HELSINKI UNIVERSITY OF TECHNOLOGY

Department of Electrical and Communications Engineering

Juha Tapio Ristimäki

MULTI-CHANNEL SERVICE PROVIDING

Thesis submitted for examination for the degree of Master of Science in Engineering

Espoo, February 20, 2002

Supervisor Professor Petri Vuorimaa

Instructors Topi Järvinen, Doctor of Philosophy

Teemu Stewen, Master of Science

HELSINKI UNIVERSITY OF TECHNOLOGY ABSTRACT OF MASTER'S THESIS

Author:	Juha Tapio Ristimäki
Name of the thesis:	Multi-channel service providing
Date:	February 20, 2002 **Number of pages:** 117
Department:	Electrical and Communications Engineering
Professorship:	Tik-111 Interactive Digital Media
Supervisor:	Professor Petri Vuorimaa
Instructors:	Topi Järvinen, Ph.D., Sonera Teemu Stewen, M.Sc., Sonera

As diversity of end devices used to access Internet is growing, the way of creating interactive services must be rethought. This thesis discusses the problems in media-independent service providing, especially treated are interactive services. Technical and also some non-technical challenges are discussed.

Extensible Markup Language (XML) is a re-usable document description language that is described with some example applications. Channel specific presentation needs can be satisfied using XML together with Extensible Stylesheet Language (XSL), a language for transforming XML document into other document.

One service development environment is introduced. Proposition for a media independent service development Framework suitable to it is developed and presented. Interactive services producing media independent XML can be integrated with the Framework. Framework identifies client for them and makes channel specific content transformations to their resulting XML documents with appropriate XSL sheets. Support for new media type or integration of new XML service into the Framework does not require touching the program code. Configuring instructions to identify new media and authoring new XSL sheets are what is needed.

Prototype of a media-independent XML producer application is presented and integrated with the Framework. It's media independent use cases are requested differently from different channels. The performance of the prototype in the Framework is tested and reported.

Keywords:	XML, XSL, WWW, WAP, HTML, Interactive services

TEKNILLINEN KORKEAKOULU DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Juha Tapio Ristimäki
Työn nimi: **Päivämäärä:**	Monikanavainen palveluntuotanto 20.2.2002 **Sivumäärä:** 117
Osasto: **Professuuri:**	Sähkö- ja tietoliikennetekniikan osasto Tik-111 Vuorovaikutteinen digitaalinen media
Työn valvoja: **Työn ohjaajat:**	Professori Petri Vuorimaa FT Topi Järvinen, Sonera DI Teemu Stewen, Sonera

Tämä diplomityö käsittelee päätelaiteriippumattomaan palveluntuotantoon liittyviä ongelmia etenkin vuorovaikutteisissa palveluissa. Palveluiden tekotapaa on arvioitava uudelleen kun Internetiin liitytään yhä moninaisemmista päätelaitteista. Työssä esitetään teknisiä ja jonkin verran myös ei-teknisiä aiheeseen liittyviä haasteita.

Extensible Markup Language (XML) on yleinen dokumenttien kuvauskieli, joka esitellään esimerkkisovelluksineen. Päätelaiteriippumattomaan XML dokumenttiin voidaan lisätä kanavakohtainen esitystapa Extensible Stylesheet Languagella (XSL), joka on kieli XML dokumentin käsittelyyn ja muuntamiseen.

Eräs palveluntuotantoympäristö esitellään. Siihen istuva monikanavaisuutta tukeva palveluntuotantokehys kehitetään ja esitellään. Kehykseen voidaan liittää XML:ää tuottavia interaktiivisia palveluja. Kehys tunnistaa päätelaitteen ja sovittaa palveluiden tuottaman XML:n päätelaitteelle sopivaan muotoon XSL:llä. Tuki uudelle päätelaitteelle tai uuden palvelun liittäminen kehykseen ei vaadi ohjelmakoodiin koskemista. Kehykseen konfiguroidaan ohjeet uuden päätelaitteen tunnistamiseen sekä liitetään tarvittavat XSL tyylisivut.

Prototyyppi XML:ää tuottavasta interaktiivisesta palvelusta esitellään ja liitetään kehykseen. Tämän palvelun päätelaiteriippumattomia use caseja kutsutaan erilailla eri päätelaitteista. Prototyypin suorituskyky kehyksessä testataan ja raportoidaan.

Avainsanat:	XML, XSL, WWW, WAP, HTML, vuorovaikutteiset palvelut

PREFACE

This master's thesis was done at Sonera Mobile Operations during the time from June 2001 to February 2002. Thesis was originated from the perception that heterogeneity of end devices accessing Internet can be an upcoming problem when content is traditionally designed for PC only.

I want to thank my foremen at Sonera for giving an opportunity to do this thesis. Special thanks belong to supervisor professor Petri Vuorimaa and to instructors Teemu Stewen and Topi Järvinen, for reading and commenting the incomplete versions of thesis. Risto Pihlajasalo and Ari Kangasharju gave valuable advice on text formatting. Thanks to other workmates for encouragement during the work.

I want to thank my family, grandparents and other relatives, girlfriend and friends for support, understanding and encouragement during my struggles in studying.

February 20th, 2002 in Helsinki

Juha Ristimäki

Table of contents

List of Figures

List of abbreviations

API	Application Programming Interface
CC/PP	Composite Capability/Preference Profiles
CORBA	Common Object Broker Request Architecture
CGW	Content Gateway
CSS	Cascading Style Sheets
DOM	Document Object Model
DSSSL	Document Style Semantics and Specification Language
DTD	Document Type Definition
eCommerce	Electronic Commerce
GML	Generalized Markup Language
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
JAVA	Object Oriented Programming Language
PDA	Personal Digital Assistant
Qwerty	The first six keys of the top letter row in PC keyboards
RMI	Remote Method Invocation
SAX	Simple API for XML
Servlet	A Java-based Web application technology
SGML	Standard Generalized Markup Language
SMIL	Synchronized Multimedia Integration Language
SMS	Short Message Service
SMSC	SMS Center
SOAP	Simple Object Access Protocol
URI	Universal Resource Identifier
URL	Universal Resource Locator
W3C	World Wide Web Consortium
WAP	Wireless Application Protocol
Web	WWW
WML	Wireless Markup Language
WWW	World Wide Web
Xforms	XML application of presenting Web forms
XHTML	Extensible HTML
XHTMLMP	XHTML Mobile Profile
XML	Extensible Markup Language
XPath	XML Path Language
XSL	Extensible Stylesheet Language
XSL FO	XSL Formatting Objects
XSLT	XSL Transformations

1 Introduction

1.1 Background and motivation

In recent years, two fundamental things have happened in telecommunication industry: the emergence of mobile communication and the Internet. 1990 can be named as the decade of these two phenomena. Because people are mobile by nature, the need for mobile communication has been obvious. Internet has given average citizen a comfortable access to global information sources and to other people by connecting the computers of the world. These two networks together have given telecommunication operators huge business opportunities and accordingly the value of telecommunications markets has grown significantly in recent years [1][2].

Now these two networks are converging. Internet access is becoming mobile also thanks to the developments in network and end device technologies [3][4]. Telecommunication in general is becoming ubiquitous after many years of voice communication being the only major application of telecommunication networks. New ways of communication like World Wide Web (WWW), electronic mail (email) and Short Message Service (SMS) have already established position. Although the amount of voice communication is limited to the amount of people in the area, the whole communication market including different information appliances and networked machines can be almost unlimited. It is not clear what will be the actual customer needs, but at least billions of dollars are invested in network capacity, fixed and mobile, in the hope of growing needs [5].

There are three different elements in the communication besides the end user: communications channel, content delivered in it and the end device presenting it to the end user. Cost, quality and availability of these set the limits to the communication market. Right technology and design, content, billing structures, earning principles and revenue sharing in the value chain can speed up the development [6] [7]. It is suggested that NTT DoCoMo of Japan has understood the dynamics of mobile Internet business better than their European counterparts in their I-mode success story [8].

The things become complex in ubiquitous communication, when the same content or service must be made available to all kind of end devices and usage preferences [9][10]. The end user of service can be machine or human with the end device best suitable for presenting the data at the moment. This heterogeneity of end devices must be served somehow. The obvious one is making as many distinct versions of the content or service as there are different presentation needs. This is justified, when the amount of different presentation needs is few. With many presentation needs, managing the service becomes inconvenient. Some degree of scalability would be in order. Time to market and quality of service are essential in the competitive market if the customers will be kept satisfied. Data cannot be in conflict between different versions. Changing the data or only its presentation must be easy. Making the existing service suitable to the latest gadget also should be convenient.

Content providers have met these problems in recent years, when trying to serve different media and end devices like SMS, Wireless Application Protocol (WAP) and WWW. Many traditional industries nowadays have the same problems too, when they have to publish their textual material like manuals and price lists in printed version and in the WWW. In the future, there can be more different presentation needs if new kind of end devices will come popular. At the moment, PDAs and digital televisions are one of these candidates [11][12].

Good thing is to find out that the visually presented content or service has two fundamental elements, the data and its presentation, body and soul [13]. When these two elements can be kept separated, data can be reused and conflicts between data in parallel versions are over. Also, making the existing service suitable to the new kind of end devices is easier, when only presentation issues must be considered and data portion stays the same. This is a human resource thing also. Traditionally, one group of people has made the data portion of the service and the other group has designed the presentation of the data. When data and presentation cannot be separated, these two groups must work in deep interaction, which creates overhead and delays especially, when they work in different organizations [14] [15].

Separation of data and presentation is possible with widely adopted standard Extendable Markup Language (XML) [16], which makes the media independent publishing less inconvenient. In future, the data and services can also be exploited more and more by machines, in electronic commerce and different agents [13][17]. This prerequisites the separation of data and presentation.

At the moment, many services are designed shortsightedly for PC user only. Mobile devices, with their low bandwidth and small displays, require us to re-think the way Web services are created and delivered. Mobile phone sales exceeded the desktop PC sales in 1998 [18]. With the increasing volume of devices offering different capabilities from desktop computers, the Internet will become diverse place. The ability to adapt services to different clients will be essential for web services that wish to prevail. [19]

1.2 Scope

This study explores the field of media independent service development from the mobile operator and service provider point-of-view. Services under examination are interactive. One-way publishing is different and maybe easier thing to cope with. Personal structure and current way of service provision of one mobile operator is taken into account, when considering new operation models. Some things are left out of the scope in order to keep the study focused and adequate in size. Continuous media like voice and movies and personalization of services to different end users are out of scope. Only the presentation layer is in scope, lower communication level issues are left out. The relevant technologies are XML, Extensible Stylesheet Language (XSL) [20] and Java [21]. Special attention is paid to exploiting public software.

1.3 Objectives

This study tries to explore the field of media independent publishing and make some conversation and spread awareness of the problem in the organization. It tries to analyze the problem and present one proposal of a media-independent web publishing framework. It is not realistic to expect that a universal solution to the problem exists, there will be shortages and bottlenecks in every solution. The growing amount of handwork cannot be eliminated, but it should be moderated if possible. At least the way things are done should be as good as possible in the world getting more complex every day. Also, the durability of solutions should be as long as possible. Aim is to think and evaluate different solutions to the existing problem and find out what is good and bad in particular solution.

Changing the course of the big boat is always difficult, so the proposals of putting the new technology in the wider use in the previously not media independently oriented

organization is one objective of this study. The existing service base can be seen as a burden, it is not designed to be media independent. One objective is to find out, whether it is possible to extend these services' lifetime by making them media independent without redesigning them from the scratch.

Finally, the study pays attention to the problem at hand and spreads out the awareness of the new technologies in the organization. The overall approach is non-technical. Justification for the approach is that it is hopefully reachable by bigger audience in the organization and hence maybe more profitable. I recognize that more technical approach may have been more valuable to technology people.

1.4 Organization of this thesis

This diploma thesis employs the following four-parted basic structure [22]:

1) Introduction

   Introduction describes the background and answers the question why the diploma thesis was written.

2) Theoretical Framework

   The "Theoretical Framework" presents the necessary background to the problem and introduces the environment of the thesis. The key technologies in media independent publishing are presented as well as the current service development environment in one Sonera unit.

3) Own Contributions

   In the "Own Contribution" part, the personal work and results are reported.

4) Conclusions

   In this part of the thesis, the results of the work are summarized and the appropriate conclusions are drawn. Evaluation is made whether the objectives were met.

2 Multi-channel publishing

2.1 Foreword

This chapter tries to outline what is meant with multi-channel publishing and what are the problems in it. Different capabilities of different end devices are presented. Some explanation is given what are the different characteristics or elements in services in networked digital media. This chapter should give needed background so that the further chapters can be put into right context.

2.2 Description and background

In multi-channel publishing or service providing, the content or service is made available to several access channels. Basically, supporting more than only one channel justify the use of term multi-channel. In other extreme, the "ubiquitous" service is available to every existing channel and end device in it. So, ubiquitous service can be utilized everywhere, and with all kinds of end devices and user preferences [23] [24]. The location or accessibility of customer, was it either human or machine, usually dictates the type of device used. The words content and service are used interchangeably as well publishing and service or content providing.

Media independent publishing is pretty modern concept. The propagation of digitalization, computers and Internet brought the opportunities and difficulties of exploiting different publishing channels into daylight. Now, WWW has become another mainstream channel for printed word and only few can do without some degree of net presence. Another new channel for information delivery is mobile handsets, with SMS and WAP services, for example. Before WWW and mobile services the printed paper, TV and radio were the only mass media for serious information delivery. There was no encouragement to synergy between these different analog systems. In author's opinion, the current consciousness of the multi-channel production has not been there before the digitalization and propagation of Internet, these started the convergence of different information channels.

The media can be different in many aspects. Communication channels and presentation layers may differ, for example. The difference may be finer than between printed paper and WWW. Different versions of mark-up language or WAP browsers [25], for example, which renders the same presentation model different from the previous version, are different media in this thesis context.

Different end devices have different features that must be taken into account in order to reach good service level. There are basically too ways of achieving good service level across different platforms. Either to make own dedicated version of the service for every end device, or trying to make a scalable solution, where one version of the service or content adapts to all end devices with some kind of intelligent processing.

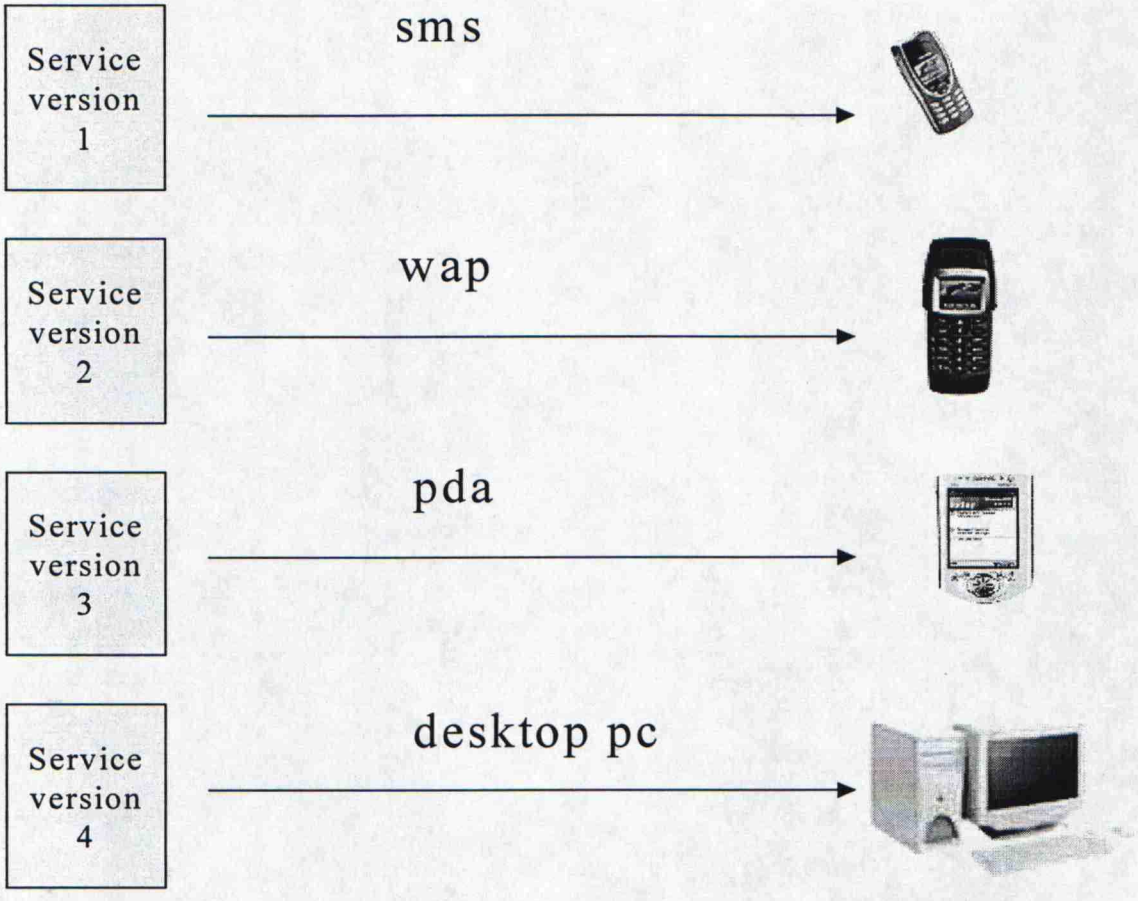


Figure 1. Service replication for different end devices.

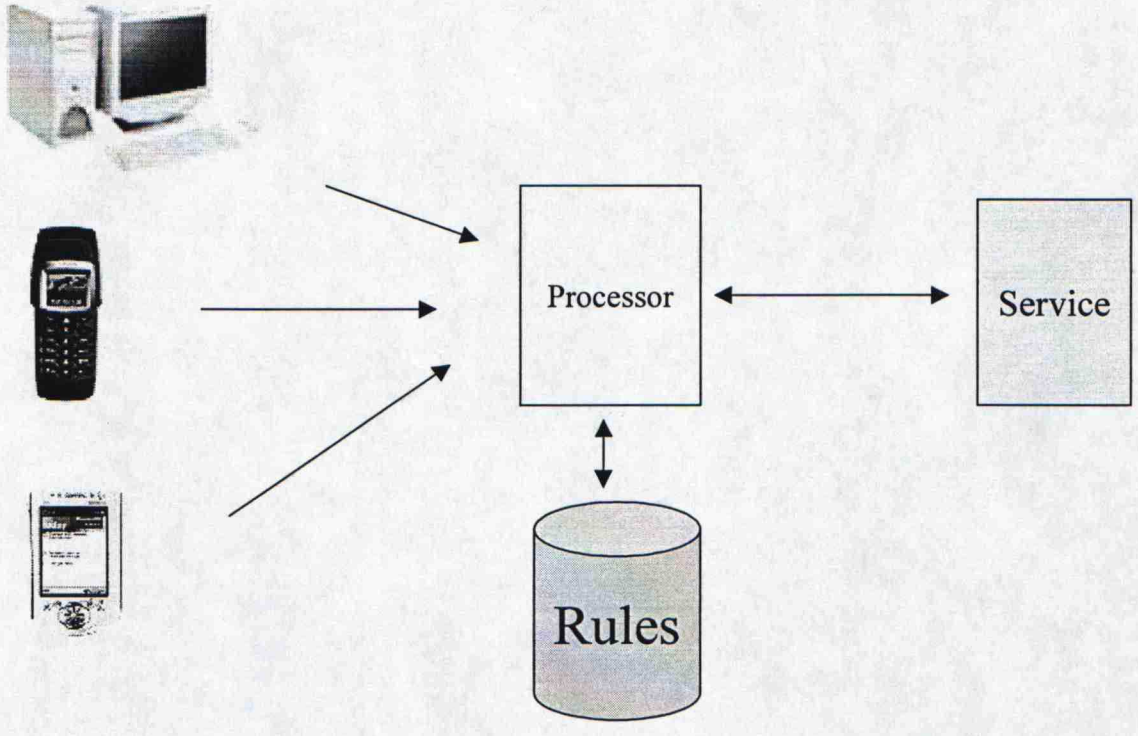


Figure 2. Service adaptation to different end devices.

2.3 *Difficulties in media-independent publishing*

2.3.1 Laborious process

The amount of handwork grows, when new end devices are taken into service portfolio. In practice, this happens many times also, when new models or browser software versions come into the market. Here is described from what components the work is composed of.

New generations of end devices come many times with totally new or improved version of mark-up language [3]. So, several mark-up-languages must be handled. It takes some time until the author is fluent with the new language or version and the best practices are found. Authoring software tools make authoring faster [26], but if the author resorts only to them from the beginning, deeper understanding of the language may not necessarily develop, in author's opinion at least. Deeper understanding can be needed, when authoring tool cannot do something extraordinary or new features come into the languages and browsers.

The effort of learning and commanding new mark-up languages is at tolerable level, but unfortunately this is not enough. Every new model and browser in it must be taken under examination because they may render the same standardized mark-up little differently [25]. In the worst case, the existing service must be reworked or replicated every time a new model comes into the market.

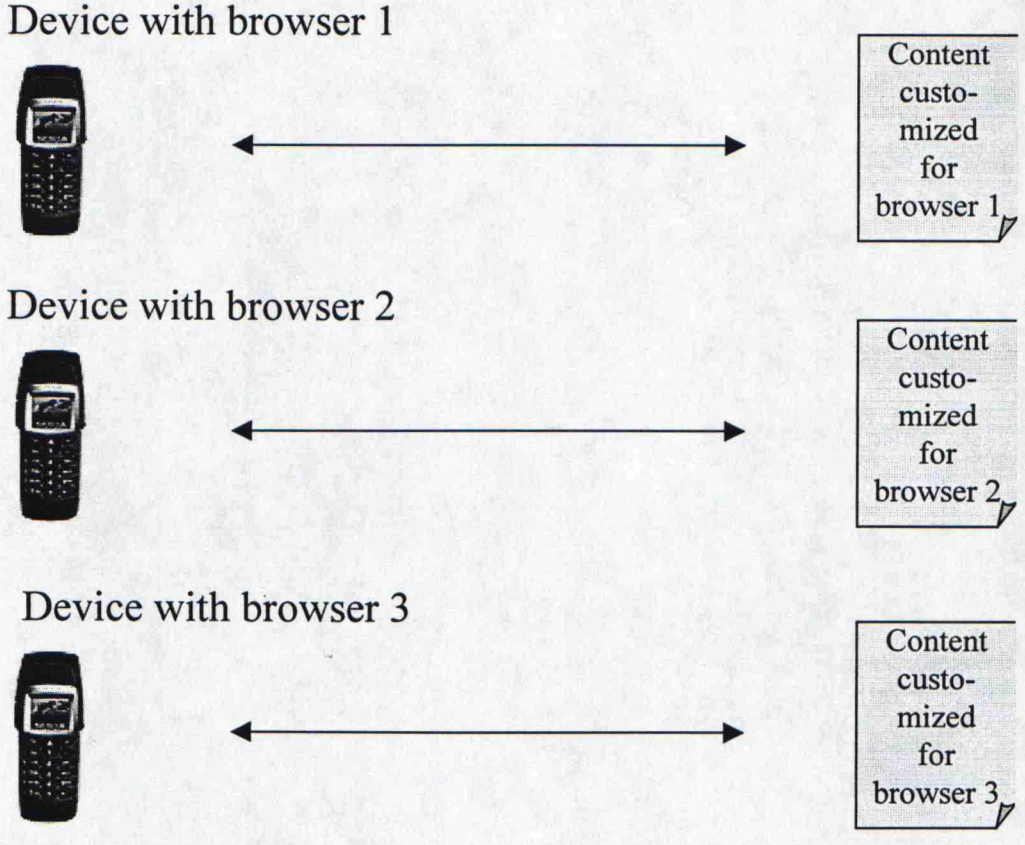


Figure 3. Approach 1, customized content.

The alternative to replicating the services optimal to every model is a general approach, an attempt to define the lowest common denominator features of different browsers. In practice this means identifying a subset of markup that is interpreted in acceptable way in every browser. Some compromises must possibly be made in layouts due to minor inconsistencies, for example, between WWW and WAP browsers of different manufacturer [25]. Good thing with that approach is some kind of scalability, when the amount and maintenance of services and pages do not grow linear with the amount of different models. Following this road has its inconveniences also, when the accumulated empiric knowledge must be documented somehow and kept up-to-date with the latest model. Also, when the page is modified to suit the latest model, it must be checked that it still functions well with the older models.

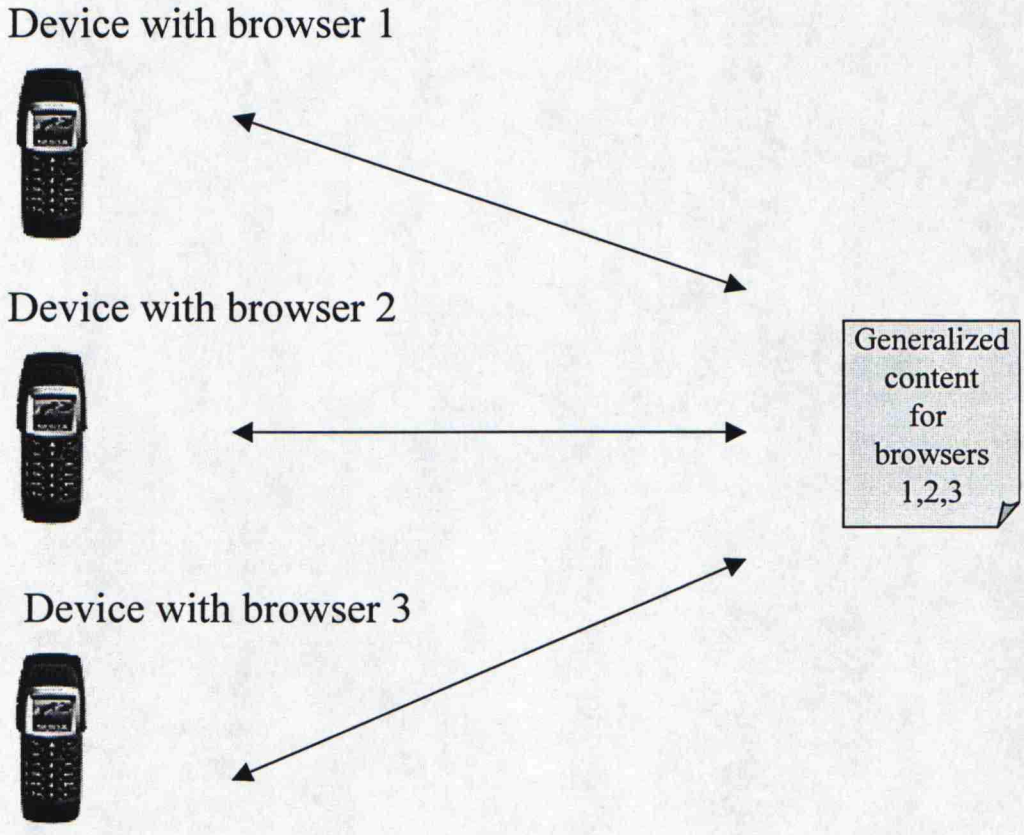


Figure 4. Approach 2, generalized content.

2.3.2 Organizational problems[14][15]

Another problem besides the amount of work is the way work is done in the organization. Overhead in development and maintenance processes and collisions between people can follow if well-defined divisions of labor and responsibilities are missing. This has been noticed, when developing interactive services, where programmers and graphic designers must co-operate. These groups have different priorities that can lead into conflict. Programmers favor robustness, maintainability and efficiency in their code, while designers are concerned about the details in presentation. Many times, when new presentations are made or existing ones are updated, programmers must take the final responsibility of the whole system, when integrating presentation into program code.

The presentation layer is it's own thing and there should be people with adequate competence in charge of that. It would be fine if the work of engineers (programmers and the like) can be separated from the work of artistic personalities (designers) who could modify the presentation with no intervention from the programmer.

This separation matches the organization of many teams. The well-defined division of labor can be missing only because the underlying technologies do not directly encourage to it. The management of work and people could be easier with clean interfaces. Effective use of internal and external resources in service development could be possible and the implementation and integrating processes can be spread across the organization or different organizations. This gives management better opportunities to control and manage the systematic service development. [27]

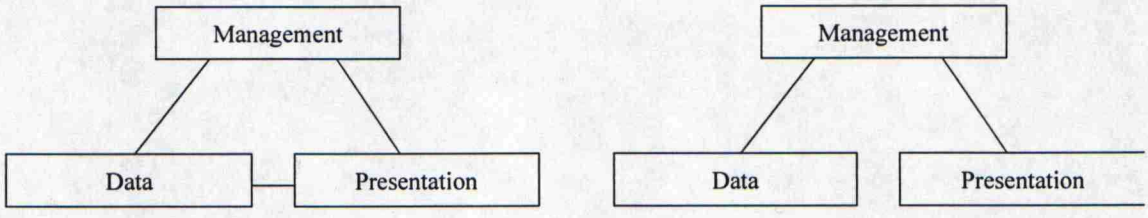


Figure 5. Organizational problems.

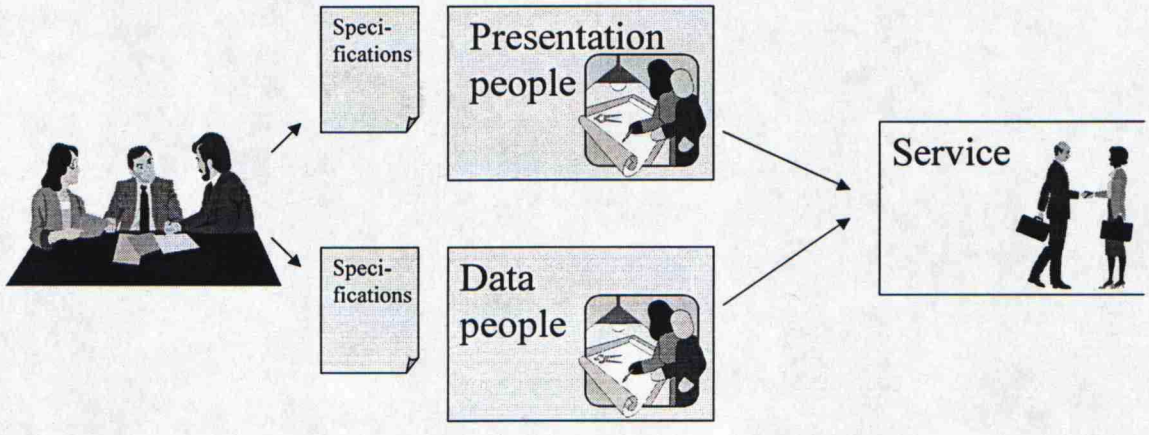


Figure 6. Division of labor between content and presentation.

