Dynamic Web-Based User-Interfaces based on Semantic Descriptions and Context Informations

Diploma Thesis

Institute of Computer Science III

by:	Merlin J. Fotsing
field of study:	Advanced Software Engineering
matriculation number:	1506897
advisor:	Prof. Dr. Armin B. Cremers
second advisor:	MdC Anne-Marie Dery-Pinna
Assistance:	Dipl.-Inf. Pascal Bihler

Abstract

According to the GSM Association, the number of mobile subscribers is growing at around 10% rate, each year [TM. 2006; cellular news.com 2008]. One of the outcomes of this is the rapid growth of mobile Internet, which is foremost due to an increasing number of sophisticated hand held devices, which are now provided with unlimited Internet access. Due to this fact, dynamic Web-based applications constitute a serious alternative to close the functionality gap of original mobile applications on different devices.

However, by delivering Web applications to mobile devices, some presentation requirements have to be fulfilled during the UI adaptation process. These issues include the content types, the network and device capabilities, such as the delivery context. To tackle the challenges of the presentation requirements in a flexible way, we propose a development approach based on an abstract SUID called *LAIM*. While using the LAIM to describe the UI, *CTs* are used to translate and adapt the abstract UI into a Web UI. Ajax is used to support the Client Server Interaction. This helps to encapsulate significant application functions and to ensure that only the data needed is retrieved from the server without reloading the whole application.

With this logic based approach, the complexity of the Web application is low and the representation of context values in logic databases could easily be included into the adaptation process. With this combination of logic programming and *SUIDs*, code re-use as well as UI usability is improved.

Zusammenfassung

Nach Angaben der `GSM Association` steigt die Anzahl der Mobilfunkkunden jährlich um etwa 10 Prozent. Durch Investitionen der Mobilfunkkonzerne in die Datennetze und die Entwicklung von neuen Endgeräten steigt die Verwendung von mobilen Anwendungen in den letzten Jahren. Die sog. Daten-Flatrates stellen die Grundlage für diese Applikationen dar. Dadurch ist es den Firmen möglich Anwendungen losgelöst von der verwendeten Hardware der mobilen Geräte zu entwickeln. Die mobilen Geräte stellen aber auch neue Herausforderungen an den Design der UI's. Dies sind unter anderem:

- Art des Inhaltes, welcher angezeigt werden soll.
- Die Netzwerk- und Gerätefunktionen der mobilen Geräte,
- und der Kontext in dem die Anwendung ausgeführt werden soll.

Um den Herausforderungen der Präsentation der Anwendung in einer flexiblen Art und Weise gerecht zu werden, schlagen wir ein Entwicklungskonzept auf der Grundlage einer abstrakten SUID, wie die in dieser Arbeit beschriebene *LAIM*-Implementierung, vor. Die Benutzeroberfläche wird mit dieser abstrakten Sprache beschrieben. Diese Beschreibung wir mit Hilfe der ebenfalls in dieser Arbeit vorgestellten *CTs* in eine Weboberfläche übersetzt.
Zur Unterstützung des Client-Server-Interaktion haben wir die AJAX Architektur verwendet. Die Vorteile von AJAX sind u.a. die Kapselung von verschiedenen Funktionen einer Applikation und die Fähigkeit nur die relevanten Dokumente von Server neu anzufragen und auch zu laden. Dies führt zu einer Verminderung der mobilen Datenlast.
Durch die in dieser Arbeit eingeführten Konzepte insbesondere der logische Programmieransatz verringert die Komplexität bei der Herstellung, Anpassung und Auslieferung von Webanwendungen auf verschiedenen Endgeräten. Da die benötigten Kontextinformationen problemlos in die logische Struktur eingefügt werden können und damit einfach in den Anpassungsprozess mit einfließen können. Dadurch wird die Wiederverwendung des Quellcodes und damit die Usability der Benutzeroberfläche gesteigert.

Danksagung

Ich danke Herrn Prof. Dr. Armin B. Cremers für die Möglichkeit diese Diplomarbeit im Bereich der Advanced Software Engineering anfertigen zu dürfen sowie für die wichtigen Einsichten und Hinweise, die ich durch ihn in diesem Gebiet erhalten habe.
Für seine fachlich wie menschlich ausgezeichnete Betreuung vor und während der Entstehung dieser Diplomarbeit möchte ich Herrn Dipl.-Inf. Pascal Bihler sehr herzlich danken. Die konstruktive Diskussion mit ihm ist das Fundament dieser Diplomarbeit, die ohne ihn nicht möglich gewesen wäre. Sowohl seine Vorschläge als auch seine Kritik haben mich stets motiviert und dabei geholfen, den Inhalt der Arbeit weiter zu verbessern.
Ebenso bedanke ich mich bei den Herren Dr. Günter Kniesel, Dipl.-Inf. Tobias Rho, Dipl.-Inf. Mark Schmatz sowie Dipl.-Math. Daniel Speicher für ihren fachlichen Rat und die Gespräche, durch die ich meinen Einblick in den Themengebieten weiter vertiefen konnte.
Außerdem möchte ich mich bei Marcel Becker, Dipl.-Inf. Mirko Esser, Willy Ngongang, Nataliya Pendzhurova, Helge Wessels und Dipl.-Inf. Daniel Wolff für die moralische Unterstützung und die Zusammenarbeit bei den Prüfungsvorbereitungen danken.
Weiterhin danke ich allen Mitarbeitern der Knowledge Discovery & Text Mining Group aus dem Fraunhofer-Institut für Intelligente Analyse- und Informationssysteme für ihre Ratschläge und Unterstützung.
Einen besonderen Dank möchte ich an meine Familie in Kamerun und meine Freundin, Frau Susan Strömbom richten, die mich immer in meinen Entscheidungen unterstützt haben und stets für mich da waren.

Contents

1 Introduction

> The Mobile Web Initiative's goal is to make browsing the Web from mobile devices a reality.
>
> *Tim Berners-Lee*

During the past years the capabilities of hand-held communication devices have grown amazingly. Almost all mobile devices are featured with advanced technology like a high-resolution color display and a tiny built-in browser to access the Web. Whereby the consequential side effect is the increasing growth of `mobile Web`. Unfortunately, there are no standard screen resolutions for mobile devices like there are for a desktop PC. Furthermore, the embedded browsers are created to take into account the low bandwidth abilities, the limited display capabilities and the lower memory capacity in hand-held mobile devices. Furthermore as the case of the desktop browser, `Web standards` interpretation differs between different mobile Web browsers. To ensure the UI usability and mobility [Johnson 1998], all these limitations and challenges are to be taken into account when delivering dynamic Web applications to specific devices. In addition, during the adaptation process, the user goals and the delivery context are also to be taken into consideration. Since mobile data transfer often costs money, the user's goal is often to access to the needed information in a specific context at the lowest cost. The context on the other hand is determined by the position and environmental factors of the user (e. g. sitting in a coffee shop, a bus, during jogging)

To deal with device constraints, with the purpose to deliver a device independent content, it is possible as a first approach to use a software system like `digestor` [Bickmore and Schilit 1997] or `Pocket News` [Hong et al. 2003], to automatically build an arbitrarily requested Web content for a specific device. But the procedural methods of these tools are not appropriate to adapt Web sites with a lot of core functionalities. In the second approach, the use of an intermediate SUID language based on `XML`, such as `UsiXML` [Limbourg et al. 2004] or `XIML`, [Puerta and Eisenstein 2002] is a better option. According to this approach, a separation between core functionality, UI requirements, and their visual appearance is guaranteed. Unfortunately, there is a relative overhead by integrating such markup languages in the adaptation process. To bridge the overhead gap at design time, the use of a relative simple semantic UI

language called `LAIM` [Bihler et al. 2008], which was defined by Pascal Bihler and Cédric Joffroy, will be examined. Using the `LAIM`, data obtained in an intermediate step of the adaptation process via powerful techniques will be parsed into a dynamic adaptable Web UI.
This research work aims to handle the user interactions with a Web application, where the application's UI is rendered `on demand` on the server and sent to the client. Using `Ajax`, just a `partial update` of the application should be needed and depending on the context of use, rendering a whole application in advance might also be possible [opengardensblog 2006; Garrett; Wei].

In section 2, we will first present, the motivation and the goals of this research paper. After that, in section 3 we will present the results of the research done in attempt to achieve the DIW goal. In order to transform the input model, we chose to use CTs instead of traditional model transformation based on XSLT. For this reason, the model transformation using CTs is presented in section 4. Thereby, we explained how the transformation rules requirements specified in section 7, are achieved. Since the UI Specification encompasses the definition of the UI functionality and the UI model, the UI functionality is specified in section 5. Thereupon, the UI model specification is made in section 6. In order to show that our research approach objective was achieved, we will present an overview of our implementation approach in section 8. Therewith, we illustrated that CTs are also suitable for model transformations.

2 Thesis Scope

In this section we will present the motivation and the goals of this research paper. Firstly, the backgrounds and scenarios are presented, in order to underline the importance of our research efforts aiming at closing of the functionality gap between original mobile and desktop applications.

2.1 Background

First of all, the main objective of this work is not to show that model transformations based on CTs are to be suited as a traditional model transformation using XSL transformations. However, the background here is to demonstrate that CTs, which are used to transform static programs, are also suitable for model transformations in a dynamic runtime environment. Nevertheless, as XSL is a W3C recommended model transformation language [Clark 1999], we explain in section 2.4 the reason why we choose CTs instead of XSL as model transformation language.

2.2 Motivation

To capture the developer's intentions, the major requirement is a declarative language, which provides a strict separation between content, structure, layout, presentation, and UI functionalities. For this purpose, there are many community languages such as RIML, DIAL or SIML. These languages fulfill the requirements of the device independence principle. For example, they allow the developer to integrate semantic meta-informations and also offer guidelines for a generic description of the dialog unit. Furthermore, they embody options for referencing context informations within the dialog.
All these allow the developer to have a full control over the adaptation process. However, the relation between the delivered unit, the presentation, and the structure is still hard-coded within the model definition [Ziegert et al. 2004; Kirda and Kerer 2004; Ku et al. 2005]. Many others are based on a series of indirections or

bindings (used to link together content and layout) [Giannetti 2002]. Other languages are platform dependent and need to be executed into a specific environment (e. g. `XAML`). Some of them are abstracts from execution environments but not from widgets (e. g. `XIML`). Furthermore, most of them are proprietary developments and the combination with other technologies such as XHTML, Ajax and JavaScript is not supported.
For this reason an abstract SUID called LAIM [Bihler et al. 2008; Bihler and Kniesel 2007] has been developed. LAIM is sufficiently abstract in order to be applicable in the definition of desktop UIs and to support the definition of adaptable Web UIs on fixed or mobile devices.
In order to solve the inability to the `Write Once and Run Anywhere (WORA)` [Rajapakse] problem for mobile applications, Web applications can be built. These have many advantages, e. g. the Web server power is used to do most of the processing. Other advantages include that an external storage on databases can be used and that the device specific UI is generated and sent to the device by the Web server on demand. All these advantages can also be exploited using the approaches listed in section 3. Nevertheless, the context and situation of use are not always part of the adaptation process and the presented languages have their limitations. Currently the adaptation process often consists of transcoding [wapreview.com Mitt; Hwang et al. 2002], selection and content serialization. Even though the context is part of the delivery content, the languages used are not developed to follow the `WORA` principles for the mobile Web.

2.3 Scenarios presentation

Nowadays people use mobile devices for social interactions and for accessing private wireless data services (maps, news, emails, etc.) nearly everywhere at any time. Mobile devices can not only serve as an instrument for social interaction, but also play a significant role in the business world. Companies have a preference for desktop application solutions with incorporated Web interfaces, since these applications can be used anywhere via the Internet. Beside companies, individuals tend to prefer mobile applications, which can also be executed on desktop PCs like the following scenario shows.
Scenario: Mobile Web-based MP3 player
Anna owns a touch-based mobile phone with wireless Internet connection. She has subscribed to a remote mp3-service which works on her phone like a traditional mp3-player, but receives all music from the Web and her phone is controllable via Web-

Interface.
In the morning, Anna likes to go for a little jogging tour. She prepares the play list for her tour in advance using her Kitchen-PC with Web connection. Here, the interface can be displayed with a lot of details, allowing extensive library browsing, cover-flow, user comments etc.
During her jogging tour, the interface is rendered on her mobile device. Taking into account the limited display size, most negligible UI elements are not displayed. With motion sensors, the device indicates a jogging context, so primary UI elements are enlarged to allow easy interaction while on the move.
Exhausted from her jogging tour, Anna finishes the trip in a small coffee shop. Sitting there, she can interact with her UI more precisely. Therefore, UI elements can be displayed with their normal size and more control components become visible on the mobile device's interface.
To achieve such a remote Web application we propose a development approach based on server-side adaptation using an abstract SUID to define the application [Bihler et al. 2008].

2.4 User Interface - (LAIM) Translation

Because of the many variants of platform- and context specific adaptations that may be necessary in arbitrary applications, there is an exponential number of possible combinations. Therefore, implementing adaptations for specific combinations is prohibitive. Instead, we need the ability to build modular and individual adaptations, and to compose those adaptations when needed. In particular, we must face the following challenges [Bihler et al. 2008; Kniesel 2006b]:

Modularity and compositionality: It must be possible to specify each adaptation separately and to compose complex adaptations by reusing existing ones. There must be no need to modify existing adaptations for this purpose. In particular, composition must not be limited to a fixed set of anticipated combinations, for which specific hooks have to be hand-coded into each adaptation.

Interaction detection and resolution: If independently developed adaptations are deployed together, they might interact in ways not foreseen by their authors. To make composition possible, it is necessary to have an automated way of detecting and resolving interactions.

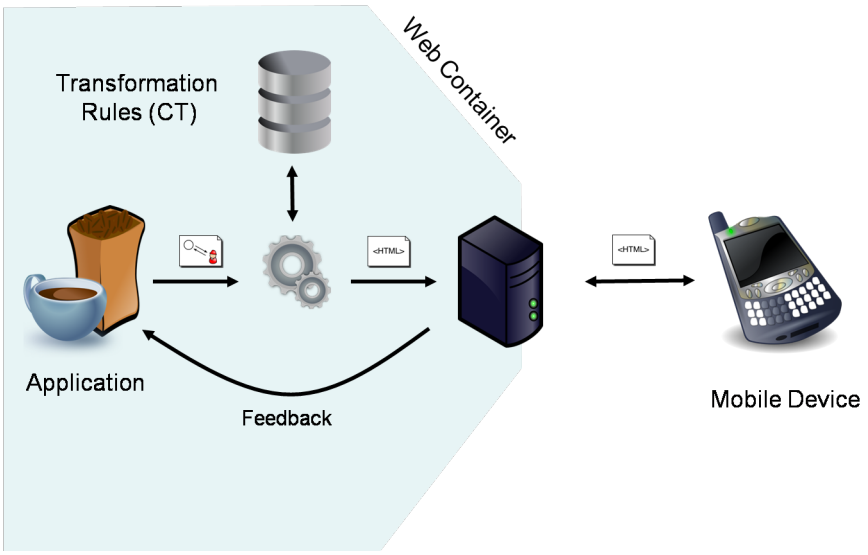

Figure 2.1: General architecture of proposed system: Semantic UI descriptions are adapted and transformed to Web standards. User events are directly delivered to the executed application.

2.5 Interaction Architecture Definition

The use of an SUID has many advantages. First of all, it helps to separate the UI adaptation process from the application core, it allows sharing of UI adaptation functionality among several applications, and finally it allows the Web application to delegate the tasks for interface adaptations to the appropriated layer.
To remain true to these principles, it has to be possible to use many adaptation layers or simply to change the given one as needed. To deal with the complexity of the adaptations required by different device capabilities and contexts, we propose the general architecture of a CT-enabled Web application as demonstrated in figure 2.1.
Referring to this architecture, the (mobile) device connects to an application server, in our case this might be an instance of Apache Tomcat. A Java Servlet acts as an entry point and interface to the Java based Web application. When it comes to UI visualization, the application hands over a semantic description of the required inputs, outputs, and actions to the rendering system. The handling chain is based on CTs. During the adaptation process, the abstract input model is translated into a concrete UI representation. The transformation is done according to the delivery context. Therefore transformation rules are defined and materialized as Prolog facts in the fact base (cp. section 7.1).

2.6 Client-Server Interactions

By using the combination of a SUID and the logic based approach to create the Web UI, the context data could easily be included in the adaptation process [Rho et al. 2006]. However when the context of use, such as the device capabilities, is part of the content delivery, the number of Web-service subscribers can increase exponentially day after day. Therefore, the server would be extremely overloaded for a highly interactive application. This leads to long processing delays, when the application would be completely rendered after each user input. To tackle this problem more automation techniques are needed to deliver interactive components that provide enhanced application function and an improved user experience [Schmidt et al. 2007]. To solve this problem we proposed a development approach including Ajax. Ajax, which is a combination of an asynchronous communication technique, JavaScript and XML, is used to encapsulate significant application functions in ways that are easy for authors to specify and control [opengardensblog 2006; alliance 2008a]. Ajax is a combination of several technologies:

Standards-based presentation Supports a presentation in the common standards using XHTML, CSS, etc.

DOM Dynamic display and interaction using the DOM, which is accessed with JavaScript.

XMLHttpRequest (XHR) For asynchronous client server communication. Thereby the XMLHttpRequest object is used to avoid page reloads.

XML and XSLT, or JSON To interchange and manipulate data.

JavaScript Programming language to pool everything together.

Using such an asynchronous communication, just the needed data can be obtained from the server without reloading the whole application. Additionally, JavaScript as a language to execute code in a browser, and XML for the ad-hoc manipulation of a Web application make it possible to support Client-Server interaction without explicitly clicking a control component on the application [Lewis 2007; alliance 2008b; Garrett].

3 State of the Art

> Gimson et al. [2003] describes the Device Independence Principles as: content and applications authored, generated or adapted for a better user experience when interacting with presentations via many different access mechanisms.
> Thereby the access mechanism, which is an intermediate between the user an the Web, can be a combination of hardware (including one or more devices and network connections) and software (including one or more user agents). This allows a user to perceive and interact with the Web using one or more interaction modalities (sight, sound, keyboard, voice etc.)

With the vision of a DIW, the *W3C* proposes a couple of authoring concepts, amongst which:

Delivery context: Device characteristics and user preferences.

Delivery unit: As a response to a single HTTP request, the delivery unit can be composed from a set of data(Markup languages such as XHTML), ECMAScript (JavaScript) and Objects like video or audio data.

Adaptation: Process of content selection, generation or modification according to the delivery context, which occurs on the server (server-side adaptation) or even on the client (client-side adaptation).

In order to achieve the DIW goal, many research attempts were done in the past. Before we start presenting their results, we will take a look at the way of capturing the device's delivery context. After that we will show how the delivery unit is obtained according to the content adaptation method and the delivery context chosen in those research projects.

3.1 Delivery Context

There are many standards to capture the device's delivery context. According to the *W3C* there are many existing approaches [Gimson et al. 2006], amongst which:

Composite Capabilities/Preferences Profiles (CC/PP): Profile to describe the device capabilities and user preferences [Kiss 2007]. According to Kiss [2007], the device capabilities and user preferences often refers to the device's delivery context and can be used to guide the adaptation of content presented to that device. This profile is often written in a RDF data model.

WURFL XML file listing capabilities of all known mobile devices [Passani 2008; Kindler 2007]. Nowadays, the WURL is provided with a framework for device detection. For this purpose the UA contained in the HTTP Header [Wagner and Paolucci] is detected via text mining or pattern matching. Using the UA, the needed device characteristics can be extracted from the database or from the XML file.

UAProf (User Agent Profile): As Open Mobile Alliance (OMA) standard [Forum 2001; Tran 2002], a UAProf which is based on CC/PP profiles, is an XML document that contains information about the UA type and device characteristics. UAProfs are stored in a server called profile repositories, which is maintained by mobile device manufacturers. Like the UA, the URL to the UAProf is also part of the HTTP Request Header.

