS 23.241 V1.0.0, 2003.

[Hje00] *Hjelm, J.* Designing Wireless Information Services. John Wiley & sons, Inc. 2000.

[HS+03] *Hofer, T., Schwinger, W., Pichler, M., Leonhartsberger, G., Altmann, J. & Retschitzegger, W.* Context-awareness on mobile devices - the hydrogen approach. Proceedings of the 36th Annual Hawaii International Conference on System Sciences, 2003. 6–9 Jan. 2003. Pp. 292–301.

[Jok03] *Jokela, T.* Authoring tools for mobile multimedia content. International Conference on Multimedia and Expo, 2003. ICME '03. Proceedings 2003, Volume 2, 6–9 July 2003. Pp. 637–640.

[KHL02] *Kimmel, J., Hautanen, J. & Levola, T.* Display technologies for portable communication devices. Proceedings of the IEEE, Volume 90, Issue 4, April 2002. Pp. 581–590.

[Kle01] *Klein, M.* XML, RDF, and relatives. IEEE Intelligent Systems, Volume 16, Issue 2, Mar/Apr 2001. Pp. 26 –28.

[KR+03] *Klyne, G., Reynolds, F., Woodrow, C., Ohto, H., Hjelm, J., Butler, M. H. & Tran, L.* Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0. W3C Proposed Recommendation 15 October 2003.
http://www.w3.org/TR/CCPP-struct-vocab/

[Kum03] *Kumar, S.* White paper: Multimodality on thin clients - a closer look at current mobile devices and the multimodal experience possible today. V-Enable Inc, White paper, 2003.
http://www.v-enable.com/products/
Multimodality%20on%20Thin%20Clients.pdf

[KuO00] *Kunz, T. & Omar, S.* A mobile code toolkit for adaptive mobile applications. Third IEEE Workshop on Mobile Computing Systems and Applications, 2000. Pp. 51–59.

[LaS99] *Lassila, O. & Swick, R.R.* Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation 22 February 1999.
http://www.w3.org/TR/REC-rdf-syntax

[LNR96] *Larrondo-Petrie, M. M., Nair, K. R. & Raghavan, G. K.* A domain analysis of Web browser architectures, languages and features. Southcon/96. Conference Record, (1996). Pp. 168–174.

[LR+03] *Larson, J.A., Raman, T.V., Raggett, D., Bodell, M., Johnston, M., Kumar, S., Potter, S. & Waters, K.* W3C Multimodal Interaction Framework. W3C NOTE 06 May 2003,
http://www.w3.org/TR/mmi-framework/

[Luo98] *Luotonen, A.* Web Proxy Servers. Prentice Hall PTR, Upper Saddle River, NJ 0758, (1998). Pp. 51–54.

[MiP01] *Mihovska, A. & Pereira, J.M.* Location-based VAS: killer applications for the nextgeneration mobile Internet. 12th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, 2001, Volume 1, 30 Sept.-3 Oct. 2001. Pp. B-50–B-54.

[MK+02] *Mandato, D., Kovacs, E., Hohl, F. & Amir-Alikhani, H.* CAMP: a context-aware mobile portal. IEEE Communications Magazine, Volume 40, Issue 1, Jan. 2002, pp. 90–97.

[MMI] Multimodal Interaction Activity. http://www.w3.org/2002/mmi/

[MT+02] *Maruyama, H., Tamura, K., Uramoto, N., Murata, M., Clark, A., Nakamura, Y., Neyama, R., Kosaka, K. & Hada, S.* XML and Java Developing Web Applications. Second Edition, Addison-Wesley, Pearson Education, Inc. (2002).

[NFP01] *Niklfeld, G., Finan, R. & Pucher, M.* Multimodal Interface Architecture for Mobile Data Services. Proceedings of TCMC2001 Workshop on Wearable Computing, Graz, 2001. http://userver.ftw.at/~niklfeld/pub/niklfeld_tcmc2001.pdf

[Nob02] *Noble, B.* System Support for Mobile, Adaptive Applications. IEEE Personal Communications, Feb. 2002. pp. 44–49.

[NOK02] Debugging Wireless J2ME/MIDP Applications: An Introduction. Forum Nokia July 9, 2002. http://www.forum.nokia.com/

[OK+01] *Okamoto, M., Kamahara, J., Shimojo, S. & Miyahara, H.* Automatic production of personalized contents with dynamic scenario. IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, 2001. PACRIM. 2001, Volume 1, 26–28 August 2001. Pp. 91–94.

[OS+01] *Ohto, H., Suryanarayana, L. & Hjelm, J.* CC/PP Implementors Guide: Privacy and Protocols. W3C Working Draft, 20 December 2001. http://www.w3.org/TR/CCPP-trust

[Pas98] *Pascoe, J.* Adding generic contextual capabilities to wearable computers. Second International Symposium on Wearable Computers, 1998. Digest of Papers., 19–20 Oct. 1998. Pp. 92–99.