2.3.3 Slow time-to-market

In strongly competitive market, quick time-to-market is crucial. Maybe more accurate, quick time-to-quality. First in the market gets usually the most sophisticated and payable customers that shape the market and tell other people of the new things. Especially, if the company is touting itself as a forerunner in the business, it is weird if more than a few rivals are on the market before it. In the best case, the first in the market gets so dominant position that its brand becomes synonymous with the product or service itself. Slow time-to-market lowers the customer's threshold to switch to the competitor. Also, some income is lost when the end devices are on the market without services.

2.3.4 The existing base

One problem is the existing base that can be totally out of date being designed restrictively only for certain end device. It is difficult to find consensus on whether to

dump old services and design them all again from the scratch or trying to modify or "wireless-enable" the existing ones. There is software for "wireless-enabling", for example, the existing content. Fully automated or configurable transcoding software tries to convert existing content to other markup languages. This can be even achieved dynamically without touching the original pages directly, which guarantees that the converted pages are always up-to-date with the master pages. In some WAP gateways for example, this kind of transcoder software is embedded. There are anyway pros and cons in using these converting tools comparing to developing all from the scratch. One of the disadvantages is the price coming usually with them. They are also non-predictable, human eye is needed to check that every page is transformed correctly. And there may be situations, where automatic transcoding is not possible at all. [25]

Another trouble is that in WWW community, the HTML pages are often poorly written (i.e., the HTML specification is not respected). Today's browser manufacturers are aware of this and they work hard to make browsers that can cope with HTML pages no matter how badly they are written. This functionality comes with increased memory and processing consumption. The web authoring tool-makers are taking advantage of the status quo. The pages they pump out can be far from standard HTML trying to optimize the behavior of the non-standard mainstream browsers. [28]

PCs can cope with this situation with large memory and fast processor, but wireless devices, for example, cannot. The browsers in thin clients are smaller and thus more strict in what they accept. They may not accept some parts of HTML at all (e.g., frames) and even then only standard HTML [11]. There is fortunately good quality public software for automatic cleaning of HTML pages [29]. The transcoding software may prerequisite that the master pages are clean HTML [28]. The page sizes and pictures are many times designed for good connections. On today's wireless connections these pages take too long to load.

2.3.5 Look-and-feel and brand

In customer business, it is important to promote the brand always, when dealing with customer. If company has built valuable brand with lots of money in traditional media, it probably wants to cultivate it also in new media. Services and content should look familiar in order to create trust and brand loyalty. Keeping look-and-feel coherent across different media and different end devices can be very difficult, because services may appear so different in different clients in the eye of the customer. In digital services, for example, the convenient user interface and positive user experience contributes to the brand loyalty [30].

2.4 Features of media

2.4.1 Capabilities in different end devices

Different end devices come with different capabilities. The set of features and capabilities that, at least, must be taken into account are output and input capabilities, available processor power and memory size, ability to handle different multimedia and picture formats, support of different mark-up, scripting and application languages [10][31].

Output

Width and height and proportions of the screen must be considered, when sending text and graphics. Text and graphics can be customized to fit in particular screen size. If screen size is very small compared to content's primary access device, only limited amount of text can be sent (summary), or text can be divided into several pieces and linked together with so-called "more buttons" [25]. The amount of text in each page should be considered. People may find it laborious or inconvenient if there is overdose of text. One approach is one view per page so that no scrollbars are needed. This approach has its disadvantages if there is very much text, since the amount of metadata for navigation increases. The advantage gained in authoring small and simple pages is, that they fit in every browser, also in browsers designed for more sophisticated end devices [25]. The Finnish mobile handset manufacturer Nokia can hopefully ease content developers' efforts with their new policy. They promise to have only four different user interfaces with four screen sizes in their upcoming phone models [32]. Content providers likely wish that other manufacturers also will follow this kind of harmonization policy. The legacy models must be still taken into account. There can be also some kind of sound output, requiring speaker (e.g., synthesized speech for handicapped, recorded speech, music) [33][31]. Also, the viewing distance must be considered. It is different between television and desktop PC, for example [34].

Input device and navigation

When designing and converting interactive services, it should be noted that all devices don't have maximum set of user interface features and capabilities. Convenient pointer device or qwerty keyboard may be missing. General choice and arrow buttons with color identifiers can be used in menus, if convenient pointer is missing [12]. Users may favor portals and bookmarks instead of entering long Universal Resource Locations (URLs), for example, if convenient keyboard is missing. One issue is that different browsers may render the same markup differently in the sense of navigability, 'one click away' on one particular device may be two or more click on another device [25]. The interaction should be convenient enough, otherwise the user may not use the service at all or uses it the first and the last time [30]. Touch screen and voice input are also possible input alternatives. Special attention should be paid to handicapped people, their disabilities must be taken into consideration, when designing the output and input solutions [35].

Supported mark-up language and its features

Appropriate customizations may have to be made to different kind of basic structures, for example, tables and forms. Some browsers may support presentation languages only partly. In future, Extensible HTML(XHTML) [36] will have different modules supporting different entities [66]. It should be communicated with client what elements are supported. Tables, forms and frames are among the most important ones to be taken into account. There may not be clean counterparts between particular tags and features in different languages, so creativity and common sense are needed, when customizing the structure and layout best suitable. It should be elaborated if rule based tag conversions are adequate [25]. Applying rules by appropriate software possibly with minor handmade corrections reduce greatly the inconvenient handwork. It should be also considered how the page appears in browsers without CSS support. Essential is that functionality and user-friendliness is maintained after transformations.

Ability to handle image and multimedia types

Conversions between different image and multimedia types should be made if needed. In case of images, resizing only may not be enough. It should be investigated, which image

types are supported in each particular agent type or browser. Processor power of particular client may dictate, whether the sophisticated pictures and multimedia is convenient to handle in it. There is software that can make image type conversions and image data reduction like color depth reduction [37]. In general, it should be considered, whether images can be entirely dropped away in smaller screens and slow connections. The reality can be occasionally blurred if services are developed and tested in high-speed connections that are not at reach of the average customer. If images are not essential in small screens, they can be replaced with links to them or with ASCII characters to depict the same things literally.

Transformed unit size

Some devices may have certain physical upper limit in transformed unit size. This is fact in current WAP phones [38]. If content designed for unlimited transform unit sizes is fed into WAP phones, very tricky division must be made with content, or it must be trimmed. In static content, the division can be simply made with links to next portion of data ("more buttons"). There are automatic tools for this in some WAP Gateways, for example [25]. Interactive content is more difficult, because the whole content in the same interaction context should be treated as one unit. Logic implemented in server software dictates the solution. Only solution may be to trim the original content so that it fits.

Session management

Session consists of series of transactions between customer and service and the transaction data involved. Application may have to keep track of all the session related data in certain kind of services like shopping cart. Client support for session management varies between different terminals. In HTTP client-server-model, session management is typically handled with HTTP Cookies, hidden parameter exchange or with URL rewrite [39]. Cookies are not supported in all current WAP clients and not at all in SMS clients. Sessions can be emulated in WAP with hidden parameter exchange like in HTTP. Fortunately, some WAP gateways store cookies on behalf of devices [25].

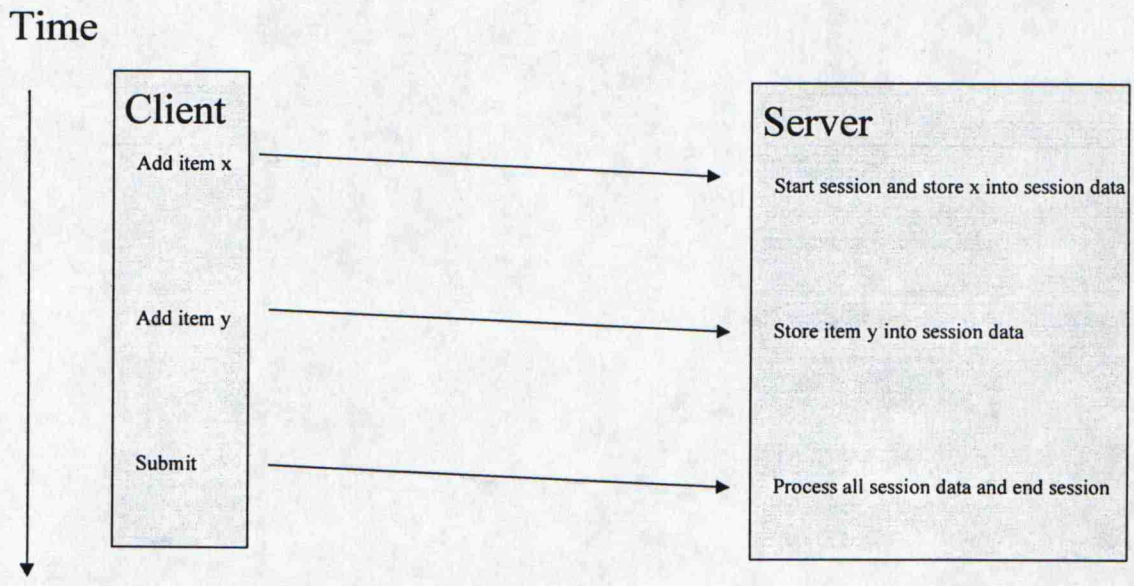


Figure 7. Session management.

Scripting languages and applets

Scripting languages intermingled in the content sent to client enable processing (e.g., checking of user input) in client's own processor. All client types do not have same scripting capabilities. For example, in current desktop HTML browsers, there is support for Javascript and, in many Wireless Markup Language (WML) browsers in WAP phones, there is support for Wmlscript. It is laborious to learn all scripting languages. Fortunately, Javascript based Ecmascript is destined to replace the various forms of scripting in the future [40]. If there is no way at all to cover the things in client that were handled by scripts in some other media, the only way is to change the application logic in server side. Server software must take the role of scripting language also. Negative effect in this is that the amount of time consuming transactions between client and server easily increases. For example, when input fields have to be sent to the server to be checked against correctness. Java applets are supported in PC HTML browsers but not for example in WAP.

```
function check() {
    if (document.sms.MSG.value.length < 1) {
        alert("Message is missing!");
        return false;
    }
    else return true;
}
```

Figure 8. Example of a scripting language (Javascript).

In future, some kind of standardization of presentation and user interface features in information appliances is needed [23]. Too much overhead and inconvenience must be tolerated today in customizing the layout and content for different devices.

2.5 Elements of visualized content [41]

Documents can be defined to have three distinctive characters in it, content, structure and presentation. These three elements can be found in any kind of documents. Presentation can be sometimes further divided into its composition and style. The context of this thesis is one popular class of electronic documents, markup documents.

2.5.1 Content

The content of document is the information of the document. Content can be text, pictures or voice, for example. If the content can be separated from the presentation, it can be presented in different ways. Content can be static or dynamic. Static content is permanent and can be stored in intermediate storage also, like in proxy servers of Internet. Dynamic content is created on demand runtime by applications (e.g., web services in Internet) and it cannot be stored in intermediate storage.

If content can be separated from presentation, it can be hold in separate storage unit, where it can be fetched into different presentations. Updates and corrections must be made to one place only and no conflicts between information in different presentations can then exist as a consequence of human error or typo.

2.5.2 Structure

The structure of the document breaks up into physical and logical structure. Physical structure tells how the different entities of the document are physically organized. Physical structure is syntax for computer, telling how logical structure can be accessed.

Logical structure is the structure and relationships of the meaningful content, and it is distinct from the physical structure. Titles, paragraphs, headers, for example, belong to logical structure. Logical structure is semantic for humans and computers reading and processing the document.

2.5.3 Presentation

The presentation of the document can be broken up into internal and external presentation. Internal presentation of the content is the low-level presentation used by computers. It can contain byte order and character encoding, for example. External presentation is the physical appearance of the document. It is wrapped around document's content in some phase to fit in the eye of the human reader. Same content can have different external presentations to different purposes and audiences. The external presentation can be further separated to its composition of different elements, and to its style (i.e., editorial layout).

Pictures, headers, paragraphs, tables, buttons, input forms, lists, in hypertext languages (e.g., HTML) also links to other documents are among these elements, for example. The document's composition of these can be presented with some markup languages (e.g., HTML).

Style can be described as the look and feel of the service. Indents, fonts, borders, colors and backgrounds, for example, construct style. These style directives are applied to desired elements in presentation. The overall impression of a group of documents can be kept familiar with common style. If style elements can be kept in separate storage, effort is smaller, when changing the style of large document base. There is couple of situations, where decision is made to change the style of the whole document base. Every now and then some renewal and face lifting may be needed, or when the company gets a new logo, name or both in a merger, for example. In HTML, for example, style can be separated with cascading style sheet (CSS) [42] technology.

3 The most important technologies

3.1 Foreword

In this chapter, few useful technologies in media-independent service development are described. Especially described is XML, which is the base for understanding and exploiting many other technologies too, like XSL. Few other relevant technologies are also described but only generally. All described technologies are recommendations or proposals of World Wide Web Consortium (W3C) and they are publicly available and not vendor specific. W3C recommended technologies have gone through long lasting and careful preparation process under the supervision of many authorized software professionals and powerful companies [43].

3.2 XML

3.2.1 What it is?

Popular description

Extendable Markup Language (XML) [16] is just a one way of describing data in a structured manner. When data is described in an agreed structured format, computers can understand it. In XML document, data being described can be anything. XML document consists of actual data and data about data, metadata, telling the meaning of the actual data to computer. XML document is human readable as well because it is text based. XML is becoming popular data storage and communicating language in the computerized world [28]. XML tells only what the content means, not how it should be presented. Together with Extensible Stylesheet Language (XSL) [17] or other mechanism, XML document can be presented and styled in desired manner.

Example of an XML document and its use

For example, one can agree with somebody upon the grammar and meaning of an XML document describing the contents of his fridge. The metadata labeling the meaning of every particular data field is enclosed in angle brackets, <> and </>. All actual content is enclosed within the beginning <label_of_data>, and corresponding ending tag </label_of_data>. One can tell about the contents of his fridge something like this:

```
<my_fridge>
    <ingredients>
    <amount_of_items>3</amount_of_items>
    <item>
        <name>milk</name>
        <left>0.5 l</left>
```

```
    </item>
    <item>
        <name>bread</name>
        <left>2 pieces</left>
    </item>
    <item>
        <name>butter</name>
        <left>one third</left>
    </item>
    </ingredients>
</myfridge>
```

Figure 9. Example of an XML document.

If human being gets this XML file telling the contents of somebody's fridge, she or he will intuitively understand what it is about. Computers cannot do this; they don't have any intuition [17]. But if they are beforehand told unambiguously the meaning of items <my_fridge> and <milk>, everything comes possible and computers can communicate with each other. By using XML, the computers in fridge and grocery could, for example, communicate with each other and agree upon the order of reserving the missing items to one's fridge.

Formal Description

> *"XML describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them. XML is an application profile or restricted form of SGML. By construction, XML documents are conforming SGML documents". [16]*

3.2.2 Origin

So, XML has its origins in Standard Generalized Markup Language (SGML) [44], but also in the proliferation of WWW and its popular markup language HTML.

SGML [19][45]

SGML evolved from the Generalized Markup Language (GML) that was developed by Charles F. Coldfarb and others in IBM in 1969. GML was the first modern markup language for marking the structure of an arbitrary set of data.

In order to treat documents electronically, it was essential that their logical structure was clearly marked. On top of that, to ensure that documents are really

interchangeable, one had to develop a common language to implement this type of representation. When there are several parties communicating or sharing documents, using the common vocabulary and syntax is justified to avoid conflicts and misunderstandings. GML was invented to be a meta-language, a language that could be used to describe other languages, their grammars and vocabularies. GML later came SGML, which was adopted in 1986 as an international data storage and exchange standard by the International Organization for Standardization (ISO).

Plenty of applications have been designed to exploit common SGML vocabularies. Their grammars and vocabularies are defined in Document Type Definitions (DTD). SGML documents respecting syntax of common DTDs, belong to a same SGML language and can be understood by applications designed for that language. With accurate respect of DTD, documents to its language can be constructed as well as applications processing them.

SGML is a powerful, but rather complicated markup language. Because it is approved by ISO, it has been widely adopted by manufacturing companies and publishers of technical information, for example. Publishers often construct paper documents, such as books, reports, and reference manuals in SGML. These SGML documents can be transformed into desired presentable format. Because SGML is text-based, and it can be transformed into other formats, it is device- and system-independent method of representing electronic documents.

Several Document Type Definitions (DTD) have been made with SGML in the industry.

Major industry DTDs (markup languages):

- ATA 2100 aircraft industry
- CALS military, aerospace
- CMC pharmaceuticals
- PCIS semiconductors
- DocBook computer software
- IBMIDDoc IBM software
- SAE J2008 automobile manufacturing
- TMC T2008 truck manufacturing
- TIM telecommunications
- EDGAR Securities and Exchange Commission
- ISO 12083 journal, book and magazine publishing
- ICADD publishing for the print-disabled
- TEI academic and scholarly publishing
- UTF news media

- HTML World Wide Web

Figure 10. Major industry DTDs [46].

Deficiencies in HTML and SGML created XML [28][19]

Another origin for XML was explicit deficiencies in HTML. Its simplicity has contributed to the current popularity of WWW, but it has several limitations. It is too restricted to fulfill the needs of interoperable and diversified WWW. The strict syntax for HTML exists, but everybody does not follow it. This has lead to not interoperability problems. The restricted vocabulary of standard HTML does not support the machine-machine communication well enough, because it is primarily designed for presentation. HTML cannot be extended and kept separate from presentation format.

SGML fulfilled the other requirements for the new WWW language, but it was not simple enough. SGML was too complex for humans and computers to be used in larger scale. With its many optional features, SGML processors were large and complicated computer programs and SGML was not human readable.

With the development of WWW and popularity of its simplistic HTML, there was a need for as simple language that will have the good features of SGML, flexibility, extendibility, optional validation and capability to transformations into other languages. This kind of language was especially needed for electronic commerce.

XML evolved and it tries to compensate HTML's and SGML's deficiencies by being as simple as HTML and as powerful as SGML.

XML design goals [16]

The design goals for XML were:

1.XML shall be straightforwardly usable over the Internet.

2.XML shall support a wide variety of applications.

3.XML shall be compatible with SGML.

4.It shall be easy to write programs which process XML documents.

5.The number of optional features in XML is to be kept to the absolute minimum, ideally zero.

6.XML documents should be human-legible and reasonably clear.

7.The XML design should be prepared quickly.

8.The design of XML shall be formal and concise.

9.XML documents shall be easy to create.

10.Terseness in XML markup is of minimal importance.

Design goals are explained more in [47].

First XML W3C recommendation is from the year 1998. The latest version is second edition from the year 2000 and it has some errors corrected in convenience to readers.

3.2.3 XML languages

As like SGML, XML is a meta-language and can be used to describe grammars and vocabularies of special classes of documents, or languages. DTDs can be used also in defining XML languages. Because XML is a subset of SGML, XML languages are also SGML languages. The relationship between these kinds of languages is depicted in Figure 11.

XML can be used also as "standalone", without DTD and validation, as long as it has well-formed XML syntax. But when there is several parties communicating or sharing XML, use of common vocabulary and syntax is justified to avoid conflicts and misunderstandings.

Plenty of applications have been designed to exploit XML already and XML can be the common base for languages of the next generation Web architecture, for communication and for presentation. Some XML applications and their languages are introduced in 3.8. XML document is described next. DTDs are described in 3.3.

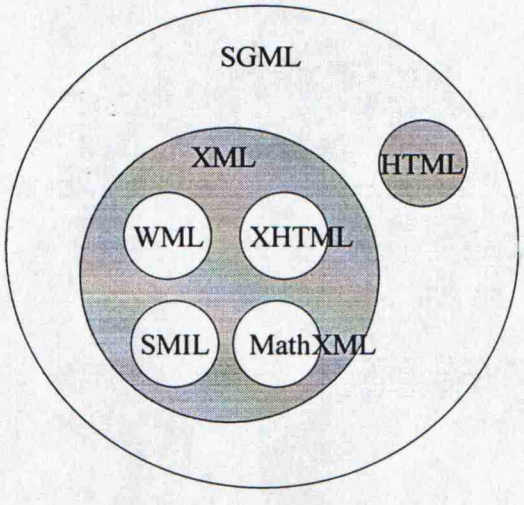


Figure 11. Families of SGML and XML languages.

3.2.4 XML document [16][19]

Definition

> *"A data object is an XML document if it is well-formed, as defined in the XML recommendation. A well-formed XML document may in addition be valid if it meets certain further constraints". [16]*

These further constraints can be stated in external or internal DTD. DTDs and well-formness of XML document are described soon.

Physical and logical structure

Each XML document has both a logical and a physical structure. Physically, the document is composed of storage units called entities. An entity may also refer to other entities, internal or external, to cause their inclusion in the document. Internal entities are located in the same file, external entities in some other file, local or remote. This means that XML document can be aggregated from several places, when appropriate. The whole document is a "root" or document entity, from which the processing or the parsing of an XML document begins.

Logically, the document is composed of declarations, elements, comments, character references, entity references, CDATA sections and processing instructions, all of which are indicated in the document by their explicit markup. The logical and physical structures in XML document must nest property to fulfill the well-formedness constraints.

Character data and markup

XML document contains text, a sequence of characters, which may represent markup or character data. Binary data must be encoded into character form, or it can be referenced from its external source. Character range is defined in recommendation. Markup contains start tags, end tags, empty-element tags, attributes, entity and character references, comments, CDATA section delimiters, document type declarations, processing instructions, XML and text declarations. All text that is not markup, is character data of the document. Markup facilitates the processing of the document (XML parser). Character data is for secondary applications employing this processor.

There are few special characters used to delimit the markup from character data. If used in character data, these delimiters must be escaped correctly in order to surpass their markup meaning. Delimiter characters < and & can not be used in their literal form in character data, it causes XML processor to misinterpret them as markup, thus ending up in wrong assumption and usually also violating the well-formedness. Markup delimiters are escaped in characted data, using their numeric character references or corresponding predefined entity references (described soon), in order to surpass their markup meaning. In CDATA sections, the end delimiter "]]>" must be escaped, in element attributes, ' and " must be used unambiguously.

Elements and attributes

Elements are the basic building blocks of XML markup. Elements' content is delimited by matching start and end tags, such as `<element>` and `</element>`, where the name of the element is inside brackets. Exception is the empty element, which can be represented also like <element/>. The content can be character data like text strings or other elements. Elements must be properly nested and these nested subelements are called child elements. The nested structure of XML document is tree-like, like branch in a tree can have multiple branches, element in an XML document can have multiple elements. Elements can have children, parents and siblings. The first or the outermost element in XML document is called "root element", which name must be "xml". Root element must contain non-empty sub-tree of other elements, the root of which is called "document element". The elements with no children are leaf elements.

Inside the beginning or empty tag can be optional attributes. Attributes are name-value pairs telling something about the element. It is up to designer, whether to place data in attributes like this:

```
<person social_security_number="01234897-xxx" sex="male"
name="Teekkari Teemu"</person>
```

or in child elements like this:

```
<person>

<name>Teekkari Teemu</name>

<social_security_number>01234897-
xxx</social_security_number>

<sex>male</sex>

</person>
```

Somebody has said that elements are the nouns of XML document and attributes their adjectives. Attributes have no tree-like structural relation to each other, like elements. The programmatic manipulation and accessibility of elements is easier. The degree of extensibility achieved with elements is not possible with attributes.

Character and entity references

XML document consists physically merely of entities. When referencing from XML document to internal or external entities, to cause their inline inclusion in the document, so-called entity references are used.

Numeric character references reference to individual characters with their Unicode character value. In XML, only ISO-Latin characters are supported in their literal form. So, other characters like Scandinavian letters, must be referenced like "&#nnn;", where "nnn" is unicode decimal value, or "&#xmmm;", where "mmm" is unicode hexadecimal value.

Entity references reference to declared entities. Syntax of reference is &entity;, where "entity" must be declared in internal or external DTD. All XML processors must recognize five predefined entities without explicit declaration. These are showed in table below.

Character	Entity reference
&	**&**
<	**<**
>	**>**
'	**'**

''	"

Figure 12. Predefined entities in conforming XML processors.

In order to surpass markup delimiters primary meaning, these delimiter characters must be escaped with predefined entity or their numeric character references. For example, < is `lt` and > is `gt`. So, syntax of referencing to these entities in is, `<` and `>`. Entity references can be used in other purposes also like in referring to often-repeated text portions or the content of external files.

Processing instructions

Processing instructions (PI) are used to provide instructions from XML document to applications processing them. These applications are called processors and they know what to do with the document or particular elements in it. One special processor is XML processor that is better known as XML parser.

For example, PI `<?XML version="1.0"?>` should be in the optional prolog of XML document and it is targeted to XML parser itself. This XML declaration PI tells XML parser that the document should be treated as a XML 1.0 document. Prolog is described soon in the structure of XML document.

Actually, PIs are used to provide instructions from XML parser to secondary processor applications. Parser parses the whole document and it must pass the PIs to their processors, where they can be further processed. In general, PIs have syntax of `<?PINAME PIDATA?>`. PINAME tells which processor is targeted with the instruction and PIDATA is fed to this processor. Ideally, this kind of policy, where additional data is put into XML document should be avoided in order to respect the design goals of simple XML.

Several processors can process XML document. You can declare in the beginning of the document what processors are used during the parsing process. Each processor then gets the PIs and data targeted to it and processes it further and maybe returns some XML data in return. For example, if data from database is fetched into XML document with mysqlprocessor, it can be marked in the beginning of the document as `<?mysqlprocessor user="matti" passwd="masa123"?>`. When XML parser notices this line, it loads into runtime memory the processor named "mysqlprocessor" and during the parsing mysqlprocessor can insert XML formatted database query results into the portions it identifies.

Comments

Comments start with `<!--` and end with `-->`. They can be placed anywhere in the XML document outside other markup. They are a way to remove some markup from use temporarily or to comment something meaningful from the authors or editors of the document. XML processor (parser) is not required to pass the comments to the applications.

Declarations

XML provides a mechanism, the document type declaration, to define constraints on the logical structure. Declarations are part of the XML recommendation, but they are described in own chapter 3.3.

CDATA sections

In CDATA sections can be included text that contains characters that would otherwise be recognized as markup. Data in CDATA block is not mark-up, so it is not parsed, or processed by XML applications. Inside CDATA block anything can be written except the premature ending CDATA notation "]]>" in its literal form. Below is an example of the CDATA block.

```
<![[CDATA <title>Example of an XML document</title>]]>
```

This feature is useful, for example, if author wants to include examples of XML markup in his document. Work-around for including binary data in XML document is to put it in the CDATA section. CDATA section is not good place to but binary data. It is possible that binary data contains premature ending notation "]]>", which causes malfunction. The binary data could be encoded with Base64 or other technique, where encoded data newer contains ">" character. With this approach, if the encoded data does not contain any other misleading markup delimiters either, binary data can be as well contained in element content. Other way is to put binary data behind link.

Well-formedness

A textual object is a well-formed XML document if it meets all the well-formedness constraints given in the XML recommendation. The logical and physical structures must nest properly, which means that no start-tag, end-tag, empty-element tag, element, comment, processing instruction, character reference, or entity reference can begin in one entity and end in another. In other words, if the start-tag is in the content of another element, the end-tag is in the content of the same element, or ending tag closes the previous beginning tag.

This is not correct:

```
<example><strong>example</example></strong>
```

This is correct:

```
<example><strong>example</strong></example>
```

Unclosed elements violate the well-formedness. Elements must always have also the ending tag. Either `</element>` or `<element/>` in the case of empty elements.

Attribute values must always be surrounded with quotation marks or apostrophes, like `id="123"` or `id='123'`. Quotation mark can appear in attribute's value, if it is enclosed with apostrophes, and vice versa. Entity or character references must be used otherwise.

Document must contain one or more elements. The outermost element is called document element, no part of which can appear in the content of any other element.

Well-formness violations are called fatal errors, and they should abort the parsing of document.

Names

XML names are case sensitive, meaning `<Element>` is considered different from `<element>`. Naming conventions are defined in recommendation, names have legal starting characters and continuing valid name characters. Names starting with "xml", or with any variation of those three characters, are reserved for XML standardization.

XML document's structure

XML document is comprised of three parts, optional prolog, body, and optional epilog. It starts with prolog, which tells the parser that document data begins and may give additional hints to parser and applications. XML document should begin with special PI called XML declaration, which specifies the version of XML being used:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

Encoding and standalone attributes are optional. Former one tells, which character encoding is used, and the latter one if the XML document has DTD for optional validation. Their default values are "UTF-8" and "yes" (i.e., no validation). Prolog can contain in addition to this other PIs, comments and document type declaration that is described in 3.3.

After the prolog comes body, which must contain one or more elements (i.e., at least the document element like stated in the well-formness rules), in the form of hierarchical tree that may contain also character data, comments and CDATA sections. After the body comes optional epilog, which can contain comments and PIs.

Well-formed vs. valid XML document

Document that respects the syntax of XML defined in recommendation is said to be well-formed. XML parser implemented in accordance with recommendations must accept well-formed XML and must not accept any other XML. If document is not well-formed it can not be parsed (i.e., understood by XML parser).

Validity is a further requirements set to XML document if so desired. An XML document is valid if it has an associated document type declaration and if the document complies with the constraints expressed in it. Validation takes some extra time in parsing so it is up to the use situation, whether it is rational to check document against validity. Validation can be done in server or in client or both.

3.3 Document Type Definitions (DTD) [16] [19]

> *"XML recommendation provides a mechanism, the document type declaration, to define constraints on the logical structure of XML documents and to support the use of predefined storage units". [16]*

There are four kinds of markup declarations:

1. element type declarations
2. attribute-list declarations
3. entity declarations
4. notation declarations

The syntactic rules of XML document's vocabulary can be defined with DTD. The allowed vocabulary, correct nesting, cardinality and order of elements can be set. The attributes and their default values can be set. Elements and their attributes can be set mandatory or optional.

Internal and External document type declarations

With DTD, each XML file can carry a description of its own format [48]. Document can be associated to its declarations in two ways. Declarations can be written in the beginning of the XML document or in its own file, where it can be referenced from the XML documents. XML prolog can contain document type declaration, in "DOCTYPE" block, which contains internal subset, and/or refers to external subset of, DTD. The DTD for a document consists of both internal and external subsets.

Syntax of document type declaration referencing to external DTD:

```
<!DOCTYPE root_element_name SYSTEM "system_identifier">

<!DOCTYPE root_element_name PUBLIC "public_identifier"
"system_identifier">
```

In the declaration, the root-element is replaced with the name of the actual root-element. An example of DTD is in appendix A.

In validating XML parsers, there is a resolution mechanism implemented, of using "system_identifier" and "public_identifier" names to locate the external DTD. "system_identifier" is the explicit and direct URL of the DTD. By using the "PUBLIC" keyword with the URI, parsers are given the opportunity to locate the DTD using their own algorithms.