These delivery context detection methods have its advantages and disadvantages. For instance, The UAProf method provides more detailed information about the wireless device and their capabilities, but it is not supported by old devices.
In this work the WURFL framework was used. The WURFL framework provides a useful API to look up the needed device characteristics from the data base. For example, as the implemented prototype is a music application, we query the database to find out if the required audio format (MP3) is supported. More precisely, whether the client is able to play sound provided in MP3 format. In addition, the other common needed capabilities comprise the:

- Screen size of the device.
- Supported image formats.
- Supported markup language (XHTML-MP, XHMTL, WML, etc.).
- Supported content type (`"application/javascript;charset=UTF-8","text/html"`, etc).
- Supported CSS standard.

3.2 Traditional Web-Application creation methods

To create a remote Web-based service accessible by specific devices, the first solution is to overcome the obvious standardization by creating a mobile-friendly Web-Application using `Mobile Web Best Practices` [Rabin and McCathieNevile 2008] and open standards. To guarantee a good user experience, many versions of the same Web site have to be created. While trying to connect to the *url* [Berners-Lee et al. 1994], one can be automatically redirected to the appropriate content directory. As there has to be a content directory created for each device type, code reuse is limited and application evolution is difficult. The second possibility is to use a software system like **digestor** [Bickmore and Schilit 1997] or **Pocket News**, to create or to translate a given Web page into a mobile-friendly one. However, the resulting Web pages are not able to support any kind of interactions between the user and the UI. Furthermore, the application is not able to adapt its response to a client-request in a device-sensitive way [Glover and Davies 2005].
The advantage of such an approach is the fast access to Web content for a specific device type. Nevertheless, the device description and capabilities are not part of the delivery context. This leads to a bad user experience with the application, since the user- and technical-goals are not tackled. In order to guarantee a good separation of concern between the UI functions and their presentation, **content adaptation** on demand provides a solution. Using the content adaptation approach the business-, user- and technical goals are taken into account to create a **One - Web** [Rabin and McCathieNevile 2008] content for all devices.

3.3 Content Adaptation

Adaptation, also called *multi serving*, is the action of transforming content by dynamically re-flowing it in accordance with the device capabilities [Wikipedia 2008]. Indeed, Content Adaptation is not just about an adaption to the device, it also takes into consideration the user preferences, context, and location, such as the network speed and bandwidth.
Currently, there are many different techniques to achieve a better device independence for Web applications. These techniques fall into three broad categories:

Intermediate or Pre-Adaptation: Based on techniques such as *LCD* [Kindler 2007]. It consists of creating many possible versions of the same Web page, which are optimized for many device classes, and version selection at runtime according to the HTTP-Header [Group 2002].

Client-side adaptation: Using this approach the adaptation occurs within the micro browser, whereas the content is delivered to the device without any prior transmission of client properties to the server [Koskimies 2004]

Server-side adaptation: The content is adapted and delivered from the server to the client at runtime due to the delivery context. This technique offers a high level of author control regarding the delivered content and enables device-independent authoring.

To separate the UI functionality from its visual appearance, *XML SUID* languages are used. To produce the most appropriate adaptation, content adaption is done on the server. Such a SUID allows the representation of the UI and its artifacts (formal functionalities) at different levels of abstraction and can contain transformation rules, which will be taken into consideration during the adaptation process.
There are many research approaches for device-independent authoring, which try to take into account and adapt the content accordingly upon delivery [Butler et al. 2002; Hanrahan and Merrick 2004; Manuel Cantera Fonseca and Hierro 2007]. According to the *DIWG*, new hand-held devices often have new capabilities and are able to exploit features of newer generations of networks, while connecting to the Web [Lewis Sond]. Due to that observation, more powerful abstraction mechanisms are required to ensure the usability of existing applications.
To solve this problem, many authors adopted the *declarative programming* [OpenWiki 2004; Torgersson 1996] concept. To fulfill the declarative programming requirements, many languages such as *XAML* [Foundation 2009], *XUL* [Goodger et al. 2008; Feldt 2007] or *JSF* [MicroSystem 2009] were created. These languages are based on a higher-level tag-based abstractions and are also sometimes called *SUI* [Tilly et al. 2007]. Besides these languages there are other SUI languages such as *UsiXML* [Limbourg et al. 2004; Francisco M. Trindade] or *XIML* [Puerta and Eisenstein 2002]. To achieve the *DIWG* goals, some server-side programming languages such as Java, C++ or *XSLT* and *XPath* are used to translate the described UI into a Web-UI.

3.4 Web Application Frameworks

In the literature there are many approaches covering the single authoring principles. All these approaches have a common base. According to Ziegert et al. [2004] a **DIWF** should offer:

A device independent markup language: Used to capture or to preserve the intention of the author.

An adaptation system: To transform the input model into a specific presentation model.

Author control over the final presentation Thereby the maintenance effort should not increase significantly.

In addition, according to Kirda and Kerer [2004], a *DIWF* should also meet some requirements. For example the proposed markup language should be platform independent, open standard, flexible, extensible, etc. The framework such as the implementation language should be platform independent. It should support the generation of applications according to the delivery context. The delivery unit should include industrial standards like XHTML to enable the use of existing development tools. In the end, it should also enable the definition and generation of content and layout for non XML Web based services.

In the context of device independent delivery there are many proposed research approaches. These approaches are classified in two trends. First, existing language profiles and content models are extended with new features in order to approach the device independent Web engineering principle. On the other hand, new languages are created to directly address those principles. Therefore, some transformation engines used to translate the designed model into the presentation domain are developed.

3.4.1 Frameworks using existing modeling Languages

3.4.1.1 CONSENSUS and the RIML

In year 2002 the **CONSENSUS**[1] project [Ziegert et al. 2004] was started. The goals of the project were:

- To provide a standardized markup language called RIML to uncover the gaps in current standards[2].
- To specify a usability guidelines for DIAD.
- The specification of a DIAD Architecture.
- The implementation of authoring and conversion tools working within the Eclipse IDE.

[1] IST-Programme / KA4 / AL: IST-2001-4.3.2. The project CONSENSUS was supported by the European Community.
[2] http://natalian.org/archives/2004/09/27/riml/

The developed RIML [Grassel et al.] is based on W3C technologies: XHTML 2.0, XForms, SMIL [Bulterman et al. 2008] and extensions. The extensions consist of new elements included in the XHTML standard to describe the functionality, the layout, the content control, the pagination, navigation, grammar, and error recovery elements for voice. A couple of authoring and conversion tools implemented to support a productive content authoring include:

XML Editor: For code completion, schema validation and also tree editor.

Frameset view: To provide abstract layout preview.

Device dependent view: For an adaptation result preview.

Preview functionality: Based on existing emulators.

The application layout definition with RIML encompasses the definition of layout module with layout container (column, row, paginatingRow, paginatingColumn) and hierarchical frame layout modules. Container properties are specified by attribute tags. The layout definition is enclosed in the *<head/>*, and the content definition-in the *<body/>* tag. The content is defined in different section elements using XHTML markups that are mapped to frames according to the layout definition in the head file. The adaptation engine is structured in two adaptation phases (Semantic and Syntactic Adaptation). The major tasks can be described as follows: The client request is received from the server (e. g. a tomcat container). According to the request the client is recognized based on the UAPROF method (cp. 3.1). In the next step the content is adapted according to the client characteristics (Content reduction). Finally, the adapted content is transformed into the targeted markup language. The Content Adaptation mechanism is based on device classes. In the content definition section optional and alternative contents, such as the pagination properties, are defined for each device class. Therefore, layout definitions for each defined content according to a group of device class must be provided in the *<head/>* tag.

3.4.1.2 W3C and the DIAL

DIAL [Smith 2007] is the language profile proposed by the *W3C* to allow consistent delivery across devices and contexts. As a subset of XHTML version 2, DIAL is an XML language profile based on an existing W3C document profile such as *XFORMS* and content selection [Lewis et al. 2007] mechanisms for device independence (DISelect) [Lewis et al. 2007]. DIAL allows to capture the author intentions, thereby iignoring the conditions under which the content should be selected or filtered. Since DIAL is a subset of XHTML version 2, it allows authors to define variable business

rules which allow content to be selected or excluded for rendering. For example, a variable rule could indicate that the content should only be shown in a certain location.

3.4.1.3 SIML state

In general, the user interacts directly with Web or desktop UI. Jansen and Bulterman [2008] presents a novel approach in which the time and the geographical information is used as the major structuring paradigm. In this approach SIML state is used to add an user-defined state or an external data model to the declarative time-based languages SIML [Bulterman et al. 2008]. The data model externalization allows it to be shared with other components. SIML is a W3C recommendation that allows authors to write interactive multimedia presentations. As a declarative language, SIML allows the author to specify the relations between multimedia objects by describing the temporal behavior of a multimedia presentation. Using SIML the author can also define the references to the media objects and the presentation layout. With SIML state, new variables and communication can facilitate the interaction with specific domain requirements. SIML state can also serve as a bridge to external UI components. Additionally, the declared variables allow full control over the selectively rendered content.

3.4.2 Frameworks defining own modeling Languages

3.4.2.1 The DIWE framework

The DIWE framework presented in Kirda and Kerer [2004] is based on the MyXML language. MyXML is an XML-based language, used to enable a separation between layout, content and application logic and also to model the application. A MyXML language compiler is used to process the language and generate static content, which is embedded in HTML or XML according to the application's specification. This language [Kerer and Kirda 2001][3] supports the definition of loops, variables, and database access functions. The framework is provided with a visual IDE (Integrated Development Environment) called MyXMLDesigner and four default runtime processors, to provide device-independent content according to the delivery context. The runtime processors provide some techniques such as:

[3]The project was supported in part by the European Commission in the Framework of the IST Programme, Key Action II, on New Methods of Work and eCommerce. Project number: IST-1999-11400 (MOTION) and in part by the Austrian Academy of Science.

Device detection: Processor for device detection and identification.

Logic Interfacing: To provide an application logic integration and invocation according to the model specified using the MyXML language.

Page splitting: To provide the intelligent layout adaptation according to the device capabilities.

Process partition: Page splitting method used to incrementally display Web forms.

To allow the reuse of existing XSL stylesheets and to reduce the maintenance overhead, the framework also provides a technique called XSL stylesheet pre-processing. As shown in [Kirda and Kerer 2004, p. 98] MyXML language is used for application specification in the design phase. In this phase the application layout as well as the application logic are defined. After the Web application has been generated and the domain logic has been written, the runtime processors are used to filter and adapt the output stream to suit to a particular device. Thus the adaptation process occurs in a later step of the deployment, and not during the implementation.

3.4.2.2 HP and the DIWAF

The DIWAF [Giannetti 2002] from HP Laboratories Bristol is another effort to solve the single authoring problem. The approach also starts with a single device-independent description, which includes meta data that can guide the adaptation process. As in the CONSENSUS project the content is defined for device classes. Here, alternative content is also defined for each device class. For completely different devices it is possible to enrich the original content with optional information. The Framework is based on a series of indirections or bindings (used to link together content and style), which is used to enable the separation between the layout, the selected content and the style. The adaptation process is based on content selection according to the element priority and alternative contents. Thereby the interaction model is defined using extended XForms to support the priorities and the binding model.

3.4.3 Applications using Web Services

Nowadays Web applications are also developed by combining different Web services. In [Kreger 2001, p. 6] a Web service is defined as an interface, which describes a collection of operations that are network-accessible through standardized XML messaging. Thereby a Web service is described using a standard, formal XML notion

called service description. Different Web services can seamlessly interoperate to carry out complex business tasks. For building such a Web application, which incorporates the use of Web services, a research approach based on declarative UI was presented in Kawash [2004]. This approach, also based on the single authoring principle, requires authors to declare the UI. Using a small engine that is deployed on the client, the UI specifications are downloaded through the Web service and interpreted on the fly according to the delivery context. The UI functionality is specified by a tailored form of finite state machines defining the:

- application states
- UI component in each state
- transition between the states
- and finally a mapping between the UI element and the Web service.

3.5 Conclusion

The research approaches presented here allow a single content authoring and an adaptation mechanism according to the delivery context. Thereby the content adaptation requirements are always the same:

- Separation of domain logic from the presentation logic using subsets of existing language profiles like XHTML, SIML or new languages (MyXML).
- The defined languages allow the developer to integrate semantic meta-information. They also allow a generic description of the delivered units. Optionally the interaction with the application logic can also be specified.
- The resulting frameworks provide the detection of device and network characteristics. The collected informations are used to guide the adaptation process.

Using these technologies the developer is required to specify the user interface only once. To generate the content according to the delivery context the developer must provide XSL stylesheets for each target family of devices and the appropriate one is selected during the runtime.

4 Model Transformations

> In Kleppe et al. [2003]; Visser [2005] model **transformation** is defined as: The automatic generation of a target **model** from a source model, according to a transformation definition. A **transformation definition** is a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language. A **transformation rule** is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language.

Moreover, model transformations also include transformations from a more abstract (e.g., LAIM modeled in a XML or an Object Oriented (OO) form) to a more concrete model (e.g., XHTML, XHTML-MP) [Mens et al. 2005]. According to the transformation principles, the syntax and the semantics of the modeling language are expressed by a meta model (e.g., XSD [Gao et al. 2009] or UML [OMG 2009]). Due to the fact that the output and the input models are expressed in different languages or concepts, an exogenous transformation also called translation [Visser et al.] is performed. According to Mens et al. [2005], the source model (LAIM) and the target model (XHTML, etc) belong to two different technological spaces. To bridge those spaces, CTs are used.

4.1 Conditional Transformations (CTs)

> **Conditional Transformations (CTs)** are a logic-based language and formalism for expressing arbitrary software transformations guarded by arbitrarily complex preconditions. The transformation part of a CT is executed for all elements that fulfill the CT conditions [Kniesel 2006a; Kniesel and Koch 2003].

As a theoretical and practical basis for endogenous model transformations [Kleppe et al. 2003] and model driven engineering [Schmidt 2006], CTs make a strict separation between model analysis (conditions) and model modifications (transformations)

```
ct (name_of_the_ct (parameters ),
   condition ((
      list_of_conditions
   )),
   transformation ((
      list_of_transformations
   ))
).
```

Figure 4.1: CTs Representation

(cp. figure 4.1). Similar to Prolog, analysis results (terms) are constructed via variable unifications. Thereby the model or the program artifacts are represented as logic facts or predicates [Kniesel 2006a; Shapiro and Sterling 1994]. However, unlike in Prolog, no side effects occur during the program analysis. In order to explain the CTs functionalities a brief overview of program element **Facts**, **Rules** and the **Queries** used in the conditional part follows.

4.1.1 Facts

According to inc. [2004], facts are the simplest form of Prolog predicates and are similar to records in a relational database. For example a fact is defined as a term of the form $'pred(arg_1, arg_2, ...arg_N).'$. Thereby the predicate name is $pred$, $arg_i, \quad i = 1, ..., N$ are predicate arguments, N is the `arity` and . the syntactical end of all Prolog clauses.

A fact (cp. listing 4.1) in CT consists of a predicate such as laimGroup, laimEnabled, or enabled and its arguments. In this example, *enabled* is a predicate of arity *0*. The arguments are legal Prolog terms. A term can be a digit, an atom (a constant beginning with a lowercase letter), a variable (a universally quantified [cs.odu.edu 2003] construct beginning with an upper letter) or a structure (e. g. aggregation of Prolog facts, lists). A universally quantified construct can be defined as a logical predicate that will be bound to a term during the program execution, depending on the context in which it is used.

Listing 4.1: *Example of Prolog Facts*

```
1 laimGroup(iplayui,root,[image,songs,player,context,play,volume,time]).
2 laimEnabled(iplayui).
3 enabled.
```

4.1.2 Rules

Rules are stored queries. They permit to describe a relationship among facts, the interference between these facts or how a fact can be deducted from another fact. Since CTs are related to Prolog, Prolog rules are universally valid in CTs (cp. listing 4.2). In the literature, Prolog rules are also called Horn clauses. A `Horn clause` [Padawitz 1988] is a clause of the form: $p_1 \cap .... \cap p_n \Rightarrow q \quad or \quad [\neg p_1, ..., \neg p_n, q]$. Thereby, $p_i \, i = 1, ..., n$ are positive atoms and q is the goal. Listing 4.2 demonstrates how a Horn clause is used. Thus, a Horn clause is a clause with at most one positive literal:

$$lhs \; :- \, rhs_1, \, rhs_2, ..., \, rhs_N.$$

Thereby lhs is the left hand side, $rhs_i \, i = 1, ..., N$ the right hand side of the clause and $:-$ is the logical implication.
In the literature the rules and the facts are also called clauses. Clauses are assertions about a Closed World. The **Closed World Assumption** [Daintith] suggests that the domain in which the model is defined is explicit and well-defined.

Listing 4.2: *Rule definition*

```
1 %<- X and Y are siblings if they have the same mother
2 sibling (X,Y) :- mother(Z,X), mother(Z,Y).
3
4 %<- mother and sibling are predicates
5  %<- merkel is the mother of angela
6 mother(merkel,angela).
7 %<- merkel is the mother of ingrid
8 mother(merkel,ingrid).
```

4.1.3 Queries

Queries are statements which are used to make assertions by pattern matching in the data base. A query pattern also called a goal can be a simple fact or a fact with one or more names replaced by variables (cp. listing 4.3). Prolog's pattern matching is called unification [inc. 2004]. A Query can also be an instruction, which is used to update the model.

Listing 4.3: *Querying the fact base in a Prolog manner*

```
1  ?- mother(merkel,angela).
2  yes.
3  ?- mother(merkel,ingrid).
4  yes.
5  ?- mother(Mother,angela).
6  Mother = merkel.
7  ?- mother(Name,Child).
8  Name = merkel,
9  Child = angela ;
10 Name = merkel,
11 Child = ingrid.
12
13 ?- sibling (angela,Y).
14 Y = ingrid.
```

4.1.4 Modules

Like in Prolog, predicates are organized in modules. This provides namespaces for predicates defined within the modules and allows users to redefine system predicates. The default module is the **user module**. The user module contains all predicates, which are not explicitly declared in a module. The predicates included in the user module are module transparent [Wielemaker 2008]. The **system module** contains all other module transparent build-in predicates.

4.1.5 The Backtracking mechanism

The backtracking mechanism from Prolog, as described in Iyengar [1994], is also used in CTs. This mechanism is used in rules (e. g. 4.1.2) to find multiple solutions to a query.

4.2 Model Representation in CTs

To represent the model a `gAST` is used. In the Literature an `AST` is defined as:

> The representation of Source Code as a tree of nodes representing constants or variables (leaves) and operators or statements (inner nodes). It reveals the lexical/syntactical structure of the program text, which is sometimes related to the semantic structure of the program.

According to Kniesel [2006b] the model `gAST` is defined as a typed and labeled graph. In this way, a program or a model is represented as a set of nodes. Each node has an UID, a type, a set of typed attributes and edges referring to other nodes. E. g. the generalized representation of a logic node is as follows:

$$\begin{aligned} node =< id, type, attr_1, ..., attr_n, edge_1, ..., edge_m > \quad & with \\ attr_i =< name_i, type_i, val_i > \quad & i = 1, ..., n \\ edge_j =< name_j, type_j, val_j > \quad & j = 1, ..., m \end{aligned}$$

To ensure the **Closed World Assumption** and to avoid redundancy in the model, model specific meta-information or meta-predicates are to be defined. The resulting representation of the model in the fact base should fulfill the `gAST` specification.

4.2.1 Meta-Predicates Definition

The `gAST` specification allows the definition of `node` and `relation facts` which is part of the `sub tree` of a node fact.

4.2.1.1 Node Facts

In this work, according to the generic way of the `gAST`, we define a node fact as a fact of the form:

$$type(id, attr_{a_1}, attr_{a_2}, ..., attr_{a_n})$$
or
$$type(id, parentid, [attr_{a_1}, attr_{a_2}, ..., attr_{a_n}])$$

Thereby, the predicate `type` is the node type (cp. 4.2), `id` is the UID of the node, $attr_{a_i} \quad i = 0, 1, ..., n$ represent references to other nodes[4] (e. g. cp. listing 4.1). The parentid is the reference to the parent node and $[attr_{a_1}, attr_{a_2}, ..., attr_{a_n}]$ represents the reference list, also called children nodes of the current node. Since each AST node of the program is represented as a logic fact, a node fact refers to its parent via its second attribute.