[PLK00] *Palviainen, M., Laakko, T. & Kolari, J.* Visual WML - a development tool for WAP applications. VTT Research Notes 2068 (2000).

[RS+01] *Rössler, H., Sienel, J., Wajda, W., Hoffmann, J. & Kostrzewa, M.* Multimodal Interaction for Mobile Environments. International Workshop on Information Presentation and Natural Multimodal Dialogue, Verona, Italy 14–15 December 2001. http://i3p-class.itc.it/papers/sienel.pdf

[SEMW] W3C Semantic Web Activity. http://www.w3.org/2001/sw/

[UAPROF] User Agent Profile Specification. Wireless Application Protocol, WAG UAPROF, Version 20-October-2001. http://www.openmobilealliance.org/

[WAE] Wireless Application Environment Specification. Wireless Application Protocol, Version 2.0, 7 February 2002. http://www.openmobilealliance.org/

[Wat94] *Watson, T.* Application design for wireless computing. Workshop on Mobile Computing Systems and Applications, 8–9 December 1994, Proceedings, Pp. 91–94.

[WCSS] WAP CSS Specification. Wireless Application Protocol, Version 26-Oct-2001. http://www.openmobilealliance.org/

[WML] Wireless Markup Language Specification. Wireless Application Protocol, Version 1.3, 19 February 2000. http://www.openmobilealliance.org/

[XHTMLMP] XHTML Mobile Profile. Wireless Application Protocol, Version 29-Oct-2001. http://www.openmobilealliance.org/

Published by

Series title, number and report code of publication

VTT Publications 515
VTT–PUBS–515

Author(s) Palviainen, Marko & Laakko, Timo	

Title

mPlaton
Browsing and development platform of mobile applications

Abstract

This report describes the mPlaton browsing and development platform of mobile applications developed at VTT Information Technology. mPlaton supports the development of client-side mobile browsing applications. It provides frameworks and solutions how to browse the content in mobile environment, present the content with profile information in separate user agent implementations with multimodal controls available and create and edit the browsing content.

The core of mPlaton include frameworks for browsing (AGB), for mobile user agents, content types and multimodal controls (MIMEFrame), and for XML editors (FEdXML). The frameworks are designed for mobile environments so that they are very light and scalable, and can be implemented by using different kinds of object-oriented languages (e.g., Java and C++). The solutions introduced in this work are implemented in Java SE, Java Personal Profile and Java MIDP platforms. The solutions are based strongly on standards (e.g., by Open Mobile Alliance (OMA) and World Wide Web Consortium (W3C)) and generic object-oriented technologies, and they are designed to be generic enough to enable the transferability to other platforms.

Keywords
mobile application development, mobile Internet, WAP, XML, XHTML, browsing, mobile platform

Activity unit VTT Information Technology, Tekniikantie 4 B, P.O.Box 1203, FIN-02044 VTT, Finland			
ISBN 951–38–6265–8 (soft back ed.) 951–38–6266–6 (URL:http://www.inf.vtt.fi/pdf/)			Project number T3SU00031
Date November 2003	Language English	Pages 98 p.	Price B
Name of project ALLLAS2		Commissioned by VTT Information Technology	
Series title and ISSN VTT Publications 1235–0621 (soft back ed.) 1455–0849 (URL: http://www.vtt.fi/inf/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	

This report describes the mPlaton browsing and development platform of mobile applications developed at VTT Information Technology. mPlaton supports the development of client-side mobile browsing applications by providing frameworks and solutions how to browse the content in mobile environment, present the content with profile information in separate user agent implementations with multimodal controls available, and create and edit the browsing content.

The core of mPlaton include frameworks for browsing (AGB), for mobile user agents, content types and multimodal controls (MIMEFrame), and for XML editors (FEdXML). The frameworks are designed for mobile environments so that they are very light and scalable, and can be implemented by using different kinds of object-oriented languages (e.g., Java and C++). The solutions introduced in this work are implemented in Java SE, Java Personal Profile and Java MIDP platforms. The solutions strongly rely on standards (e.g., by OMA and W3C) and generic object-oriented technologies and they are designed to be generic enough to enable the transferability to other platforms.

Tätä julkaisua myy	Denna publikation säljs av	This publication is available from
VTT TIETOPALVELU	VTT INFORMATIONSTJÄNST	VTT INFORMATION SERVICE
PL 2000	PB 2000	P.O.Box 2000
02044 VTT	02044 VTT	FIN–02044 VTT, Finland
Puh. (09) 456 4404	Tel. (09) 456 4404	Phone internat. +358 9 456 4404
Faksi (09) 456 4374	Fax (09) 456 4374	Fax +358 9 456 4374

ISBN 951–38–6265–8 (soft back ed.)
ISSN 1235–0621 (soft back ed.)

ISBN 951–38–6266–6 (URL: http://www.vtt.fi/inf/pdf/)
ISSN 1455–0849 (URL: http://www.vtt.fi/inf/pdf/)