Using internal subset increases the size of XML document. If the same DTD is common to several documents, keeping one external DTD saves from conflicts and from writing the same DTD into every document. With the policy of using external DTDs, the control over DTDs can be left only to the owners of information. Internal subset must be transmitted always, even though the validation is not needed. External DTD can be cached also for efficiency. If external and internal subset have common declarations, internal ones supersede the external ones.

Deficiencies with DTDs

DTDs are considered to have several limitations:

1. they are difficult to write and understand

2. they are not extensible

3. they do not provide support for namespaces

4. there is no support for data types

5. there is no support for inheritance

When XML is used in more programmatic situations, these limitations become increasing important. XML Schemas have been designed to pass the problems with DTD.

DTD is SGML based language definition language but it is not XML, it is in Extended Backus Naur Form (EBNF), and cannot been parsed and used for validation with plain XML parsers. Validation is optional feature in XML parsers. DTDs need their own parsers, and it is not possible to inquire into DTD with Document Object Model (DOM) application (described soon).

3.4 Schemas [19]

Schemas are XML documents, and their programmatic runtime manipulation (e.g., possible via DOM) is more powerful. Even though XML parsers can parse Schemas, all of them do not yet support its vocabulary. Other valuable XML advantages in Schemas are extensibility and reusability. It is possible to borrow and inherit to Schema from other Schemas, and they provide better support for aggregation and reuse of components. Single document can relate to several schemas.

XML namespaces can be used together with Schemas. They are needed to avoid name collisions, when using several declarations together. There can be many declarations for the same element. Namespace is a collection of names, identified by a URI reference, which are used in XML document as element types and attribute names [49].

Schemas enable a little more precision on the structure of document, and on the definition of the data types of its elements and attributes. The only data type in XML is text, and with DTD, only few data types other than text are supported. In Schemas, there are several useful predefined simple data types, like date and time. Complex data types can be formed by combining simple data types. In modern programming languages, these kind of data types are also used, as well as in the real world. There are two normative references of Schemas at the moment [50][51]. Example of a schema is in appendix D.

3.5 XML parsers, SAX and DOM [19][28]

Parser is the module that parses the XML document and provides it to other applications, like XSL transformer. There are two major types of XML Application Programming Interfaces (APIs): event-based and tree-based. One popular API for both of these API types exists. Event-based SAX (Simple API for XML) and tree-based DOM (Document Object Model) are wide spread XML APIs. These APIs are used for reading and manipulating the XML documents programmatically. XML parsers and processors can use these interfaces.

SAX API is event-based, read-only and stream oriented. SAX parser traverses the XML stream and reports parsing events (such as the start and end of elements) directly to the

application through callbacks. Event handler methods catching the different parsing events do what they have to do when encountering, for example, the start or end or characters of an element. So, in SAX, basically the context is only the opened elements, the document is only examined, not cached into memory. Therefore, SAX is useful for examining and filtering large files, for example. SAX application must build its own data structures if the data is needed afterwards. It is a push model (i.e., document's content is pushed to the application). SAX does not belong to any standard body, company or even individual. It can be implemented and used by anyone. SAX development is coordinated by D. Megginson, and its specification can be found at web. [52]

In DOM, the whole document is read into the memory, DOM maps the XML document into an internal tree structure, DOM node tree. In SAX, only the current node is the context, but in DOM, random access to whole XML document is possible [53]. SAX parser and its events can be used in constructing the DOM model, for example [54]. There is no standard to the tree construction mechanism, at least in DOM level 1 specification. Every item in the document is treated in DOM as a node; elements, attributes, comments, PIs and the character data.

DOM is not any platform or programming language specific, it only defines the methods that must be implemented in DOM parsers. Methods in DOM API give applications full-featured access to XML data. The objects in the DOM allow the application to read and navigate, search, modify, add to and delete from a document. With DOM, the manipulation of the whole XML document (e.g., sorting) is possible in the application. The DOM is very memory-intensive, and may not be the best platform for the manipulation of large XML files. With DOM, large documents can exhaust the memory and processor. So, applications using DOM should manage memory to avoid excessive swapping and failure. One solution is to use SAX, when manipulating large documents [53]. Also, DOM provides only generic data structure of the XML data, if more sophisticated data structures and stronger typing are needed, SAX can be used. DOM is a pull model (i.e., active application asks what is in the document). DOM level 1 API is the recommendation maintained by W3C [55]. DOM level 2 comes with increased functionality and it comprises five specifications available at www.w3c.org.

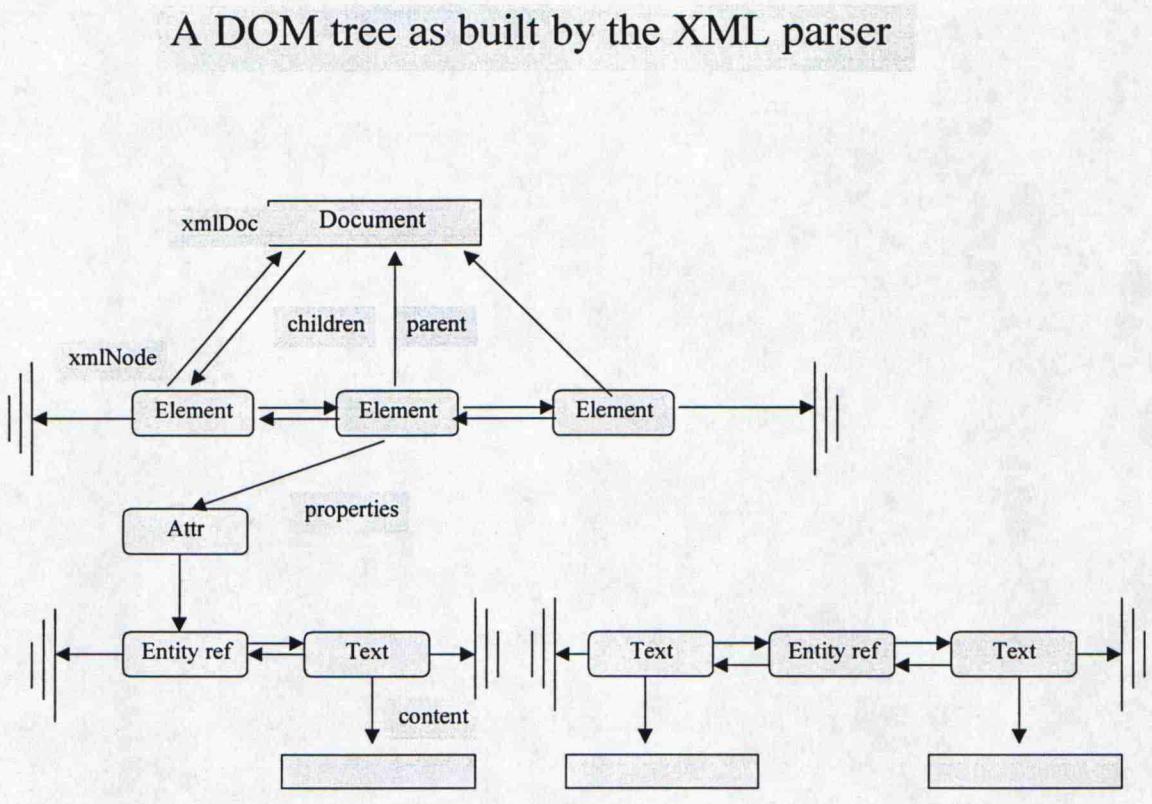


Figure 13. A DOM tree as built by the XML parser [56].

3.6 Transforming XML, XSL [17][19]

Extensible Stylesheet Language (XSL) is one application of XML and it has its own grammar and vocabulary that are understood by applications called XSL transformers. Data in XML document has certain fixed XML format that is understandable by some applications or people only. There can be other applications and people that do not understand or favor the same format. XML document consists of data only and it lacks the presentation. The same data can be presented in different ways. In this kind of situations, XML transformations are needed. XML document can be transformed beforehand or on-demand. With transformation capability, XML documents are re-usable, dynamic and platform and application independent.

XSL transformations are defined with a language for expressing stylesheets that contain the transformation and formatting directions. Designers use XSL stylesheets to express their intentions about how XML document should be restructured and formatted. XSL specification consists of two separate parts; XSL Transformations (XSLT) is a transformation language for transforming XML documents' structure, XSL Formatting Objects (XSL FO) is a vocabulary for specifying formatting semantics of XML document for display. Both are XML languages. The conceptual XSL processing model is depicted in Figure 14. XPath is a third related specification and it is used in XSLT for accessing the parts of XML document. XPath has its own non-XML syntax. XSL can be used, for example, to transform XML document into rendition object (e.g., HTML document), into another XML document or into plain text. The input document must be well-formed XML, but the result document can be anything. The XSLT and XSL FO are next described separately.

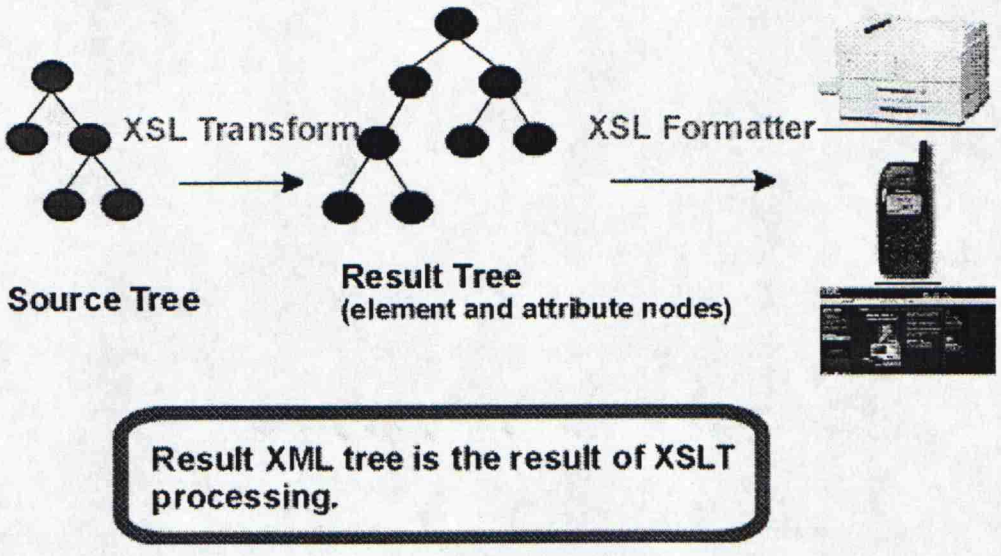


Figure 14. The conceptual XSL processing model [17].

XSLT [57]

XSLT is for transforming XML documents into other types of documents, or into other XML documents. It has its own recommendation outside the XSL recommendation [57]. With XSLT desired elements and attributes can be changed into other objects. XSLT can add completely new elements and attributes or it can remove some of them. It can rearrange, filter and sort the elements, and test and make decisions about, which elements to display. XSL uses XSLT to transform an XML source tree into an XML result tree or an XML source document into an XML result document. In the transformation process, XSLT uses XPath to access specific parts of the document [58].

XSLT processor is the application making the XML transformation. The original XML document and XSL sheet are converted into an internal model, which is a tree-like structure. This tree-structure is independent of any API to access it (e.g., DOM can be used). After the trees are formed, processor goes through the source tree (i.e., the tree from original XML document) node by node from the root node, and looks for a matching template in the style sheet tree. When matching template is found, processor transforms the node according to the rules in the template, results are put into a result tree. At the end, the result tree can be translated into a desired output format with XSL formatter if wanted (XSL FO) or it can be serialized and send to other application or rendition machine.

In presentational uses, presentation markup of some rendition language (e.g., HTML) is authored into XSL sheet and it is wrapped around the XML data in transformation. So, if transforming XML data into HTML presentation, for example, XSL sheet will contain all the needed HTML mark-up. But there are also intermingled references to the data in XML document that is desired to be included in the result document. XML data can be transformed into other pure data format, like into other XML vocabulary and grammar (i.e., DTD).

Data in XML document can be fetched into result document or just referenced. XSLT processor can do some further calculations and processing with XML data also. Or it can rework the XML structure. There are many useful internal functions in XSLT and

in its reference language XPath. With these, almost anything can be done to transform the content. XSL sheets are well-formed XML documents themselves, so anything that can be done with XML document, can be done with XSL sheet. The process of converting and optionally validating XML document into some other format is depicted in the pictures below. In the first picture, the result tree is missing before the result document.

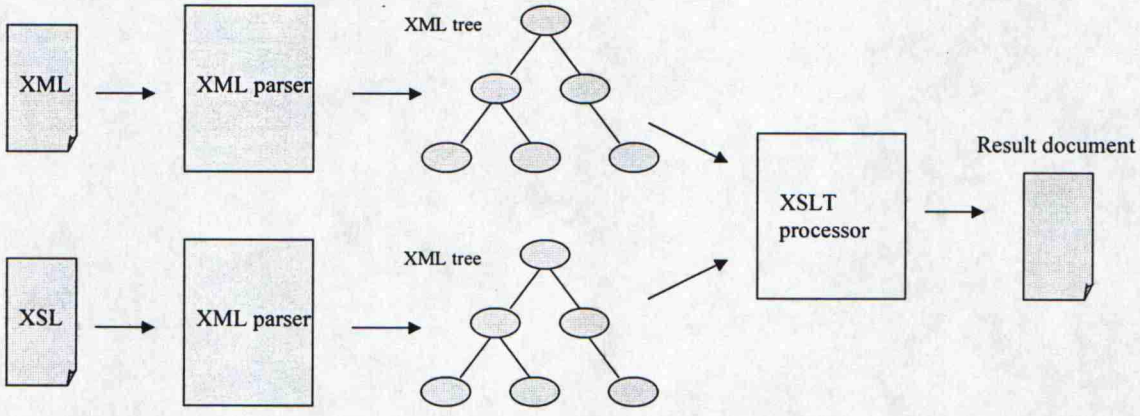


Figure 15. XML/XSLT transformation process.

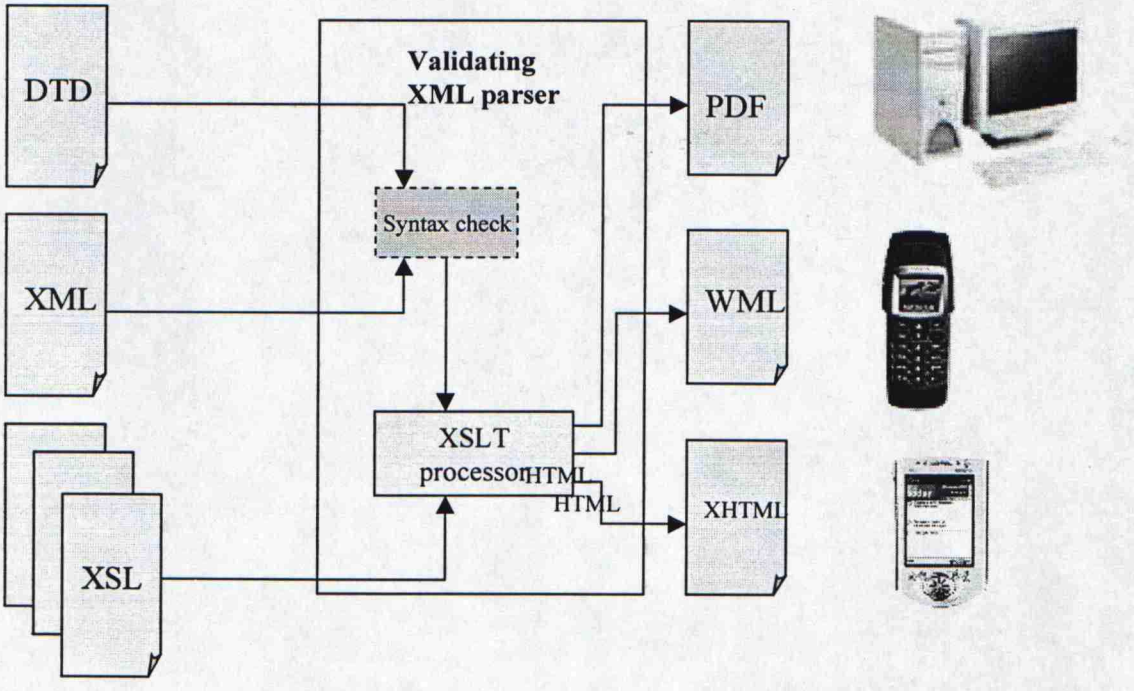


Figure 16. XML/XSLT transformation process with DTD validation.

XPath [59][60]

XML Path Language (i.e., XPath) is an expression language used by XSLT to access or refer to parts of an XML document. XPath patterns are used to identify or select elements and attributes with certain criteria. In patterns, element names are delimited with slashes in a directory like manner. With Xpath the whole document tree can be referenced and queried for transformation purposes. Specific requirements for searched

elements can be set. Boolean operators, mathematical functions and basic string handling methods are part of Xpath. XPath models an XML document as a tree of nodes. There are different types of nodes, including element nodes, attribute nodes and text nodes.

XSL FO [28][59]

XSL Formatting Objects (XSL FOs) vocabulary is defined in the actual XSL specification, it is an XML vocabulary for specifying formatting semantics to XML documents. Rules for displaying different elements are set. XSL FOs has its roots in CSS and Document Style Semantics and Specification Language (DSSSL), but are more sophisticated [10]. If FOs are used, XSL transformer should have both parts of XSL specification implemented. XSLT processor with XSL FO namespace support can output FO documents. These can be viewed, browsed or printed with XSL FO capable processor (formatter), which understands the vocabulary and can render FOs into page layout.

Pagination, titles, tables, lists and links, for example, are among the basic FOs. The XSL FO specification is large and complex with large number of elements and attributes, the complete vocabulary is hard to learn and implement. Because XSL FO viewer is complex software to build, and many potential users are satisfied with CSS, XSL FO is not as popular part of XSL as XSLT. It is suitable to large-scale printing purposes due its sophistication. One implementation of XSL FO is made by Apache XML project, FOP [61] is a software that can convert FOs to PDF, among other formats.

XSL vs. CSS [19][62]

These two stylesheet techniques share the name, but have a little common. With XSL much more can be done than with CSS. Both are declarative languages, but XSL has the XML syntax allowing its tree structure manipulated through DOM like API. CSS is for styling purposes only, while with XSL, more precise with XSLT, the structure of XML document can be also reworked (e.g., sorting). CSS has different format and cannot be parsed with XML parser. CSS can be used with XSL also, XSL sheet can generate CSS, or CSS sheet can be referenced from the result document.

There is lot of discussion and disagreements, whether the XML, CSS and DOM used together is a better way to handle transformations and styling than XML and XSL. If XML document is transformed into some presentation format, like XHTML, its data can be lost in some degree. It is not possible to access its structured data anymore. With CSS and DOM, the XML data can be accessed and manipulated through DOM dynamically in client also, in response to user actions, for example. DOM can be accessed from Javascript, for example. It is an event based programmatic way of doing things. The declarative XSL transformation is made only once, and there is no dynamics afterwards, it declares the transformed document only in relation to the original document. If the result document of XSL transformation is DOM suitable (e.g., XHTML, strict HTML), it can be accessed through DOM, but some data may have been "lost", because data cannot be re-engineered from presentation anymore.

There is although some interoperability problems between different implementations of DOM, the function of the same DOM accessing scripts may differ across implementations in different platforms [10]. XSL is considered more portable solution,

and may be easier to write. Every client does not have DOM support at all. XSL transformation can be done in server.

3.7 Benefits of XML [19]

Below are presented some of the advantages inherent in using XML.

Extensibility

Originally chosen XML structure and vocabulary can be extended later if appropriate. So, the later use of XML is not bound to its original use.

Simple structure and syntax

XML document has simple structure and syntax, for humans and computers. There are only few rules that must be respected in order to write well-formed XML documents. If validity constraints are set, the task becomes a little harder.

It is rather easy to implement XML processors and applications accessing and manipulating the well-formed XML structure. Processors can be kept in moderate sizes capable of being utilized in modest devices also.

Optional validation

With validation, independent groups of people can agree to use a common schema for interchanging data. Applications can use standard schemas to verify that the data received from the outside world is valid. [48]

Validation is optional. It is up to the user to consider, when the validation is needed. The drawback in validation is the overhead in processing, it can be left away in applications, where this overhead is not justified. If there is critical behavior in XML application, validation can be reasonable. It is easier to check the correctness of large documents with validating parser than manually.

Modularity and re-usability

XML is designed to be modular. Content can be included and excluded in any manner. The XML can be aggregated from several entities located in several places. By keeping the shared data portions in one place only, updating is easier and conflicts between data are eliminated. Also disk space is saved.

Maybe more important aspect of modular data modeling is the better control achieved. Data can be portioned after the ownership of the data. Different parts of data may have different update cycles or different update authorities. The responsibility of the data can be put, where it belongs.

Division of labor is possible and several authors can work simultaneously increasing the throughput. When interfaces are clean, the integration is fluent. This topic was discussed also in 2.3.2.

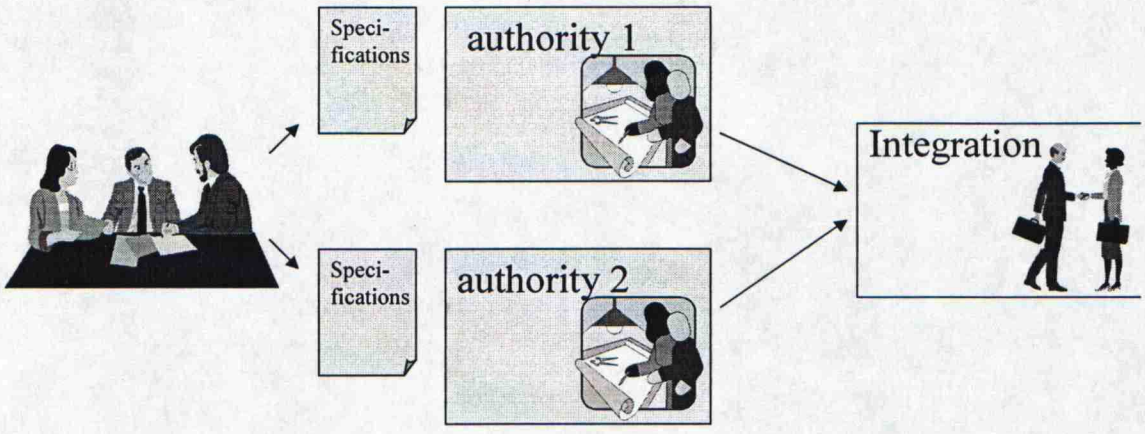


Figure 17. Data portioning after the ownership of information.

Text based

Since XML is text-based, it is human understandable and platform independent.

Hierarchical data

The relationship between data can be described in hierarchical form, like in databases. Together with inclusion mechanism this provides powerful data modeling for all kind of data.

Popular and available

XML has gained popularity in recent years. There is commercial and public software available for almost all purposes. It is safe to invest in XML based solutions, because every vendor nowadays supports XML.

Different audiences come available [13]

With XML, the coupling between the server application and the client is considerably looser [19].

Dynamic adaptation to different presentation needs is hard if content is irreversible engaged with particular presentation (e.g., HTML). There are different end devices and groups of people needing their own presentations. For example, blind people cannot use text-based services. They can have braille machines interpreting digital content to their language. Or if content could be converted into speech, they could use the service [33]. Adaptation of content to different end devices is another application of keeping data and presentation separate.

One audience is the world of networked computers. Computers do not have intuition, they do not either appreciate the presentation at all. Data is all they need. This problem is concrete to Internet's search engines. They must investigate the whole page to find out, whether there is some word. The task is huge and still the outcome is questionable, because there is no way of telling the context of the searched data. In XML, schemas and DTDs can be used to describe the content. HTML's meta tags help the search engines in some degree [63][64], but this problem description serves well as an example.

Machine-to-machine communication between web services has failed to become popular, because applications do not understand each other. In XML, the protocol can be communicated from server application to the client with DTDs or schemas. Web services, for example, in eCommerce can communicate.

3.8 XML applications

3.8.1 XML in communication

Integrator [28]

XML can be used in communicational purposes between different information systems. With XML, the different systems, new or legacy, can be integrated. One of the advantages of XML is that is it text based, so there is no platform dependency in using it. Communicating counterparts can agree on common DTDs or schemas used in their shared XML messages and documents.

The important thing in accelerating the acceptance of XML is the popularization of industry and field specific languages. Electronic commerce, mathematics and some industry trade organizations have created their own XML languages to be deployed between players in these fields. Field specific XML languages can become de facto documenting and communication languages in computer society if good DTDs or Schemas can be designed and distributed in collaborative way. Especially electronic commerce has suffered from the lack of common data formats. There are DTD repositories and efforts of defining industry wide standard DTDs or schemas.

SOAP [28]

XML can be used also to integrate program code. W3C effort in this is the working draft for Simple Object Access Protocol (SOAP). SOAP messages convey method invocations and responses in agreed XML format between different systems. Unlike Common Object Request Broker Architecture (CORBA) and Remote Method Invocation (RMI), SOAP is text based.

CC/PP [65]

The Composite Capability/Preference Profiles is a framework for content negotiation between client and server. The goal of the CC/PP framework is to specify how client devices express their capabilities and preferences to the server that originates content. The origin server uses the "user agent profile" to produce and deliver content appropriate to the client device. In addition to computer-based client devices, particular attention is being paid to other kinds of devices such as mobile phones. Like with industry DTDs and schemas, the repository of device profiles must be organized somehow.

3.8.2 XML in presentation

In the scope of media independent publishing, there are some evolving XML technologies of special interest.

XHTML [28][36]

The w3C's XML version of HTML 4.01 is called XHTML. HTML pages cannot be processed with XML tools, but XHTML pages can because they are well-formed XML. XHTML is considered to be the language of the WWW soon. It has several modules [66], different clients can support different amount of these modules and communicate that to the WWW application. XHTML Basic is the common subset of all XHTML versions and it is going to be a basis for XHTML Mobile Profile (XHTMLMP), the presentation language of WAP 2.0 planned to be launched in 2002 [3]. HTML tidy is the software helping to convert HTML to XHTML [29]. HTML syntax is not anymore respected and platform independence of HTML pages has suffered from this. XHTML is kind of a new attempt to keep pages and browser implementations standardized and also moderate in size.

WML [38]

Wireless Markup Language, presentation language of WAP 1.x specifications, specified by WAP Forum. WML is reminiscent with HTML, but its is meant for tiny screens in mobile phones. WML will be replaced in WAP 2.0 by XHTMLMP, but the legacy content can be transformed to it with XSLT [3]. WML is different from popular HTML and wide community of HTML authors cannot capitalize enough on their prior knowledge of HTML. In author's opinion, this can be one reason why WAP has not succeeded so well.

SMIL [67]

Synchronized Multimedia Integration Language is W3C XML language for multimedia presentation. It enables convenient authoring of interactive audiovisual presentations. SMIL can be used for multimedia presentations, where streaming audio and video must be integrated with static images and text, for example. X-Smiles is an open source Java based XML browser with SMIL support. It is developed at the Helsinki University of Technology [68][69].

XForms

XForms is W3C working draft for making interactive service deployment more feasible by making forms platform independent. The aim of XForms is to separate user interface (presentation) from the data and logic (purpose) allowing the same form to be completed by users on a computer desktop, PDAs or mobile phone, for example. [70][71]

3.9 Example solutions for media-independency

The problem of serving several mark-up languages and their different versions was discussed in 2. There are different technical approaches to achieve media-independence. Few basic principles are introduced next for interactive services, but the same principles apply to static content also.

Unscalable solutions

In unscalable solutions, linear increase in handwork is not avoided, when amount of supported media types increases. An advantage is that there are no compromises and restrictions in final presentations and the results are predictable.

In XML/XSL solution, own XSL sheets must be made to every media type in every service, making this solution in this regard unscalable. The authoring and managing of large amount of XSL sheets cannot be avoided. XML and XSL are the extra skills that must be commanded in addition to proprietary presentation languages of every media type. XSL has programmatic features and it is not as easy to learn by graphic designer as HTML or WML is, for example. XSL transformations are memory and processor intensive, caching and sophisticated use of memory and parsing technologies is needed especially if large documents are transformed. Advantage with XML/XSL is that content, was it static or dynamic, is always in pure data format and separated from the presentation in standard way. There are no limitations, where content can be used or accessed from. XSL transformation is also location independent. If it can be done in client or some XSL transformation proxy, scalability in processing and memory use can be achieved. Server application is saved from processing, but it must still give link to XSL sheet.

In template technique, templates are documents in some media type's proprietary presentation language (e.g., in HTML) having placeholders for dynamic values. Application populates these placeholders with dynamic values in runtime and returns complete documents. This approach is not scalable either (i.e., large amount of templates), but no additional XSL knowledge is needed. Disadvantage is that if pure data format is needed, one additional template for XML, for example, is needed.

In XML/XSL and template approach, same URL can be used and communicated for users in all media types. Third alternative is to make separate version of application to different URL for every media type. But if every media type have its own URLs, the marketing communication, for example, will come inconvenient.

Scalable solutions

In scalable solutions, some automation is implemented that tries to adapt the original content to different media. Degrees of automation and configuration vary.

Full automation can be achieved with ideal transcoding software that converts original content into some other format(s). In some WAP gateways, for example, are embedded HTML-to-WML transcoders. Special rules are applied to the original content in transcoding. According to these rules content's markup and data is transcoded as authentic as possible to a result content (i.e., original document's elements are transcoded to analogous or nearest comparable elements in result document). The meaning of original element may change if clear one-to-one mapping do not exist in result markup, some elements may be dropped away. [25]

Applying transcoding rules can lead to diverse quality in output anyway. The transformation depends on the original document, if it has syntactic violations, the transcoding may fail. It should be manually checked if new content is transcoded successfully. There is not one-to-one mapping between all markup languages, so some compromises may have to be made and still the result is not predictable. Advantages in converting content dynamically from the original content are that only one base of

documents must be managed and all converted content is automatically up-to-date. This solution is scalable, if the manual checking of process is not taken into account. [25]

Original content may be usable in some other device if some parts could be clipped away from it. There is software with embedded visual tools, where original content is fetched and clipped portions are marked graphically. Original pages stay untouchable. [72]

One solution is to put intelligence into original documents also. Markup can be put into the original content as directives to different media types. This is less automatic than the former, but more predictable. Tailored application reads this markup and according to its internal rules processes the directives and outputs desired format. In the most predictable and laborious model, one document has explicit content inside custom tags for every media type. Advantage is, that content for every media type can be kept in one file which eases management. There is commercial software for this with integrated emulators for different media as well as authoring tool support [73]. Scalability in this approach is achieved in management, not so much in authoring effort.