[4] https://sewiki.iai.uni-bonn.de/research/ctc/start

Listing 4.4: *Node facts definition in CTs*

```
:- multifile ast_node_def/3. % ast_node_def of arity 3 may be defined in many files

%general predicate definition
ast_node_def(context,predicateName,[ % a predicate is always used in a context call language
    ast_arg(id,       mult(1,1,no ), id, [predicateName]),
    ast_arg(parent,   mult(1,1,no ), id, [parentnode]) %reference to the parent element
]).

ast_node_def(context,predicateName,[
    ast_arg(id,       mult(1,1,no ), id, [predicateName]),
    ast_arg(parent,   mult(1,1,no ), id, [parentNode]), %reference to the parent element
    ast_arg(children,     mult(0,*,ord ), id, [id]) %ordered list of reference elements
]).

%definition of laimInput and laimGroup predicates
ast_node_def('Laim',laimInput,[
    ast_arg(id,       mult(1,1,no ), id, [laimInput]),
    ast_arg(parent,   mult(1,1,no ), id, [laimGroup])
]).

ast_node_def('Laim',laimGroup,[
    ast_arg(id,       mult(1,1,no ), id, [laimGroup]),
    ast_arg(parent,   mult(1,1,no ), id, [laimGroup]),
    ast_arg(children,     mult(0,*,ord ), id, [id])
]).
```

4.2.1.2 Relation Facts

A relation fact is designed or represented just like a node fact (cp. 4.2.1.1). Actually a relation fact is used to express optional node fact's attribute values. The attribute in the first position is always the UID of a given node fact (e. g. cp. listing 4.5):

$$name(id, [attrVal_{a_1}, attrVal_{a_2}, ..., attrVal_{a_n}])$$

Where $attrVal_{a_i}$, $i = 0, 1, ..., n$ represents the attribute value of a given node fact with UID id. A relation fact can also be used to express the connection between many other node facts. It serves to organize foreign keys as in a relation data base (e. g. cp. listing 4.6):

$$name(attr_{a_1}, attr_{a_2}, ..., attr_{a_n})$$

Name is the relation name or the relation type and $attr_{a_i}$, $i = 0, 1, ..., n$ represents the UID of a given node fact.

Listing 4.5: *Relation Fact in CTs*

```
:- multifile ast_relation/3. % ast_relation of arity 3 may be defined in many files

%general predicate definition
ast_relation(context,predicateName,[ % a predicate is always used in a context call language
    ast_arg(parent, mult(1,1,no ), id, [predicateName]) % parent id: required
]).

ast_relation(context,predicateName,[ % a predicate is always used in a context call language
    ast_arg(parent, mult(1,1,no ), id, [predicateName]), % parent id: required
    ast_arg(attrVal, mult(1,1,no ), attr, [atom]) %reference to the parent element
]).

ast_relation(context,predicateName,[
    ast_arg(id, mult(1,1,no ), id, [predicateName]), % parent id: required
    ast_arg(parent, mult(1,1,no ), attr, [atom]), %reference to the parent element
    ast_arg(children, mult(0,*,ord ), attr, [atom]) %ordered list of reference elements
]).

%definition of laimDefaultValues and laimGroup predicates
ast_relation('Laim',laimEnabled,[
    ast_arg(id, mult(1,1,no ), id, [id]) % <-- convention!!!
]).

ast_relation('Laim',laimDefaultValues,[
    ast_arg(id, mult(1,1,no ), id, [laimInput]), % convention
    ast_arg(defaultValues, mult(0,*,ord), attr, [atom]) %ordered list of laimInput default
        values
]).
```

In listing 4.6 a jsConnecto serves to imply the relation between a jsLabel and a jsInput.

Listing 4.6: *Another relation fact in CTs*

```
%jsConnecto predicate definition
%a jsConnecto serves to imply the relation between a jsLabel and a jsInput

ast_relation('Intermediate',jsConnecto,[
   ast_arg(id, mult(1,1,no ), id, [jsLabel]), % convention
    ast_arg(connecto, mult(0,1,no ), id, [jsInput])
]).
```

Using the `gAST specification` the model can be translated into logical facts. The result of the translation serves as input model for the transformation. To transform the input model into the target domain, **transformation rules** have to be defined. They describe how a model in the source language can be transformed into a model in the target language.

4.3 Transformation Rules in CTs

Since CTs are purely declarative, transformation rules have to be built to transform the input model into the target model. These rules are used in the condition part of the CTs and are materialized as a set of analyses and preconditions. Since transformation rules are part of CTs, henceforth in this case study we consider a CT as a transformation rule (cp. 4.1). To formulate the transformation rules there are two proposed CTs syntaxes[5]:

Native CTC Syntax Used to create individual CTs and CT sequences.

Sugared CTC Syntax Also used to create individual CTs and CT sequences but is more concise than the native syntax.

The native and the sugared CTC Syntax have the same meaning and consist of:

Head Name of the CT and its parameters.

Condition According to the common Prolog syntax the condition part consists of an arbitrary logic expression.

Transformation Consists of conjunctions of addition (add), deletions (delete) or term replacement (replace).

Variables A universally quantified parameter (cp. 4.1.1).

Constants Also a parameter which starts with a lower case letter or is enclosed in simple quotes.

At the start of this work, the sugared CTC syntax was not included in the stabled release, so that we choose to use the native CTC syntax. However the most important differences between the native and the sugared CTC syntax is the use of terms as operators (e. g. ct, condition, transformation, etc.) [6] in the native syntax and on the other hand the explicit use of algebra operators (e. g. ::, »>, ++, etc.)[7]. Listing 4.7

[5] https://sewiki.iai.uni-bonn.de/research/ctc/language
[6] https://sewiki.iai.uni-bonn.de/research/ctc/native_ct_syntax
[7] https://sewiki.iai.uni-bonn.de/research/ctc/sugared_ct_syntax

demonstrates the use of CTs in the native syntax. Variables such as constants are found in the head, condition such as the transformation part. The head of the CT is:

```
translateLaimGroupWithDepthTWO(Id,2,'false')
```

The condition part is enclosed in `condition(...)` and the transformation part in `transformation(...)`. In the common CTC syntax an UID creation part which is enclosed in `idcreation(...)` is also available. However, this part is not used, since arbitrary ids are created in the UID creation part. Hence, CTs are used for runtime model transformations on demand and the created ids are used to control the Client-Server interaction. Instead, common ids are just transformed into new ids. E. g. the terms:

```
transform_id('js', Id, NewId)
```
or
```
jsGenerateIdList(Id,NewChildIdList,ChildIdList,0,'false')
```

are used to transform the `Id` passed as parameter in the CT head or the `ChildIdList` found by querying the data base into a `NewId` or into a list of new reference ids (`NewChildIdList`).

Listing 4.7: *Sugared CTC Syntax in used*

```
1  ct( translateLaimGroupWithDepthTWO(Id,2,'false'),
2    condition((
3      laimGroup(Id,ParentId,ChildIdList),
4      jsGenerateIdList(Id,NewChildIdList,ChildIdList,0,'false'),
5      transform_id('js', Id, NewId),
6      transform_id('js', ParentId, NewParentId),
7      (laimElementTag(Id,DefineTag) -> Tag = DefineTag; Tag = 'div')
8    )),
9    transformation((
10     add( htmlNode(NewId,NewParentId,NewChildIdList,Tag) )
11   ))
12 ).
```

By developing the transformation rules some requirements have to be met (cp. 7). Therefore the granularity of each rule should be guaranteed when defining a rule as a CT. In addition to the granularity, the composition of the defined rules should be possible. According to the fundamental principles of software engineering, programs should be developed incrementally by defining several units and their interfaces,

and then by composing those units. Based on this notion of program composition, O'Keefe [1985] interprets logic programs as algebraic elements and modeling their composition in terms of algebraic operators. In Bugliesi et al. [1994] this conceptual approach is used to extend the syntax of programs with new logical connectives that support abstraction mechanisms and to define a variety of powerful composition mechanisms.
The composition mechanism in CT is provided by CT sequences[8]. They are used to compose individual and unitary CTs into an executable sequence. The composition mechanisms allowed by the CTs syntax are as follows:

Orseq The OR sequence denotes a sequence without history propagation.

Propseq In PROP sequence successful substitutions are propagate from left-to-right.

Negprop Failed substitutions are propagate from left-to-right in the NEGPROP sequence.

Andseq Finally, the AND sequence denotes a bidirectional history propagation.

Beside OR, PROP, NEGPROP and AND sequences, which are the four main sequence operators, a LOOP sequence defined syntactically as a loop(first, second) can also be used. This sequence denotes a fix point iteration of the sequence passed in the second argument. In order to compose CT sequences, the CTC proposes the definition of procedures. Procedures are introduced with ctseq (CT Sequence). Listing 4.8 demonstrates the definition of a procedure.

Listing 4.8: *CT sequences in used*

```
1 ctseq( ctseqHead(ctseqParameters),
2   orseq(
3     ct( ctHead1(firstCTparameters) ),
4     andseq(
5       ct( ctHead2(2ndCTparameters) ),
6       ct( ctHead3(3thCTparameters) )
7     )
8   )
9 ).
```

4.4 Conclusion

In conclusion CTs provide a theoretical and practical basis for model transformations. Similar to Prolog the software artifacts are represented as logic facts [Kniesel 2006a;

[8] https://sewiki.iai.uni-bonn.de/research/ctc/native_ct_syntax

Kniesel and Koch 2003]. Unlike Prolog during the CTs execution, the model analysis and the model update do not interfere. The transformation part of a CT is executed for all elements that fulfill the condition part of the CT. Compared to other model transformation approaches CTs provide a unique combination of features [Bihler et al. 2008; Kniesel 2006a], amongst which:

Purely Declarative Unlike imperative programs with side-effects (including Prolog programs that manipulate their own factbase) CTs have a well-defined, model-theoretic semantics. Hence CTs are easy to compose, analyse and optimize automatically.

Composable Different composition operators, of which some are unique to CTs, allow creation of complex programs from simple, reusable units. Composition is possible even if not anticipated by the designers of the existing CTs. No hooks need to be built into CTs to enable their reuse in unanticipated contexts.

Analysable Automated analysis and resolution of interactions between CTs is possible [Kniesel and Bardey 2006; Kniesel 2008] thus providing the basis for the related interaction analysis of adaptations expressed by CTs.

Multi-Directional A CT with N arguments is like $N!$ functions, since each argument can be used as input or output. E. g. the same CT procedure (ctseq) can be used to transform an input element into a `jsInput` element from any type with the referring node elements. This depends only on the way the CT is called or on the input model which is transformed without the need to rewrite the CT procedure.

5 User Interfaces Specification

The UI Specification encompasses the definition of the UI functionality and the UI model. This chapter is dedicated to the UI functionality and the presentation logic specification. The UI model specification will be done later in section 6.

5.1 UI Functionality

In this section we will firstly present the UI functionality specification approaches, defined in the literature. Thereby we try to evaluate which methods can be integrated in our development approach. Then, we will focus on the UI functionality specification approach that we have adopted.

5.1.1 State of the Art

In the literature many UI specification approaches are presented. Amongst others, there is the method based on conceptual pattern language JUST-UI for UI specification [Molina et al. 2002]. According to Molina [2004], JUST-UI is described as a pattern language extension to the OO-Method [Pastor et al. 1997]. The OO-Method allows the UI specification with a set of graphical models. This OO-Method provides an OO software production environment and is based on two components:

Conceptual model Used to capture the system structure.

Execution model: An automated translation model. It is used to generate the data sources and the logical modules in the desired implementation environment and language [Gómez et al. 2001].

JUST-UI also proposes a series of mappings which helps to convert the abstract UI specification into a concrete implementation for different devices. A UI code generator uses these mappings to produce the UI code.
The user-centered approach called the OO-H (Object-Oriented-Hypermedia) method [Gómez et al. 2001] is also based on the OO-Method. It proposes a methodology to

combine the software artifacts (structure, behavior and presentation) in order to produce the final software product. Like the JUST-UI approach, the OO-H method is based on a pattern catalog language. These patterns are used to capture the abstract interaction model between the user and the application. Furthermore, the pattern catalog proposes a set of constructs that effectively cover the problems identified within Web environments.
In this work approach the evaluation of the pattern catalog proposed in the OO-H method is out of scope. Nevertheless, the OO-H method approach seems to facilitate the reuse of design experiences and the consistency among the different interface modules. In order to solve well-know hypermedia application development problems, the OO-H method also proposes a set of interaction patterns.
To design the UI interface the OOHDM [Schwabe and Rossi 1995] approach proposes to break up the hypermedia application development into four steps. Thereby each step focuses on a particular design concern. This concept is used in Karimpour et al. [2008] to build Web-based Applications based on the J2EE Technology. By mapping the design artifacts created in OOHDM into J2EE components, the interface usability increased and the development time was improved.

5.1.2 Adopted Approach

The results of the above presented research approaches can be used for large application development projects. However, unlike in Karimpour et al. [2008] where the MVC architecture is used, we chose to use the MVP [Potel; Law 2007] instead. MVP is used to extract the business logic from the presentation logic. By using an interface to interact with the view, the MVP is also used to separate the UI specification from the UI implementation.
A UML state diagram [OMG 2009] can be used to model our business processes. In this way, we found the modeling concept by [Kawash 2004, p. 2] better than the simple use of state diagrams. Kawash [2004] has defined a form of extended FSM called UIFS. The UIFS was used in that research to specify the UI functionality for a Web service. As we know, a Web service UI can also be a part of a Web application. So we adopted the idea found there to specify the UI functionality used in our research approach.
For our purpose a UIFS is also defined as a triple $(\mathcal{S}, \mathcal{T}, \mathcal{M})$, where:

$\mathcal{S}$ is a set of states, with an initial (state after the first application rendering) and an implicit final state (by session expiration or even when the application is

leaved). Each state in $\mathcal{S}$ is also a collection $\mathcal{U}$ of UI components with an assignment function $\mathcal{A}$ defined on $\mathcal{U}$. More precisely, the application running on the server is always on a specific state $\mathcal{S}$. According to this state, the whole UI or just a part of the UI is rendered and sent to the client. The UI consists of UI components. Each UI component is linked with JavaScript callback functions used to control the UI interactions.

$\mathcal{T}$ is in this case just a labeled transition function on $\mathcal{S}$. E. g. a function like *play*() to play a song, when the *play* event is initiated.

$\mathcal{M}$ is defined as a mapping that associates a UI component (the id of that component), with an event type and the method executed on the server.

Two event types are defined here:

Input Change Used to execute a server side method according to the user selection.

Output Change Used to trigger a content update of the corresponding UI component.

Similar to the hidden control principles declared in that work [Kawash 2004, p. 3], UI elements produced using the UI description in LAIM can also be hidden elements (cp. 6), when they were described as disabled. Whether an element is hidden or not depends also on the state of the application. E. g. during the load time an image can be enabled (to show that the application is loading) and after the application is completely rendered the image can be disabled. To deal with such interaction asynchronous Ajax call-back functions are used to manipulate the DOM [Hors et al. 2004].

5.2 Presentation Logic

The presentation logic specification covers the definition of the UI presentation on the client, such as the definition of a UI update strategy. Since the delivery context plays a significant role, the adaptation engine should be designed to deliver content and presentation in accordance to the One Web Principle (OWP) requirements.

5.2.1 One Web Principle

Rabin and McCathieNevile [2008] defines One Web as the possibility to make the same information and services available to users irrespective of differences in presentation capabilities and access mechanisms. This means that the provided service

should be accessible on a wide range of devices. Thereby, the delivered content must be thematically coherent and must provide a good user experience when accessed from different devices. However, as suggested in the *W3C* guidelines, it does not mean that precisely the same information must be available in exactly the same representation across ubiquitous devices.

5.2.2 UI Presentation Specification

The presentation definition begins during the interface design. Each UI component has an UID. This UID is used to identify the component either on the client and on the server side or also to determine the CSS [Bos et al. 2009] stylesheet. During the stylesheet definition, the Mobile Web Best Practices [Rabin and McCathieNevile 2008] guidelines should help to improve experience.

5.2.2.1 Content Presentation

To ensure the *OWP*, we provide the mechanism to produce the final presentation domain according to the UI capabilities. This mechanism provides the ability to define element tag according to the targeted markup language. Thus, the developer can define element tags, attributes and attribute values according to the target domain. To specify the element tags common markup profiles can be used [Axelsson et al. 2006; Oshry et al. 2007]. OMA presents an overview about the OMA standards. For a rapid start, the developer can also use the WML, cHTML, and XHTML-MP comparison approach done in Woo and Jang [2008]. The developer network [Network 2009] also proposed a style guideline for beginners. Using the W3C or the OMA recommendations [Bos et al. 2009; Alliance 2006] the layout presentation can be defined. Doing this the CSS definition is be performed using the id attribute values or defining a class attribute. To define CSS attribute value, the application property file can also be used. However, the property file is just used to declare the binding between the elements and the attribute values. The final layout definition in CSS is up to the developer.

5.2.2.2 UI update Strategy

Since Ajax is used to support the Client-Server interaction (cp. 2.6) with asynchronous and partial refreshes of the application, an update strategy should be developed to contend the UI issues. The UI update strategy is elaborated according

to the UI functionality specification defined in section 5.1. Having the functionality specification defined, an overview of the states of the application is done. After the application states are defined the developer can choose how the application will be updated. Furthermore, having the overview, he can decide whether a full or partial application update is required.
For example, according to our prototype, the first state consist of the application after the first client request. Another state can be the one after the user clicked on the play button. So the transition from the first request to the playing mode is labeled by an play event. In the playing state the image associated with the play button and the value of the end time node should be changed. Interactively the value of the starting time node should also be changed according to the playing progress. Therefore, we provide an abstract Output element with content type JSScript. As a content value instruct the CSS and the JavaScript to change the visual appearance, to instantiate an elapse time counter and to update the end time value.

5.3 Conclusion

The UI specification encompasses the definition of the UI functionalities, presentation logic and an UI model. To specify the application logic we chose to specify the domain logic after the UIFS method presented in Kawash [2004]. If presentation issues occur, the developer should use the W3C or the OMA style guidelines. Finally the way the application will be updated is up to the developer. Then the appropriate update strategy depends on the application issues. For more information on the design strategies using Ajax architecture, a large overview is presented in Ort and Basler [2006]. Thereby all advantages and disadvantages of each method are explained.

6 Language for Abstract user Interface Modeling (LAIM)

To handle different end-user preferences and different computing platforms where the UI application contexts become important, SUIs come in use. They facilitate an extensive reuse of UI components. Herein a separation between the application logic from its visual presentation and also from the UI requirements is guaranteed. SUI can also be used to allow UI interoperation which increases productivity, while the UI components are shared between many applications as in the Microsoft office packet. Due to the numerous advantages of the SUI, we generate dynamically an adapted UI based on a Semantic User Interface (SUI) [Tilly et al. 2007] description [Tilly et al. 2007]. For this purpose LAIM was build. In order to build a powerful model which fulfills the user's and the developer's requirements, the approach mentioned in Ludewig [2003] was taken into consideration.

6.1 LAIM Specification

LAIM [Bihler et al. 2008] is a very basic SUID language. It consists of four main elements and was designed after the principle of the design pattern named composite pattern [Freeman et al.] (cp. figure 6.1). The component interface mapped as an UIElement contains the definition of the required UI attribute elements, which are contained in all other Elements. Based on the semantics of the visual UI components, the LAIM elements can be described as follows:

Input Control element associated with an Input Data Type. Since LAIM is an abstract SUI, if the Force Default attribute was not defined, this would allow users to input text or to choose among several default choices. Thereby the restricted number of possible selectable values are specified with the Min and Max selection attributes (cp. listing 6.3).