Alternative to authoring explicit content for every media type into the same document, is using media type independent custom tags having general meaning [74]. Original document consists of custom elements and attributes that are converted into result format according to media type specific rules. For example, custom element for picture could be <picture source=URL/> and it is converted to in HTML to PC and to <a href=URL /> in HTML for PDA device. This idea can be implemented, for example, with XML/XSL in a scalable manner, only one XSL sheet per media type. The content can be initially produced into pure data format in XML. Then XML must be first transformed into this intermediate form (i.e., with custom tags) and finally with second transformation to the media type specific result format.

In the two latter models original content is customized into a non-standard format which can cause harm if deciding to return to some original formats. Common to all solutions is that media type must be first identified. Media types must be well known beforehand, nothing happens fully automated. When new media type comes into market, rules for its conversion or transcoding must be designed.

Transcodings and custom tag manipulations can be implemented with XSLT transformations or with customized SAX or DOM applications.

3.10 Supporting software

In media independent publishing, some special software modules are relevant. Web publishing framework helps in service creation and providing. If XML documents are requested from or responded to the network then also the network software is needed. XML content can be generated or fetched from database.

Web publishing framework [27]

Web publishing framework, like Cocoon, is the software that uses XML parser and XSL transformer, or XSLT processor, in its inner implementation. It is an example of the XML/XSL model described recently. It takes care of two-phase process of XML parsing and XSL transformation in a transparent way. It hides some needed functionality behind it. Like formatting the output, setting the appropriate mime-type for the served

content, identifying the requesting client type and associating it with suitable XSL sheet. It may help in editing XML and XSL documents.

XSL transformations are memory and processor intensive. Web publishing frameworks are trying to cope with scalability problem as well as possible. They possibly cache XSL transformers in their internal tree-structures in runtime memory or file system to make transformation process faster. Or they try to optimize the transformations with SAX and DOM to achieve better performance.

Also, the configuration of client identification and presentation specific attributes is made easy in web publishing frameworks. XML content can be aggregated and produced from different sources and it can be manipulated and formatted. The whole web publishing framework, consists mainly of processing XML through APIs and making the management of documents and the whole distributed web site more convenient.

Web Server

Web server is the link between Internet and standalone computer. It responses to the requests coming from network. Web server serves the addressed content from static files or from dynamic applications, like Java Servlets.

Java Servlets

Java Servlet technology is an easy way to connect applications through web server to the Internet. Web server connects to the specific Servlet application through Servlet engine container that is attached to the web server by specific module. Servlet engine can hold HTTP sessions and arranges HTTP requests and responses from the web server to its Servlet applications and vice versa.

Databases and XML

Database connection is sometimes needed, when XML data is parsed from database data. In latest versions of databases, direct XML insertion and fetching is possible without any external XML parsing.

Content cleaner

Non-standard HTML may have to be cleaned into standard HTML, or into XHTML. Some end devices cannot cope with non-standard HTML. If HTML is wanted to be processed with XML tools, XHTML conversion is needed.

Public software

XML parser (e.g., Apache Xerces), XSLT processor (e.g., Apache Xalan), XSL FO processor (e.g., Apache XML FOP), database (e.g., MySQL), content cleaner (HTML Tidy), web server (e.g., Apache Web Server) and Servlet engine (e.g., Apache Tomcat) are all possible to get free as an open source or public software. Of course the operating system is needed also but it as well is freely available today (e.g., Linux). The software mentioned in parentheses are examples of popular free software. The programming language for tying these parts together is needed. If you want to manage without programming then some web publishing framework like Apache Cocoon is good choice

in certain usage preferences. All mentioned software exists for Java programming language.

3.11 Relevant technical out of scope issues

All relevant issues in service development must be taken into account in real life, but in this thesis only the presentational features are in scope. Some major out of scope issues are introduced here briefly.

User personalization

There are at least three different personalization contexts in service development. The one is based on end device profile and two other ones on user preferences. In latter, the general user settings and preferences can be considered distinct from personalization between different use cases or usage contexts like outdoors, office hours or car. Different combinations of general settings can be active in different contexts. Nokia's director Heikki Huomo has said that in future's ubiquitous telecommunication the "myDomain" is roaming between different contexts. The contexts have to be configured, accessed and taken into account by service providers and other communicating parties. The configuration can contain the rules of customer's visibility to and awareness of other parties in ubiquitous communication society. The big problem is how to make the decision of context switch automatic. Example of non-automatic context switch is when changing profile in Nokia's mobile phones. [75]

If user related preferences are taken into account they usually overrule the devices default preferences. One question in user personalization is, where to store all user-related data. Some kind of profile is needed and it must be communicated to the content origin server. User data can be on/off like data telling whether images should be sent or not [65]. The other extreme can be that every user has her own XSL sheet for each service to make presentation just as she wants it, for example. XML/XSL techniques fit fine for this purpose as well, but there will be much more data to manage, store and communicate. Scalability can be a problem.

Communication path dependencies

Different end devices are usually behind different communication link. Among the quality parameters of the communication link are at least speed, throughput, security, continuity, error-rate, etc. Some services cannot function well if some quality parameter is weak. For example, if you drive into the tunnel and wireless connection disappears some harm may occur in your web based electronic commerce session. Something can be done in client and server software to eliminate the poor communication link quality. One very concrete issue is how much data can be sent when throughput and speed of the path varies. Still the scope is only how the data appears on the screen of the end user.

Continuous media

Continuous media like voice and video are their own science. They are too big issues to be handled in this thesis. With text and images the synchronization and timing do not have to be considered at all. Multimedia sets many requirements on the operating systems, synchronization, scheduling and context switching between different media types are important [76]. Streaming media is the context of W3C SMIL language.

Today WWW and WAP browsers support only text and images so it is at that sense also not relevant to consider continuous media.

4 The service development environment

4.1 Foreword

In this chapter, the existing service development environment is introduced. The overall architecture and pieces of software and technologies in it are presented. The three-tier architecture is reviewed and linked to our own architecture.

Programming level policies and practices used in the development process are described. The service development process is depicted.

The rewrite of the existing Addrbook service is part of the own contribution of this thesis. So, the existing old Addrbook service is introduced at the end of this chapter in order to make the subsequent chapter of rewriting the service easier to follow.

Special attention is paid into presentation issues. They are presented occasionally with more accuracy than others, because they are the focus of this thesis.

4.2 The general environment

General service-providing environment is composed of three different gateways and several backend servers for different purposes. In the following, they are introduced in the same order as the customer's service request is handled.

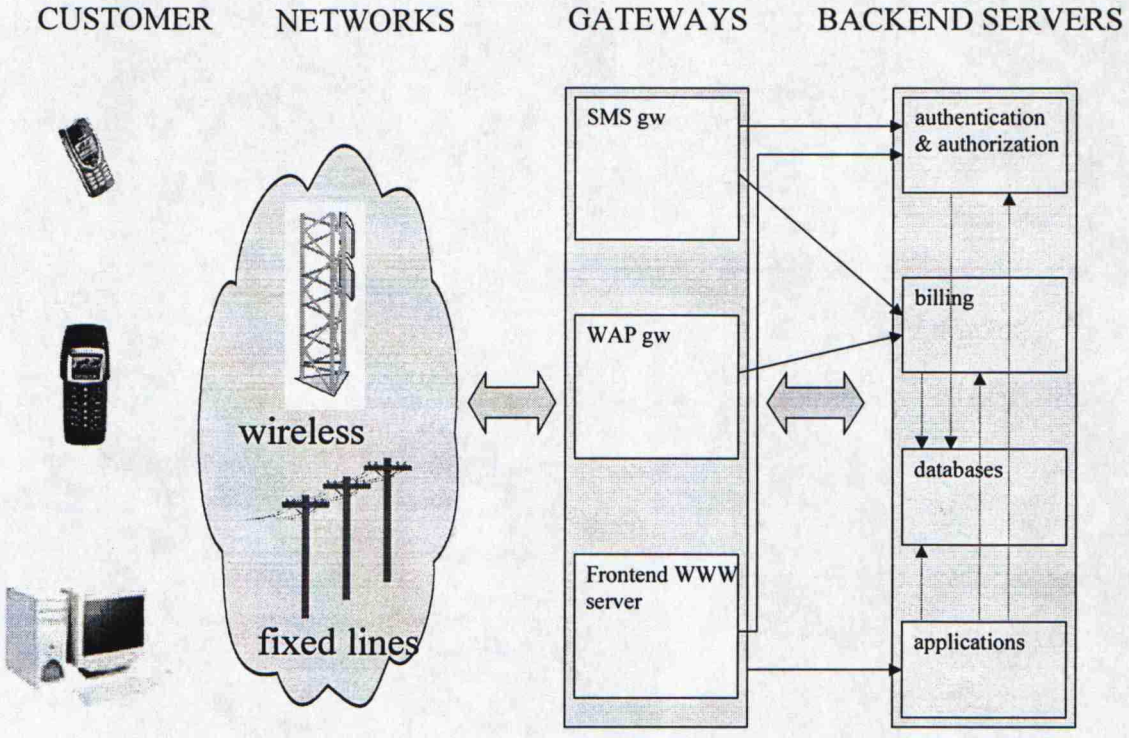


Figure 18. Service network.

4.2.1 Gateways

Software that connects different information systems together is generally called gateway. Gateway must know addresses and data transfer protocols of the systems it connects. It enables the communication between clients in different end systems.

Because the service network has three different end communication systems, it has three different gateways. These gateways connect the mobile and desktop customers to the service network. The most visible gateway is the frontend WWW server of the WWW portal (at the moment, it's URL is www.sonera.net). The other ones are for SMS and WAP customers. Gateways are sometimes good places to put some common functionality like billing, authentication and authorization. In WAP gateway, there is an automatic transcoder trying to transcode from HTML to WML.

Frontend WWW server

Frontend WWW server is a link connecting the public Internet (WWW) customers to the service network. Frontend WWW server is the only place in the service network that is made visible to the public Internet. It takes care of some common routines like authentication, authorization, encrypting/decrypting, logging and statistics. It knows the internal service network's addresses so that it can redirect the HTTP requests from Internet to the backend servers.

Content Gateway (SMS)

Gateway called Content Gateway (CGW) connects the SMS customers to the service network. CGW authenticates and authorizes customer and takes care of the session related invoicing. It makes protocol conversion from SMS Center's (SMSC) output protocol to HTTP and redirects the SMS requests into the service network. [77]

WAP gateway

WAP gateway connects the WAP customers to the service network. WAP gateway authenticates and authorizes the customer, makes protocol conversion from WTP to HTTP and redirects the requests to the service network. [25]

4.2.2 Backend servers

Backend servers are the networked servers of the service network that are not visible to the public Internet. Some of them are in test use and some in dedicated production use. The three gateways know the addresses of backend services and can employ their services and redirect the incoming customer requests to them. In the following, the different kinds of backend servers in use at the moment are presented.

Authorization server

Authorization is needed in order to bar certain subscribers from using certain service classes they are not allowed to use. Authentication is used in mobile network and WWW frontend. Regulator has given strict instructions that must be obeyed in payable services [78]. Customer has rights to impose bar classes for his subscriptions and they must be obeyed. Gateways and in some cases also applications call authorization

services and react to them. Services in authorization server have HTTP interface. In some of the services, there is subscriber-related data stored between sessions. When customer resigns his subscription, all this data must be deleted. If this is not taken care of, the next subscriber who gets the resigned subscription number after certain quarantine period, will have the data belonging to earlier subscribers. After resignation, authorization server requests these kinds of services and triggers the deletion of their related subscriber data.

Application servers

Some of the servers are dedicated to end user applications. They have HTTP interface implemented with Java Servlets. In addition, some older cgi-bin applications are still in use. Application servers are called with HTTP through gateways. Also, the HTML and WML pages (i.e., templates) are on the same servers as applications using them.

Database servers

Subscriber and service related data is stored in database, where it can be fetched, updated and removed. Databases are maintained in dedicated servers. The applications in application servers manage the service related database operations through Java Database Connectivity (JDBC). JDBC connects the Java runtime environment conveniently to the proprietary Database Management System (DBMS) through its JDBC driver. Other programming languages have also their own database connectivity technologies.

Billing server

Depending on the service and access type the billing function is carried out in different places. Applications or gateways call billing server after successfully served customer requests. Billing server contains ticket databases that are later used in invoicing and revenue sharing among partners using the network.

4.3 Three tier architecture and the system

The service development system employs three-tier philosophy in some degree. Three-tier architecture is composed of three logical layers: data, logic and presentation. In good architecture, these layers are insulated from each other in as many aspects as possible. This insulation comes in practice with clean and standard interfaces. Interfaces are needed at least when connecting, disconnecting, moving and scaling these layers smoothly in changing environment.

4.3.1 Data

Data is maintained in some data storage like files or databases. Managing large amount of interrelated data is very difficult task. Databases hide the lowest data layer nicely. DBMSs are so sophisticated and established software that it is not very wise to even try to handle its responsibilities by own methods like managing data in files.

The scalability of the service depends on at least two things. Whether the processor capacity and runtime memory of the logic part is scalable and whether the data storage

and management layer is scalable. With databases the scalability and management of the data layer is under control.

Other unquestionable advantage achieved with databases is that data in it can be utilized in standard way from every application, re-usability is achieved. If this critical layer was not behind the clear and standard interface like Simple Query Language (SQL), it was hard to tell other applications how to access the data. SQL is the de facto universal database language that hides the complexity of DBMS. Standard interface to managing data is missing, when, for example, plain files are used.

Location of databases is also insulated. They can be utilized over the network as well as from local file system.

4.3.2 Logic

The middle tier or the logic layer is more abstract notation and it is implemented with programming languages in applications or with scripting languages in markup pages, for example. Application logic handles the business cases in it and can call other logic in other or underlying systems. In good architecture, the interfaces between different logic parts are clean and standardized. Applications are written in Java in our systems. Logic in Java Servlets is accessible through HTTP interface, which is standard and easy to use. Servlets can use Java classes in the same file system and they can employ other networked logic and so do what they have to do. Logic that has no network interface like socket or Servlet interface must be installed in every filesystem and set in the classpath.

In the system, logic layer also connects lower data tier to the upper presentation tier. It fetches and manages database data and feeds it into the presentation layer. The data-logic interface is standard JDBC/SQL and it functions well. The logic-presentation interface is self-developed template technology introduced in the next section.

In ideal architecture logic components can be utilized from different presentation layers and also in other direction. There are some problems in logic-presentation interface that is based on template use. The logic is designed and optimized to special end device and mark-up language features only. Logic should be first and foremost designed flexible to function in desired use-cases, not only in certain end devices. In the system, the insulation and interfaces between logic and presentation could be improved. Consequence is otherwise lack of flexibility and re-usability in new end devices.

Another problem is that these logic components, like Servlets, cannot communicate with each other at the moment. Their responses are in HTML, or in other presentational markup, that mixes the data irreversible into the presentation. The pure data needed in other logic components is impossible to re-engineer from these markups.

Clustered servers make the logic layer scalable independent of the other layers. Load balancing and clustering together makes sure that quality-of-service can be maintained at tolerable level.

4.3.3 Presentation

At presentation layer, presentation language like HTML or WML presents the service or content to the customer. In SMS, the text messages are the only presentation layer related thing.

The clean division into three-tier architecture was broken little in above mentioned logic-presentation interface with using presentational markups. The division breaks also inside the presentation layer itself. There is logic also in the presentation layer. Logic is present in the form of scripting languages like Javascript intermingled in mark-up language.

The advantage of using scripting in presentation level in telecommunication services is that it can be processed locally in client's processor. Scripts save also from unnecessary telecommunication traffic in cases, where customer input must be checked and validated. It is justified to check customers input locally in client rather than send it costly into remote processor for validation. In this sense, this break of rule is justified.

The approach, where logic is implemented with scripting languages in presentation layer (e.g., Javascript), has its disadvantages also. If it is later noticed that every client do not anymore have the supposed scripting features available, the logic must be implemented in server-side logic layer.

Presentation components may be harder to make re-usable and this may be possible only with compromises and restrictive one-size-fits-all approach.

4.4 Policies and practices in the service deployment

Some policies and practices have formed in the development process during the years and they are evolving all the time. Some of the policies are encouraged to follow when developing services. Doing some things in standard way makes development, maintenance and also further development easier.

4.4.1 Guidelines

There is a set of service development guidelines passed to the subcontractors. They find their way automatically into the consciousness and routines of our own developers as well. Some procedures must be implemented in predefined way in order to make things work together. Authorization and billing are among these procedures. They are the critical functions that can be totally ignored in services if they are not communicated systematically in early phase. In practice, there are certain interfaces in the system that must be called or implemented. When doing the things according to guidelines all needed functionality is achieved, and it is achieved without extra overhead. [79][80]

4.4.2 Common classes

Some common routines are maintained in common classes. They are recommended to use in guidelines with documented features and application programming interfaces. Subcontractors are provided with these common classes. Use of these classes is the safe and convenient way to handle common things like logging, database connection pooling, SMS sending, authorization and billing, for example. User interface is also among these.

4.4.3 User interfaces

The user interface or presentation layer is kept in file system as proprietary templates [81]. There are templates for HTML and WML. SMS messages have no templates,

because they are so simple. Templates are HTML and WML pages that include placeholders for application data. Applications load templates and replace placeholders with actual data fetched from database or elsewhere. Needed functionality in template handling is written in special common Template class. Guidelines contain recommendations of WML design, because there are known inconsistencies in WML between different phones.

```
<INPUT TYPE="HIDDEN" NAME="ID" VALUE="$(id)">

<INPUT TYPE="HIDDEN" NAME="FIRSTNAME" VALUE="$(firsname)">

<INPUT TYPE="HIDDEN" NAME="LASTNAME" VALUE="$(lastname)">
```

Figure 19. Example of a template.

Applications load templates and put runtime values into placeholders with the help of Template Java class. In the example above, three real values were put into template placeholders by application. $(id) could be converted into 10000123, $(firstname) into "Pekka" and $(lastname) into "Virtanen".

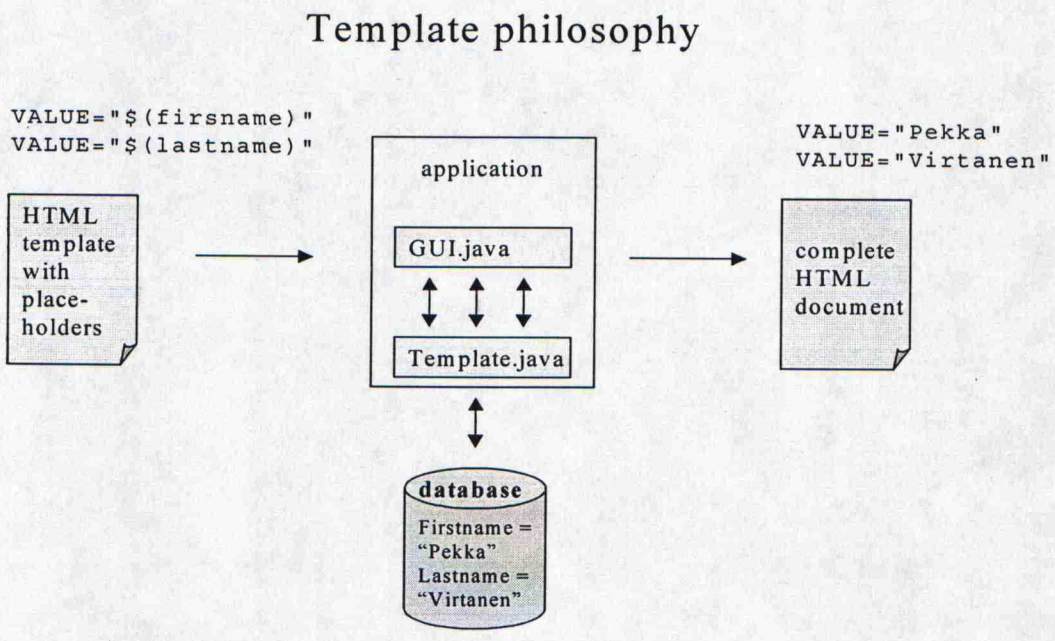


Figure 20. Depiction of the template philosophy.

Other alternative is to hardcode the presentation language in the code. Unfortunately, that leads to programming code, which is very difficult to understand and trace. Another disadvantage comes up, when the presentation layer is changed and the code must be rewritten and recompiled. WWW designers or other non-technical people are not familiar with this kind of procedures. Unfortunately, there are also some older services that have certain amount of hardcoded presentation. Specific recommendations to designing user interfaces are in [82].

4.4.4 Stylesheets

CSS is used in the WWW portal [82]. By using it, the style and look-and-feel can be kept separate in its own file. Changes made in this file reflect to the whole portal immediately. It is impossible to do things by hand in every HTML page. Disadvantage is that some client may not support CSS.

4.4.5 The WWW portal practice with frames

One special thing called frames is worth mentioning. HTML framesets are employed in the portal. In frameset technology, the browser view is composed of more than one page. These pages are kept in their own URLs, where browser gets them and composes the whole view. Only the changing frames must be loaded during the navigation in the site using frames. This practice and the advantages of it are described in the following.

The main or middle frames in services are dynamically generated from the application. They are composed of only the few essential things that build the user interface [81]. These pages must be changed seldom or never because they do not contain any extra communication or brand related issues. Changes in these pages must be made in the applications (if hardcoded) or their templates. Applications can not process the templates if their structure is broken by mistake. Using this frame approach, the programmers are not bothered constantly with fault detection and correction issues, because the middle frame coming from application is insulated from all others.

The surrounding left and upper frames in the services are fetched from separate files. Our business people can independently maneuver the content in all these pages. These pages are changed often. If they want to change the look-and-feel in the whole portal, it is done in CSS files only and the changes are reflected also into application pages automatically.

Keeping portal wide menus and other common features in one place, the changes must be made into this one place only and the data is safely coherent. Less file capacity is also needed as side effect. Common pages can be conveniently used from every frameset view.

In practice only the middle frame returned from the application is changed between the transactions in typical service. If all data on the screen, especially many large images in surrounding frames, must be reloaded after every transaction, it would be slower to use our services. The loading of the middle frame is light operation because it does not contain so much data.

But there are unfortunately also disadvantages in using frames. All clients do not support them. This problem is hard to bypass. One solution is to give links to every individual frame belonging to the frameset. By doing this, the look-and-feel and user friendliness can be lost.

4.4.6 Serving different end devices

There are two ways to serve three different end devices. The first one is to copy the application and make little end device related conversions and change the templates, for example, from HTML to WML. Another way is to keep the logic in one Servlet and

put conditions on which end device to serve. End device must be identified and correct templates must be chosen for presentation.

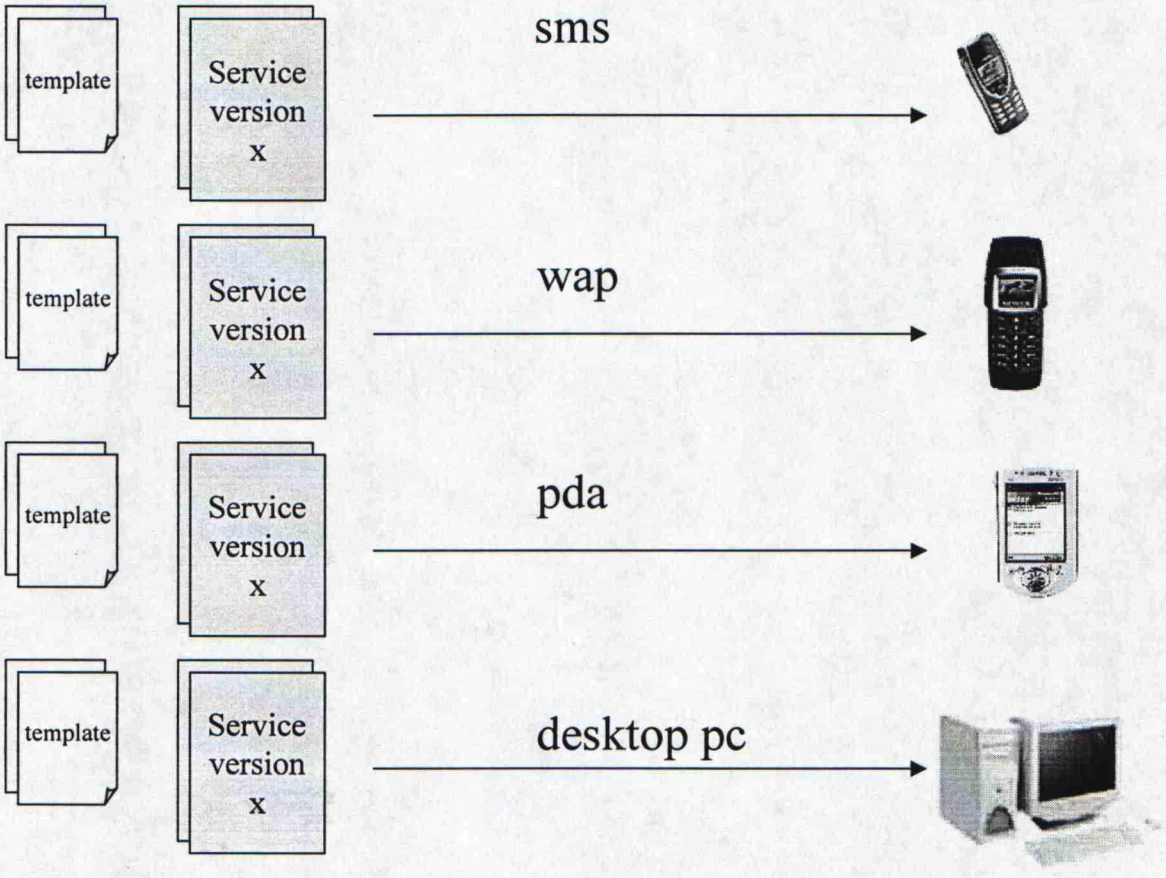


Figure 21. Serving different end devices with appropriate templates.

Some times the whole logic must be considered differently with different end devices. In WAP, for example, the amount of data that can be sent in one unit is limited. When feeding data into WAP phones it must be split in smaller units at the server. Logic must keep in memory what piece of the whole data was sent recently and what will be the next piece. In this case, the mere copying and tuning is not enough.

The latter case belongs to the problematic described in the logic portion of the three-tier-architecture in section 4.3.2. More precise, it arises from the difficulties of designing logic-presentation interface and from the difficulties of designing re-usable and flexible logic.

4.5 Process description

The phases in our development process adhere to the traditional programming process model.

reasonable to go further with the service, requirements may be further elaborated and the service is done in-house or at partners or subcontractors under the supervision of customer.

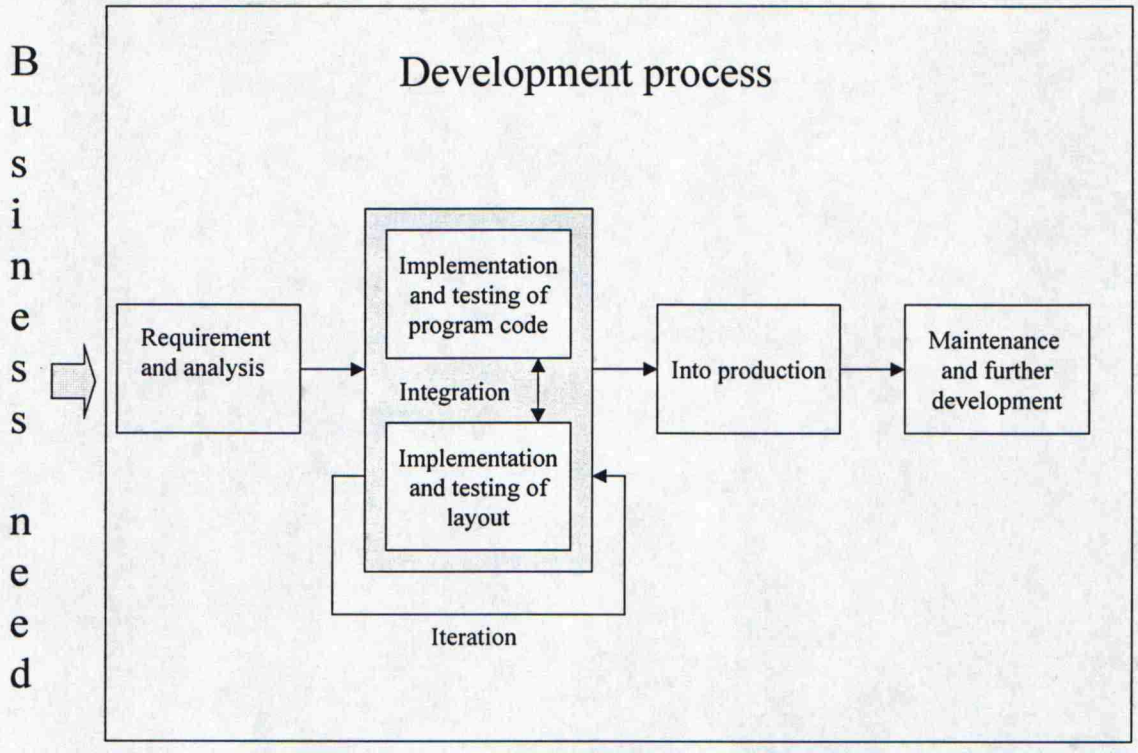


Figure 22. Diagram of the development process.

4.5.2 Implementation and testing

Implementation is done in-house or by subcontractor. Services are implemented in test environment that is equivalent with the production environment. Subcontractors develop software in accordance to guidelines [79] and requirements in their own premises and periodically come to customer's premises to test its integration. When they are finished, the whole software is tested against requirements and usually after some iteration accepted.

The implementation phase can be separated into two distinct tasks: application programming and layout design. Different people can accomplish these tasks and there is a WWW design team, which can be used. They are in charge of the look-and-feel of the branded portal, also.

Guidelines into template philosophy are applicable to both parties. In addition, there is one common class available that implements the interface to templates.

When using WWW design team, they make templates with placeholders in accordance to guidelines to make them work with applications. Or they just make the HTML pages and give pages to programmers, who convert them into templates by inserting the placeholders. WWW team does not make WML pages at the moment, programmers or subcontractor do them.

The subcontractor usually makes the layout itself, because the integration can cause too much overhead if layout and application are implemented in separate units.

The incremental testing of service requires some kind of user interface from the beginning. Application programmers make at least it by themselves, usually.

4.5.3 Into production

Sometimes service goes first through a trial period, where it is put into the pilot use among chosen customers or in the own organization. At the end, the service is launched into the production with the simultaneous business units marketing efforts. The software is moved from the test environment to the final place.

4.5.4 Maintenance and further development

Maintenance and further development can be done in-house or at subcontractor. Applications developed by subcontractors are inclined to leave in their control more naturally. The maintenance and further development task has lot to do with the agreements made with the subcontractor, whether customer get the possession of the source code or not.

The daily monitoring of services is separate function from these and it is done in-house.

4.6 Old Addrbook service

The development of the multi-channel prototype of the Addrbook service is described in 6. The existing Addrbook service is the base for the multi-channel version, and it is introduced here.