Output Defines an output of any kind, depending on the content and the content type of the current component, such as the context information. This UI component is not associated with a control element (cp. listing 6.2).

Action Another control element, wherewith users directly interact with the application. It allows triggering of program functionalities and can be visualized as menus, buttons, or as an object control.

Group Container for other leaf elements and mapped as the composite. The leaf elements (Output, Input and Action) representing the real UI elements can be set into a context. This might help the transformation engine to structure the rendered UI and decide upon the proper visualization of the elements within the group. Enhancing the transformation engine with logic data based on the targeted platform, the running environment and context informations, the group element can also serve to group many UI components from the same type into a single UI component.

This is similar to the model definition presented in [Kawash 2004, p. 2]. In this model, the Input element is also used to collect the user's input. The Output element is used for content presentation. The Controls defined here as Action elements are used to initiate a user event and to control the interaction between the user and the interface. The classic software life-cycle models usually consist of many activities [Nutt 1995]. In the requirement analysis and specification step, an XML document (cp. figure 6.2) can be used to capture the UI description. Therefore, the UI description is built according to the meta model specification, which on its own is built using the XML schema specification [Gao et al. 2009] (cp. figure 6.2). During the runtime the defined UI representation is translated into an object-oriented model. In favor of the model description the UI is optionally extendable. To generate the adaptable UI the object-oriented model is translated into logical Prolog facts. These facts are used to create the UI using CTs.

6.2 LAIM - Attribute Semantics

The LAIM model allows the specification of common attributes found in all UI components, and on the other hand -the specification of specific UI attributes.

6.2.1 Semantic of common UI Attributes

Referring to the program artifacts, each UI element owns an **id**, which is the UID for each UI component with its **caption** translated as labels in order to visualize it. Depending of the UI Requirements, UI components can be set to **enabled** or disabled (cp. listing 6.1). As defined in Giannetti [2002], to guarantee the content

selection depending of the context of use, the priority attribute was provided. It is an optional attribute, which can be defined for each UI component. In the object-oriented view of the abstract model, this attribute is automatically inherited from the superclass, when the priority definition is not contained in the subclasses.

6.2.2 Semantic of specific UI Attributes

To fulfill the UI requirement the LAIM specification 6.1 allows the definition of specific UI attributes. These attributes are specified with their expected attribute types. Since an object-oriented view of the model was adopted, the valid data types are the same as in the Java world. Nevertheless, a collection in LAIM can be interpreted as list of String in the object-oriented model. For instance, the LAIM - Input defines an Input Data Type attribute of type String. Then, the Default Values are syntactically restricted to a set of Strings.
In this work the target domain differs from the source domain. So the required Data Types also differ from the Data type usually used in an object-oriented system. E. g. in order to print out an image in XHTML, the image location is required. When implementing with Java the source location is also required, but the internal representation is done with a Buffered Image. Semantically an image is viewed as a content in the target domain. For this purpose LAIM allows the specification of an Output UI Element with Content Type image. Defining an Output UI from type image, the content attribute must refer to the location where the output can be found (cp. figure 6.2). The Output caption can be interpreted in this way as the alternative text attribute value of the image tag in the output model such as XHTML.

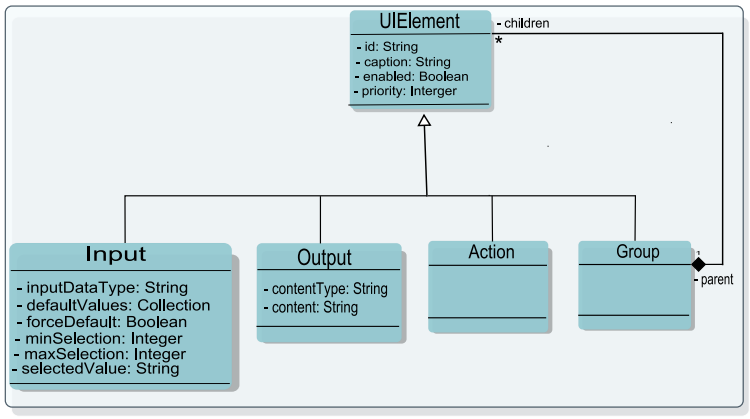

Figure 6.1: LAIM Architecture: The elements used to represent a user interface semantically in LAIM

```
<?xml version="1.0" encoding="UTF-8"?>
<laim caption="Mail" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="LAIM-Schema-lite.xsd" priority="1000">
   <group id="menu" semantic="main-actions" caption="Menu" priority="250">
      <action id="button_new" caption="New" semantic="new-element"/>
   </group>

   <input id="mail_listing" caption="Mails:" forceDefault="true" inputDataType="string"
      minSelection="0" maxSelection="-1" semantic="selection" priority="500"
      selectedValue="Bugs Bunny - Fresh hare">
      <defaultValues>
         <defaultValue>Bugs Bunny - A carrot - 12/01/2008</defaultValue>
         <defaultValue>Bugs Bunny - Fresh hare</defaultValue>
      </defaultValues>
   </input>
   <output id="mail_view" caption="Content:" contentType="image"
      semantic="element-view" priority="250">
      <content>/src/images/mypicture.jpg</content>
   </output>
</laim>
```

Figure 6.2: XML representation of a LAIM sample, thereby the XML Schema is located on the file system in LAIM-Schema-lite.xsd

Listing 6.1: *UIElement Attributes Schema*

```
1 <!ELEMENT UIELEMENT -- (AUIType) -- >
2 <!ATTLIST UIELEMENT
3   id          %id;       #REQUIRED -- client/server- UID component identifier --
4   caption     CDATA      #OPTINAL -- label used to view the UI element --
5   enabled     %boolean;  #OPTINAL -- set the element as enabled or hidden --
6   semantic    CDATA      #OPTINAL -- semantic, depending from the UI application context --
7   priority    CDATA      #OPTINAL -- priority, used to specify the UI component selection --
8   >
```

Listing 6.2: *LAIM Output Attributes Schema*

```
<!ELEMENT OUTPUT -- (AUIType) -- >
<!ATTLIST OUTPUT
  contentType CDATA #OPTINAL -- specifier the content type --
  content     CDATA #OPTINAL -- content shawn on the UI --
  >
```

Listing 6.3: *LAIM Input Attributes Schema*

```
<!ELEMENT INPUT -- (AUIType) -- >
<!ATTLIST INPUT
  inputDataType CDATA   #OPTINAL -- specifier the input data type --
  defaultValues %Collection #OPTINAL -- initial values --
  forceDefault  %boolean;  #OPTINAL -- user input is allow or not--
  minSelection  CDATA   #OPTINAL -- how many default values are to choose --
  maxSelection  CDATA   #OPTINAL -- max initial values to choose --
  selectedValue CDATA   #OPTINAL -- set the pre-selected value --
  >
```

6.3 Conclusion

To handle different end-user preferences and different computing platforms LAIM was specified. As a representation of the real world, this model must meet three criteria according to Stachowiak [1973]; Ludewig [2003]:

Mapping criterion There is an original object or phenomenon that is mapped to the model. In the sequel, this original object or phenomenon is referred to as "the original".

Reduction criterion Not all the properties of the original are mapped onto the model, but the model is somewhat reduced. On the other hand, the model must mirror at least some properties of the original.

Pragmatic criterion: The model can replace the original for some purpose, i.e. the model is useful.

Firstly, LAIM is used abstractly as the representation of the presentation domain. The final presentation depends on the platform on which the application is executed. The final application is built using the model specification and acts as an abstraction of the real world. Even if the created model reflects only the fantasy of the author, LAIM offers the specification of the fictitious model. So the **mapping criterion** is satisfied.
To define a model, LAIM provides the definition of a component class hierarchy 6.1.

Each component of this hierarchy contains common UI Attributes such as specific UI Attributes. To build the final model, which is in fact the original model, the abstract model interpretation is up to the developer. Since the abstract model is not a one-to-one mapping of the original model, the semantics of each component and its attributes becomes important. Consequently the **reduction criterion** is also satisfied.
During the development process many UI requirements are to be taken into account. Among others, different end-user preferences and different runtime platforms are to be taken into consideration. On the other hand, usability and code reuse play a significant role. So it is not possible to create different compatible models for each platform to guarantee a good user experience. For this purpose the created model can replace the original. According to Ludewig [2003] the **pragmatic criterion** comes true.
In conclusion LAIM is assumed to be a model, as the LAIM specification serves as model specification to define an abstract part of the real world.

7 Transformation Rules Definition

According to the model transformation definition seen in section 4, transformation definition or transformation rules have to be defined in order to translate the source model into the target model. To execute this correctly, the transformation rules have to fulfill some requirements. This section is devoted to the definition of rules requirements and demonstrates how the formulated rules are constructed and used to translate the source model.

7.1 Rules Requirements

During the work we discovered that the structure of the abstract model definition (LAIM) can be represented with FSMs. By using such an abstraction, the transformation rules can be formulated in a simple way. To guarantee a flexible application development, the transformation **Rules** should be:

Business related The specification should only pertain to the business logic.

Granular Should be suitably coarse-granular.

Composable The composition of the rules should be interpretable.

Context free As few assumptions as possible on the output data type.

In Engels et al. [2008], the above rules are used to design the interfaces and the operations in a *SOA*. In our case, we use these requirements to build the transformations definition in a way, such that the rules are coarse-granular, composable and context-free, so that the result of the transformation process is business related.
In section 4.3, we presented how the transformation definitions are built using CTs. Thereby, we observe how to build coarse-granular CTs rules and how to build complex transformation rules using CT Sequences. As CTs rules are built in a generic way, they can be assumed to be context free. Finally, according to the author, we demonstrated by constructing a prototype in section 8 how business related results are obtained. Doing this, the UI requirements, such as the delivery context were taken into consideration.

7.2 Transformation Definitions

LAIM was specified in section 6. As LAIM is a very abstract SUI used to define the model, the semantics of the attributes nodes plays a significant role during the model transformations. In this section we will explain a possible semantics interpretation of the attribute nodes.

7.2.1 LAIM-Group Transformation

The target language profiles (e. g. XHTML) propose numerous presentations, and layout constructs (table, list, checkbox etc.), which are not defined in LAIM. Since LAIM is only used to describe the presentation in an abstract way, there is no possibility to decide a priori which output element was described. To determine which UI element was intentionally described, the semantics of the elements should be examined. In this way, it should be possible to generate complex output UI element like a table, a menu structure or even XForms [Boyer 2009] components. In order to solve this problem, the author can add semantic information into the transformation rules as follows:

Position The position of the LAIM Group node according to the root node (cp. figure 7.1 and 7.3).

Type of children The author can add instructions to check if the children of the Group node are of the same type (e. g. Action or Output cp. figure 7.2 and 7.3).

Siblings In the same way, the transformation engine can check if the Group node belongs to a group of group nodes, whereby the type of children of the siblings is also taken into consideration.

Priority The priority, which is a LAIM-Group attribute, is used to determine, whether the Group node and its children node is to be translated or not.

For instance, to generate a XForms component the UI description can be done as seen on figure 7.1. Since XForms are composed of a model (model definition, behavior, and contains) and user interface (input fields definition and how to display the form element), LAIM can be used to describe the UI. The model definition can be generated and combined with the UI during the transformation process.

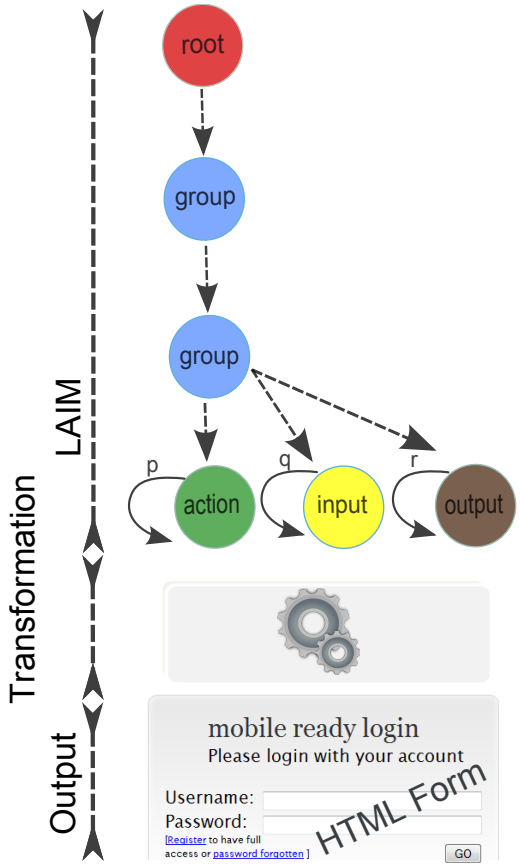

Figure 7.1: Description of a LAIM-Group containing different element types. According to this description the output can be an XForms component. The transformation result is related to the semantics of the LAIM-Group element. The other described UI components are also transformed according to their semantics description.

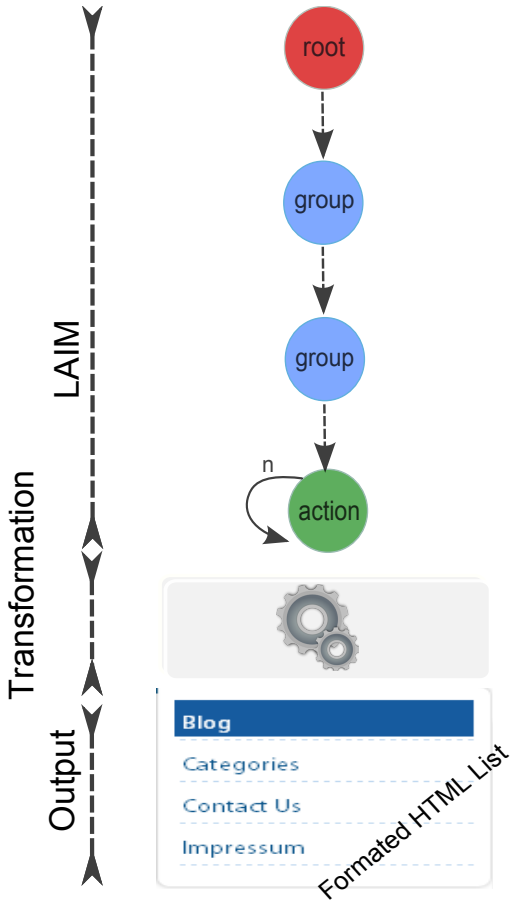

Figure 7.2: Transformation of a LAIM group node with the referring action node into a list container.

7.2.2 LAIM-Action , -Output Transformation

In the same way, the author can also add semantics information on the LAIM-Action and -Output transformation rules. Thus, the LAIM-Action or -Output transformation depends on the:

Siblings To generate a list of elements, the author can add transformation instructions into the transformation rules according to the siblings of the transformed element. Doing this, he can check whether all siblings contained in a group node are of the same type (cp. figure 7.2).

7.2.3 LAIM-Input Transformation

The LAIM input transformation is done after the LAIM-Input specification such as the semantics of its attributes (cp. section 6.2). For example, figure 7.3 shows a possi-

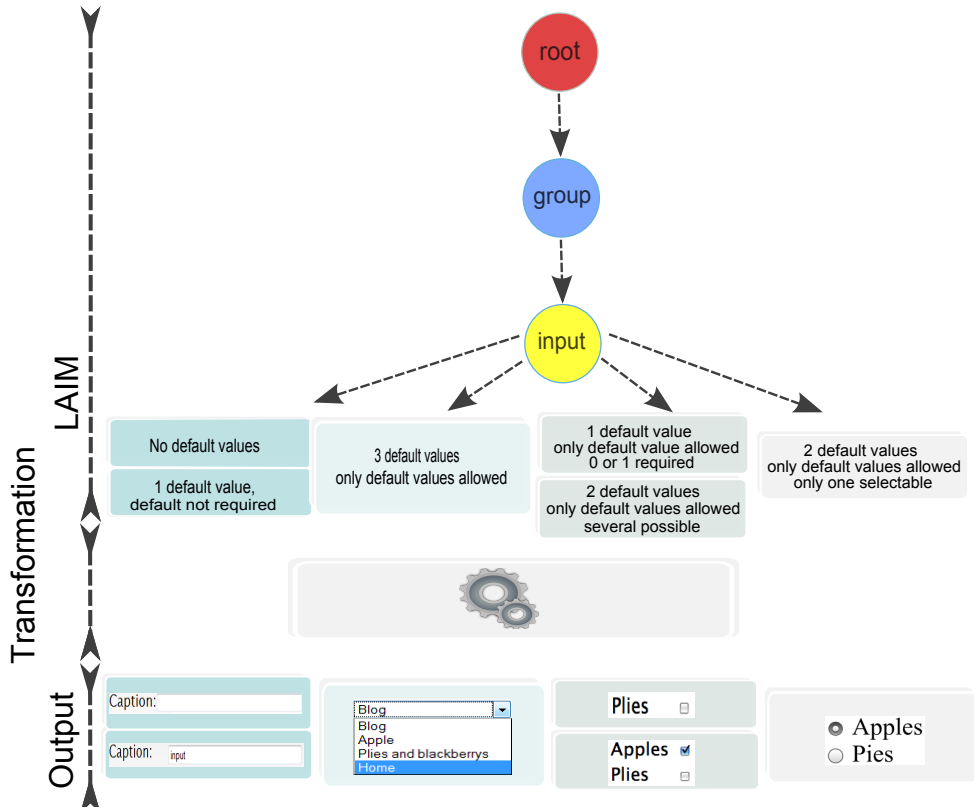

Figure 7.3: Transformation of a LAIM group with the referring input nodes.

ble transformation of an Input UI element. The transformation definitions according to the attribute semantics are also listed on the figure. Each rule is represented as a CT (cp. figure4.3) and the composition is done via CT sequences.

7.3 Conclusion

In conclusion, the semantics of the described UI elements can be used to enrich the transformations rules. Thus, it must be possible to generate specific presentation constructs according to the presentation domain.
During the transformation process, to ensure the granularity of each rule, each artifact is enclosed in one CT (cp. listing 7.1). Complex artifacts, which are related to the presentation requirements, are interpreted using CT sequences (cp. listing 7.2).

Listing 7.1: *Condition Transformation of a LAIM group node element*

```
/**
* ct( translateLaimGroupWithDepthOne(+Id,+Depth) ) is det.
* choose via pattern matching to translate the laim group node
* with depth one, into the corresponding intermediate node
**/
ct( translateLaimGroupWithDepthOne(Id,1),
  condition((
    %find the element with the corresponding Id
    laimGroup(Id,ParentId,ChildIdList),
    laimGroup(ParentId,root,_),
    %function to translate the children ids into new id list
    laimNodeTranslator:jsGenerateIdList(Id,NewChildIdList,ChildIdList,0,''),
    % translate laim group id into new id
    transform_id('js', Id, NewId),
    transform_id('js', ParentId, NewParentId),
    %tag to enclose the children elements during the final translation step
    (laimElementTag(Id,DefineTag) -> Tag = DefineTag; Tag = 'div')
  )),
  transformation((
    add( htmlNode(NewId,NewParentId,NewChildIdList,Tag) )
  ))
).
```

Listing 7.2: *Conditional Transformation Sequence*

```
/**
* translateLaimSemanticAndEnabledNodeForLaimGroupRoot(+ID) is det!
* CTSeq to translate the referring LAIM Group node Semantic, Enabled and Caption
**/
ctseq( translateLaimSemanticAndEnabledAndCaptionNode(Id),
  orseq(
    ct( translateLaimNodeSemantic(Id) ),
    orseq(
      ct( translateLaimNodeEnabled(Id) ),
      ct( translateLaimNodeCaption(Id) )
    )
  )
).
```

8 Implementation Details

The implementation consists of several stages. The first one is the design of a conform LAIM UI sample called Model (cp. figure 6.2) specified in section 6. According to the interaction architecture (cp. section 2.6) the second step checks which logic procedure will be executed on the client. Afterwards the model has to be refined. The last step consist of the real implementation on the server side.