4.6.1 What it is?

The Addrbook service is an electronic address book that helps mobile subscriber to keep her contact data of her friends up to date. Customer can utilize her contacts in daily situations and in some other WWW portal services also. These two ways of exploiting contact data are equally important. Customer cannot access other customers' contacts at the moment.

The Addrbook extends the memory and features of mobile phone. It enables unlimited storage of featured contact data in service providers data systems. In addition to the storage of personal names and phone numbers in the typical mobile phone, also the postal addresses and email addresses can be stored in contacts, for example.

4.6.2 Features and functions

One Addrbook contact item is a collection of the following personal properties: first name, last name, alias, phone number, fax number, electronic mail address, street address and postal address. The last name is the only mandatory field.

Contacts can be fetched with SMS, WAP and WWW. New contacts can be added in WAP and WWW only. As well, in WAP and WWW only, when the contact is found, it can be edited or removed. The set of personal properties belonging to a contact can be fetched with a keyword. Keyword can be first name, last name, first and last name together, phone number or alias. Incomplete keyword can be also given if the whole

word is not correctly remembered, for example. All matching contacts are returned for a keyword. In WWW, the list of contacts in which the last or first name begins with the given character can be fetched at once, but this is just the special case belonging to the incomplete keyword given.

In WWW portal, contact data can be exploited from the other services in portal that may need person related data, like e-mail addresses or phone numbers, when sending e-mail or SMS. This functionality is implemented completely in WWW clients (i.e., in HTML pages with scripting language). There is no supporting server side functionality for this and so, in WAP portal, this kind of co-operation lacks at the moment because needed scripting language support is not currently available in every WAP client.

4.6.3 Technical background

There are two parallel software in Addrbook service. One is for WWW and the other one is for SMS and WAP together. There was originally only the WWW version, the SMS/WAP version came later. One reason for making WAP/SMS version separate from WWW was the difference in the logic between WWW and WAP/SMS. In WWW, there is no upper limit to the amount of sent data. Meanwhile in WAP, the limit is maximum WML deck size. Another reason for the distinct WAP/SMS version was that in WWW version, there was too much hardcoded HTML in the program code. Both are Java Servlets running in the Jserv Servlet engine under Apache Web Server. Contact data is stored in Oracle database and JDBC is used in Servlets to handle all database operations. Templates are used in making HTML and WML output.

4.6.4 User interfaces

The following picture examples depict best the functionality in WWW, SMS and WAP.

WWW

The start view in WWW user interface is introduced first. One can decide whether to fetch contacts or create new ones.

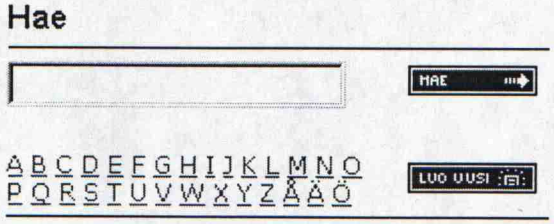


Figure 23. WWW user interface 1.

If the contact data of Teemu Teekkari is wanted, one can enter the keyword "Teemu", for example, and select "Hae" ("Fetch"):

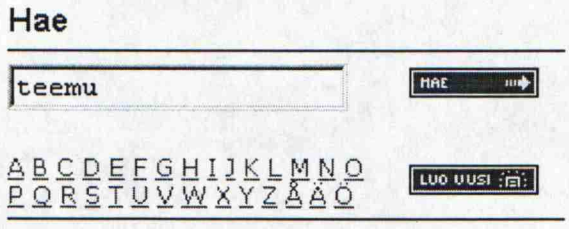


Figure 24. WWW user interface 2.

The results are listed, this time only one matching contact was found:

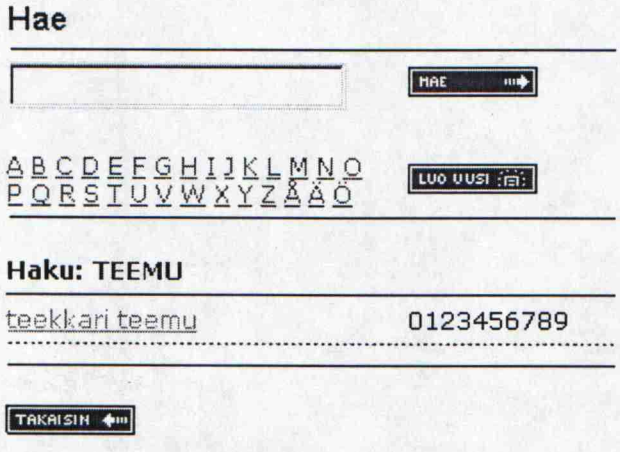


Figure 25. WWW user interface 3.

By selecting "Teemu Teekkari" the whole contact is fetched.

Figure 26. WWW user interface 4.

If one wants to edit some fields or remove the whole contact one have to go further by selecting "muokkaa" ("edit").

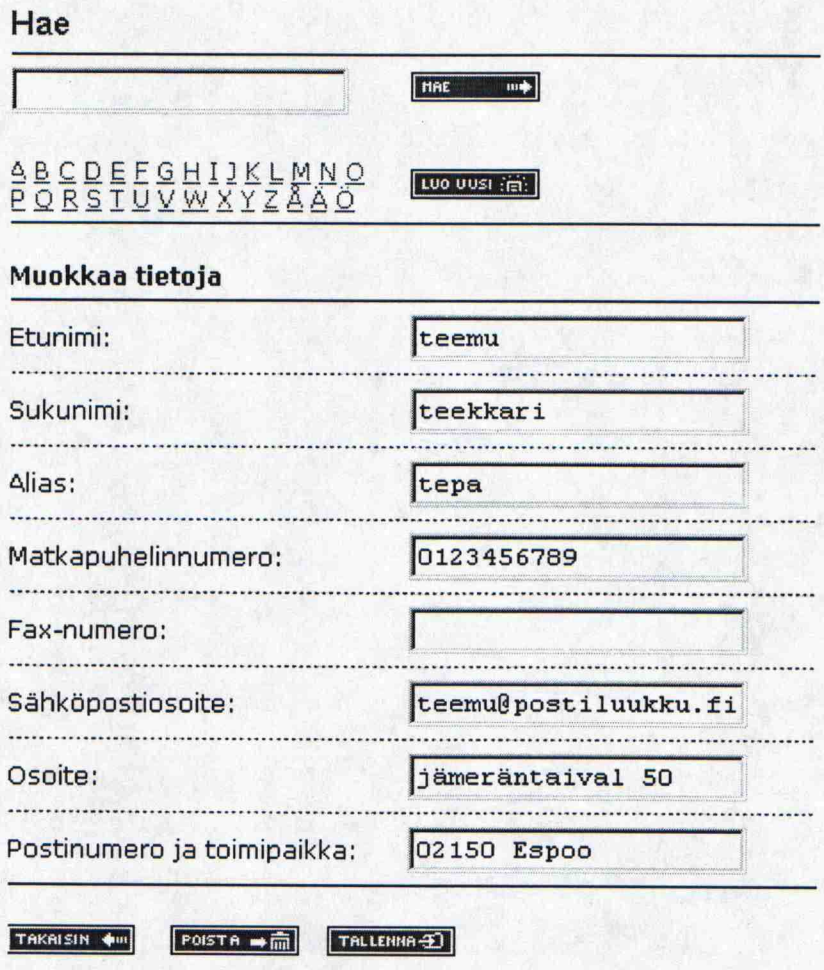


Figure 27. WWW user interface 5.

The creation of new contact data is like editing empty contact and it is straightforward. Identification number for a contact is generated when the contact is created. This id is a key for a contact in database table.

WWW popups

When contact data is exploited from other WWW portal services, the user interfaces look quite same as the previous ones, but they are rendered into separate "popup" window. Contact data is fetched into the popup window and its values can be moved with Javascript to the fields of original page. Popup windowing is a built-in feature in PC HTML browsers, where new windows into the same browser context can be popped up from the page in actual HTML browser by pushing the button or selecting the link. Popup window pops up in a separate browser window with the desired URL content. There are no framesets in these popup windows, like in normal browser windows in the portal. The policy of using frames in portal was discussed in 4.4.5. The content in popup windows is little different from middle frames in framesets. All the pictures of WWW user interfaces above were just these middle frames. There are own templates in application for popup use cases, the upper parts with title "Kontaktit" and instructions below it are different from templates used in framesets. For example, the corresponding popup view to Figure 25. is in Figure 28.

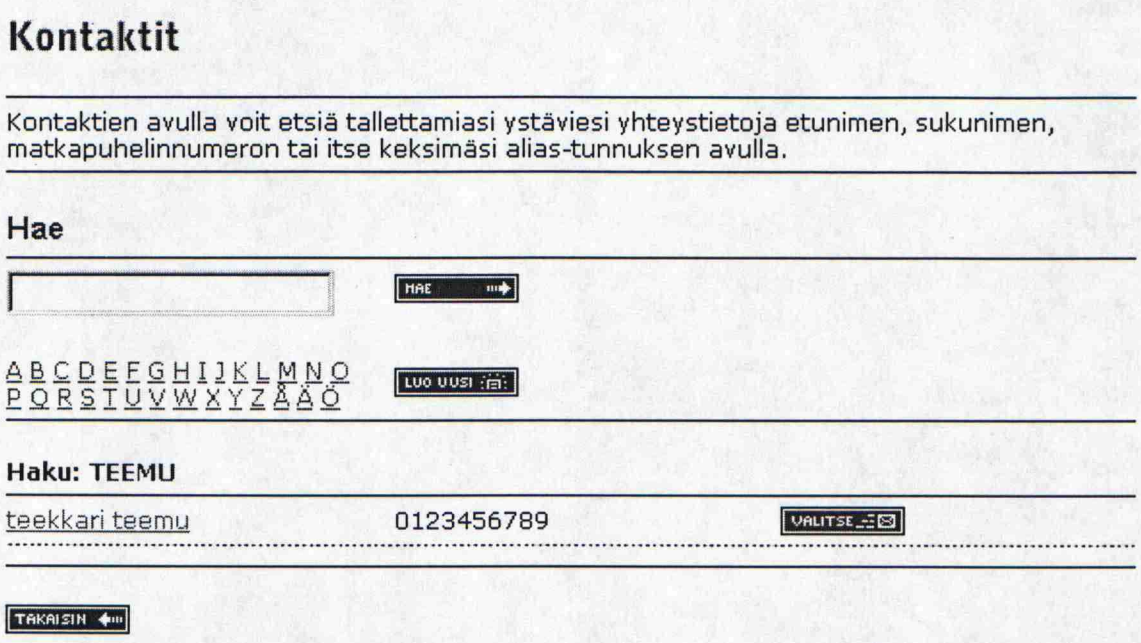


Figure 28. WWW user interface 6, the popup template.

SMS

In SMS, the only use case is the fetching of a contact by sending the following SMS message into service short number:

KONTAKTIT X (where X is first name, last name, "first name#last name", incomplete first or last name, alias or phone number).

Message sent to service:

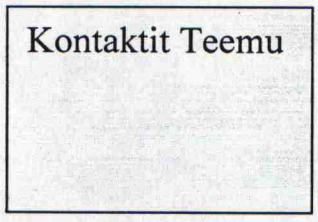


Figure 29. SMS user interface 1.

And response received from service:

Figure 30. SMS user interface 2.

The rest of the response message would continue in the next lines. The maximum length of the response is set to three 160 character messages.

WAP

In WAP, the user interfaces are like in the pictures below.

This start view is the user interface for fetching and creating new contacts:

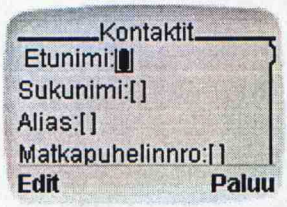


Figure 31. WAP user interface 1.

The lower part of the same view (scrolled down):

Figure 32. WAP user interface 2.

Results are returned in this format:

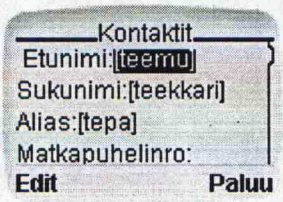


Figure 33. WAP user interface 3.

The lower part of the screen:

Figure 34. WAP user interface 4.

If more than one contact was found with the keyword, the lower part has additional "hae seuraava" ("fetch next") hyperlink.

The deficiencies in the interactive services, like the existing Addrbook, and in the whole portal are described next.

4.7 Deficiencies in the WWW portal

4.7.1 Designed for PC and frames only

The WWW portal fits to the current situation, where majority of customers use PC browsers with frame support. Portal unfortunately does not adapt into other kind of end devices. This kind of situation is not tolerable in the long run, when additional Internet access methods get more popular day by day [4].

4.7.2 Frame context

The whole WWW portal is based on using the frames. The frame practice with its justified historical reasons was described in 4.4.5. The WWW user interfaces described in 4.6.4 were only the main or middle frames in the portal frameset, except the one in Figure 28. In real life, the view is composed of many frames like in Figure 35., where the middle part is the same as in Figure 23. and is returned from the Addrbook application.

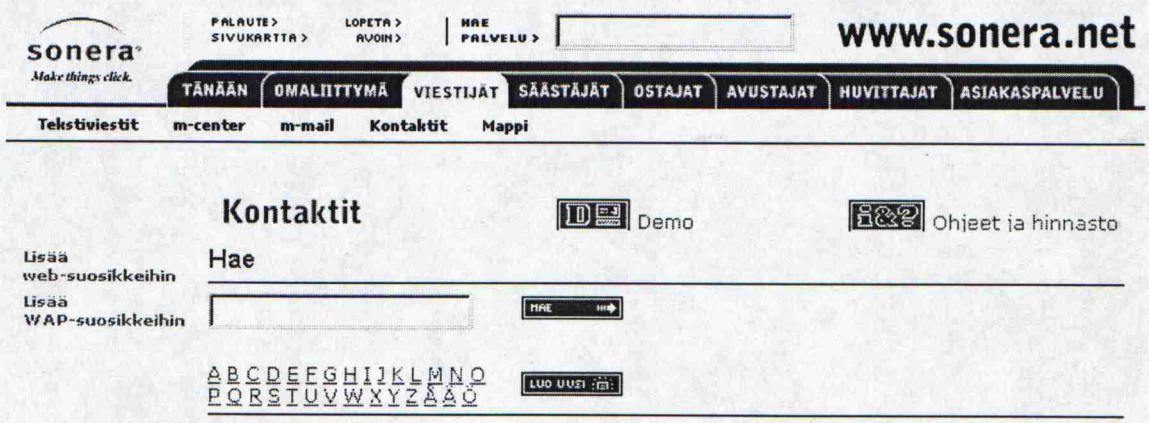


Figure 35. WWW portal with frames.

The individual static pages and templates in applications are designed to be used in frame context. The presentation and information in them is not adequate to standalone use. If individual frame is loosed from its frameset and presented alone without other frames, it probably lacks the information and illustration needed in good quality service experience. In Addrbook, the popup window was an example of this. For example, user directions, links to other services, prices, logos, advertisements and things like that can be decentralized in different frames in frameset.

4.7.3 Collective client identification

If different clients are served with customized content, one thing is for sure, these client types must be first distinguished.

Client identification should be done in a collective manner in one place, for example in web server module or Servlet. Configuration parameters making unambiguous identification possible must be collected and updated for this module.

4.7.4 Static pages

When serving static pages, client identification could be followed by client specific URL rewrite response or by explicit response with that same URL's content, like in Figure 36. In explicit approach (i.e., alt. b in figure), additional roundtrip can be avoided.

In both of these solutions, mappings of actual URLs to requesting clients for outside visible "virtual" URLs must be maintained. In simple approach, behind virtual URL is the page containing client-to-URL mappings. Every URL's content was checked by module, whether it is a client-to- URL mapping or primary content. Virtual URL can be also distinguished without investigating the syntax of its content, from some unambiguous filename suffix, for example. If nonstandard suffices are not accepted, all requests must go through this module for syntax check.

Without this kind of dynamic redirection mechanism, different URLs have to exist parallel for different clients, which is not convenient.

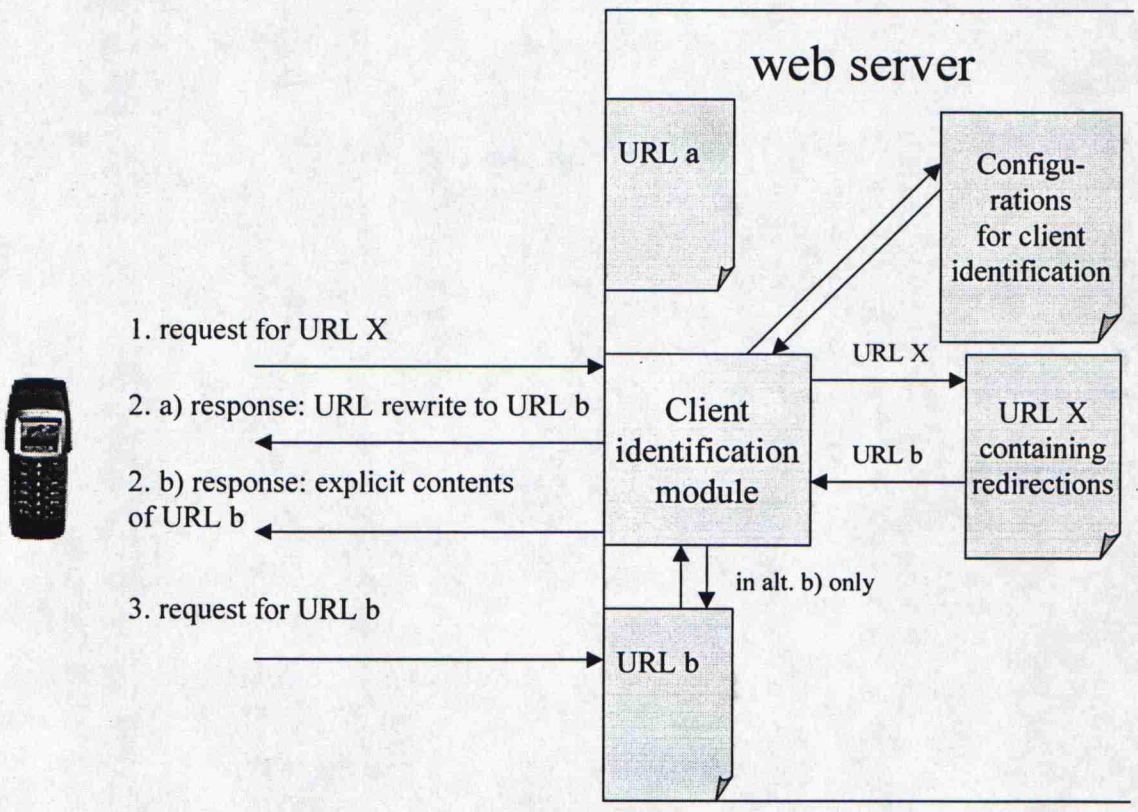


Figure 36. Client identification module with static pages.

4.7.5 Interactive services

Collective client identification module would be robust and effective solution for applications also. They were saved from doing this critical task in each one separately. Identification module could put the needed client data in HTTP header, for example.

Using this information, requesting client can be associated to its templates in application. Application maintains list of templates for different end devices. Configuration of this association should be convenient. Adding support for a new end device (i.e., templates) should be done without touching the program code.

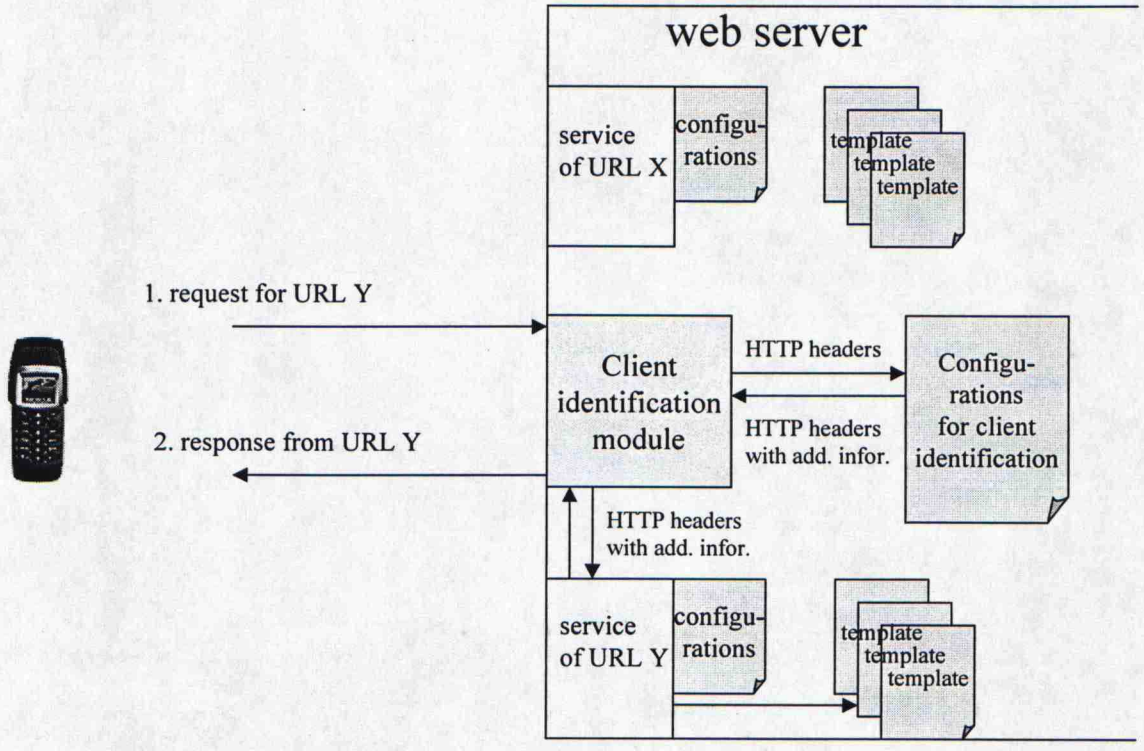


Figure 37. Client identification module with interactive services.

These improvements alone do not help if new end device requires different logic from the existing ones. Then some additional, maybe nontrivial coding has to be done also. Or separate versions of application can serve clients having different client-server logic.

4.7.6 URLs to framesets

Many URLs in portal point to framesets. In some clients, framesets are worked around by rendering the frames in frameset sequentially or giving links to each frame. The result of these kinds of work-arounds is not necessarily same as the intention that content's author had. In server-side solutions, authors would have the control and results were predictable.

If standalone pages replacing the corresponding framesets were authored, and client identification mechanism implemented, these standalone pages could be returned for clients configured to have no frame support. URL rewrite or explicit response with that specific standalone page could be used like described.

Special case is the framesets, where some frame is dynamically generated by application. In these cases, the standalone page must be a template behind that application. If there are frames from more than one application in original frameset, the resulting standalone page must be aggregated from all of these applications somehow.

4.7.7 Why Addrbook?

These same problems apply as well into any other portal service, because they all are based on same prerequisites, frames and PC user. The Addrbook service is not better or worse than any other services and there is no other reasons why Addrbook was chosen the prototype except the background of the writer.

4.7.8 What can be done?

In next chapters, the problem is elaborated and it is find out if there are any solutions. The proposition of the multi-channel service development framework is presented. After that, the end device independent prototype of the Addrbook is developed using the framework. It should adapt to the requests from different clients. The frame problem is analyzed into certain degree and some proposals made if possible. The frame problem appears to be harder.

5 Multi-channel development framework

5.1 Foreword

This chapter proposes a multi-channel service development framework. It addresses the problems described in 4.7.

The framework implements client identification and template association for interactive content. Support for new client type can be added without touching the program code. The templates are replaced in Framework with XSL style sheets. Cocoon web publishing system is used for serving static content.

Alternative was just to make a stand-alone multi-channel version of Addrbook service. It can be however more valuable if the multi-channel intelligence is put into a separate independent system. This system can then also be utilized from other services.

A short description of the framework is given in 5.4.1.

The framework will be nothing more than a prototype of solving media independence problem. This is so big challenge that it would be unrealistic to expect more. John Bentley has said:

> *"...you often don't really understand the problem until after*
>
> *the first time you implement a solution. The second time, maybe you*
>
> *know enough to do it right."*

Let's keep this thought in the mind during the own contribution part of the thesis.

5.2 Alternatives to making own solution

There are many web publishing frameworks and transcoding machines available, commercial and public, to solve the multi-channel problem. Commercial products are expensive, making such a big investment requires some elaboration to the problem at first. It is difficult to judge which one is better than the other. When making own "web publishing framework", the technologies come familiar at least, maybe this helps to judge different ways of solving the problem. Framework utilizes XML and XSL.

5.3 General requirements

General requirements to the multi-channel service development framework are listed. These requirements are in the scope of this thesis' main problem: end device independence.

The framework should be media independent. It should have no presuppositions of the end devices it will serve. To be realistic and honest, there will be shortages in implementation, because nobody can make this kind of all-round apparatus perfect. And when the only experience is accumulated from serving PC, WAP phone and SMS, the end result will be a little biased towards optimization in these, I believe.

All kinds of services should be able to set into this framework. Again, our experience is from interactive services, where services react dynamically to user inputs. The static content should as well be served from the framework.

The framework should be modular and extensible. If some piece of it looks poorly written or outdated, it should be easily replaced. New features and methods of doing things should be easy to add beside existing ones.

The aim is to do understandable and commonsense programming code and avoid complexities and needless optimization in wrong places. The organization should be clear. Solution should be easy to comprehend so that it is easy to use and redevelop by others as well. The integration of services with the framework should be simple.

Configuration should not be difficult and laborious. Good examples and documentation will help in this.

There are commercial products available to the problem but they cost money. Public software is favored instead.

Solution should exploit only future proof technologies.

In production use, performance and scalability are musts. In the prototype framework these requirements have been secondary. Memory and processor intensive XSL transformations can be performance bottlenecks. This is difficult problem and it is wrestled with by experts [27].

5.4 Framework and its Components

At first short description of the framework is given for those who are not interested in details. Technologies and pieces of public software used in Framework are presented. Framework and its six modules are introduced in detail after that.

5.4.1 Short description

The framework acts as a proxy in front of actual services. The services can be anywhere and anything until they are reachable. Framework gets original requests as HTTP and delivers them to the interfaces of actual services. It takes the responses back from the services and converts them into the XML if they already are not in XML. At the end, framework makes client and service specific XSL transformations and gives results back in HTTP response to the original requester.

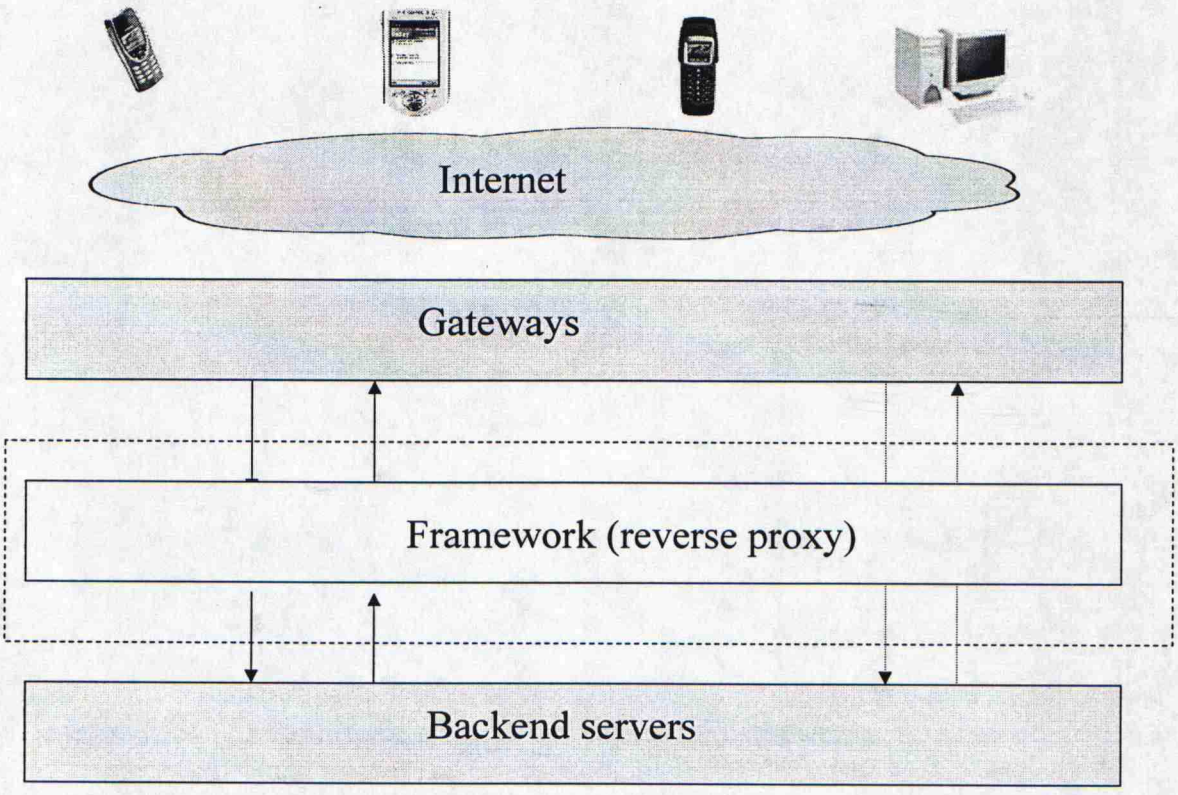


Figure 38. Picture of the multi-channel framework.

Framework identifies the client type and does also some general tasks on behalf of the actual services. It adds subscriber and customer data in the request that services may need. Framework has internal cache for the XSLT processors to accelerate its function. Since framework adds the presentation layer to the services, they do not have to have any intuition or knowledge of the client or presentation.

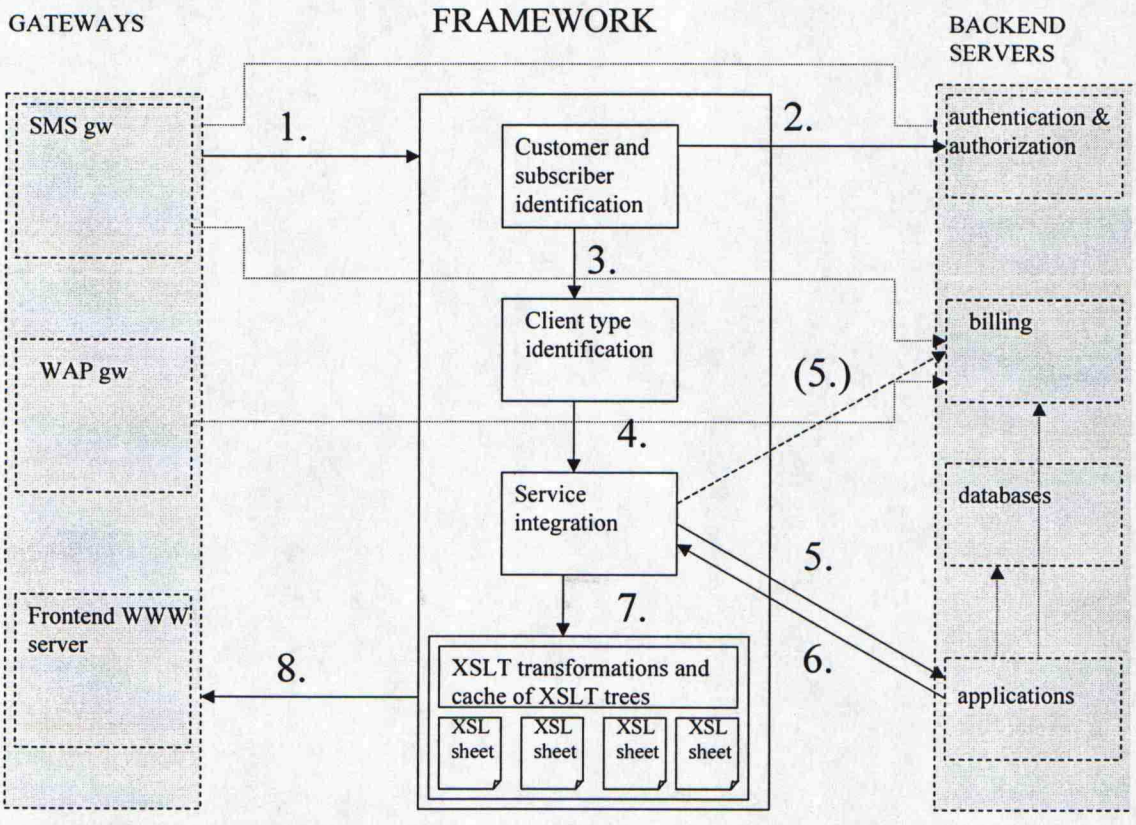


Figure 39. Closer picture of the multi-channel framework.

5.4.2 Chosen solution and technologies

Framework's solution is to transform XML with XSL sheet into final presentation form. Technologies used in the framework are HTTP, Java, Java Servlets, XML and XSLT.

5.4.3 Public software

Web server is from Apache, XML parser is Apache Xerces, XSLT processor is Apache Xalan and Servlet engine is Jserv. Cocoon is used with static content. All of these are publicly available free software. In addition they have established position and good reputation. Jserv is not developed anymore, corresponding free Servlet engine is Apache's Jakarta Tomcat.

5.4.4 Modules

The framework is composed of Cocoon, configuration module, logging module, customer and subscriber identification module, client type identification module, service integration module and XSL transformation module. Cocoon is not examined here. The self-made modules are introduced next by introducing the configuration data related to every module. With this approach the modules come familiar in practical sense and, at the end, the picture of the whole framework hopefully will be formed. How to use the framework hopefully becomes clear as well. The order of modules in introduction is from general ones to service specific ones.

5.4.5 Configuration module

Configuration module consists of configuration file and an object of Java Properties class that provides an access to configured items. Configuration file contains specific items for every module. Module specific items are presented in connection with the modules they belong to. Configuration items are presented in tables with the id and name of the item, example value of it and a short explanation.

Example configurations are only for html and wml media types. Example of a complete configuration file is presented in appendix E.

5.4.6 Logging module

Framework has logger class that writes logs in the configured log directory. Logging facilitates monitoring of how things are going and it also helps in detecting errors and correcting them. Statistics can be generated from the data in log files if wanted.

Id	Configuration item name	Value	Explanation
Log.1	LOGPATH	/services /framework/log/	Absolute path to log directory.
Log.2	LOGFILEPREFIX	Framework	Prefix of the log files, log files are generated on daily basis, and the prefix is appended with date like "framework18102 001".
Log.3	LOGFILESUFFIX	Log	Suffix of the daily log files, like framework181020 01.log.

Example listing from the log directory:

```
-rw-r--r--    1    apachegr    125762    Oct        1    16:04
framework1102001.log

-rw-r--r▒ -    1    apachegr    202502    Nov        1    23:06
framework02102001.log
```

5.4.7 Subscriber and customer identification module

Customer is the party, who is in charge of the payments in the service use. Subscriber is the actual end user of the service and has his own subscriber identity. Usually, services are only subscriber based, but sometimes the customer identity is needed also. Different restrictions can be posed on subscriber or customer basis. Several subscribers can be under the same customer. Customer can be private person or company. The barring classes related to this were discussed in 4.2.2.

At the moment, only the subscriber identification is put inside the HTTP request in three gateways (i.e., WWW, WAP and SMS). If customer identification is needed also, this module fetches it from the authorization server with the subscriber identity from the gateway as a key.

Configurations tell how the subscriber identities are put into the HTTP request in the gateways and thus how the framework can get access to them:

Id	Configuration item name	Value	Explanation
Sub.1	html_MSID_carrier	http_header	The id is carried in the HTTP header in html media type.
Sub.2	html_MSID_http_header_name	cookie:Sonera-msisdn	Name of the HTTP header carrying the value of the id. If the header name is "cookie", also specific cookie parameter name can be given if there are more than one parameter in the cookie.
Sub.3	wml_MSID_carrier	http_query_param	The id is carried in the HTTP query parameter in wml media type.
Sub.4	wml_MSID_http_query_param_name	msid	Name of the HTTP query parameter carrying the value of the id.

Also, how the framework carries the identity data forward to the service integration modules is configured. Subscriber id is carried by default but customer id is carried and fetched only if directed in service level configurations:

Id	Configuration item name	Value	Explanation
Sub.5	msid_param_name	msid	Subscriber identity is put into the value of HTTP parameter "subsid".
Sub.6	custid_param_name	custid	Customer identity is put into the value of HTTP parameter "custid".

5.4.8 Client identification module

Client identification needs a more extensive introduction because it is not so transparent.

Client type portfolio

First, it must be decided, which client types or end devices are served. All these client types must be identified also. Next decision is, which client types of them are fed with the same presentation (i.e., put under the same media type category in the configuration file).

In this example configuration, only two different client types are identified and served appropriately, html and wml. This means that no distinction is made between different clients under html or wml categories. So, there can be only one "compromise" html presentation in every service that has to fit nicely to all html clients (i.e., clients in the media type html). Exactly the same applies to wml.

As well it is possible to make further distinction between different html browsers or wml phone makers and feed tailored presentations into each one. The configuration or the framework itself is not a problem in this approach. When the amount of distinctly served client types increases, the work in designing the mark-up language pages and maintaining them increases also. It is up to the user, which approach to choose. In this module description the example case is with no further distinction.

The supported media types are configured like this:

Id	Configuration item name	Value	Explanation
Cli.1	media_types	html,wml	Media types that this framework recognizes.

Cli.2	media_type_default	html	Default media type, if media type is not identified.

If there were tailored presentations for Microsoft Explorer and Netscape HTML browsers the media types for them could be htmlMS and htmlNetscape, for example.

Client type identification methods

Some raw handwork is needed, when examining how to identify different clients. HTTP requests must be examined, whether there is some unambiguous data identifying the different client types. The complete references can and should be also obtained from the client manufacturers and other authorities. The systematic approach is needed to avoid unidentified clients. The handwork is good for verification.

The only identification method needed at the moment in our services is based on the contents in HTTP header "user-agent". There are no other methods implemented in the module. The identity could be put also in the HTTP query parameter value, or in other header field, for example. Prerequisite is that client identification module is fed with the HTTP requests that contain the client identity. Identity must be put there prior in gateways or somewhere else if the client itself do not do it.

So framework identifies client types (media types) after these instructions in the configuration file:

Id	Configuration item name	Value	Explanation
Cli.3	client_identitificati on_method	http_header	If there will be other identification methods also, this is needed.
Cli.4	client_identity_http _header	user_agent	Name of the HTTP header that is examined for identifying the media type.

Client resolving from the user-agent header

Specific strings inside the header identify the client. These strings must be searched from the header in specific search order.

The strings "Mozilla", "MSIE" and "Netscape" appear in the main stream HTML browsers. Thus, html media type is deduced if some of these strings appear in the header.

If the further distinction is needed between different browsers, (e.g., between Microsoft and Netscape) the search order is important and it must be like this:

1. MSIE
2. Mozilla
3. Netscape

"MSIE" must be searched first because it is the only value identifying MS Explorer. String "Netscape" or "Mozilla" can be in the headers of the both. If "MSIE" is found the browser is deduced to be Microsoft, otherwise Netscape [83] [27]. In wap media type, there are many strings identifying different clients, the most popular strings are configured in the following order:

1. Nokia
2. Wap
3. Ericsson

Id	Configuration item name	Value	Explanation
Cli.5	user-agents_html	MSIE, Mozilla, Netscape	Strings that identify the html client type(s) in "user-agent" header.
Cli.6	user-agents_wml	Nokia, Wap, Ericsson	Strings that identify the wml client type(s) in "user-agent" header.

Content types for different media

After the client type is resolved the appropriate content type (MIME type) is set to the HTTP response. These are configured like this:

Id	Configuration item name	Value	Explanation
Cli.7	content-type_html	text/html; charset=ISO-8859-1	Content type for media type html.
Cli.8	content-type_wml	text/vnd.wap.wml; charset=ISO-8859-	Content type for media type wml.

		1	
Cli.9	content_type_default	text/plain	Default content type if media type is not identified.

5.4.9 Service integration module

The use of framework is not restricted to services having HTTP interface. The configurations related to HTTP interface have nothing to do with actual services. Actual services can have some other interface as well. This configured HTTP interface is used in transmitting information between the Framework and its service integration modules, not between Framework and its actual services.

Actual services must be somehow integrated to the framework, or framework must be configured to use these services. And because services might have different interfaces, Framework connects to them always through special adapters called service integration modules. These modules hide the possibly different interfaces in underlying services. Integration modules can as well convert requests' formats into format understandable by the underlying service. Example of integration module will be implemented in next chapter, when making prototype of Addrbook service.

The interface

Service integration modules must implement "SoneraService" interface that Framework knows. It has these three public methods:

1. public HttpServletResponse service(HttpServletRequest request) throws IOException;

The services are called from the Framework through this method.

2. public void setProps(Properties props);

The own configuration module is given to the service integration module.

3. public void setLog(Log log);

The reference to the Framework's logging module is given to the service integration module.

Service identification from the request

When services are called through the framework, there must be some way to distinguish, which service, or service adapter from the framework point-of-view, is called. Only the framework's address is visible outside. The HTTP requests to the framework must have HTTP parameter "service" telling the service integration module's name to the framework.

Id	Configuration item name	Value	Explanation
Serv.1	service_param	service	HTTP query parameter name that identifies the service integration module to be called.

Service integration modules installed in the Framework are listed like this:

Id	Configuration item name	Value	Explanation
Serv.2	services	/services/addrbook/conf/addrbook.cfg	Configuration files of the service integration modules in the framework.

So, the service integration modules are listed in the form of their configuration modules, or configuration files. These configuration files contain tree items targeted to the framework. These are the class name implementing the SoneraService interface for the service, the identifying name to that class object and note if service needs customer id in addition to subscriber id.

Id	Configuration item name	Value	Explanation
Serv.3	loaded_class	addrbook.addrbookService	Class that implements the SoneraService interface.
Serv.4	name	addrbook	Name that identifies the service integration module. Ref:Serv.1
Serv.5	is_cust_id_based	no	No customer id is needed in this service.

Framework loads dynamically during its startup all the service integration module classes defined in the configurations. It puts these adapter objects into the hashtable, from where they are fetched during the runtime with a key that is configured in Serv.4.

Service integration modules' internal functioning

Service integration modules' configuration files contain some service specific configurations that are not used in the framework level unlike the previous Serv.1-Serv.5 items.

Service integration module must have some way to get access to the actual service it is responsible of. Integration modules get all needed input data from the framework in standard HTTP requests. Modules go on by calling the actual services in appropriate ways.

At the moment, the only module implementation is for service having HTTP interface. Socket messaging, Java RMI, CORBA, SOAP or other ways would be possible also, but there are not any implementations at the moment. The URL of the actual service is given like this in configurations:

Id	Configuration item name	Value	Explanation
Serv.6	url	http://machine.xyz.company.net:portnumber/servlets/XMLAddrbook	Internet address or URL of the called service.

If there are additional connecting methods or protocols, there will be appropriate configurations to the module. Only configuration items Serv.1-Serv.5 have to be similar in every service integration module, because they are used by one and the same framework. Module's internal configuration item Serv.6 instead can be anything, because it is not visible to the framework.

At the end, modules give services' responses back to the framework in XML in HTTP responses. If the actual service do not respond with XML, then its integration module has to make the needed conversions in formatting.

Service level media type support

When framework supports some client types, it means only that it can identify these, not more. In service level configurations, the support of media types tells what different client types particular service supports. Service level client type set is subset of the framework client type set. Prerequisites to supporting media type in service level are that there is needed logic built in the actual service or in its adapter and there are appropriate XSL sheets available. Latter is optional, because it can be that the XSL transformation is not needed at all.

Id	Configuration item	Value	Explanation

	name		
Serv.7	media_types	html,wml	Supported media types in the service. Prerequisite: These must be identified in the Framework level also. Ref:Cli.1

Requirements for the services

Logic in services should be built independent of the client type. It is not allowed to put any conditions on client types in the service logic. Starting point should be that services do not know anything of the requesting clients. This means, for example, that no presuppositions of script language support should be in logic. The maximum set of use cases and functionality should be implemented in services although some of these may not be needed in every client type. These principles encourage the media independence. For example, the fetching of data should be flexible, when some client types can possibly take only a little amount of data at once, incremental fetch operations should be implemented.

Some things like session handling and billing can be done in the actual service or in its integration module.

In the next chapter, the end device independent prototype of the Addrbook service is developed. The requirements to the media independent service logic are then in focus.

5.4.10 XSL transformation module

The XSL transformation module is in charge of the presentation layer of the services in the framework. It uses Apache Xerces XML parser and Apache Xalan XSLT processor to do this. Framework does not need at the moment XSL FO capabilities and XSL FO vocabulary is not supported in Xalan. So, when using the phrase XSL transformation within this framework, only the XSLT part (i.e., XSLT processor) of it is actually meant. Framework gets XML data from service modules and feeds it back to the client devices after XSL transformation.

The XSL transformations can be service and its internal use case specific.

Service may have different use cases that have different responses with different presentations to be wrapped to them. Usually, the use case is identified in the HTTP query parameter "CMD". In SMS though, it can be so that the use case must be parsed in nonstandard way from the user input, so the determination of use case is done in service integration module instead of common framework because it cannot be generalized. The use case is parsed and put into the unambiguous element in resulting XML in service integration module. The XSL module can get it from there for making decision on which XSL sheet to apply.

Id	Configuration item name	Value	Explanation
Xsl.1	usecase_element_name	CMD	The name of the XML element in response that contains the use case specific to the service called.

Next level would be response code. It is common that there are different kinds of responses in the same use case. For example, successful and unsuccessful ones may need different presentations. The response code is also one XML element in XML response. The element must be unambiguous under the root element. The response code level XSL sheet support is not yet implemented in the framework, but it is in the example configurations. The different presentations for different response codes can be achieved at the moment only by putting conditions in XSL sheets.

Id	Configuration item name	Value	Explanation
Xsl.2	resp_code_element_name	response_code	The name of the XML element in response that contains the response code specific to the use case and service called.

XSL transformations are service related. It is up to the designer, how he groups the presentations of the specific service into separate XSL sheets. It is possible to put all presentations in the same XSL sheet file with conditions on which part of the XSL sheet is used in which circumstance (i.e., with different values of CMD and response_code). Another, maybe cleaner, way is to put XSL for every use case (and response code) in own files. This is analogous how things are done in plain WWW HTML services also.

NOTE:

Service integration modules own configuration file contains directions to XSL module regarding to service's XSL transformations.

Example configurations hopefully help to clarify how XSL transformations are managed:

Id	Configuration item name	Value	Explanation

Xsl.3	XSL_PATH_html	/services/addrbook/xsl/html/	The path to the directory, where all XSL sheets are kept.
Xsl.4	XSL_PATH_wml	/services/addrbook/xsl/wml/	The path to the directory, where all XSL sheets are kept.

There must be XSL path configured one by one for every media type that the service integration module supports. These paths can be same or different.

Example listing from the /services/addrbook/xsl directory:

```
drwxrwxr--        4096      Nov 21 12:38      html

drwxrwxr--        4096      Nov 21 08:22      wml
```

If XSL sheets to different use cases (and response codes) are kept in separate files, they are configured like this:

Id	Configuration item name	Value	Explanation
Xsl.5	html_CMDS	fetchonewithid, update,saveupdated, fetchall, remove, new, savenew, fetchwithkeyword	Use cases in the service. Prerequisite: XML element in response carrying these use cases must be configured in the Framework. Ref:Xsl.1
Xsl.6	html_pages	fetchonewithid.xsl, update.xsl, saveupdated.xsl, fetchall.xsl, remove.xsl, new.xsl, savenew.xsl, fethwithkeyword.xsl	File names containing these XSL sheets.
Xsl.7	onewithid.xsl	onewithid	The page-to-use-case mapping. Can be one to one or one to many.

There must be corresponding configuration items (Xsl.7) for every individual XSL sheet file defined in xsl.6.

Example listing from the /services/addrbook/xsl/html directory:

```
-rw-r--r--  1926  Jan 25 11:53      fetchonewithid.xsl
-rw-r--r--  1662  Jan 25 11:53      fetchwithkeyword.xsl
-rw-r--r--  3928  Jan 25 11:53      new.xsl
-rw-r--r--   605  Jan 25 11:53      remove.xsl
-rw-r--r--  1041  Jan 25 11:53      savenew.xsl
-rw-r--r--   749  Jan 25 11:53      saveupdated.xsl
-rw-r--r--  3914  Jan 25 11:53      update.xsl
```

Corresponding configurations must be for every other media type as well. In the example, configurations were for two media type, html and wml.

With wml media type the approach is different from html. In wml, all XSL transformations are kept in one XSL sheet. Like already said, only one XSL sheet can contain transformation rules for the whole service, i.e., for every use case (and response code). This is achieved by putting conditions in the XSL sheet. Advantage in this approach is that the amount of files remains low. The configurations with wml are:

Id	Configuration item name	Value	Explanation
Xsl.8	wml_CMDS	fetchonewithid, update,saveupdate, fetchall, remove, new, savenew, fetchwithkeyword	Use cases in the service. Prerequisite: XML element in response carrying these use cases must be configured in the Framework. Ref:Xsl.1
Xsl.9	wml_pages	wml-all.xsl	File names containing these XSL sheets.
Xsl.10	wml-all.xsl	fetchonewithid, update,saveupdate, fetchall, remove, new, savenew, fetchwithkeyword	The page-to-use-case mapping. Can be one to one or one to many.

Example listing from the /services/addrbook/xsl/wml directory:

```
-rw-r--r--    6920        Jan 25 12:02            wml-all.xsl
```

Simple XSLT processor cache

To accelerate the XSL transformations, XSLT processors are kept in internal cache during runtime. When XSL sheet is edited, the framework automatically reloads the corresponding processor into the memory in the next request (i.e., converts it into internal tree-structure), after having detected that it has been modified. The cold start of the framework takes couple of seconds, but the runtime performance is better. The size of the cache should be managed somehow.

XSLT processor resolution

In recapitulation, how the module can get right XSLT processor? XSLT processors are kept in two-dimensional hashtable and they all have one XSL sheet behind them. The first key to the hashtable is service name, and the next key is absolute file name of the XSL sheet. The absolute file name of the needed XSL sheet can be built up using the configurations together with resolved media type, service name and use case.

5.5 Roadmap for framework

The first thing that must be done if this framework is used in production is absolutely testing its performance.

1. testing performance

Also, general functionality and robustness should be tested. In next chapter, the new version of the Addrbook service is implemented to test the framework in some degree.

The requirement "performance and scalability" in 5.3 was stated as secondary requirement, focus being in media independence related issues. The presented simple XSLT processor cache improves performance somewhat. There are severe deficiencies in this cache anyway. There is no upper limit to the cache memory size yet, and as well there is no policy in which processors are kept in cache and which are removed.

2. cache sizing and cache's performance optimizing

Framework can be developed in other sense also and couple of things has come into mind. One of the requirements in 5.3 was "modular and extensible". This requirement can be fulfilled better with dynamic class loading capabilities in essential classes.

3. dynamic class loading for XML parser and XSLT processor

4. XSL sheet location given as input parameter also

5. dynamic class loading for subscriber and customer identification module, configuration and logging module, simple interfaces to all of these

Only improvement that has been already made is the method of identifying different media types. The user-agent header is not adequate because there are so many different

browsers and values. The same value can exist in WWW browsers' and WAP browser's user-agent header. WAP clients are distinguished from others from one special header set in WAP gateway.

Other shortages can come into daylight in larger scale production use, not necessarily before.

6 Prototype of the multi-channel Addrbook

This chapter describes an end device independent prototype of the Addrbook service, XMLAddrbook, and its integration to the framework. User interfaces for three different media types are made to it. The performance of this prototype within the framework is tested and reported.

6.1 Description of the prototype

The existing Addrbook software was template driven as depicted in Figure 40. There is separate Servlets for WWW and WAP/SMS, because the logic in WWW version was not initially designed to support WAP use.

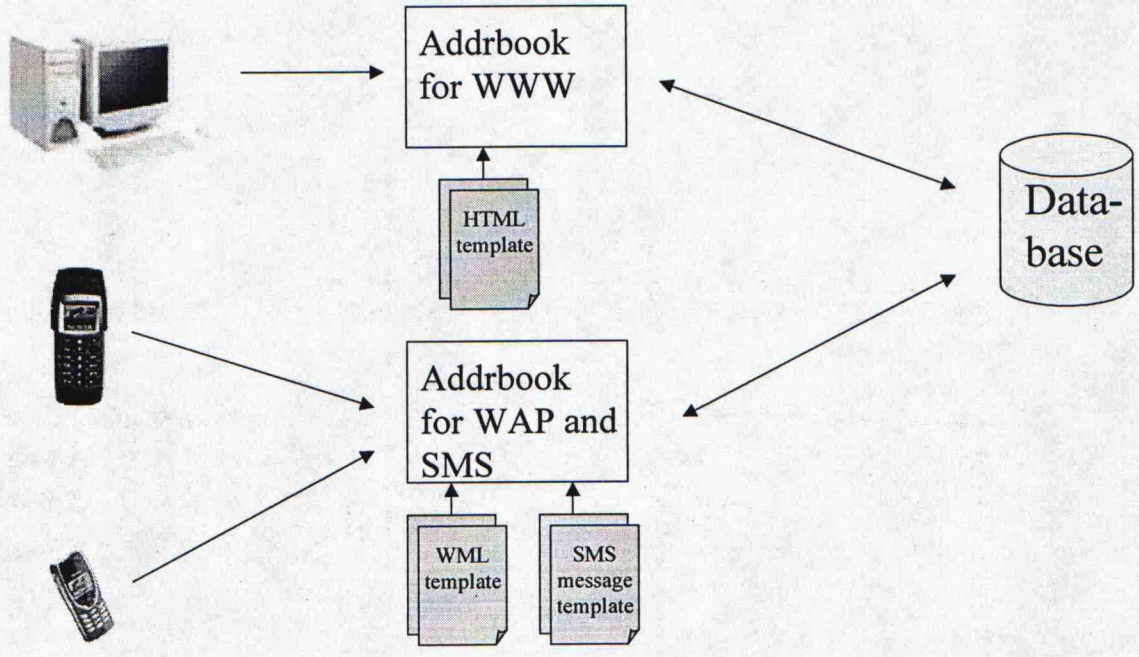


Figure 40. Architecture in existing Addrbook.

XMLAddrbook is a Servlet returning only XML (i.e., it does not have any templates). More precise, it does not know anything else either about the end device requesting it. It can be requested from other services as well, if presentation to the XML is needed, framework like developed in chapter five can be used.

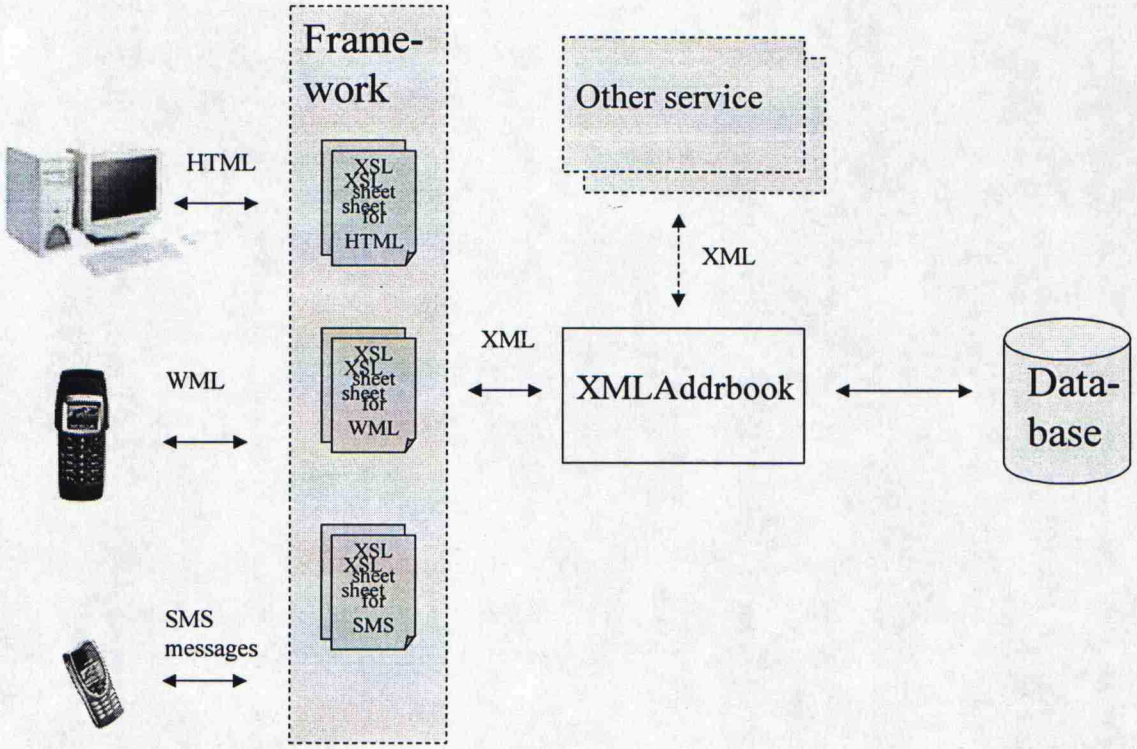


Figure 41. Architecture in XMLAddrbook.

Logic in XMLAddrbook

In this prototype, interfaces from three media types are made to XMLAddrbook. These media types are www, wml and sms. Framework identifies them as depicted in 5.4.8. Hopefully the logic in XMLAddrbook does not restrict its use from other media and other services as well.

XMLAddrbook has seven use cases, where contacts can be fetched, updated, removed or added. Three different use cases are implemented for fetching the contacts. With these seven use cases, the logic can be implemented for different media types as best suitable. If these use cases are not enough, additional ones can be implemented. They are rather straightforward to implement, because the presentation or end device level issues must not be considered at all.

XMLAddrbook identifies the use case from the incoming HTTP request parameter "CMD". Contact owner is identified from request parameter "msid". For each use case, specific XML document is generated and returned. The table below tells, which input parameters XMLAddrbook needs in each of the use cases. The elements, that are placed in the returned XML document, are in the last column:

Use case (i.e., value of the "CMD" parameter)	Other parameters	Returnable
Fetchwithkeyword	msid, keyword, from_index, to_index	(0-n) * (id, from_index, to_index, first name, last name, alias, phone number, fax, email, street address and postal address), keywork, CMD,

		msid
Fetchonewithid	msid, id	First name, last name, alias, phone number, fax, email, street address and postal address, CMD, msid
Update	msid, id	Like in "fetchonwithid"
Saveupdated	msid, id, first name, last name, alias, phone number, fax number, email, street address, postal address	CMD, msid
Remove	msid, id	CMD, msid
Savenew	msid, first name, last name, alias, phone number, fax number, email, street address, postal address	CMD, msid
New	msid	CMD, msid

When fetching contacts, given keyword can be like in the existing Addrbook, complete or incomplete first name, last name, first and last name, alias or phone number.

All other use cases are trivial except the "fetchwithkeyword". If large amount of contacts is found for a given keyword(s), all of them do not necessarily fit into one transfer unit in all media types (e.g., WAP and WML deck). Parameters "from_index" and "to_index" are for sequencing large amount of contacts into smaller transfer units. The contacts are returned in alphabetical order, and these two index parameters refer to it. Amount of returned contacts is the difference between "from_index" and "to_index" in corresponding request. XMLAddrbook puts the next values of "from_index" and "to_index" in its response, from which the request or link to the next portion in session can be constructed. In the last retrieval, there can be fewer contacts than that fixed amount. As a sign of last response, the value of "from_index" is "end". This kind of functionality is needed in WAP, for example. In WWW, all matching contacts can be fetched at once as well, if desired. This happens with setting "to_index" big enough.

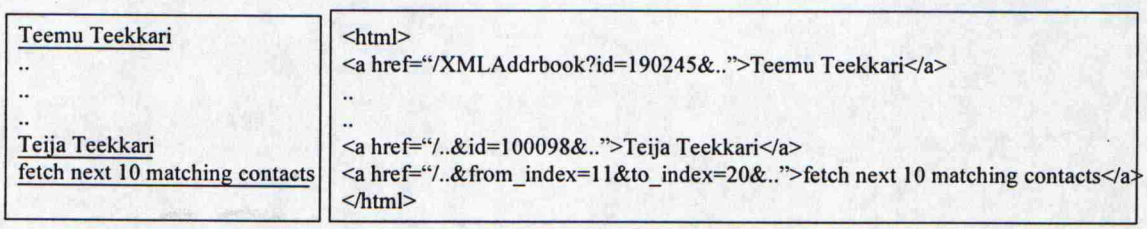


Figure 42. Use of sequential fetches.

SMS gateway portions and cuts the SMS response if needed. Not more than three SMS messages are returned for one SMS request, each one carrying 160 characters maximum. XMLAddrbook or its XSL sheets either do not have to care about this.

In use cases new, savenew, saveupdated and remove, no processing happens in XMLAddrbook. Only the values of "CMD" and "msid" are returned back. Actual content comes from appropriate XSL sheets. "CMD" and "msid" are always returned, because XSLT transformation unit needs the former to fetch correct XSL sheet (Figure 39.) and latter may be needed in presentation (i.e., in XSL transformations).