8.1 Model Definition

In the requirement analysis and specification phases the model is created. To ensure that the application will be delivered upon the user requirements, it is recommended to use the user-centered design (UCD) approach [Visciola 2003; Vredenburg 2008]. Thereby the focus is put on the users, their tasks, and the context in which the application is used. In order to ensure the usability and a good user experience, Degler [2006] proposes to divide the target audience into groups, then usability can only be defined for a specific group of users and context. In an UCD project there are four main activities [Kirakowski and Collins 1999; ISO13407 1999]:

Requirement gathering Specify the context of use.

Requirement specification Specify the user and organizational requirements.

Design Produce design solutions.

Evaluation Evaluate designs against user requirements.

The outlined activities can also serve as a process to define the model. As the final model is an abstract model, the evaluation takes place only after the model transformation. During the requirement gathering and specification the basis model is designed (Figure 6.2 shows the representation of a basis model in XML format). Since many contexts of use have to be supported, the priority attribute must serve as content selection. Towards the model design in XML format, the model is translated into an object-oriented one (cp. figure 6.1).

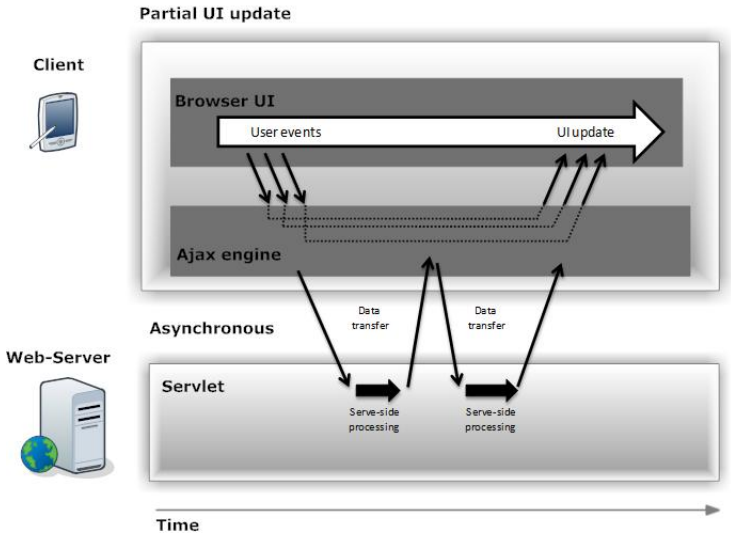

Figure 8.1: Standard Ajax web application model.

8.2 Client Side

To guarantee a great user experience an Ajax Web application model (cp. section 2.6) is used. Figure 8.1 shows a common Ajax web application model [Garrett; Wei]. To build such an architecture the control elements must be provided with JavaScript function calls. For instance, by implementing the prototype we choose the function $clicked(element_{Id})$ as a general function for all control elements of type button. For all Output UI components, which were translated into control elements, we choose the function named $outputChanged(element_{Id})$ or also $player\_media(content, element_{Id})$. For example see listing 8.1. Thereby the $element_{Id}$ is the id of the triggered element. As a prototype, we choose to build a music application. Doing this, the real music player, which resides on the client needs to be initialized with the needed content before the client feedback is sent to the server. For this purpose we choose to embed the JavaScript code with the method call $player\_media(content, element_{Id})$.

Listing 8.1: *Embedded JavaScript function in XHTML code.*

```
1 <div id="js_play_i">
2   <input type="button" id="js_play" name="js_play" value="play" ONCLICK="clicked('
        js_play');"/>
3 </div>
4 <div id="js_song_list_0">
5   <a id="js_song_list_0_1"
6     href="javascript:void player_media('/RenderTest/musics/101 Beyonce - If I Were A Boy.
          mp3','js_song_list_0_1');"
```

```
7       class="songs">Beyonce - I'am Sasha Pierce ... - If I Were A Boy</a>
8   </div>
```

To send the feedback to the server, we provide a XMLHttpRequest Object (cp. listing 8.2), which is initiated and processed from the client. The Feedback is composed of three major parameters:

Action: Denoted an ONCLICK or an ONCHANGE event.

Id: Component UID.

Url: The active content.

The Component UID is the only required element, because the application resides completely on the server and without the Id, there is no way to find out which component was activated.
After the requirements on the client side were specified, the model can be refined. In the next step, the proper implementation can begin.

Listing 8.2: *The XMLHttpRequest Object used for Client-Server Interaction.*

```
1  /**
2   *  http://www.prototypejs.org/api/ajax/request
3   * '/RenderTest/HelloWorld' entry point location on the Server,
4   * this location is also the Servlet URL.
5   **/
6  new Ajax.Request('/RenderTest/HelloWorld', {
7    method: 'post',
8    parameters: {
9      action: action,
10     id: elementId,
11     url: currentURL
12   }
13 });
```

8.3 Server Side

Referring to the proposed architecture, the device connects to an application server, in the case of this work prototype this might be an instance of Apache Tomcat. A Java Servlet acts as an entry point and interface to the Java-based Web application. This Servlet monitors the client Request/Response and differs between two HTTP Request methods:

Get Method: Used by the first application call, to request a representation of the application. In the process, according to the UI requirement, the application is instantiated, the predefined model is loaded and translated, and the translated UI is sent to the client without any side effects. As a content type, $text/html; charset = UTF-8$ is attached to the response.

Post method: Used to submit data included in the body of the request to be processed to the Java Servlet resource. This method is sent from the XMLHttpRequest object (cp. listing 8.2) by any other client requests. The side effect is the client-side partial presentation update. This occurs via the instruction $application/javascript; charset = UTF-8$, which brings the client to execute the response code.

Ajax allows developers to update parts of the application without needing to refresh the whole application. This is done using an asynchronous HTTP call back to the server and client-side JavaScript to update specific parts of the application. For instance, by clicking on a control element, its Id is sent as a parameter to server. According to this Id, there are two application update strategies:

Component update: Usually defined using $< div\, id = "element_{Id}" >$ tags. Only the required part of the abstract model is transformed and sent back to the client.

UI update: The entire application is modified or updated according to the context of use, the user and UI requirements.

When it comes to UI visualization, the application hands over a semantical description of the required inputs, outputs, and actions to the rendering system. A handling chain based on the Conditional Transformation Core processes the abstract input and arranges a concrete UI representation based on rules persisted in a Prolog database (cp. section 7.1).

8.4 Transformation Chain

The transformation is composed of several steps (cp. figure 8.2). The first one is the definition of a SUI according to section 8.1 and 6.1. Upon SUID, the model is translated into SUID facts. As the delivery context [Tran 2002] is also part of the requirements, the context data must be collected and included in the Prolog fact base.

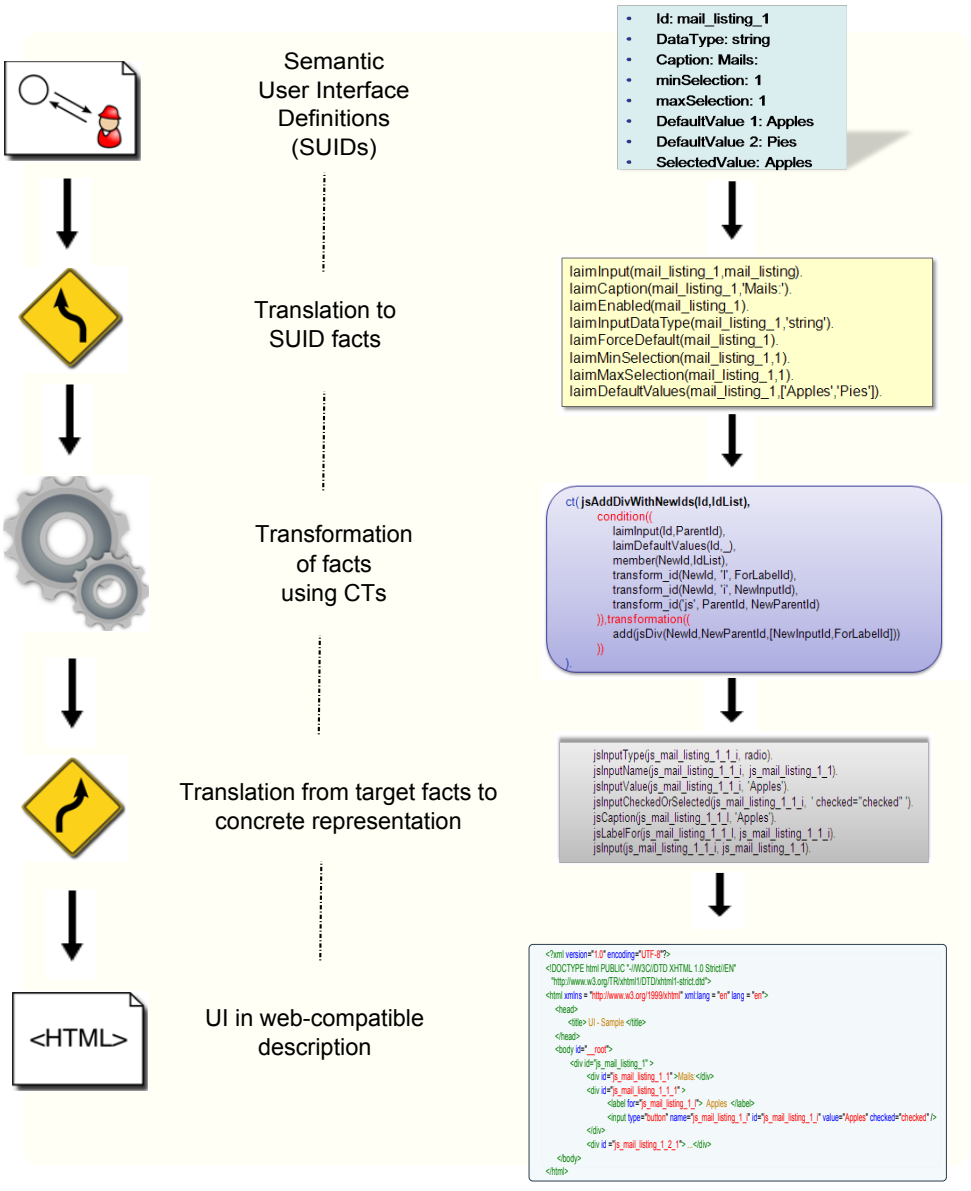

Figure 8.2: A detailed view of the LAIM transformation chain.

8.4.1 Delivery Context Detection

The user preference [Wagner and Paolucci] and the context of use, such as the UI requirements, can also be translated into facts and included into the user module (cp. 4.1.4). To detect the delivery context, we used the WURFL framework presented in section 3.1. Doing this we extracted information such as the supported markup language and so on. The supported markup is used for example as input variable to parse the application property file. The extracted information is parsed and included into the fact base, which contains the LAIM model translated into Prolog facts. For instance, facts like `laimUICapabilities(mp3,'true').` and `laimElementTag(volume_group,'div').` are added. Thereby `true` means that the mp3 sound format is supported, and the related node tag of the element with `id volume_group` will be a `div` tag.

8.4.2 Meta-Model Definitions

The transformation chain includes two transformation steps: The transformation into SUI facts and Intermediate facts. For each of these transformation steps, a meta model has to be defined.

8.4.2.1 SUI Meta-Model Definition

In order to transform the abstract LAIM model specified in section 6, the LAIM meta model needs to be expressed using the `gAST specification` (cp. section 4.2.1). The meta model is used to describe the internal structure of each LAIM model facts, which fulfill the meta model specification. The meta model specification consists of node facts (cp. section 4.2.1.1), and relation facts (cp. section 4.2.1.2). For instance, listing 4.4 shows a possible definition of the node fact according to the LAIM-Input element definition. During the LAIM specification the attribute's element was classified in two groups: common and specific. This distinction is only for a specification purpose and will not be considered anymore. However, by the meta model definition all attributes are translated as relation facts. Listing 4.5 shows the definition of relation facts according to the LAIM-Input element. Any other relations are be defined after the same principle.

8.4.2.2 Intermediate Meta-Model Definitions

Due to the platform and the target model (e. g. XHTML, XHTML-MP, WML, WAP, etc.) diversity, the transformation process was split into four steps:

1. Model definition.
2. Translation of the defined model into Prolog facts called intermediate facts.
3. Transformation of the intermediate facts using CTs into the target model specific facts.
4. Translation of the target model specific facts into a specific presentation domain.

Referring to the transformation chain (cp. figure 8.2) the last step consist in translating the CT output into the presentation domain. Beforehand, according to the Prolog process model, the CT output model needs to be conformed to a meta model. This meta model is called intermediate model and is related to the final presentation domain. Hence, to define the meta model, the presentation requirements are to be specified. For instance, to print an input element according to the presentation using XHTML syntax (cp. listing 8.3), the input caption is sometimes enclosed between the label tag. Depending on the input type, some attributes (id, name, value, etc) are required, some others (checked="checked", class, etc.) are not required. Depending on this syntax, the meta model definition is made. All other node and relation facts are defined after the same principle or convention. A representation of clauses depending to such meta model definition is illustrated in listing 8.6

Listing 8.3: *Input element according to the XHTML presentation requirements*

```
<label for="js_song_list_1">
  Beyonce - If I Were A Boy
</label>
<input type="radio" id="js_song_list_1" name="song_list_name"
  value="Beyonce - If I Were A Boy" checked="checked"
  ONCLICK="clicked('js_song_list_1');"/>
```

Listing 8.4: *Intermediate meta model definition principle.*

```
/*
* node fact definition
**/
ast_node_def('Intermediate',jsInput,[
      ast_arg(id,       mult(1,1,no), id,   [jsInput]),
      ast_arg(parent,   mult(0,1,no), id,   [id])
]).

/*
* relation fact definition
**/
ast_relation('Intermediate',jsInputType,[
    ast_arg(id,       mult(1,1,no), id,   [jsInput]),
      ast_arg(inputType,   mult(1,1,no), attr,   [atom])
]).

ast_relation('Intermediate',jsInputName,[
    ast_arg(id,       mult(1,1,no), id,   [jsInput]),
      ast_arg(inputName,   mult(1,1,no), attr,   [atom])
]).

ast_relation('Intermediate',jsInputValue,[
    ast_arg(id,       mult(1,1,no), id,   [jsInput]),
      ast_arg(inputValue,    mult(1,1,no), attr,   [atom])
]).

ast_relation('Intermediate',jsInputChecked,[
    ast_arg(id,       mult(1,1,no), id,   [jsInput]),
      ast_arg(inputChecked,   mult(1,1,no), attr,   [atom])
]).
```

8.4.3 Transformation into SUI Facts

After the meta predicates are defined, the model can be translated into SUI predicates (cp. listing 8.5). The first predicate is the node fact. All other predicates are relation facts and denote the relation to the LAIM-Input node fact.

Listing 8.5: *LAIM-Input predicates translated after the user requirements.*

```
/**
* LAIM-Input predicates
**/
laimInput(song_list,songs).
laimCaption(song_list,'Songs:').
laimSemantic(song_list,'').
laimEnabled(song_list).
laimPriority(song_list,250).
laimForceDefault(song_list).
laimMinSelection(song_list,1).
laimMaxSelection(song_list,1).
laimDefaultValues(song_list,['Beyonce - If I Were A Boy','Beyonce - Halo',
        'Beyonce - Disappear','Beyonce - Broken-Hearted Girl']).
```

8.4.4 Transformation using CTs

In a further step CTs are used to translate facts into intermediate facts (cp. listing 8.6). According to the transformation definition (cp. figure 7.3) each transformation's artifacts are enclosed in a CT. The composition of these rules are made using CT sequences. For Instance to encapsulate the transformation sequence a PROPSEQ 4.3 is used (cp. listing 8.6).

Listing 8.6: *Intermediates Prolog facts related to the presentation domain*

```
1  jsInput(js_song_list_1, js_song_list).
2  jsInput(js_song_list_2, js_song_list).
3  jsInput(js_song_list_3, js_song_list).
4  jsInput(js_song_list_4, js_song_list).
5
6  htmlNode(js_song_list, js_songs, [js_song_list_1, js_song_list_2, js_song_list_3,
       js_song_list_4], div).
7
8  jsInputType(js_song_list_1, radio).
9  jsInputType(js_song_list_2, radio).
10 jsInputType(js_song_list_3, radio).
11 jsInputType(js_song_list_4, radio).
12
13 jsInputName(js_song_list_1, song_list_name).
14 jsInputName(js_song_list_2, song_list_name).
15 jsInputName(js_song_list_3, song_list_name).
16 jsInputName(js_song_list_4, song_list_name).
17
18 jsInputValue(js_song_list_1, 'Beyonce - If I Were A Boy').
19 jsInputValue(js_song_list_2, 'Beyonce - Halo').
20 jsInputValue(js_song_list_3, 'Beyonce - Disappear').
21 jsInputValue(js_song_list_4, 'Beyonce - Broken-Hearted Girl').
22
23 jsInputChecked(js_song_list_1, ' checked="checked" ').
24 jsInputChecked(js_song_list_2, '').
25 jsInputChecked(js_song_list_3, '').
26 jsInputChecked(js_song_list_4, '').
27
28 jsCaption(js_song_list_1, 'Beyonce - If I Were A Boy').
29 jsCaption(js_song_list_2, 'Beyonce - Halo').
30 jsCaption(js_song_list_3, 'Beyonce - Disappear').
31 jsCaption(js_song_list_4, 'Beyonce - Broken-Hearted Girl').
```

At first the processor checks if the Input component is needed. If not, no transformation will be done, and the whole input transformation process will be skipped. Neither the node fact nor the relation facts of this node fact will be transformed. If an element needed to be translated, the fact is stored in the priority element, which is semantically defined as the content selection attribute. An example of that is the case when the element priority is smaller than a given value. In listing 8.7 the first CT is used to check the element priority. During the translation process, we have also differentiated between how many default values were listed. It is essential, to ensure a good transformation. Since LAIM is an abstract model definition, the input element is used to describe any kind of input control related to the presentation domain. As the semantics of the described elements plays a significant role, this is the only way to provide a good transformation process. However, the second CT is used to detect initially if some default values are given. After the number of default values is determined, the input node is translated. In addition, the last CT of the listing is used to translate the default values depending of the transformation rules (cp. figure 7.3). In order to transform the other relation facts, another CT sequence comes in use. This sequence is built in almost the same manner as the sequence presented in listing 8.7. The final result of this transformation step, according from the predicates listed on listing 8.5 are demonstrated on listing 8.6.

Listing 8.7: *CT sequence transformation example.*

```
/*
* At first check if the element is needed ->
* ct( checkElementPriority(+Id,?Result) ). the result is stored
* in Result as fail or true. If the result is true, then the element
* is needed and the node fact, such as the relation facts will be
* translated. During the translation we also differentiated between how many
* default values are listed. It is essential in order to ensure a
* good transformation.
**/
ctseq( translateLaimInputNode(Id,Result),
  propseq(
    ct( checkElementPriority(Id,Result) ),
    orseq(
      ct( translatelaimInputwithNoDefaultValueOrWithOneDefaultValue(Id,Length,
          NewChildIdList, Result) ),
      ct( generateFactsForEachDefaultValues(Id,Length,NewChildIdList, Result) )
    )
  )
).
```

The output of the transformation step presented below is used to create the final presentation. As defined in section 8.4.2.2 the created facts should fulfill the intermediate model specification. The transformation allocation in a a couple of steps (cp. section 8.4.2.2) has the advantage of code reuse. Then it is up to the developer to create the model in which the presentation will be done, this according to the WORA principle (cp. section 2).

8.4.5 Transformation into the Presentation Domain

In the last transformation step, the intermediate model is translated into the presentation domain. To guarantee a good user experience some principles like the One Web Principle (OWP) should be ensured. For this, the presentation logic was define in section 5.2. To define element tag according to the presentation domain, the developer can define an application property file and override the default application properties. For instance the default element tag for a group node is defined as a `div` tag. The group node can be redefined as a `block` for the output in VoiceXML [Oshry et al. 2007] in the property file. To override the common tag nodes, it is also possible to define a tag nodes for each Ids. To provide the element's tag definition according to the target domain variations, the classification can be done by splitting the file into sections beginning with the markup instructions like `#BEGIN LANG = XHTML_ADVANCED` and `#END LANG = XHTML_ADVANCED`, or define a configuration file for each targeted domain. Thereby `XHTML_ADVANCED` is one of the device markup language extracted by querying the WURFL. This instruction denotes that the integrated browser supports HTML, such as the W3C XHTML standards.
After the tag nodes are defined, the developer can proceed with the CSS definition. The application property file can also be used to declare the CSS class values, by binding the element id with the attribute or the class value. During the model transformation the class value is queried and outputed as a value of the class attribute. For example listing 8.8 demonstrates the intermediate facts translation into the presentation model. Thereby, the UI element is created according to the declared node attributes and the attribute value is declared in the application property file. Since the application provides a way to define node tags for different target languages, a model according to each syntax can be automatically generated on demand.

Listing 8.8: *Translation in the output model depending on the presentation domain.*

```
1 jsDivVisitor(Parent,before,Id, _Node,DefferedJsCommandsIn,DefferedJsCommandsOut,
      OutputStream) :-
2   htmlNode(Id, Parent, _ChildIdList, TagNode),
3   (%check if the html node is refferencing some attribute with value
```

```
4         (htmlNodeAttibute(Id, Attibute),not(Attibute='')),
5         (htmlNodeAttributeValue(Id, AttValue);AttValue='') ->
6           format(atom(AttWithValue),'~a="~a"',[Attibute,AttValue]);
7         AttWithValue=''
8     ),%write the html node with the given tag and the reffering  attribute  with value
9     (
10        htmlNodeCSSClassAttibute(Id,Class,ClassValue) ->
11          format(atom(CSS),'~a="~a"',[Class,ClassValue]);
12        CSS=''
13    ),
14    format(atom(FormatString),'<~a id="~a" ~a ~a>',[TagNode,Id,AttWithValue,CSS]).
```

8.4.5.1 Translation into the target Domain

In the last transformation step, an output writer, created using Prolog, is used to translate the result of the CTs transformation process into the output model. Thereby, the predicates are translated according to UI requirement, the delivery context and also to the client server interaction requirements described in section 2.5. For example, listing 8.9 shows the transformation rule, used to translate the predicates, resulting from the CTs transformation step (cp. 8.6). The produced output is similar to the output outlined in listing 8.1. Due to the hierarchical definition of the components element in the presentation domain, the output writer is based on the visitor pattern principle [Freeman et al.].

Listing 8.9: *Predicates translation into the target domain.*

```
1  /*
2  * jsVisitor(jsInput,before,+Id,_,_,_,+OutputStream).
3  * output writer
4  **/
5  jsVisitor(jsInput,before,Id,_,_,_,OutputStream) :-
6    (jsCaption(Id, Caption);Caption=''),
7      (jsInputType(Id, InputType);InputType='text'), %get the input type
8      (jsInputName(Id, InputName);InputName=Id),%get the input name
9    (jsTagName(Id,TagName);TagName='input'),%get the transformed tag name or write input as
         the default one
10   (%check if some attribute with values was provided for this element and or write the default
         one
11     (htmlNodeAttibutes(Id, Attibutes),length(Attibutes, Length1), not(Length1=0)),
12     (htmlNodeAttributeValues(Id, AttValues), length(AttValues, Length2), not(Length2=0))
           ->
13       formatAttributeWithValues(Attibutes,AttValues,0,
14     Length1,_Result,_TmpResult,AttWithValue); %attribute and attribute values parser.
15       AttWithValue=''
```

```
16      ),%write the html node with the given tag and the reffering  attribute  with value
17      (%if the input type is  radio
18        InputType = 'radio' ->
19          format(atom(LabelwithCaption),'<label for="~a">~a</label>',[Id,Caption]);
20          (%if the input type is  checkbox
21            InputType = 'checkbox' ->
22              format(atom(LabelwithCaption),'<label for="~a">~a</label>',[Id,Caption]);
23            LabelwithCaption = Caption
24          )
25      ),
26      (jsInputValue(Id,  InputValue);InputValue=''),
27      (jsInputChecked(Id, InputChecked);InputChecked=''),
28      transform_id(Id, 'i',  NewId),
29      format(atom(FormatString),
30        '~a<~a type="~a" id="~a" name="~a" value="~a" ~a ONCLICK="clicked(''~a'');" ~a
              />',
31        [LabelwithCaption,TagName,InputType,Id,InputName,InputValue,InputChecked,Id,
              AttWithValue]),
32    %write the result  onto the output stream
33      write(OutputStream,FormatString).
```

8.5 Object-Oriented Principle Simulation

During the development, transformation rules were defined (cp. section 7). According to the CT programing model, to ensure the granularity of the rules, we enclose each transformation artifact in one CT. To build complex transformation rules, CT sequences are used. However it is not always possible to express an artifact using only one CT. For instance, the focus is put on the LAIM-Group transformation (cp. section 7.2.1). The LAIM-Group transformation can depend on the position (depth) of the group element, such as the type of children element and the siblings of the group element.

In an OO programing approach, the artifact about the position of the element can be expressed using *method overloading*. Method overloading occurs when a class contains more than one method with the same name, but a different signature. This mechanism is provided from strong typed OOP programming languages. Which one of these methods is used is resolved at compile time.

As in Prolog, the parameters of queries are not typed in CTs. Variable binding is done using unification depending on the context in which it is used. To simulate the *method overloading* mechanism, CT sequences also come in use. For example the Ct sequence showed in listing 8.10 is used to simulate the overloading mechanism in

CTs. The transformation sequence is used to encapsulate the transformation of the laimGroup's clause depending on the position of the group component. Therefore, after the CT sequence head `ctseq( translate...DepthTWO(Id,Depth) )` is called, the variables Id and Depth are unified and one of the outlined CTs is executed according to the type of the children elements. To ensure that only one of the enclosed CTs is executed, the condition tail of each CTs inspects the type of the children, and the transformation is only executed if the type of children is the one, which was estimated.

Listing 8.10: *Method overloading simulation in CTs.*

```
1  /*
2  * ctseq( translateLaimGroupWithDepthTWO(+Id,+Depth) )
3  * CT Sequence used to simulate the method overloading mechanism in CT
4  * Translate translate LaimGroup with Depth equals 2
5  **/
6  ctseq( translateLaimGroupWithDepthTWO(Id,Depth),
7    orseq(
8      %children element are from different types
9      ct( translateLaimGroupWithDepthTWO(Id,Depth,false) ),
10     %children element are from the same type
11     ct( translateLaimGroupWithDepthTWO(Id,Depth,true) )
12   )
13 ).
```

9 Evaluation approach

During the development process and before the application was deployed, an evaluation took place via two mechanisms: An automated validation based on the recommendations of the *W3C* [W3C 2009] and a manual test or human revision J. et al. [2008].

9.1 Testing

To ensure that the implemented application works properly on all devices, the verification phase started by using emulators, on which an acceptance test was done after each iteration of the development process. Here, the objective was just to demonstrate that, according to the developer, a standard conform content was produced. Using emulators, the evaluation allows a desktop testing without loading the application on the device. However it is usually not sufficient to test on device emulators, for emulators behave slightly different than real devices on a real network do. For this reason, the device testing phase was done focusing on a few good mainstream device classes [Wayner 2009]. Furthermore to evaluate the usability of our technique, we compared it with the RIML development approach.

9.2 Comparison with the RIML Techniques

The RIML is based on existing standards. The document created using these standards is augmented with meta-information for the adaptation engines. The adaptation uses context informations in addition to the semantics and syntax of meta data to prepare the device-specific presentation.

9.2.1 RIML Document Definition

The document definition in RIML encompasses the content, the layout, the document structure and the presentation definition (cp. listing 9.1). The layout enclosed

in the head tag is defined using layout container, such as column and row. By defining the layout the presentation properties can be defined using attribute tags. The content definition is done in the body tag using XHTML markups. Using the frames defined within the layout, the corresponding content is mapped. The RIML allows the content pagination. Therefore the section element can be used. This represents semantic hints for page splitting. Other semantic information like user preferences and corresponding devices classes can also be added. To define the device classes, a usability research from the user point of view was carried out. Thereafter, devices with similar behavior were grouped into categories. Using the device classes the author can create different layouts for different device classes. The appropriate layout is chosen at runtime depending on the delivery context.

Listing 9.1: *Layout description with RIML.*

```
<?xml version="1.0" encoding="UTF-8"?>
<html>
    <head>
        <riml:layout eccdc:deviceClassOneOf="DeviceClass1,DeviceClass2">
            <riml:column riml:id="main-column1">
                <riml:frame riml:id="navigation-frame" riml:paginate="false"/>
            </riml:column>
        </riml:layout>
        <smil:switch>
            <riml:layout eccdc:deviceClassOneOf="DeviceClass1,DeviceClass2">
                <riml:column riml:id="main-column1" riml:minWidth="400">
                    <riml:frame riml:id="logoframe" riml:paginate="true"/>
                </riml:column>
            </riml:layout>
            <riml:layout eccdc:deviceClassOneOf="DeviceClass3,DeviceClass4">
                <riml:row riml:id="root-container">
                    <riml:column riml:id="CA">
                        <riml:frame riml:id="f1" riml:minWidth="40"/>
                        <riml:frame riml:id="f2" riml:minWidth="40"/>
                    </riml:column>
                    <riml:riml:frame riml:id="f3" riml:minWidth="50"/>
                </riml:row>
            </riml:layout>
        </smil:switch>
    </head>
    <body>
        <section riml:frameId="navigation-frame">
            <riml:navigation>
                <riml:navigation-links riml:scope="productframe" riml:links="previous"
                    riml:linksValue="relative-order"/>
                <riml:navigation-links riml:scope="productframe" riml:links="next"
                    riml:linksValue="relative-order"/>
            </riml:navigation>
        </section>
        <section riml:frameId="logoframe" eccdc:deviceClassOneOf="DeviceClass4">
            <object data="images/jupiter_logo.jpg" type="image/jpg"/>
        </section>
        <section riml:frameId="logoframe" eccdc:deviceClassOneOf="DeviceClass1">
            <p><strong>Jupiter - Notebooks</strong></p>
        </section>
        <section id="s1" riml:frameId="f1"><!-- section content --></section>
        <section id="s6" riml:frameId="f3"><!-- section content --></section>
    </body>
</html>
```

To provide interactive web applications, the RIML model permits form support based on XForms to separate data definition from UI definition. XForms was designed to be executed on a wide range of devices that might not be able to support a complex and inconvenient JavaScript engine, including mobile phones. Many technologies supporting the creation of XForms documents are presented in Dubinko [2009]. However, using XForms during the model definition leads to increased development efforts [B'far 2004, pp. 351]. For instance, to generate WML from XForms the adaptation engine needs to be able to translate XForms events into WMLScripts. The developer also needs to ensure that the XHTML MIME type `(application/xhtml+xml)` is supported.

9.2.2 LAIM Document Definition

A LAIM document is defined independently from user and domain point of view. The model design is done focusing on the relevant domain aspects. From a domain independent point of view, each UI is composed of an aggregation of three main categories of UI elements:

Input Used to collect user inputs.

Controls or Action Elements Used to initiate user-events.

Outputs Used to output messages to the user.

Using these elements to define the interface, all domain specific presentation elements are excluded from the defined model. Since the target domain proposes many syntactic elements with the same behavior, our framework proposes writing a configuration file in which the expected element tags are defined. For instance, in order output a caption, the author can choose to enclose it in a `<div/>` container or in a `<span/>` tag. In this way, if a `<span/>` tag is expected, this should be defined in the configuration file. Otherwise the caption is printed using by default a `<div/>` tag. Referring to the implementation details (cp. 8) all other information can be stored in property files or even XML files.

9.2.3 RIML Adaptation Engine

The major adaption engine tasks encompass the receiving of a client's request, the recognition of the device type using the UAPROF technique, the content adaptation according to the device properties, and the transformation of the adapted content into the presentation domain. The adaptation engine itself is composed of two sections:

the semantic and syntactic adaptation. Thereby, the semantic adaptation represents the adaptation of the content, depending of the device capabilities. The syntactic adaptation is the adaptation according to the markup supported from the device (WML, XHTML, VoiceXML). Since the RIML framework provides the definition of optional and alternative contents, the appropriate one is selected during the semantic adaptation process. During this step, the user's request is analyzed. For example, the analysis of the HTTP request determines whether a new page is requested, or if the presented actual content should be rendered again. For the syntactic adaption the author should provide different stylesheets for different target markup languages.

9.2.4 Prototype Engineering

As we have suggested in Section 8.1, the prototype is built after the UCD approach. This is to ensure that the application will be delivered upon the user requirements. Here, the information about the users, their tasks, and the context in which the application will be used, was collected. In this way, a single device independent description was developed. Doing this, we identified two contexts of use: The full mode and the restricted mode. In the full mode, all UI components are available and in the restricted mode, just a few of them are displayed 9.1. Therefore, the user's transparent adaptation is done on the server-side according to the priority of the described elements and the device capabilities.

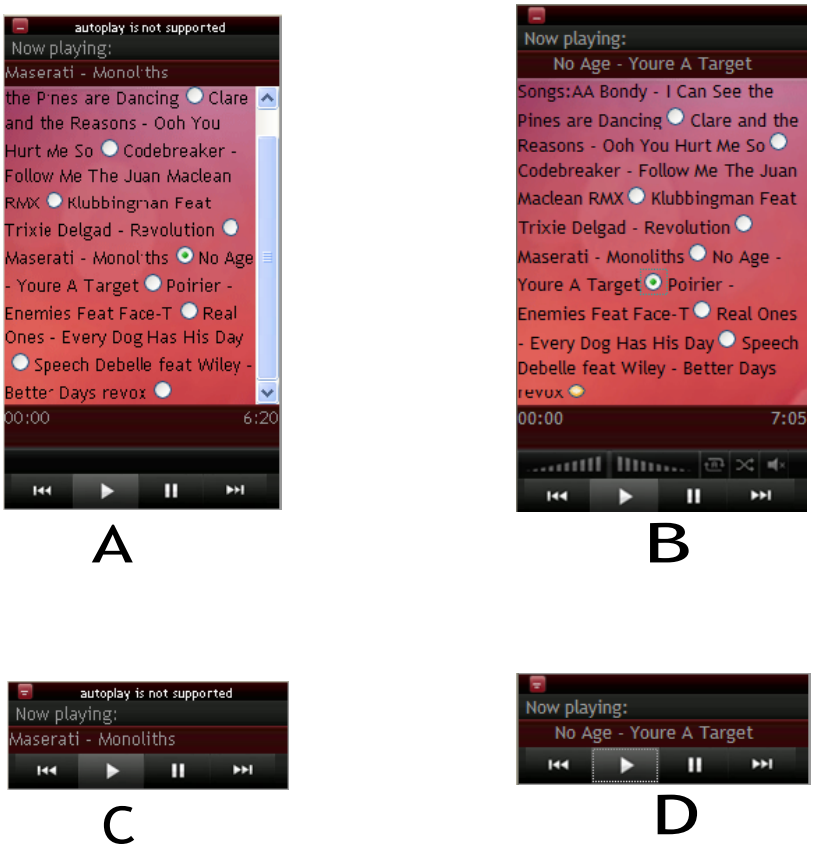

Figure 9.1: The music application (Prototype) is represented in four contexts of use: A represents the music player rendered with a gecko's or opera's browser engine. Where B represents the application rendered with a MSIE's browser engine. C and D represent the application in the restricted mode. Indeed, the integrated media browser plugin from A and C, on which the application is running does not support the auto play back function.

10 Discussion

The UI specification encompasses the definition of the UI model and functionalities. A subset of W3C standards like RIML or SIML can serve as a model definition language. During the content definition, many research approaches provide the ability to define optional and alternative content. Defining optional content for each device class is somehow in contradiction with the "author once, present anywhere principle". Even though it allows the author to prepare special versions of content for different devices, defining optional content for each device class leads to an increase in maintenance cost.
The RIML language profile also provides the possibility to integrate basis content control elements coming from SIML. These control elements are used by the markup mapper to choose the appropriate XSLT stylesheet. Therefore, the designer must provide stylesheets for each targeted markup language. Proving many stylesheets is also one negative development aspect: Then in the course of evolution of the application, each defined style must be improved. Furthermore, when developing for a new category of devices, the specified model, such as layout and stylesheet definitions, should be customized.
In our approach, an abstract SUID was used to describe the interface. Optionally, the author can use property files to map the defined model with domain specific informations. The information contained in these files is translated at runtime and forwarded in the transformation process. The advantage of such an approach is that the configuration properties, as well as the defined input model, can be shared by many applications. The UI presentation depends exclusively on the CSS style definition. In order to improve the UI usability, the target domain is produced without any styles and layout information.
On the other hand, the development with RIML can be more convenient. The RIML framework provides a couple of authoring and conversion tools. Furthermore, the RIML language profile is based on open standards. Therefore, it can be used by anyone while maintaining compatibility among different implementations. Since a standard is open, it is also possible to implement extensions in some future version. Since XSL transformations are popular, it should be very easy to find a developer who can maintain the application.

However, because of the many variants of platform- and context specific adaptations that might be necessary in arbitrary applications, there is an exponential number of possible combinations. Therefore, implementing adaptations for specific combinations is prohibitive. A development with CTs and Prolog provides the ability to build modular and individual adaptations and to compose those adaptations when needed. Furthermore, using the LAIM to define the UI, a strict separation between content, layout, presentation and domain logic is guaranteed. Configuration files are used to map the UI definition with optional and additional domain specific properties. Using such an approach, it can possible to change the presentation whenever needed without changing the running code.

11 Outlook

Web applications can be developed by combining different Web services. When a model is described with LAIM, a strict separation between the presentation logic and the domain logic is provided. In this way the usability of the UI is guaranteed. The described model, as well as the define properties, can be shared along the service interfaces. Accordingly, LAIM can be used to create adaptable UIs coming from different sources at runtime. Doing this, it is possible to use the development with CTs not only for output generation and UIs adaptations, but also for UI composition.
At the current state of our research, the designer should write many CSS stylesheets, which should be chosen according to the output medium. The alternative should be to write the style using relative values. In the future release it should be possible to define the stylesheet once and to transform the given one according to the delivery context and to the UI requirements. It should also be possible to specify the UI functionalities using FSMs. Using property files to capture the specified UI, it should also be possible to configure the behavior of the application without the need to modify the running code. Thus configurable, adaptable UIs can be created using LAIM to describe the UI and CT to transform the described model according to the delivery context.
In a future prospective, LAIM can be used to create highly interactive Web UI in combination with XUP [Yu et al. 2006] instead of Ajax.
Another goal of modern Web 2.0 development [opengardensblog 2006] is the creation of Web applications by transforming the existing ones into context sensitive applications [Hong and Lee 2006]. For that reason a research study about how LAIM can be used to create real-time Web contents adaptation can be carried out.
Since most of the prefetch technologies used in the current cache service schemes did not take into consideration the systematically different requirements of the delivery context of the Client-Server interactions, as well as of the global coordination, the research result presented in Zhang et al. [2008] should be combined with our development approach to guarantee the best user experience.

Annex

11.1 Software used by the author

In the following, there is the list of tools used in this work.