XSL sheets

Next step is to author all needed XSL sheets. Below are the sheets authored for each media type supported by XMLAddrbook (i.e., www, wml, sms):

www:

-fetchonewithid.xsl

-update.xsl

-saveupdated.xsl

-remove.xsl

-new.xsl

-savenew.xsl

-fetchwithkeyword.xsl

wml:

-addrbook-all-wml.xsl

sms:

-addrbook-all-sms.xsl

So, each use case can have its own XSL sheet or several use cases can have common sheet containing presentations for all (5.4.10).

Inside each media type, the logic can be implemented as wanted by calling the use cases in XMLAddrbook and constructing desired user interfaces with XSLT. In SMS, there is no form-like user interface, the fetching of contact is initiated by one SMS message. In WWW and WAP, it happens in two phases where links to matched contacts are returned first. Selecting one of these links fetches the whole contact. After that, the contact can be removed or fetched for edition. When fetched for edition, form is returned prefilled with the contact values. In use case "new", empty form is returned. In WWW and WAP, ten contact links are returned by once, this amount could be bigger.

6.2 *Integration with the Framework*

When service is ready to be used, these steps must be done in integration:

1. Implement service integration module class for service
2. Tell the framework, where to locate this class
3. Tell the framework, which XSL sheet to use, with which use case (i.e., with which value of the parameter "CMD" that is included in the XML response)
4. Restart the framework Servlet

Service integration module for XMLAddrbook

Service integration module class must implement "SoneraService" interface, which was described in (5.4.9). The purpose of this interface was to enable systems having different interfaces to communicate.

XMLAddrbook understands only HTTP request format. So, its service integration module must convert SMS request format to HTTP request format.

Service integration module does two tasks in XMLAddrbook:

1. Conclude, which use case the request conveys and put it into the value of HTTP request parameter "CMD".
2. Converse the SMS gateway request format into the HTTP request format understandable by XMLAddrbook.

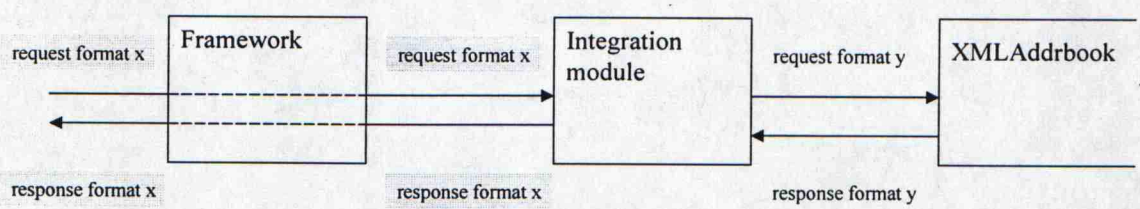


Figure 43. Format converter between requests and responses.

User delimits different fields in SMS requests with space and '#' characters. For example, syntax of the request in use case "savenew" is like this (in finnish):

L Eetunimi#Ssukunimi#Aalias#Ppuhelinnumero#Ffaxnumero#Sosähköpostiosoite

#Lolähiosoite#POpostiosoite

First parameter "L" identifies the use case and it is delimited with space character from the other actual contact fields, that are delimited with '#'.

This is converted into this:

CMD=savenew&Fname=etunimi&lname=sukunimi&alias=alias&phone=puhelinnumero &fax=fax-numero&email=sähköpostiosoite&address1=lähiosoite&address2=postiosoite

In fetching, when giving first name and last name as keyword, they are delimited with '#'. Integrator module converts "firstname#lastname" to "fetchparam1=firstname&fetchparam2=lastname".

Framework locates the class for integration module with the help of configuration item Serv.2 in 5.4.9:

Services:/services/addrbook/conf/addrbook.cfg

The module specific configurations are in "/services/addrbook/conf/addrbook.cfg" (5.4.9). These are:

1. Loaded class for the service integration module (implements SoneraService interface)
2. The URL, where the XMLAddrbook can be requested from
3. The name of the service
4. Media types supported
5. Path of the XSL sheet directories for each supported media types
6. The mappings between XSL sheets and use cases (items xsl.5 and xsl.8)

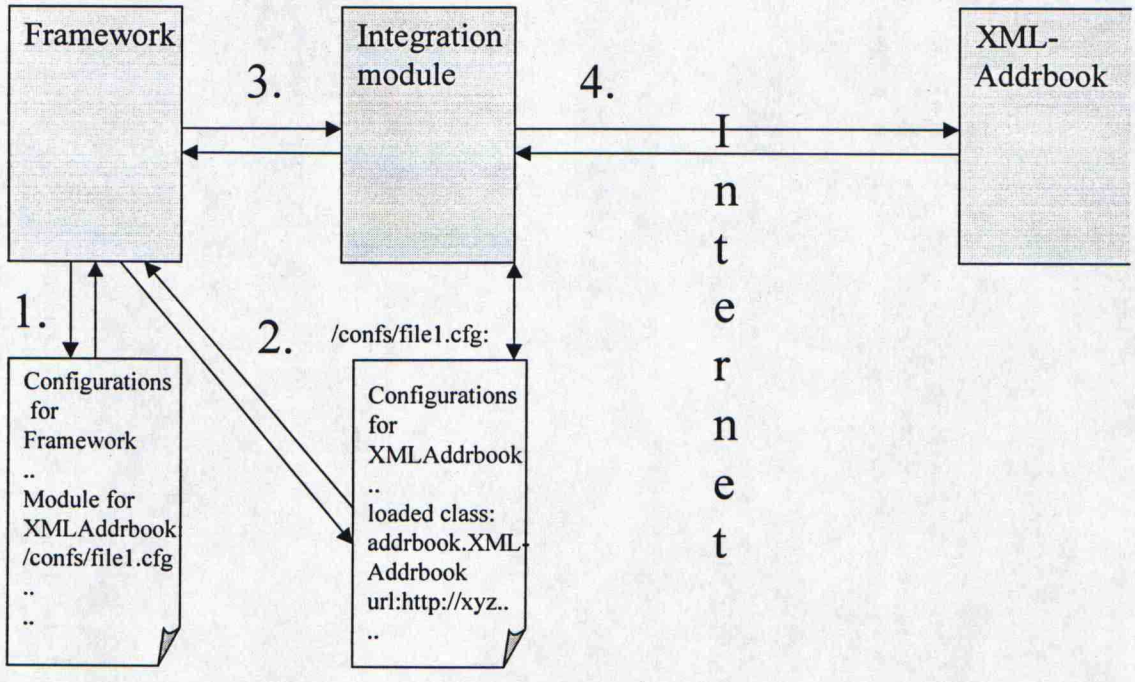


Figure 44. How framework locates XMLAddrbook.

These will be the real configurations, they are in appendix F also:

1. Loaded_class:xmladdrbook.IntegrationModule

2. url:http://machine.x.y.net:8090/xmladdrbook/XMLAddrbook

3. name:XMLAddrbook

4. media_types:html, wml, sms

5. XSL_PATH_html:/opt/xmladdrbook/xsl/html/

   XSL_PATH_wml:/opt/xmladdrbook/xsl/wml/

   XSL_PATH_sms:/opt/xmladdrbook/xsl/sms/

6. html_CMDS:fetchwithkeyword, fetchonewithid, update, saveupdated, remove, new, savenew

   html_pages:fetchwithkeyword.xsl fetchonewithid.xsl, update.xsl, saveupdated.xsl, remove.xsl, new.xsl, savenew.xsl

   fetchwithkeyword.xsl:fetchwithkeyword

   fetchonewithid.xsl:fetchonewithid

   update.xsl:update

   saveupdate.xsl:saveupdate

   remove.xsl:remove

   new.xsl:new

   savenew.xsl:savenew

   XSL_PATH_wml:/opt/xmladdrbook/xsl/wml/

   wml_CMDS:fetchwithkeyword,fetchonewithid,update, saveupdate, remove, new, savenew

   wml_pages:xmladdrbook-all-wml.xsl

   xmladdrbook-all-wml.xsl:fetchwithkeyword, fetchwithid, update, saveupdated, remove, new, savenew

   XSL_PATH_sms:/opt/xmladdrbook/xsl/sms/

   sms_CMDS:fetchwithkeyword,savenew

   sms_pages:xmladdrbook-all-sms.xsl

   xmladdrbook-all-sms.xsl:fetchwithkeyword,savenew

If new media type is added to XMLAddrbook service, let's say "pda", only configurations that must be made into integration module are:

media_types:www, wml, sms, pda ("pda" added to medit_types)

XSL_PATH_pda:/opt/xmladdrbook/xsl/pda/ (the XSL style sheet path of pda)

pda_CMDS:

pda_pages:

and the last item(s), the style sheet to use case mappings (one-to-one, or one-to-many), like above with www, wml and sms.

Frame problem

Frame problem can be solved in the Framework by making additional media type and associated XSL sheets for browsers without frame support.

Use of the service

1. Start (WWW, WAP)

The static start pages in WWW and WAP come from Cocoon web publishing network. In SMS, there is no start page.

From start page, use cases "fetchwithkeyword" or "new" can be initiated. In start phase, the XML document contains only PIs for HTML and WML stylesheets.

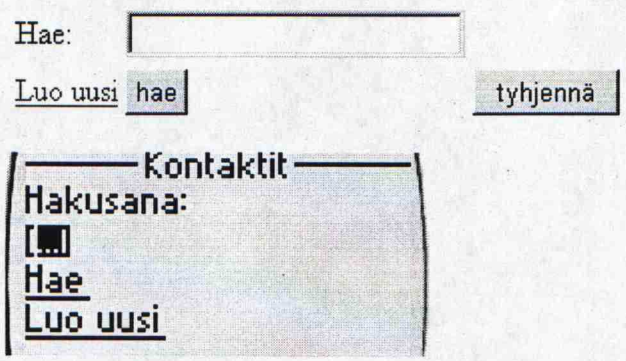


Figure 45. User interface 1 in WWW and WAP.

2. Fetchwithkeyword (WWW, WAP, SMS)

In SMS, matching contacts are fetched right away for a keyword in SMS. There is no intermediate phase with links to matching contacts in SMS. Contacts are fetched exactly like in 4.6.4. The XSL sheet constructing all the WAP user interfaces is in appendix C.

In WWW and WAP, there is an intermediate phase. List of links to contacts that match the given keyword is returned. Link contains first name and last name and it hides unambiguous id for contact.

1. Teekkari Teemu
2. Teikäläinen Teija
3. Tossavainen Teemu
4. Turunen Tuija

Takaisin

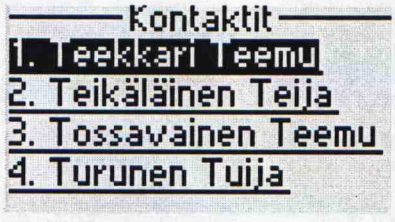


Figure 46. User interface 2 in WWW and WAP.

3. fetchonewithid (WWW, WAP)

Contact is fetched against its id for read-only purpose. Ids are first fetched with "fetchwithkeyword".

ETUNIMI:	Teemu
SUKUNIMI:	Teekkari
ALIAS:	Tepa
PUHELIN:	0123456789
FAX:	
SDHKVPOSTIOSOITE:	teemu@postiluukku.fi
LDHIOSOITE:	Jämeräntaival 50
POSTIOSOITE:	02150 Espoo
Muuta	Poista

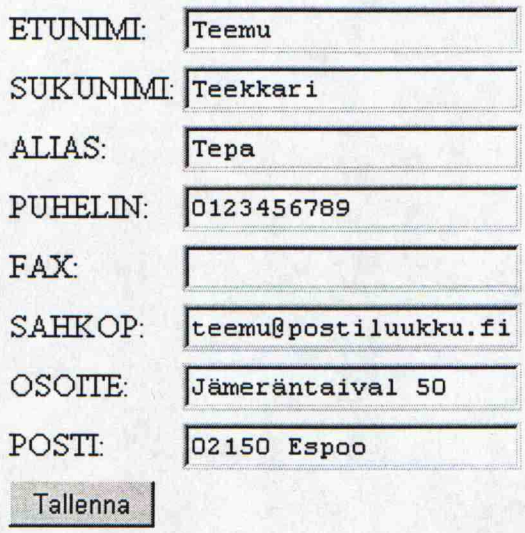
Takaisin

(second picture is from the bottom of the page)

Figure 47. User interface 3 in WWW and WAP.

4. update (WWW, WAP)

Contact is fetched against its id for editing (i.e., it is fetched to the interactive form, where the old values of the contact fields are prefilled and can be edited).



(second picture from the bottom of the page)

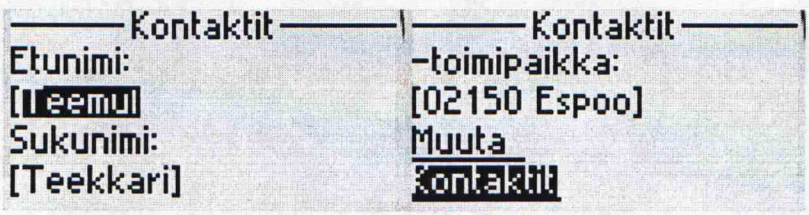


Figure 48. User interface 4 in WWW and WAP.

5. saveupdated (WWW, WAP)

The edited contact is saved.

6. Remove (WWW, WAP)

Contact is removed against its id. Note, in the existing version, contact must be fetched for update first, after which it can be removed. In this multi-channel version, contact can be removed right after its read-only fetch.

7. new (WWW, WAP)

An empty form is returned to the user, where she can fill the values for a new contact item.

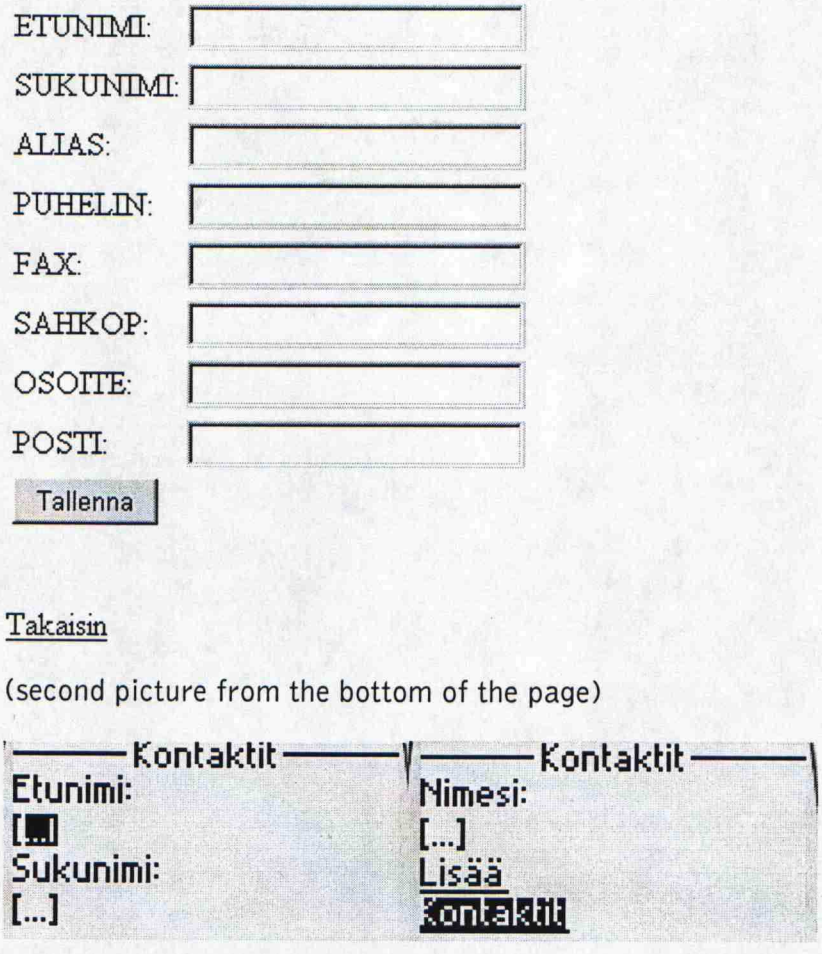


Figure 49. User interface 5 in WWW and WAP.

8. savenew (WWW, WAP, SMS)

A new contact is saved to database. In SMS, the syntax was like (in finnish):

L Eetunimi#Ssukunimi#Aalias#Ppuhelinnumero#Ffaxnumero#Sosähköpostiosoite

#Lolähiosoite#Popostiosoite

Figure 50 and Figure 51 show how these 8 use cases are related.

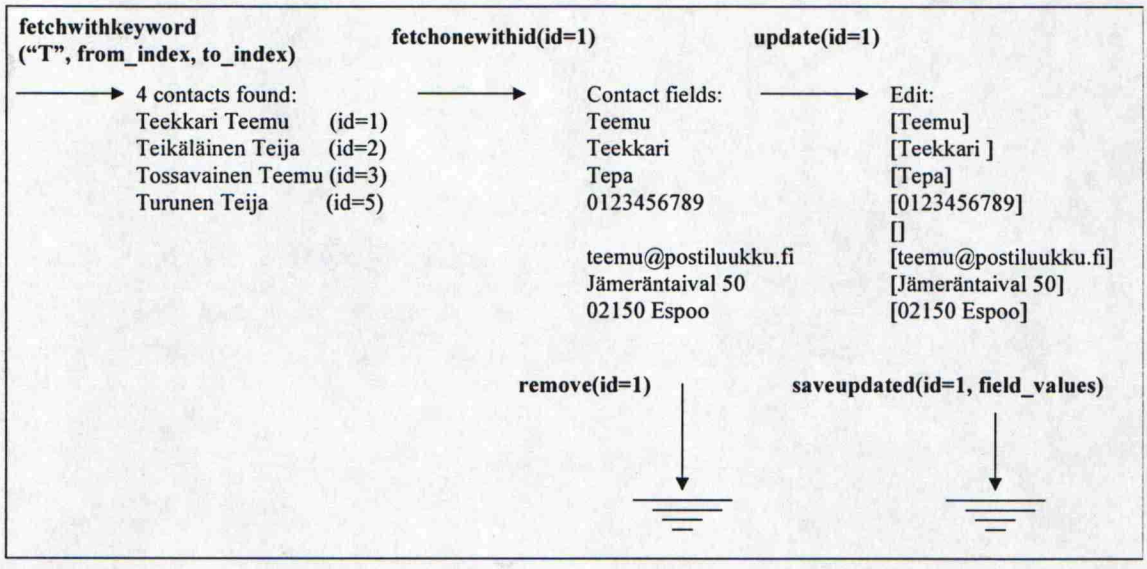


Figure 50. Use case 1: WWW and WAP.

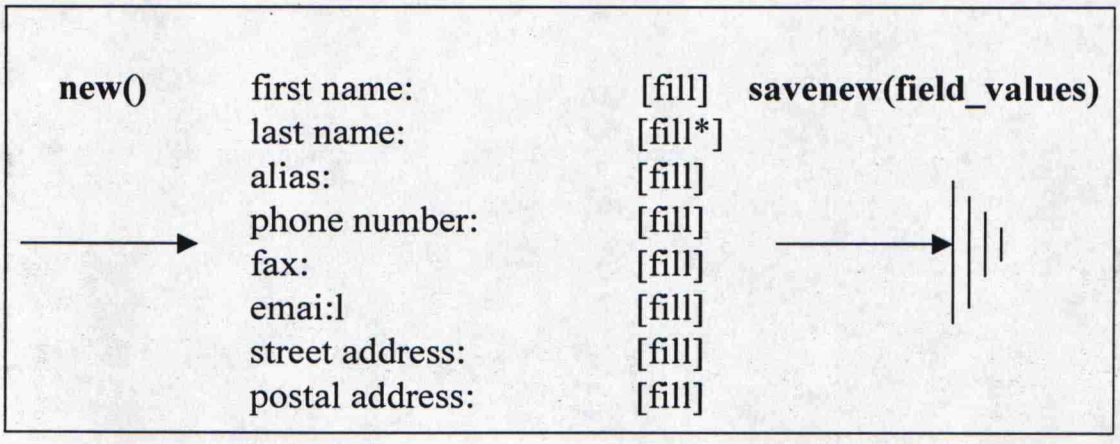


Figure 51. Use case 2: WWW and WAP.

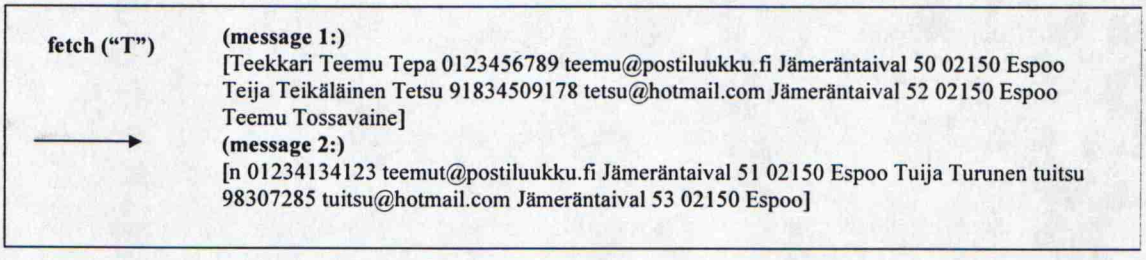


Figure 52. Use case 3: SMS.

6.3 Testing and evaluation

6.3.1 Platform

Framework and XMLAddrbook was running on PC having the following configurations.

Memory	256 MB SDRAM (PC-100)

Processor	INTEL PENTIUM III 700 MHZ
Operating System	Linux Red Hat 6.2 i386
Web server	Apache 1.3.12
JDK	1.2.2
JSDK	2.0
Cocoon	1.8.2
Xerces	1.2
Xalan	1.2
Database	Oracle 8.05

6.3.2 Testing tool

Framework's performance was tested with benchmark tool Apache Bench (ab). The ab version in this test was the one coming with Apache distribution 1.3.12 [84].

6.3.3 Test cases and their results

Framework, XMLAddrbook, Apache and ab were all running in the same machine. Database was in other machine.

Two test cases were run. First one was for XMLAddrbook service's performance in Framework and the second one for only XSLT transformations in Framework. Both test cases were for use case "fetchwithkeyword". In the second test, the XMLAddrbook was not requested, but the transformed XML document was identical to document returned from XMLAddrbook in that use case. During both test cases, it was concurrently checked from the framework's log file, that service was responding correctly to every request.

In both test cases, XSLT transformations were made as synchronized operations, because there was only one transformer that had to be synchronized.

In real use, the response time experienced by the end user is bigger due to extra delays in network. The only network delay during this test was in test case 1, from database operations, database resided in remote machine.

Test case 1

First test case one was for XMLAddrbook service's performance in Framework. This is the performance experienced in real use of the service without network delays. First test case was run for WWW, WAP and SMS user separately, because they have different XSL sheets and transformations. In ab, the HTTP headers can be set, this enabled the imitating of three different media types, www, wml and sms.

Only concurrency tests were run, and they were run ten times for 1, 5, 10, 20 and 40 concurrent users. The average test results can be seen in Figure 53. Performance test results of Framework with XMLAddrbook.

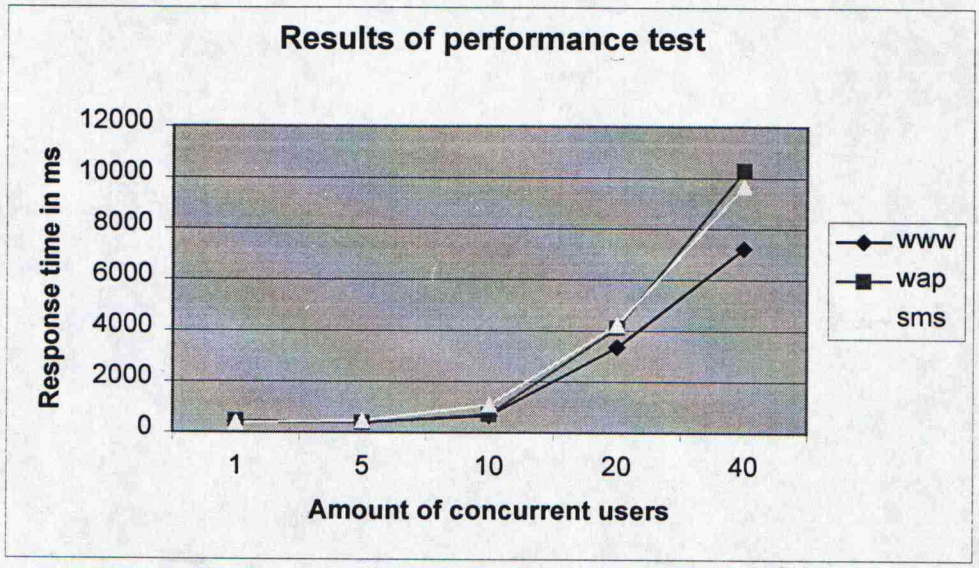


Figure 53. Performance test results of Framework with XMLAddrbook.

The XMLAddrbook did stand the concurrent load of about 40 users in the Framework in test environment. During the tests there was no remarkable load from other users. When the amount of concurrent users were increased from the 40, service crashed down. This is not caused from Framework or its XSLT transformations, because test case 2 shows that they can stand that load. So, the XMLAddrbook is the bottleneck. There are time-consuming database operations without database connection pooling.

The little differences between response times of WWW, WAP and SMS can result from the different sizes of XSL sheets that were transformed in the test. WWW sheet was smaller than SMS and WAP. As described in 5.4.10, media type www had separate style sheets for every use case. Media types wml and sms had only one style sheet with conditions on which part of it is applied on particular use case. WML and SMS style sheets were bigger and had more conditions, and so the transformation took more time.

To be realistic, in this type of service the load will be at most few concurrent users. The requests from the simultaneous users are many times intermingled, because it takes time when user fills the fields, for example. The response time of about 0,5 seconds with five concurrent users is tolerable.

Test case 2

The performance of XSLT transformations only was tested in isolation, with static XML documents. The DOM object of the static XML document was constructed before test and kept in memory, also the XSLT transformer was constructed from the XSL sheet in advance. So, in test case 2 the only processing was the XSLT transformation. Only concurrency tests were run, and they were run ten times for 1, 5, 10, 20 and 40 concurrent users, and thirty times for 100, 150 and 200 concurrent users. The results can be seen in Figure 54.

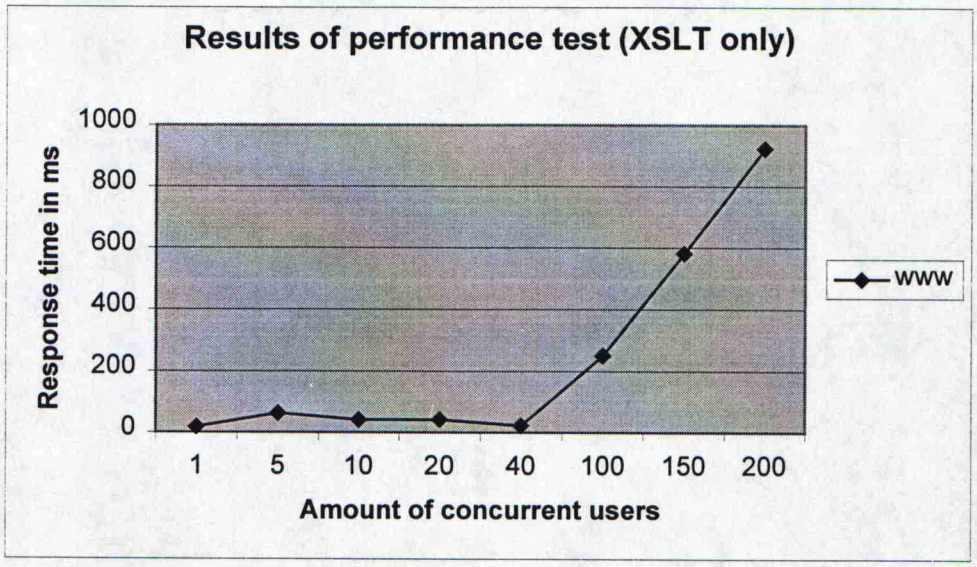


Figure 54. Performance test of XSLT only.

The response times were much better with only XSLT transformations. Not until with 300 concurrent users, the framework could not stand the load and some responses were not responded anymore. The response time started to increase between 50-100 concurrent users. Without synchronization of XSLT transformations, the response times could be lower.

7 Conclusions

7.1 Findings

The Framework works fine with prototype service (i.e., XMLAddrbook). The service has been in production in the Framework since 11/2001 and no problems have appeared. Framework should be used in other services also before it can be said more about its performance. The Framework's XSLT performance appears to be sufficient with XMLAddrbook at least. If the performance is not adequate with more services in the framework, it could be made better by getting rid of synchronization of XSLT transformers. If memory fills up with more services, the file system could be used to cache XSLT transformers. If these improvements are not enough, then Cocoon should be taken into use with interactive services as its XML producers, and checked whether it can fix the performance problem.

One finding is that Framework should be simpler. For others than its designer, Framework can be too difficult to exploit, at least in the beginning. This has become clear by the interview with colleague. Especially the XSL sheet-to-use case mapping practice is too complicated to configure and comprehend.

Alternative to Framework is Cocoon, it is may be easier to configure and its performance due to sophisticated parsing, transformation and caching is surely better. In Cocoon 2, the SAX processing is used efficiently to increase its performance. XMLAddrbook and other XML services can be used as XML producers in Cocoon. There is simple interface for producers in Cocoon and it is not a problem to implement it.

The Framework has few differences compared to Cocoon. XML producers [27] (e.g., XMLAddrbook) in Framework do not know explicitly their XSL sheets. In Cocoon, the producer puts explicitly PIs for all its XSL sheets in the beginning of produced XML document. Cocoon chooses right PI from these with its client identification data and makes appropriate XSLT transformation. Framework can be configured to identify also subscriber and customer on behalf of its XML producers. Cocoon cannot do this without changes.

One finding is that maintaining large amount of XSL sheets is as inconvenient as maintaining corresponding amount of HTML or WML pages. Basically, authoring own XSL sheets for every media type can solve the media-independence and frame problem. But the penalty is that huge amount of XSL sheets must be authored. Authoring XSL sheets is rather laborious. In author's and some colleagues' opinion, it is more difficult than making HTML or WML alone and it would be too hard for web designers to learn it.

In author's opinion, making static content media independent is an easier task than making that for interactive services. There is logic in interactive services that may have to be considered differently for different media. Even if good transcoding product is found, it cannot do anything to the logic implemented in Servlets, for example.

The presentation layer maybe cannot be made scalable with only XML and XSL, but with XML, the logic in services can. If service returns only XML and does not know anything about the requesting client, the program code becomes simpler and easier to maintain. The logic is easier to implement and improve when there is no presentation-related issues in program code.

One finding is, in author's opinion, that XHTML should be used instead of HTML. The future browsers and devices will have better support for XHTML. Again in author's opinion, nothing is lost, if HTML is replaced with XHTML. This is not a remarkable relief to the overall problem of media-independence, though.

Existing base can be tried to convert with some transcoding machine. But it is hard to predict without long lasting trial period which one is the right solution. The complete scalable solution without compromises, in which the handwork in authoring the presentation does not increase in linear relation to supported media types, is hard to imagine in author's opinion.

7.2 Were objectives met?

Objectives were met only partly. Technologies that can help in achieving media-independence became familiar during this thesis, especially XML and XSL. Also the future candidate technologies under development became familiar in certain degree. Author learned lot from designing the framework and one prototype service into it. This knowledge can be hopefully exploited in the future in the organization, where this thesis was written.

Framework for multi-channel service production was developed and the result is not so extraordinary, but works fine in one service at least.

One objective was to compare and evaluate different solutions and products to the problem. Cocoon was the only product that was examined in practice. The commercial products were left without explicit consideration in this thesis. Some example solutions were introduced and judged. Solutions behind different products can be anyway understood and evaluated better now when some experience and knowledge in the field has cumulated.

XML was studied quite thoroughly during this thesis. Afterwards it has come into mind, that maybe more prototype applications to multi-channel problem should have been made. Less theory and more practice. Programming some prototype implementations after the examples described in 3.9 could have given valuable practical experience.

Adapting of XML into big organization was overestimated objective. In the organization, maybe more realistic objective could have been getting rid of bad HTML or frames, for example.

No solution was proposed for automatic conversion of the existing content base into media-independent content. Only imaginable example approaches were shortly described. In the Framework's XML/XSL solution, all existing content must be converted manually into XSL. In static pages, HTML Tidy can be used to modernize non-standard HTML into XHTML.

At the beginning, it was easy to enumerate fine objectives. Being afterwise, it should have been better if the problem has been defined more exactly in the beginning. The media-independence is too big a problem field to cover in one thesis.

References

[1] European Commission, "Sixth Report on the Implementation of the Telecommunications Regulatory Package," Annex 1 – Telecommunications market data, Chart 8, Per capita telecommunications expenditure trends in EU, USA and Japan, COM(2000)814, 7 December 2000, http://europa.eu.int/ISPO/infosoc/telecompolicy/implrep6/Size+growth-en.pdf.

[2] Telecommunications Industry Association, "Applications to Drive Growth for Network and Enterprise Markets," 2001 Multimedia Telecommunications Market Review and Forecast, P.A. Release 01-03/01.16.01, http://www.tiaonline.org/pubs/press_releases/index.cfm?parelease=01-03.

[3] WAP Forum, "WAP 2.0 Technical White Paper," http://www.wapforum.org/what/WAPWhite_Paper1.pdf, August 2001.

[4] eTForecasts, "Internet Users Will Surpass 1 Billion in 2005," Feb 2001, http://www.etforecasts.com/pr/pr201.htm.

[5] European Commission, "The Introduction of Third Generation Mobile Communications in the European Union: State of Play and the Way Forward," Annex 2 – Overview of 3G Licensing in the Member States, COM(2001), 141, 20.3.2001 2000, http://europa.eu.int/information_society/topics/telecoms/radiospec/mobile/docs/pdf/3gcomen.pdf.

[6] Sumanta Deb, SETLabs Analytics, Infosys Technologies Limited, "How can enterprises leverage 3G's complex value chain and its uniqueness, 2000 Wireless Access Technologies," http://www.watmag.com/technologies/3G_UMTS/3G-008/3G-008.html.

[7] NTT DoCoMo, http://www.nttdocomo.com/.

[8] MercuryRed Wireless News, Issue #1, March, 2001, "Lessons from I-mode," http://www.mercuryredwireless.com/pdf/3g_futures.pdf.

[9] P. Vuorimaa, J. Teirikangas, and J. Vierinen, "Ubiquitous multimedia services with XML," accepted to the 1st Int. Conf. Universal Access in Human-Computer Interaction, UAHCI, New Orleans, Louisiana, August 5-10, 2001.

[10] O. Marttila and P. Vuorimaa, "XML based mobile services," the 8th Int. Conf. in Central Europe on Computer Graphics, Visualization, and Interactive Digital Media, WSCG'2000, Czech Republic, Feb. 7-10, 2000.

[11] S. Patient, "Think Small, Think PDA," Jun 2000, http://www.webdevelopersjournal.com/articles/think_pda.html.

[12] J. Vierinen and P. Vuorimaa, "A browser user interface for digital television," in Proc. the 9th Int. Conf. in Central Europe on Computer Graphics, Visualization and Computer Vision, WSCG'2001, Czech Republic, Feb. 5 - 9, 2001, pp. 174-181.

[13] M. Floyd, "Separating Body from Soul," webTechniques, Jul 2000, http://www.webtechniques.com/archives/2000/07/floyd/.

[14] B. Marchal, "Making teams work, via XML and XSL," Nov 2000, http://www-106.ibm.com/developerworks/xml/library/x-applied/?dwzone=xml.

[15] B. Marchal, "How Java Programmers and HTML Designers Can Collaborate Using XML," Netscape Communications Corporation 1999, http://developer.netscape.com/viewsource/marchal_xml.htm.

[16] T. Bray et al., "Extensible markup language (XML) 1.0 (second edition)," W3C Recommendation, Oct. 6, 2000. http://www.w3.org/TR/REC-xml.

[17] J. Bosak and T. Bray, "XML and the Second-Generation Web," Scientific American, May 1999, http://www.sciam.com/1999/0599issue/0599bosak.html.

[18] C. Weyrich, "Orientations for Workprogramme 2000 and beyond," Sep 1999, http://www.bit.ac.at/IST/istag-vision.pdf.

[19] D. Martin et al, "Professional XML," Wrox Press Ltd, Jan 2000.

[20] S. Adler et al., "Extensible stylesheet language (XSL) version 1.0," W3C Recommendation, Oct. 15, 2001, http://www.w3.org/TR/xsl/.

[21] Sun Microsystems, http://java.sun.com.

[22] Laakso T.I., "How to write a Diploma Thesis," Helsinki University of Technology, 1999.

[23] R. Gimson et al., "Device Independence Principles," W3C Working Draft 18 Sep 2001, http://www.w3.org/TR/di-princ/.

[24] S. H. Maes and T. V. Raman, "Position paper for the W3C/WAP Workshop on the Multi-modal Web," Last modified Oct 2000, http://www.w3.org/2000/09/Papers/IBM.html.

[25] C. Arehart et al., "Professional WAP," Wrox Press 2000.

[26] I. Vatton, "Welcome to Amaya, W3C's Editor/Browser," Last Updated Dec, 2001.

[27] Apache Software Foundation, The Apache XML project, Apache Cocoon, http://xml.apache.org/cocoon/.

[28] C. F. Goldfarb, P. Prescod, "The XML handbook," 3rd edition, Prentice Hall 2001.

[29] D. Ragget, "Clean up your Web pages with HTML TIDY," Last modified Jan 2002, http://www.w3.org/People/Raggett/tidy/.

[30] Business Week online, "One the Web, Experience is the Brand," Oct 1999, http://www.businessweek.com/ebiz/9910/dm1029.htm.

[31] M.H. Butler, "Current technologies for Device Independence," Apr 2001, http://www.hpl.hp.com/techreports/2001/HPL-2001-83.pdf.

[32] Forum Nokia, "Nokia Wap phones, introduction to the Wap phones," 2001, registration needed to the site, http://forum.nokia.com/wapforum/main/1,,1_1_75,00.html.

[33] A. Teppo and P. Vuorimaa, "Speech interface implementation for XML browser," accepted to the 2001 International Conference on Auditory Display, Espoo, Finland, July 29-August 1, 2001, pp. 272-275.

[34] MetaTV inc., "FORD iTV Cross-Platform TV Service," http://www.metatv.com/downloads/ford_profile.pdf.

[35] W. Chisholm, "Web Content Accessibility Guidelines 1.0," w3C Recommendation, May 1999, http://www.w3.org/TR/WAI-WEBCONTENT/.

[36] S. Pemberton et al., "XHTML 1.0: The Extensible HyperText Markup Language, A Reformulation of HTML 4 in XML 1.0," W3C Recommendation, Jan 2000, http://www.w3.org/TR/xhtml1/.

[37] M. Parkkonen, "Implementation of multi channel info services for different distribution channels," Helsinki University of Technology, Electrical and Communications Engineering, Masters Thesis, 2001.

[38] Wireless Application Protocol (WAP) Forum, "WAP 1.2 Specification Suite," 1999.

[39] M. Honkala, "Using XML to Develop Applications for WAP and WWW environments," Helsinki University of Technology, Department of Computer Science and Engineering, Master Thesis, 2000.

[40] European Computer Manufacturers Association (ECMA), "ECMAScript Language Specification," 3rd edition Dec 1999, ftp://ftp.ecma.ch/ecma-st/Ecma-262.pdf.

[41] K. Kivikoski, "XML-verkkotekniikan mullistaja? Standardiperheen esittely ja tilanne Suomessa 1999," Teknillinen Korkeakoulu, koulutuskeskus Dipoli, 1999.

[42] H. Wium Lie et al., "Cascading Style Sheets, level 1," W3C Recommendation 17 Dec 1996, revised 11 Jan 1999, http://www.w3.org/TR/REC-CSS1.

[43] I Jacobs, "About the World Wide Web Consortium (W3C)," Last updated Dec 2001, http://www.w3.org/Consortium/.

[44] International Organization for Standardization, Standard Generalized Markup Language (SGML), ISO 8879.

[45] C.M. Sperberg-McQueen and Lou Burnard, "A Gentle Introduction to SGML," Sep 1999, http://www.tei-c.org/P4beta/index.htm.

[46] J. Bosak, "An SGML-Based Web Server," Fifth International World Wide Web Conference 1996 Paris, France, Last updated May 1996, http://www5conf.inria.fr/fich_html/slides/dday/sgml/overview.htm.

[47] N. Walsh, "What Do XML Documents Look Like," Oct 1998, http://www.xml.com/pub/a/98/10/guide0.html.

[48] Refsnes Data, "Introduction to DTD," http://www.w3schools.com/dtd/dtd_intro.asp.

[49] T. Bray et al, "Namespaces in XML," W3C recommendation, Jan 1999, http://www.w3.org/TR/REC-xml-names/.

[50] H. S. Thompson et al., "XML Schema Part 1: Structures," W3C Recommendation, May 2001, http://www.w3.org/TR/xmlschema-1/.

[51] P. V. Biron et al., "XML Schema Part 2: Datatypes," W3C Recommnendation, May 2001, http://www.w3.org/TR/xmlschema-1/.

[52] D. Brownell, "About SAX," Last modified Jan 2002, http://www.saxproject.org/.

[53] B. McLaughlin, "Java and XML," O'reilly, 2000.

[54] P. Vuorimaa and T. Ropponen, and N. von Knorring, "X-Smiles XML browser," the 2nd International Workshop on Networked Appliances, IWNA'2000, New Brunswick, NJ, USA, Nov. 30 – Dec. 1, 2000.

[55] V. Apparao et al., "Document Object Model (DOM) Level 1 Specification," Oct 1998, http://www.w3.org/TR/REC-DOM-Level-1/.

[56] D. Veillard, "The XML C library for Gnome, The tree output," Last modified Feb 2002, http://www.xmlsoft.org/tree.html.

[57] J. Clark et al., "XSL Transformations (XSLT)," W3C Recommendation, Nov 1999, http://www.w3.org/TR/xslt.

[58] Refsnes Data, "XSL languages," http://www.w3schools.com/xsl/xsl_languages.asp.

[59] H. Williamson, "XML, The Complete Reference," Osborne/McGraw-Hill, 2001.

[60] J. Clark et al., "XML Path Language (XPath)," W3C Recommendation, Nov 1999, http://www.w3.org/TR/xpath.

[61] The Apache Software Foundation, "FOP," http://xml.apache.org/fop/index.html.

[62] M. Leventhal, "XSL Considered harmful," http://www.xml.com/lpt/a/1999/05/xsl/xslconsidered_1.html.

[63] D. Ragget et al., "HTML 4.01 Specification," W3C Recommendation 24 Dec, 1999, http://www.w3.org/TR/html401.

[64] O. Lassila et al., "Resource Description Framework (RDF) Model and Syntax Specification," W3C Recommendation, 22 Feb 1999, http://www.w3.org/TR/REC-rdf-syntax/.

[65] M. Nilsson et al., "Composite Capabilities/Preference Profiles: Requirements and Architecture," W3C Working Draft 21 Jul 2000, http://www.w3.org/TR/CCPP-ra/.

[66] M. Altheim et al., "XHTML 1.1 – Module-based XHTML," W3C Recommendation 31 May 2001, http://www.w3.org/TR/xhtml11/.

[67] J. Ayars et al., "Synchronized Multimedia Integration Language (SMIL 2.0)," W3C Recommendation, Aug 2001, http://www.w3.org/TR/smil20/.

[68] X-Smiles, http://www.x-smiles.org.

[69] K. Pihkala, N. von Knorring, and P. Vuorimaa, "SMIL in X-Smiles," to the Int. Conf. on Distributed Multimedia Systems, Taipei, Taiwan, Sept. 26-28, 2001.

[70] M. Dubinko, "Xforms 1.0," W3C Working Draft, Jan 2002, http://www.w3.org/TR/xforms/.

[71] M. Honkala and P. Vuorimaa, "XForms in X-Smiles," in Proc. the 2nd Int. Conf. on Web Information Systems Engineering, Kyoto, Japan, Dec. 3-6, 2001.

[72] Cisco, Cisco CTE1400 series, Content Transformation Engine, Product Overview, http://newsroom.cisco.com/ekits/cte_1400/cte1400_white_paper.pdf.

[73] MobileAware, Everix Mobility Server, Last modified Jan 2002, http://www.mobileaware.com/products.htm.

[74] Volantis Systems, Volantis Mariner, Adaptive Tag Library, http://www.volantis.com/voladptag.jsp.

[75] H. Huomo, "Ubiquitous Communication," presentation held at Telecommunication Forum 9th Oct 2001, at Helsinki University of Technology, Last updated Dec 2001, http://www.tct.hut.fi/opetus/s38001/s01/materiaali/2/presentations.shtml.

[76] P. Vuorimaa, "An XML based mobile software architecture," in Proc. 2nd Int. Symposium on Mobile Multimedia Systems & Applications, MMSA2000, Delft, The Netherlands, Nov. 9-10, 2000, pp. 150-156.

[77] Sonera, "Content Gateway Overview," version 2.1 Feb 2001, http://www.sonera.fi/yrityksille/matkapuhelinpalveluntarjoajat/.

[78] Finnish communications Regulatory Authority, Order 35 F/2001 M, "Order of the barring classes in telecommunications," http://www.ficora.fi/suomi/document/THK35E2000%20M.pdf.

[79] Sonera, "Development guidelines for Sonera.net environment," Internal Publication, Jan 2001.

[80] Sonera, "Sonera.net billing guideline (java API/TDB)," Internal Publication, Sep 2001.

[81] Sonera, "Ohjeistus palveluiden toteuttamiseksi Sonera.net ympäristöön," Internal Publication, Jan 2001.

[82] Sonera, "Sonera.net WWW ohjeistus," Internal Publication, Jan 2001.

[83] J. Zawinski, "user-agent strings," Mar 98, http://www.mozilla.org/build/user-agent-strings.html.

[84] Apache Software Foundation, Apache HTTP server project, http://httpd.apache.org/.

[85] Refsnes Data, http://www.w3schools.com/schema/schema_example.asp.

APPENDICES

APPENDIX A. XML document

XML document used in use case "fetchonewithid" in XMLAddrbook

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<kontaktit>
<msid>0405909005</msid>
<CMD>onewithid</CMD>
<url>/kontaktit/kontaktit.xml</url>
<service_url>/kontaktitsmswap/XSLServlet?service=Kontaktit</service_url>
<info>Kontakti haettu id:n perusteella</info>
<response_code>2</response_code>
<kontakti>
<id>1000099818</id>
<ETUNIMI>Teemu</ETUNIMI>
<SUKUNIMI>Teekkari</SUKUNIMI>
<ALIAS>Tepa</ALIAS>
<PUHELIN>0123456789</PUHELIN>
<FAX />
<SAHKOP>teemu@postiluukku.fi</SAHKOP>
<OSOITE>Jämeräntaival 50</OSOITE>
<POSTI>02150 Espoo</POSTI>
</kontakti>
</kontaktit>
```

APPENDIX B. DTD for the XML document

XML document is used in use case "fetchonewithid" in XMLAddrbook, XML document obeying this DTD can be seen in APPENDIX A.

```
<!ELEMENT kontaktit (msid, CMD, url, service_url, info, response_code, kontakti)>

<!ELEMENT msid (#PCDATA)>

<!ELEMENT CMD (#PCDATA)>

<!ELEMENT CMD (#PCDATA)>

<!ELEMENT CMD (#PCDATA)>

<!ELEMENT CMD (#PCDATA)>

<!ELEMENT kontakti (id, ETUNIMI, SUKUNIMI, ALIAS, PUHELIN, FAX,
SAHKOP, OSOITE, POSTI)>

<!ELEMENT id (#PCDATA)>

<!ELEMENT EUNIMI (#PCDATA)>

<!ELEMENT SUKUNIMI (#PCDATA)>

<!ELEMENT ALIAS (#PCDATA)>

<!ELEMENT PUHELIN (#PCDATA)>

<!ELEMENT FAX (#PCDATA)>

<!ELEMENT SAHKOP (#PCDATA)>

<!ELEMENT OSOITE (#PCDATA)>

<!ELEMENT POSTI (#PCDATA)>
```

APPENDIX C. XSL sheet for the XML document

XSL sheet is used in XMLAddrbook's media type "wml", in its all use cases

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output doctype-system="http://www.wapforum.org/DTD/wml_1.1.xml" doctype-
public="-//WAPFORUM//DTD WML 1.1//EN" />

<xsl:template match="kontaktit">

<wml>

<template>

<do type="prev">

<prev />

</do>

</template>

<card id="a" newcontext="true" title="Kontaktit">

<p>

<xsl:choose>

<xsl:when test="(CMD='fetchwithkeyword')">

<xsl:if test="(response_code='3')">Anna hakusana (nimi, alias tai
puhelinnumero).</xsl:if>

<xsl:if test="(response_code='2')">

<xsl:for-each select="kontakti">

<a>

<xsl:attribute name="href">

<xsl:value-of select="/kontaktit/service_url" />

&msid=

<xsl:value-of select="/kontaktit/msid" />

&id=

<xsl:value-of select="id" />
```

```
&CMD=onewithid
</xsl:attribute>
<xsl:value-of select="@index" />
.
<xsl:value-of select="SUKUNIMI" />
<xsl:text />
<xsl:value-of select="ETUNIMI" />
</a>
<br />
</xsl:for-each>
<xsl:if test="(from_index != 'end')">
<a>
<xsl:attribute name="href">
<xsl:value-of select="/kontaktit/service_url" />
&from_index=
<xsl:value-of select="from_index" />
&to_index=
<xsl:value-of select="to_index" />
&CMD=hae&fetchparam1=
<xsl:value-of select="fetchparam1" />
&fetchparam2=
<xsl:value-of select="fetchparam2" />
</xsl:attribute>
hae seuraavat
</a>
</xsl:if>
</xsl:if>
<xsl:if test="(response_code='1')">Ei hakua vastaavia kontakteja</xsl:if>
```

```
</xsl:when>
<xsl:when test="(CMD='fetchonewithid')">
Etunimi:
<xsl:value-of select="./kontakti/ETUNIMI" />
<br />
Sukunimi:
<xsl:value-of select="./kontakti/SUKUNIMI" />
<br />
Alias:
<xsl:value-of select="./kontakti/ALIAS" />
<br />
Matkapuhelin:
<xsl:value-of select="./kontakti/PUHELIN" />
<br />
Fax:
<xsl:value-of select="./kontakti/FAX" />
<br />
Sähköposti:
<xsl:value-of select="./kontakti/SAHKOP" />
<br />
Lähiosoite:
<xsl:value-of select="./kontakti/OSOITE" />
<br />
Postiosoite:
<xsl:value-of select="./kontakti/POSTI" />
<br />
<xsl:element name="a">
<xsl:attribute name="href">
```

```
<xsl:value-of select="/kontaktit/service_url" />
&CMD=muuta&id=
<xsl:value-of select="./kontakti/id" />
</xsl:attribute>
Muokkaa
</xsl:element>
<xsl:element name="a">
<xsl:attribute name="href">
<xsl:value-of select="/kontaktit/service_url" />
&CMD=poista&id=
<xsl:value-of select="./kontakti/id" />
</xsl:attribute>
Poista
</xsl:element>
</xsl:when>
<xsl:when test="(CMD='update' or CMD='new')">
Etunimi:
<input title="kirjoita" name="ETUNIMI" emptyok="true">
<xsl:attribute name="value">
<xsl:value-of select="kontakti/ETUNIMI" />
</xsl:attribute>
</input>
Sukunimi:
<input title="kirjoita" name="SUKUNIMI" emptyok="true">
<xsl:attribute name="value">
<xsl:value-of select="kontakti/SUKUNIMI" />
</xsl:attribute>
</input>
```

Alias:

<input title="kirjoita" name="ALIAS" emptyok="true">

<xsl:attribute name="value">

<xsl:value-of select="kontakti/ALIAS" />

</xsl:attribute>

</input>

Matkapuhelin:

<input title="" name="PUHELIN" emptyok="true" format="*N">

<xsl:attribute name="value">

<xsl:value-of select="kontakti/PUHELIN" />

</xsl:attribute>

</input>

Fax:

<input title="" name="FAX" emptyok="true" format="*N">

<xsl:attribute name="value">

<xsl:value-of select="kontakti/FAX" />

</xsl:attribute>

</input>

Sähköposti:

<input title="kirjoita" name="SAHKOP" emptyok="true">

<xsl:attribute name="value">

<xsl:value-of select="kontakti/SAHKOP" />

</xsl:attribute>

</input>

Lähiosoite:

<input title="kirjoita" name="OSOITE" emptyok="true">

<xsl:attribute name="value">

```
<xsl:value-of select="kontakti/OSOITE" />
</xsl:attribute>
</input>
Postiosoite ja -toimipaikka:
<input title="kirjoita" name="POSTI" emptyok="true">
<xsl:attribute name="value">
<xsl:value-of select="kontakti/POSTI" />
</xsl:attribute>
</input>
<xsl:if test="(CMD='uusi')">
Lähetä kontaktille numerosi:
<select name="SENDSMS">
<option value="no">Älä lähetä</option>
<option value="yes">Lähetä</option>
</select>
Nimesi:
<input title="kirjoita" name="ANAME" maxlength="30" emptyok="true" />
</xsl:if>
<anchor>
<xsl:if test="(CMD='update')">
Muuta
<br />
</xsl:if>
<xsl:if test="(CMD='new')">Lisää</xsl:if>
<go href="http://seadra.mobile.sonera.net:8090/servlets/XSLServlet" method="post">
<xsl:if test="(CMD='update')">
<postfield name="CMD" value="saveupdated" />
</xsl:if>
```

```
<xsl:if test="(CMD='new')">
<postfield name="CMD" value="savenew" />
<postfield name="SENDSMS" value="$(SENDSMS)" />
<postfield name="ANAME" value="$(ANAME)" />
</xsl:if>
<postfield name="service" value="Kontaktit" />
<postfield name="ETUNIMI" value="$(ETUNIMI)" />
<postfield name="SUKUNIMI" value="$(SUKUNIMI)" />
<postfield name="ALIAS" value="$(ALIAS)" />
<postfield name="PUHELIN" value="$(PUHELIN)" />
<postfield name="FAX" value="$(FAX)" />
<postfield name="SAHKOP" value="$(SAHKOP)" />
<postfield name="OSOITE" value="$(OSOITE)" />
<postfield name="POSTI" value="$(POSTI)" />
<postfield>
<xsl:attribute name="name">id</xsl:attribute>
<xsl:attribute name="value">
<xsl:value-of select="kontakti/id" />
</xsl:attribute>
</postfield>
</go>
</anchor>
</xsl:when>
<xsl:otherwise>
<xsl:if test="(response_code='-4')">Palvelussa on tilapainen häiriö, yritä hetken
kuluttua uudestaan!</xsl:if>
<xsl:if test="(CMD='savenew')">
```

```
<xsl:if test="(response_code='3')">Kontakti on lisätty. Matkapuhelinnumerosi on
lähetetty kontaktille.</xsl:if>

<xsl:if test="(response_code='2')">Kontakti on lisätty.</xsl:if>

<xsl:if test="(response_code='4')">Kontaktin matkapuhelinnumero ja/tai nimesi
puuttuu. Anna numero ja/tai nimi.</xsl:if>

<xsl:if test="(response_code='6')">Annoit kontaktille lähetettävään tekstiviestiin
nimesi mutta et valinnut viestiä lähetettäväksi. Valitse viesti lähetettäväksi.</xsl:if>

<xsl:if test="(response_code='5')">Sukunimi on pakollinen tieto. Anna
sukunimi.</xsl:if>

</xsl:if>

<xsl:if test="(CMD='saveupdated')">

<xsl:if test="(response_code='2')">Kontaktin tiedot on muutettu.</xsl:if>

<xsl:if test="(response_code='5')">Kontaktia ei voitu tallentaa koska siitä puuttui
pakollinen sukunimi.</xsl:if>

</xsl:if>

<xsl:if test="(CMD='remove')">Kontakti on poistettu.</xsl:if>

</xsl:otherwise>

</xsl:choose>

<xsl:element name="a">

<xsl:attribute name="href">

<xsl:value-of select="url" />

</xsl:attribute>

Kontaktit

</xsl:element>

</p>

</card>

</wml>

</xsl:template>

</xsl:stylesheet>
```

APPENDIX D. Example of an XML Schema

This example is taken from [85].

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

        <xs:element name="shipOrder" type="order"/>

        <xs:complexType name="order">
         <xs:sequence>
          <xs:element name="shipTo" type="shipAddress"/>
          <xs:element name="items" type="cdItems"/>
         </xs:sequence>
        </xs:complexType>

        <xs:complexType name="shipAddress">
         <xs:sequence>
          <xs:element name="name" type="xs:string"/>
          <xs:element name="street" type="xs:string"/>
          <xs:element name="address" type="xs:string"/>
          <xs:element name="country" type="xs:string"/>
         </xs:sequence>
        </xs:complexType>

        <xs:complexType name="cdItems">
         <xs:sequence>
          <xs:element name="item" type="cdItem"
                  maxOccurs="unbounded"/>
         </xs:sequence>
        </xs:complexType>
```

```
<xs:complexType name="cdItem">
 <xs:sequence>
  <xs:element name="title" type="xs:string"/>
  <xs:element name="quantity" type="xs:integer"/>
  <xs:element name="price" type="xs:decimal"/>
 </xs:sequence>
</xs:complexType>

</xs:schema>
```

Example of an XML document conforming to this Schema[85]:

```
<?xml version="1.0"?>
<shipOrder>
 <shipTo>
  <name>Tove Svendson</name>
  <street>Ragnhildvei 2</street>
  <address>4000 Stavanger</address>
  <country>Norway</country>
 </shipTo>
 <items>
  <item>
   <title>Empire Burlesque</title>
   <quantity>1</quantity>
   <price>10.90</price>
  </item>
  <item>
```

```
    <title>Hide your heart</title>
    <quantity>1</quantity>
    <price>9.90</price>
  </item>
 </items>
</shipOrder>
```

APPENDIX E. Example configurations for the Framework

```
LOGPATH:/services/framework/log/

LOGFILEPREFIX:framework

LOGFILESUFFIX:log

# media types that Framework recognizes

media_types:html,wml

media_type_default:html


html_MSID_carrier:http_header

html_MSID_http_header_name:Cookie:Sonera-msisdn


wml_MSID_carrier:http_query_param

wml_MSID_http_query_param_name:MSISDN

############################################################

# media types are identified from the value of http header
# user_agent_header, the #order is significant

############################################################

user-agents_html:MSIE, Mozilla, Netscape

user-agents_wml:Nokia, Wap, Ericsson

############################################################

# content types for different media
```

```
############################################################c
ontent-type_html:text/html; charset=ISO-8859-1

content-type_wml:text/vnd.wap.wml; charset=ISO-8859-1


############################################################

#       http query parameter names that identify the

#       service and use case to be called

#       for example:

#
        http://x.y.z/servlet/XSLServlet?service=Addrbook&CM
        D=update........

############################################################

service_param_name:service

use case_element_name:CMD

resp_code_element_name:response_code

#  http url param where the subscriber id is appended to the
original query string

msid_param_name:msid

# http url param where the customer id is appended to the
original query string

custid_param_name:custid

############################################################

#       configuration files of the services
```

```
#         in these files are use cases (CMD-parameter values)
          linked with XSL-sheets

##################################################################

services:/services/addrbook/conf/addrbook.cfg
```

APPENDIX F. Example configurations for the service integration module in the Framework

```
# class that fetch the content from the actual service

loaded_class:xmladdrbook.IntegrationModule

# name that identifies the service in http query parameter
"service"

name:XMLAddrbook

# url that holds the actual service

url:http://machine.x.y.net:8090/xmladdrbook/XMLAddrbook

#########################################################

# supported media types

#########################################################

media_types:html,wml,sms

#########################################################

# http query parameter "CMD" tell which XSL-sheet is used

# different media types can have different CMD:s

#

# these configurations must be for each media type:

# 1. XSL_PATH_mediatype:

# 2. mediatype_CMDS:

# 3. mediatype_pages:

# 4. one-to-one/one-to-many mapping (xsl sheet file-use
case/ xsl sheet
```

```
# file-use cases):

#########################################################

XSL_PATH_html:/opt/xmladdrbook/xsl/html/

html_CMDS:fetchwithkeyword, fetchonewithid, update,
saveupdated, remove, new, savenew

html_pages:fetchwithkeyword.xsl fetchonewithid.xsl,
update.xsl, saveupdated.xsl, remove.xsl, new.xsl,
savenew.xsl

fetchwithkeyword.xsl:fetchwithkeyword

fetchonewithid.xsl:fetchonewithid

update.xsl:update

saveupdate.xsl:saveupdate

remove.xsl:remove

new.xsl:new

savenew.xsl:savenew


XSL_PATH_wml:/opt/xmladdrbook/xsl/wml/

wml_CMDS:fetchwithkeyword,fetchonewithid,update, saveupdate,
remove, new, savenew

wml_pages:xmladdrbook-all-wml.xsl

xmladdrbook-all-wml.xsl:fetchwithkeyword, fetchwithid,
update, saveupdated, remove, new, savenew
```

```
XSL_PATH_sms:/opt/xmladdrbook/xsl/sms/

sms_CMDS:fetchwithkeyword,savenew

sms_pages:xmladdrbook-all-sms.xsl

xmladdrbook-all-sms.xsl:fetchwithkeyword,savenew
```