- **Eclipse**
  Open Source IDE
  Website: http://www.eclipse.org/
- **The Prolog Development Tool**
  Website: https://sewiki.iai.uni-bonn.de/research/pdt/users/download
- **Aptana Studio**
  Web development environment, that combines powerful authoring tools for HTML, CSS, and JavaScript.
  Website: http://aptana.com/
- **LaTeX**
  This work was written in LaTeX. TeX Live has been used as LaTeX distribution and as editor, TeXnicCenter has been used.
  Websites: http://www.tug.org/texlive/acquire.html,
  http://www.texniccenter.org/

List of abbreviations

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
AST	Abstract Syntax Tree
CC/PP	Composite Capabilities/Preferences Profiles
CSS	Cascading Style Sheets
CT	Conditional Transformation
CTC	CT Core
CTSeq	CT Sequence
DIAD	Device Independent Application Development
DIAL	Device Independent Authoring Language
DISelect	Content Selection for Device Independence
DIW	Device Independent Web
DIWAF	Device Independent Web Application Framework
DIWE	Device-Independent Web Engineering
DIWF	Device Independent Web Framework
DIWG	Device Independence Working Group
DOM	Document Object Model
FSM	Finite State Machine
gAST	generalized Abstract Syntax Tree
HP	Hewlett-Packard
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
J2EE	Java 2 Enterprise Edition
JSF	JavaServer Faces
JSON	JavaScript Object Notation
LAIM	Language for Abstract Interface Modeling
LCD	Lowest Common Denominator
MP3	MPEG-1 Audio Layer 3
MSIE	Microsoft Internet Explorer
MVC	Model View Controller

MVP	Model View Presenter
OMA	Open Mobile Alliance
OO-H	Object-Oriented-Hypermedia
OOHDM	Object-Oriented Hypermedia Design Method
OWP	One Web Principle
PC	Personal Computer
Prolog	PROgramming in LOGic
RDF	Resource Description Framework
RIML	Renderer-Independent Markup Language
SMIL	Synchronized Multimedia Integration Language
SOA	Service Oriented Architecture
SUI	Semantic User Interface
SUID	SUI Description
UA	User Agent
UAProf	UA Profile
UCD	User-Centered Design
UI	User Interface
UID	Unique identifier
UIFS	UI Functionality Specification
UML	Unified Modeling Language
URL	Uniform Resource Locator
UsiXML	User interface eXtensible Markup Language
W3C	World Wide Web Consortium
WCSS	Wireless CSS
WML	Wireless Markup Language
WORA	Write Once and Run Anywhere
WURFL	Wireless Universal Resource File
XAML	eXtensible Application Markup Language
XHR	XMLHttpRequest
XHTML	eXtensible Hypertext Markup Language
XHTML-MP	XHTML-Mobile Profile
XIML	eXtensible Interface Markup Language
XML	eXtensible Markup Language
XPath	XML Path Language
XSD	XML Schema Definition Language
XSL	eXtensible Stylesheet Language
XSLT	XSL Transformation
XUL	XML User interface Language

XUP ..... eXtensible User interface Protocol

List of Figures

Listings Index

Bibliography

Alliance 2006

ALLIANCE, Open M.: Wireless CSS Specification / Open Mobile Alliance (OMA). Version: 2006. http://www.openmobilealliance.org/technical/release_program/docs/Browsing/V2_3-20080331-A/OMA-WAP-WCSS-V1_1-20061020-A.pdf. 2006. – Forschungsbericht 5.2.2.1

alliance 2008a

ALLIANCE, Openajax: *Introducing Ajax and OpenAjax*. Internet. http://www.openajax.org/whitepapers/IntroducingAjaxandOpenAjax.php. Version: 02 2008. – Last Visited: 2009.02.08 2.6

alliance 2008b

ALLIANCE, Openajax: *Introduction to Mobile Ajax for Developers*. Internet. http://www.openajax.org/whitepapers/IntroductiontoMobileAjaxforDevelopers.php. Version: 02 2008. – Last Visited: 2009.02.08 2.6

Axelsson et al. 2006

AXELSSON, Jonny ; BIRBECK, Mark ; DUBINKO, Micah ; EPPERSON, Beth ; ISHIKAWA, Masayasu ; MCCARRON, Shane ; NAVARRO, Ann ; PEMBERTON, Steven: XHTML 2.0 / w3c. Version: 2006. http://www.w3.org/TR/2006/WD-xhtml2-20060726. 2006. – Forschungsbericht 5.2.2.1

Berners-Lee et al. 1994

BERNERS-LEE, T. ; MASINTER, L. ; MCCAHILL, M.: Uniform Resource Locators (URL) / tools.ietf.org. 1994. – Tech Report. – Edition: September 7, 1994; URL:http://tools.ietf.org/html/draft-ietf-uri-url-07 3.2

B'far 2004

B'FAR, Reza: *Mobile Computing Principles: Designing and Developing Mobile Applications with UML and XML*. New York, NY, USA : Cambridge University Press, 2004. – ISBN 0521817331 9.2.1

Bickmore and Schilit 1997

BICKMORE, Timothy W. ; SCHILIT, Bill N.: Digestor: Device-independent Access

to the World Wide Web. In: *Digestor:* (1997). http://citeseer.ist.psu.edu/old/bickmore97digestor.html 1, 3.2

Bihler et al. 2008

Bihler, Pascal ; Fotsing, Merlin ; Kniesel, Günter ; ; Joffroy, Cedric: Using Conditional Transformations for Semantic User Interface Adaptation. In: Kotsis, Gabriele (Hrsg.) ; Taniar, Daviv (Hrsg.) ; Parede, Eric (Hrsg.) ; Khalil, Ismail (Hrsg.) ; November (Veranst.): *The 10th International Conference on Information Integration and Web-based Application and Services (iiWAS2008)*. Linz : ACM, Inc, September 2008, S. 677 – 680 1, 2.2, 2.3, 2.4, 4.4, 6.1

Bihler and Kniesel 2007

Bihler, Pascal ; Kniesel, Günter: Seamless Cross-Application Workflow Support by User Interface Fusion. In: Christina Brodersen, Susanne B. (Hrsg.) ; Klokmose, Clemens N. (Hrsg.): *Multiple and Ubiquitous Interaction*. DAIMI PB-581, University of Aarhus, 2007 2.2

Bos et al. 2009

Bos, Bert ; Çelik, Tantek ; Hickson, Ian ; Lie, Hakon Wium: Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification / W3C. Version: 2009. http://www.w3.org/TR/2009/CR-CSS2-20090423/. 2009. – Forschungsbericht 5.2.2, 5.2.2.1

Boyer 2009

Boyer, John M.: XForms 1.1 / W3C. Version: 2009. http://www.w3.org/TR/2009/PR-xforms11-20090818/. 2009. – Forschungsbericht 7.2.1

Bugliesi et al. 1994

Bugliesi, Michele ; Lamma, Evelina ; Mello, Paola ; Paola: Modularity in logic programming. In: *Proceedings of the eleventh international conference on Logic programming*. Cambridge, MA, USA : MIT Press, 1994. – ISBN 0–262–72022–1, S. 15–17 4.3

Bulterman et al. 2008

Bulterman, Dick ; Jansen, Jack ; Cesar, Pablo ; Mullender, Sjoerd ; Hyche, Eric ; DeMeglio, Marisa ; Quint, Julien ; Kawamura, Hiroshi ; Weck, Daniel ; Pañeda, Xabiel G. ; Melendi, David ; Cruz-Lara, Samuel ; Hanclik, Marcin ; Zucker, Daniel F. ; Michel, Thierry: Synchronized Multimedia Integration Language (SMIL 3.0) / W3C. Version: 2008. http://www.w3.org/TR/2008/REC-SMIL3-20081201/. 2008. – Forschungsbericht 3.4.1.1, 3.4.1.3

Butler et al. 2002

BUTLER, Mark ; GIANNETTI, Fabio ; GIMSON, Roger ; WILEY, Tony: Device Independence and the Web. In: *IEEE Internet Computing September/October 2002 pp 81-86* HPL-2002-249 (2002), 5. http://www.hpl.hp.com/research/papers/2003/device_independence.pdf 3.3

Clark 1999

CLARK, James: XSL Transformations (XSLT) Version 1.0 / W3C. Version: 1999. http://www.w3.org/TR/1999/REC-xslt-19991116. 1999. – XML Language transformation 2.1

cs.odu.edu 2003

CS.ODU.EDU: *Quantification - Forming Propositions from Predicates.* Internet. http://www.cs.odu.edu/~toida/nerzic/content/logic/pred_logic/quantification/quantification.html. Version: September 2003. – Last update: Dienstag, 23. September 2003 17:12:42 4.1.1

Daintith

DAINTITH, John: *closed-world assumption.* Internet. http://www.encyclopedia.com/doc/1O11-closedworldassumption.html. – A Dictionary of Computing. 2004. Retrieved September 17, 2009 from Encyclopedia.com 4.1.2

Degler 2006

DEGLER, Duane: Preliminary Analysis of Users and Tasks for the Semantic Web. In: *User Focus 2006 conference proceedings* (2006). http://www.designforcontext.com/publications/lb_users_tasks_semantic_web.pdf 8.1

Dubinko 2009

DUBINKO, Micah: *XML.com: Ten Favorite XForms Engines.* http://www.xml.com/lpt/a/1281. Version: 09 2009. – Internet: September 10, 2003; XML.com Copyright © 1998-2006 O'Reilly Media, Inc. 9.2.1

Engels et al. 2008

ENGELS, Gregor ; HUMM, Andreas Hessand B. ; JUWIG, Oliver ; LOHMANN, Marc ; RICHTER, Jan-Peter: *Quasar Enterprise: Designing Service-oriented Application Landscapes.* Auflage 2008.03.01. dpunkt Verlag, 2008 (ISBN-978-3-89864-506-5) 7.1

Feldt 2007

FELDT, Kenneth C. ; FELDT, Kenneth C. (Hrsg.): *Programming Firefox Building Rich Internet Applications with XUL.* First Edition April 2007. O'Reilly, 2007 (ISBN 978-0-596-10243-2). – 511 S. 3.3

Forum 2001

FORUM, WAP: WAG UAProf Version 20-Oct-2001 / OMA. Version: 2001. http://www.openmobilealliance.org/tech/affiliates/wap/wap-248-uaprof-20011020-a.pdf. 2001. – Forschungsbericht. – Wireless Application Protocol WAP-248-UAPROF-20011020-a 3.1

Foundation 2009

FOUNDATION, Windows P.: *(XAML) Extensible Application Markup Language - Windows Presentation Foundation*. Internet. http://msdn.microsoft.com/en-us/library/ms747122.aspx. Version: 2009. – Online; Freiday, 30-January-2009 14:45:34 3.3

Francisco M. Trindade

FRANCISCO M. TRINDADE, Marcelo S. P.: UsiXML4ALL - A Multiplatform Software Development Tool. 3.3

Freeman et al.

FREEMAN, Elisabeth ; SIERRA, Kathy ; BATES, Bert: *Head First design patterns*. O'Reilly (9780596007126) 6.1, 8.4.5.1

Gao et al. 2009

GAO, Shudi (. ; SPERBERG-MCQUEEN, C. M. ; TECHNOLOGIES, Black M. ; THOMPSON, Henry S.: W3C XML Schema Definition Language (XSD) / W3C. 2009. – Forschungsbericht. – http://www.w3.org/TR/2009/CR-xmlschema11-1-20090430/ 4, 6.1

Garrett

GARRETT, Jesse J.: *Ajax: A New Approach to Web Applications*. Internet. http://adaptivepath.com/ideas/essays/archives/000385.php. – Last Update: February 18, 2005 1, 2.6, 8.2

Giannetti 2002

GIANNETTI, Fabio: Device Independence Web Application Framework (DIWAF). In: *W3C Device Independence Authoring Techniques Workshop*. St Leon, Germany, September 2002 2.2, 3.4.2.2, 6.2.1

Gimson et al. 2003

GIMSON, Roger ; FINKELSTEIN, Shlomit R. ; MAES, Stéphane ; SURYANARAYANA, Lalitha: Device Independence Principles / W3C. Version: 2003. http://www.w3.org/TR/2003/NOTE-di-princ-20030901/. 2003. – Forschungsbericht 3

Gimson et al. 2006

GIMSON, Roger ; LEWIS, Rhys ; SATHISH, Sailesh: Delivery Context Overview

for Device Independence / W3C. Version: 2006. http://www.w3.org/TR/2006/NOTE-di-dco-20060320/. 2006. – Forschungsbericht 3.1

Glover and Davies 2005

GLOVER, T. ; DAVIES, J.: Integrating device independence and user profiles on the Web. In: *BT Technology Journal* Volume 23, Number 3 / Juli 2005 (2005), Jully, Nr. 1358-3948 (Print) 1573-1995 (Online), S. 239–248. http://dx.doi.org/10.1007/s10550-005-0045-y. – DOI 10.1007/s10550–005–0045–y 3.2

Gómez et al. 2001

GÓMEZ, Jaime ; CACHERO, Cristina ; PASTOR, Oscar: Conceptual Modeling of Device-Independent Web Applications. In: *IEEE MultiMedia* 8 (2001), Nr. 2, S. 26–39. http://dx.doi.org/http://dx.doi.org/10.1109/93.917969. – DOI http://dx.doi.org/10.1109/93.917969. – ISSN 1070–986X 5.1.1

Goodger et al. 2008

GOODGER, Ben ; HICKSON, Ian ; HYATT, David ; WATERSON, Chris: *XML User Interface Language (XUL) 1.0*. Internet. http://www.myxaml.com/wiki/ow.asp?DeclarativeVsImperativeProgramming. Version: 2008. – Last edited, 2008.07.09 10:22:58 3.3

Grassel et al.

GRASSEL, Guido ; LAUFF, Markus ; SPRIESTERSBACH, Axel ; WASMUND, Michael: *Definition and prototyping of a Renderer-independent ML* 3.4.1.1

Group 2002

GROUP, Web C.: Content Adaptation with the Add-on Technique / Web Commerce Group. Version: 2002. http://www-mit.w3.org/2002/07/DIAT/posn/wcg/wcg.html. 2002. – Forschungsbericht 3.3

Hanrahan and Merrick 2004

HANRAHAN, Rotan ; MERRICK, Roland: Authoring Techniques for Device Independence / W3C Working Group. Version: 2004. http://www.w3.org/TR/2004/NOTE-di-atdi-20040218/. 2004. – Forschungsbericht 3.3

Hong and Lee 2006

HONG, Youn-Sik ; LEE, Ki-Young: A Real-Time Web Contents Adaptation for Mobile User. In: *Computational Science and Its Applications - ICCSA 2006*, 2006, 249–258 11

Hong et al. 2003

HONG, Youn-Sik ; PARK, In-Sook ; RYU, Jeong-Taek ; HUR, Hye-Sun: Pocket News : News Contents Adaptation For Mobile User. (2003). www.ht03.org/papers/pdfs/10.pdf 1

Hors et al. 2004

HORS, Arnaud L. ; HÉGARET, Philippe L. ; LAUREN WOOD, SoftQuad ; NICOL, Gavin ; ROBIE, Jonathan ; CHAMPION, Mike ; BYRNE, Steve: Document Object Model (DOM) Level 3 Core Specification / W3C. Version: 2004. http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407. 2004. – Forschungsbericht 5.1.2

Hwang et al. 2002

HWANG, Yonghyun ; SEO, Eunkyong ; KIM, Jihong: WebAlchemist: A Structure-Aware Web Transcoding System for Mobile Devices. In: *Mobile Search Workshop, May 7, 2002, Honolulu, Hawaii, USA. ACM 1-58113-449-5/02/0005.* (2002). http://dx.doi.org/http://davinci.snu.ac.kr/Download/msw02.pdf. – DOI http://davinci.snu.ac.kr/Download/msw02.pdf 2.2

inc. 2004

INC., Amzi!: *Adventure in Prolog.* Internet. http://www.amzi.com/AdventureInProlog/. Version: March 2004. – Last Updated: Freitag, 21. Mai 2004 19:24:17 4.1.1, 4.1.3

ISO13407 1999

ISO13407: *Human-centred design processes for interactive systems.* Internet, 1999 8.1

Iyengar 1994

IYENGAR, Sudharsan R.: Optimal Backtracking based on Failure-bindings in Prolog, 1994, S. 1005–1025. – Proc. ICCI94, 1994 Int. Conf. on Computing and Information 4.1.5

J. et al. 2008

J., Diaz ; I., Harari ; P., Amadeo: WW3C mobile web best practices evaluation of an educational website. In: *Information Technology Interfaces, 2008. ITI 2008. 30th International Conference* ISSN: 1330-1012, ISBN: 978-953-7138-12-7, INSPEC Accession Number: 10140489, Digital Object Identifier: 10.1109/ITI.2008.4588447, Current Version Published: 2008-08-05 (2008), 421-426. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4588447 9

Jansen and Bulterman 2008

JANSEN, Jack ; BULTERMAN, Dick C.: Enabling adaptive time-based web applications with SMIL state. In: *DocEng '08: Proceeding of the eighth ACM symposium on Document engineering.* New York, NY, USA : ACM, 2008. – ISBN 978–1–60558–081–4, S. 18–27 3.4.1.3

Johnson 1998

JOHNSON, Peter: Usability and Mobility; Interactions on the move. (1998). http://www.dcs.gla.ac.uk/~johnson/papers/mobile/HCIMD1.html. – First Workshop on Human Computer Interaction with Mobile Devices; GIST Technical Report G98-1; 21-23rd May 1998. 1

Karimpour et al. 2008

KARIMPOUR, Habib ; ISAZADEH, Ayaz ; MOSHKENANI, Mohsen S.: Object-Oriented Hypermedia Design and J2EE Technology for Web-based. In: *Issues in Informing Science and Information Technology* 5 (2008), 729–740. http://proceedings.informingscience.org/InSITE2008/IISITv5p729-740Karim475.pdf 5.1.1, 5.1.2

Kawash 2004

KAWASH, Jalal: Declarative user interfaces for Handheld Devices. In: *WISICT '04: Proceedings of the winter international symposium on Information and communication technologies*, Trinity College Dublin, 2004, 1-6 3.4.3, 5.1.2, 5.3, 6.1

Kerer and Kirda 2001

KERER, Clemens ; KIRDA, Engin: *Web Engineering*. Springer Berlin / Heidelberg, 2001. – 135–147 S. http://dx.doi.org/10.1007/3-540-45144-7_14. http://dx.doi.org/10.1007/3-540-45144-7_14. – ISBN: 978-3-540-42130-6 3.4.2.1

Kindler 2007

KINDLER, Dipl.-Inf. M.: *Entwicklung mobiler Web-Anwendungen*. internet. http://www.cityexperience.net/site/uploads/media/t3n_wurfl_einfuehrung_final.pdf. Version: 2007.. – © yeebase 2007. www.t3n-magazin.de 3.1, 3.3

Kirakowski and Collins 1999

KIRAKOWSKI, Jurek ; COLLINS, Karen: *An introduction to ISO 13 407*. Internet. http://www.ucc.ie/hfrg/emmus/methods/iso.html. Version: September 1999. – Copyright EMMUS 1999. Last updated: September 29, 1999. 8.1

Kirda and Kerer 2004

KIRDA, Engin ; KERER, Clemens: DIWE: A Framework for Constructing Device-Independent Web Applications. In: *UMICS 2004, LNCS 3272* Volume 3140/2004 (2004), S. 96110. – L. Baresi et al. (Eds.): UMICS 2004, LNCS 3272, pp. 96110, 2004. Springer-Verlag Berlin Heidelberg 2004 2.2, 3.4, 3.4.2.1

Kiss 2007

KISS, Cédric: Composite Capability/Preference Profiles (CC/PP): Structure

and Vocabularies 2.0 / W3C. Version: 2007. http://www.w3.org/TR/2007/WD-CCPP-struct-vocab2-20070430. 2007. – Forschungsbericht 3.1

Kleppe et al. 2003

Kleppe, Anneke G. ; Warmer, Jos B. ; Bast, Wim ; Co, Addison-Wesley P. (Hrsg.): *MDA Explained: The Model Driven Architecture(TM): Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc.„ 2003. – ISBN 9787115118127. – Boston, MA, USA, rev and revised edition, April 2003 4, 4.1

Kniesel 2006a

Kniesel, Günter: A Logic Foundation for Conditional Program Transformations / Computer Science Department III, University of Bonn, Germany. 2006 (IAI-TR-2006-01, ISSN 0944-8535). – Technical report 4.1, 4.4

Kniesel 2006b

Kniesel, Günter: A Logic Foundation for Program Transformations / CS Dept. III, University of Bonn. Germany, January 2006 (IAI-TR-2006-01). – Technical report. –
Online: http://www.cs.uni-bonn.de/~gk/papers/IAI-TR-2006-1-kniesel-CTS.pdf 2.4, 4.2

Kniesel 2008

Kniesel, Günter: Detection and Resolution of Weaving Interactions. In: *Transactions on Aspect-Oriented Software Development (Special issue 'Dependencies and Interactions with Aspects')* LNCS (2008) 4.4

Kniesel and Bardey 2006

Kniesel, Günter ; Bardey, Uwe: An Analysis of the Correctness and Completeness of Aspect Weaving. In: *Proceedings of Working Conference on Reverse Engineering 2006 (WCRE 2006)*. Benevento, Italy : IEEE, October 2006. – ISBN 0–7695–2719–1, 324-333 4.4

Kniesel and Koch 2003

Kniesel, Günter ; Koch, Helge: Program-independent Composition of Conditional Transformations / Computer Science Department III, University of Bonn, Germany. 2003 (IAI-TR-03-1, ISSN 0944-8535). – Technical report. – Updated version published in: Science of Computer Programming, special issue on Program Transformation, 52 1-3 (2004), p. 9-51, Elsevier Science, 2004 4.1, 4.4

Koskimies 2004

Koskimies, Oskari: *Metadata for Client-side Content Adaptation*. Internet. http://web5.w3.org/2004/06/DI-MCA-WS/submissions/position-nokia.

html. Version: 2004. – Disclaimer: Opinions and views expressed in this paper belong to the author and do not necessarily represent the views of Nokia. 3.3

Kreger 2001

KREGER, Heather: *Web Services Conceptual Architecture (WSCA 1.0)*. Internet. http://www.cs.uoi.gr/~zarras/mdw-ws/WebServicesConceptualArchitectu2.pdf. Version: May 2001. – IBM Software Group 3.4.3

Ku et al. 2005

KU, Tai-Yeon ; PARK, Dong-Hwan ; MOON, Kyeong-Deok: Device-Independent Markup Language. In: *Computer and Information Science, ACIS International Conference on* 0 (2005), S. 508–512. ISBN 0–7695–2296–3 2.2

Law 2007

LAW, Derek: Taligent MVP in interactive statistical graphics. In: *Computational Statistics* Bd. 23 Department of Statistics, The University of Auckland, Physica Verlag, An Imprint of Springer-Verlag GmbH, 2007, 487–495 5.1.2

Lewis 2007

LEWIS, Rhys: Mobile Ajax and Application Adaptation. In: *W3C/Open Ajax Alliance Workshop on Mobile Ajax, September 2007* (2007). http://www.w3.org/2007/06/mobile-ajax/papers/volantis.lewis.pdf. – Volantis Systems Ltd. Position Paper 2.6

Lewis Sond

LEWIS, Rhys: W3C Mobile Web Initiative Workshop Device Independence and the Mobile Web Initiative / W3C Device Independence Working Group (DIWG). Version: Sonday, 17. October 2004 23:35:03. http://dx.doi.org/http://www.w3.org/2004/10/MWIWS-papers/DIWGAndMobileWeb.html. Sonday, 17. October 2004 23:35:03. – Texhreport 3.3

Lewis et al. 2007

LEWIS, Rhys ; MERRICK, Roland ; FROUMENTIN, Max: Content Selection for Device Independence (DISelect) 1.0 / W3C. Version: 2007. http://www.w3.org/TR/2007/CR-cselection-20070725/. 2007. – Forschungsbericht 3.4.1.2

Limbourg et al. 2004

LIMBOURG, Quentin ; VANDERDONCKT, Jean ; MICHOTTE, Benjamin ; BOUILLON, Laurent ; FLORINS, Murielle ; TREVISAN, Daniela: USIXML: A User Interface Description Language for Context-Sensitive User Interfaces. In: LUYTEN, K. (Hrsg.) ; ABRAMS, M. (Hrsg.) ; LIMBOURG, Q. (Hrsg.) ; VANDERDONCKT, J. (Hrsg.) ; ACM (Veranst.): *Proceedings of the ACM AVI'2004 Workshop "Devel-*

oping User Interfaces with XML: Advances on User Interface Description Languages" (Gallipoli,May 25, 2004) ACM, 2004, 55-62 1, 3.3

Ludewig 2003

Ludewig, Jochen: Models in software engineering an introduction. In: *Software and Systems Modeling* Volume 2 (2003), March, 5-14. http://dx.doi.org/10.1007/s10270-003-0020-3. – DOI 10.1007/s10270–003–0020–3 6, 6.3

Manuel Cantera Fonseca and Hierro 2007

Manuel Cantera Fonseca, Javier S. Ignacio Marin Prendes P. Ignacio Marin Prendes ; Hierro, Juan J.: *Declarative Models for Ubiquitous Web Applications Morfeo-MyMobileWeb - Position Paper*. Morfeo-MyMobileWeb Project, 04 2007. http://www.w3.org/2007/02/dmdwa-ws/Papers/jose-m-c-fonseca.html. – position paper 3.3

Mens et al. 2005

Mens, Tom ; Czarnecki, Krzysztof ; Gorp, Pieter V.: 04101 Discussion - A Taxonomy of Model Transformations. In: Bezivin, Jean (Hrsg.) ; Heckel, Reiko (Hrsg.): *Language Engineering for Model-Driven Software Development*. Dagstuhl, Germany : Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2005 (Dagstuhl Seminar Proceedings 04101). – ISSN 1862–4405 4

MicroSystem 2009

MicroSystem, SUN: *JavaServer Faces Technologies - SUN MicroSystem*. Internet. http://java.sun.com/javaee/javaserverfaces/reference/. Version: 2009. – Last modified, 2009.01.26 22:51:29 3.3

Molina 2004

Molina, Pedro J.: User interface generation with OlivaNova model execution system. In: *IUI '04: Proceedings of the 9th international conference on Intelligent user interfaces*. New York, NY, USA : ACM, 2004. – ISBN 1–58113–815–6, 358–359 5.1.1

Molina et al. 2002

Molina, Pedro J. ; Meliá, Santiago ; Pastor, Oscar: JUST-UI: A User Interface Specification Model. In: *Proceedings of Computer Aided Design of User Interfaces* (2002), 323–334. http://users.dsic.upv.es/~west/TIMIU'02/ficheros/Molina-CADUI2002.pdf. – CADUI'2002, Les Valenciens (2002) France 5.1.1

Network 2009

Network, Openwave D.: *Openwave Developer Network - Support - Guides & References:*. Internet. http://developer.openwave.com/dvl/support/

documentation/guides_and_references/index.htm. Version: 09 2009. – Last updated: Dienstag, 8. September 2009 16:37:32 5.2.2.1

cellular news.com 2008

NEWS.COM cellular: *Worldwide Mobile Cellular Subscribers to Reach 4 Billion Mark Late 2008*. internet. http://www.cellular-news.com/story/33811.php. Version: 09 2008 (document)

Nutt 1995

NUTT, Gary J.: Software engineering process model: a case study. In: *COCS '95: Proceedings of conference on Organizational computing systems*. New York, NY, USA : ACM, 1995. – ISBN 0–89791–706–5, S. 324–335 6.1

O'Keefe 1985

O'KEEFE, Richard A.: Towards an Algebra for Constructing Logic Programs. In: COHEN, J. (Hrsg.) ; CONERY, J. (Hrsg.) ; IEEE Computer Society Press (Veranst.): *SLP*. Boston, Massachusetts : Proceedings of IEEE Symposium on Logic Programming, July 15-18 1985 (IEEE-CS 1985), S. 152–160. – ISBN 0-8186-0636-3 4.3

OMA

OMA: *Material from Affiliates - Wireless Application Protocol*. Internet. http://www.openmobilealliance.org/Technical/wapindex.aspx#wap20. – Last visited: Dienstag, 8. September 2009 16:01:16 5.2.2.1

OMG 2009

OMG: *OMG Unified Modeling LanguageTM (OMG UML), Superstructure Version 2.2*. Internet. http://www.omg.org/cgi-bin/doc?formal/09-02-02. Version: 02 2009. – OMG Document Number: formal/2009-02-02; Standard document URL: http://www.omg.org/spec/UML/2.2/Superstructure; Normative machine-readable files:n ptc/08-05-07, ptc/08-05-08, ptc/08-05-09, ptc/08-05-10, ptc/08-05-11, ptc/08-05-12. 4, 5.1.2

opengardensblog 2006

OPENGARDENSBLOG: *Mobile web 2.0: AJAX for mobile devices why mobile AJAX will replace both J2ME and XHTML as the preferred platform for mobile applications development*. Internet. http://opengardensblog.futuretext.com/archives/2006/01/mobile_web_20%_a.html. Version: 01 2006. – Online: January 1, 2006 1, 2.6, 11

OpenWiki 2004

OPENWIKI: *Declarative Vs. Imperative Programming*. Internet. http://www.myxaml.com/wiki/ow.asp?DeclarativeVsImperativeProgramming. Version: 2004. – Last edited July 16, 2004 3.3

Ort and Basler 2006

Ort, Ed ; Basler, Mark: *Ajax Design Strategies*. Internet, October 2006. – Last update: Dienstag, 21. Juli 2009 01:21:35 5.3

Oshry et al. 2007

Oshry, Matt ; Auburn, RJ ; Baggia, Paolo ; Bodell, Michael ; Burke, David ; Burnett, Daniel C. ; Candell, Emily ; Carter, Jerry ; McGlashan, Scott ; Lee, Alex ; Porter, Brad ; Rehor, Ken: Voice Extensible Markup Language (VoiceXML) 2.1 / W3C. Version: 2007. http://www.w3.org/TR/2007/REC-voicexml21-20070619/. 2007. – Forschungsbericht 5.2.2.1, 8.4.5

Padawitz 1988

Padawitz, Peter: *Computing in Horn Clause Theories*. Texts in Theoretical Computer Science. An EATCS. Springer Berlin, 1988 (Series 16). – 700 S. – isbn: 978-3-540-19427-9 4.1.2

Passani 2008

Passani, Luca: Global Authoring Practices for the Mobile Web. (2008). http://www.passani.it/gap/ 3.1

Pastor et al. 1997

Pastor, Oscar ; Insfrán, Emilio ; Pelechano, Vicente ; Romero, José ; Merseguer, José: OO-METHOD: An OO Software Production Environment Combining Conventional and Formal Methods. In: *CAiSE '97: Proceedings of the 9th International Conference on Advanced Information Systems Engineering*. London, UK : Springer-Verlag, 1997. – ISBN 3–540–63107–0, S. 145–158 5.1.1

Potel

Potel, Mike: *MVP: Model-View-Presenter The Taligent Programming Model for C++ and Java*. Internet. http://www.wildcrest.com/Potel/Portfolio/mvp.pdf. – VP & CTO Taligent, Inc. 5.1.2

Puerta and Eisenstein 2002

Puerta, Angel ; Eisenstein, Jacob: XIML: a common representation for interaction data. In: *IUI '02: Proceedings of the 7th international conference on Intelligent user interfaces*. New York, NY, USA : ACM Press, 2002. – ISBN 1–58113–459–2, S. 214–215 1, 3.3

Rabin and McCathieNevile 2008

Rabin, Jo ; McCathieNevile, Charles: Mobile Web Best Practices 1.0 / W3C. Version: 07 2008. http://www.w3.org/TR/2008/REC-mobile-bp-20080729/. 2008. – Language 3.2, 5.2.1, 5.2.2

Rajapakse

RAJAPAKSE, Damith C.: Techniques for De-fragmenting Mobile Applications: a Taxonomy. http://www.comp.nus.edu.sg/~damithch/files/SEKE2008.pdf 2.2

Rho et al. 2006

RHO, Tobias ; SCHMATZ, Mark ; CREMERS, Armin B.: Towards Context-Sensitive Service Aspects. Version: July 2006. http://roots.iai.uni-bonn.de/research/logicaj/downloads/papers/RhoSchmatz-OT4AmI06.pdf. In: *ECOOP Workshop 06*. 2006 2.6

Schmidt 2006

SCHMIDT, Douglas C.: Guest Editor's Introduction: Model-Driven Engineering. In: *Computer* 39 (2006), Nr. 2, 25-31. http://dx.doi.org/http://doi.ieeecomputersociety.org/10.1109/MC.2006.58. – DOI http://doi.ieeecomputersociety.org/10.1109/MC.2006.58. – ISSN 0018–9162 4.1

Schmidt et al. 2007

SCHMIDT, Kay-Uwe ; STOJANOVIC, Ljiljana ; STOJANOVIC, Nenad ; THOMAS, Susan: On Enriching Ajax with Semantics: The Web Personalization Use Case. (2007). http://www.eswc2007.org/pdf/eswc07-schmidt.pdf. – SAP Research, CEC Karlsruhe and the FZI Forschungszentrum Informatik 2.6

Schwabe and Rossi 1995

SCHWABE, Daniel ; ROSSI, Gustavo: The object-oriented hypermedia design model. In: *Commun. ACM* 38 (1995), Nr. 8, 45–46. http://dx.doi.org/http://doi.acm.org/10.1145/208344.208354. – DOI http://doi.acm.org/10.1145/208344.208354. – ISSN 0001–0782 5.1.1

Shapiro and Sterling 1994

SHAPIRO, Ehud ; STERLING, Leon: *The Art of PROLOG, 2nd Edition: Advanced Programming Techniques (Logic Programming)*. 2nd Edition. MIT Press, 1994 (Logic Programming). – 549 S. – ISBN: 978-0262691635 4.1

Smith 2007

SMITH, Kevin: Device Independent Authoring Language (DIAL) / W3C. Version: 2007. http://www.w3.org/TR/2007/WD-dial-20070727/. 2007. – Forschungsbericht 3.4.1.2

Stachowiak 1973

STACHOWIAK, H.: *Allgemeine Modelltheorie*. Springer, Berlin, Germany, EU, 1973 6.3

Tilly et al. 2007

TILLY, Károly ; BARANYI, Szabolcs ; PORKOLÁB, Zoltán: Semantic User Interfaces. (2007). www.fluidbusiness.org/events/tilly.pdf 3.3, 6

TM. 2006

TM., GSM A.: Universal Access How Mobile can Bring Communications to All / GSM Association TM. Version: 2006. http://www.gsmworld.com/documents/universal_access_full_report.pdf. 2006. – Forschungsbericht (document)

Torgersson 1996

TORGERSSON, Olof: *A Note on Declarative Programming Paradigms and the Future of Definitional Programming.* http://www.cs.chalmers.se/~oloft/Papers/wm96/wm96.html. Version: 1996. – Edition: Tue Mar 19 11:17:10 MET 1996 3.3

Tran 2002

TRAN, Luu: Developing Device Independent Java Applications with JSR 188. In: *W3C Workshop on Device Independent Authoring Techniques* (2002). http://web5.w3.org/2002/07/DIAT/posn/sun-jsr-188.html. – Last Update: Freitag, 13. September 2002 18:29:51 3.1, 8.4

Visciola 2003

VISCIOLA, Michele: Reflections on the user centered design (UCD) perspective in research on wireless applications. In: *Ubiquity* 4 (2003), Nr. 8, S. 1–1. http://dx.doi.org/http://doi.acm.org/10.1145/772541.772542. – DOI http://doi.acm.org/10.1145/772541.772542 8.1

Visser 2005

VISSER, Eelco: A survey of strategies in rule-based program transformation systems. In: *Journal of Symbolic Computation* 40 (2005), Nr. 1, 831 - 873. http://dx.doi.org/DOI:10.1016/j.jsc.2004.12.011. – DOI DOI: 10.1016/j.jsc.2004.12.011. – ISSN 0747–7171. – Reduction Strategies in Rewriting and Programming special issue 4

Visser et al.

VISSER, Eelco ; MENS, Tom ; WALLACE, Malcolm: *Program-Transformation: Program-Transformation.Org: The Program Transformation Wiki.* Internet. http://www.program-transformation.org/Transform/ProgramTransformation.. – EelcoVisser - 03 May 2001, 01 Apr 2002 - TomMens - 9 Apr 2004 - MalcolmWallace - 07 May 2004 4

Vredenburg 2008

VREDENBURG, Karel: *User-Centered Design.* Internet. https://www-01.ibm.

com/software/ucd/ucd.html. Version: 02 2008. – Copyright IBM Corporation 2007 8.1

W3C 2009

W3C: *Markup Validation Service*. Internet. http://validator.w3.org/. Version: 2009. – Id: index.html,v 1.94 2008-03-19 18:54:15 ot Exp 9

Wagner and Paolucci

WAGNER, Matthias (Hrsg.) ; PAOLUCCI, Massimo (Hrsg.) ; DoCoMo Communications Laboratories Europe (Veranst.): *Enabling Personal Mobile Applications through Semantic Web Services*. Future Networking Lab DoCoMo Communications Laboratories Europe Munich, Germany, 3.1, 8.4.1

wapreview.com Mitt

WAPREVIEW.COM: *How Web to Mobile Transcoding Should Work | Wap Review:*. internet. http://wapreview.com/blog/?p=516. Version: January Mittwoch, 21. Januar 2009 14:15:33. – internet blog 2.2

Wayner 2009

WAYNER, Peter: *A developer's-eye view of smartphone platforms*. http://www.infoworld.com/article/09/01/20/03TC-phone-platforms_1.html. Version: 01 2009. – Internet: 2009-01-20T00:00:00-08:00 9.1

Wei

WEI, Coach K.: *AJAX: Asynchronous Java + XML?* Internet. http://www.intranetjournal.com/articles/200508/ij_08_23_05a.html. – Last Update: 8/23/2005 1, 8.2

Wielemaker 2008

WIELEMAKER, Jan: *SWI-Prolog 5.6.60 Reference Manual*. Kruislaan 419, 1098 VA Amsterdam The Netherlands Tel. (+31) 20 5256121: Human-Computer Studies (HCS, formerly SWI), 2008. http://gollem.science.uva.nl/SWI-Prolog/Manual/ 4.1.4

Wikipedia 2008

WIKIPEDIA: *Content adaptation - Wikipedia The Free Encyclopedia*. Internet. http://en.wikipedia.org/w/index.php?title=Content_adaptation\&oldid=258749779. Version: 2008. – Online; accessed 28-January-2009 3.3

Woo and Jang 2008

WOO, Jongwook ; JANG, MinSeok: The Comparison of WML, cHTML, and XHTML-MP in m-Commerce. In: *JSW* 3 (2008), Nr. 7, 22–29. http://www.academypublisher.com/jsw/vol03/no07/jsw03072229.html 5.2.2.1

Yu et al. 2006

Yu, Jin ; Benatallah, Boualem ; Casati, Fabio ; Saint-Paul, Regis: OpenXUP: an alternative approach to developing highly interactive web applications. In: *ICWE '06: Proceedings of the 6th international conference on Web engineering*, ACM, 2006. – ISBN 1–59593–352–2, 289–296 11

Zhang et al. 2008

Zhang, Xiaowei ; Cao, Donggang ; Tian, Gang ; Chen, Xiangqun: Data Prefetching Driven by User Preference and Global Coordination for Mobile Environments. In: *Grid and Pervasive Computing, International Conference on* 0 (2008), S. 145–150. http://dx.doi.org/http://doi.ieeecomputersociety.org/10.1109/GPC.WORKSHOPS.2008.35. – DOI http://doi.ieeecomputersociety.org/10.1109/GPC.WORKSHOPS.2008.35. ISBN 978–0–7695–3177–9 11

Ziegert et al. 2004

Ziegert, Thomas ; Lauff, Markus ; Heuser, Lutz: Device Independent Web Applications The Author Once Display Everywhere Approach. In: *ICWE 2004, LNCS 3140* Volume 3140/2004 (2004), July, Nr. 978-3-540-22511-9, S. 244255. – 0302-9743 (Print) 1611-3349 (Online) 2.2, 3.4, 3.4.1.1

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich meine Diplomarbeit mit dem Thema

> *Dynamic Web-Based User-Interfaces based on Semantic Descriptions and Context Informations*

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Hereby, i affirm that the diploma thesis on

> *Dynamic Web-Based User-Interfaces based on Semantic Descriptions and Context Informations*

was created exclusively by myself and no sources and auxiliary means other than the ones stated were used. All materials or text which is extracted literally or analogously from other published work is explicitly stated.

Bonn, den 21/09/2009

Merlin J. Fotsing

Index