Multi-Dimensional Context-Aware Adaptation of Service Front-Ends

Project no. FP7 – ICT – 258030

Deliverable D.3.2.1

ASFE-DL: Semantics, Syntaxes and Stylistics (R1)

Due date of deliverable: 30/09/2011

Actual submission to EC date: 30/09/2011

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination level		
PU	**Public**	**Yes**

Document Information	
Lead Contractor	Telefónica I+D
Editor	Telefónica I+D
Revision	v.1.0 (30/09/2011)
Reviewer 1	W4
Reviewer 2	UCL, ISTI-CNR
Approved by	Telefónica I+D
Project Officer	Jorge Gasós

Contributors	
Partner	**Contributors**
TID	**Mari Carmen Rodríguez Gancedo, Francisco Javier Caminero Gil**
UCL	**Vivian Motti**
ISTI-CNR	**Fabio Paternò, Carmen Santoro, Lucio Davide Spano**
SAP	**Safdar Ali**
W3C	**Dave Raggett**
CTIC	**Javier Rodríguez, Cristina González, Ignacio Marín**

Changes			
Version	**Date**	**Author**	**Comments**
0.1	02/08/2011	TID	Section-headers only version of document
0.2	14/09/2011	TID/all	First contributions sent to TID
0.3	16/09/2011	TID	First consolidated version
0.4	21/09/2011	W4	First internal revision
0.5	26/09/2011	TID/all	Second contributions sent to TID

0.6	27/09/2011	TID	Second consolidated version, final draft
0.7	28/09/2011	W4	Final internal Revision
1.0	30/09/2011	TID/All	Final version of the document

Executive Summary

This report describes the initial work in SERENOA in the field of Description Language (DL) for the User Interfaces (UI). The deliverable presents a summarized state of the art in the field of such languages, followed by a description of the three main ones that will inform the final DL for the project, UsiXML, MARIA and IDEAL2. A description of the DL follows, focused in this document on the Abstract UI level of the Cameleon framework, that is, the level which describes UIs in an abstract manner without specifying the particulars for each context and platform. The elements that conform to the AUI description are also discussed. The document ends with a set of conclusions and future lines of work close the document.

Table of Contents

1 Introduction

1.1 Objectives

In this document, we will present the first steps towards a UI Description Language (DL) for SERENOA. This will provide the remainder of the tasks in the project with a unified approach towards the description of the User Interfaces. The focus of the project is SFE (Service Front End) adaptation, so all the elements and strategies that will be presented for the DL will have the easing of the adaptation process a primary objective.

In this document we will focus on the abstract level of the DL. This abstract level, based upon the categorization presented in the Cameleon Framework [Calvary2002], refers to the first implementation level of UIs, immediately after the conceptualization of the Tasks & Concepts to be covered in the interface. Thus, at this level of abstraction we will define platform independent descriptions of the User Interface elements required to fulfill these tasks and concepts. By focusing on this primary level, we will provide a starting description point to begin other implementation tasks in the project such as the runtime or the authoring tool.

Some of the partners in the consortium have done in the past research in the field, including authoring or co-authoring full specifications of description languages such as UCL's UsiXML [Limbourg2004], ISTI/CNR's MARIA [Paternò2009] and CTIC's IDEAL2 [Cantera2010]. Thus, for the sake of efficiency, we will base our SERENOA-DL upon them. This is reflected in the extended State of the Art sections for these languages and also in the description of SERENOA's own solution.

1.2 Audience

Being a public deliverable, this document will be available outside the confines of the project's consortium and is intended to be of interest to the following parties:

a) Members of the consortium, who will find here a detailed description of the fundamentals of the Description Language that the project is to use in the future.
b) Researchers in the relevant fields: adaptation of SFEs, UI theorists, descriptive languages and medium-scale project software engineering.
c) EC officials that will use the information in this document as an account of the activities taken in the project tasks that inform this work.

1.3 Related documents

This document is related to the following project deliverables:

- **D1.2.1 Architectural Specifications (R1)**, in which the architecture of the system is laid out and an overview of how the different pieces of engineering to be produced will fit together in the final system.
- **D2.2.1 CARFO (R1)**, dedicated to the establishment of the first principles of an ontology for the SERENOA framework. In this ontology, some of the concepts that will inform the elements that conform to the SERENOA-DL are defined and their relationships discussed.
- **D2.1.1 CARF and CADS (R1)** in which the theoretical foundation of the project is laid out, therefore affecting as well the definitions of some concepts that will be expressed using the DL
- **D3.1.1 Reference Models Specification (R1),** the main topic of which is the definition of the main models for operation in the adaptation process for SERENOA.
- **D4.5.1 Authoring Environment (R1)** outlines the first concepts of the Authoring Environment for SERENOA applications. In its due time, this module will have to use the syntax of the SERENOA-DL to describe applications, so the discussion of concepts in this initial version is relevant in this report.

Future revisions of the above documents and of this report will be closely related as well and these relationships will be documented in their introductory sections.

1.4 Organization of this document

In section 2, we present the state of the art in Description Languages for User Interfaces. We briefly summarize key achievements in the field before going much deeper into the three main languages that will inform SERENOA-DL, namely UsiXML, MARIA and IDEAL2. Upon examination of the characteristics of the three, we go in section 3 to describe the general outline of the SERENOA-DL roughly corresponding to its metamodel and a brief comparison with the authoring tool requirements from such a DL. In this section we also list the elements that comprise the language itself, including interactors, compositions that may group interactors together and finally behaviours and events that can be specified in the language for the UI elements. We end the document in section 4 with a summary of the achieved work, some lines of the future work to be undertaken in the project task and some short outline of the standardization efforts that are active in SERENOA regarding this topic and with a section describing the references used in the document.

2 State of the art

This section briefly revises the state of the art in User Interface Description Languages (UI-DL) and then provides more detail regarding the three languages that compose the consortium's portfolio of authored and co-authored languages in this field (UsiXML, MARIA and IDEAL2). In the coming sections we will describe the main characteristics of SERENOA-DL which, in turn, has been influenced by the knowledge in the consortium and these languages.

A User Interface Description Language is a set of syntactical rules, semantic symbols and stylistic restrictions that are used to provide a framework for describing a UI. They may relate to different levels of description, from the more abstract (the tasks that the user is to accomplish using the UI or the strategies that are presented) to the more concrete, including fine-grained description of the layout of components and the look and feel that they present to the user.

The UI-DLs thus present a way of separating the processing and business logic of a system from its user experience aspects and, as such, are extremely useful in the successful definition of an application. If adequately constructed, they also present evident engineering and industrial advantages, as the teams for intensive algorithm programming and those that deal with user aspects may be kept separated and focused on their main roles with minimum overlap.

As expected, a wide number of initiatives have been raised in order to produce standardized languages for describing User Interfaces. These have been extremely varied in factors such as their scope, intended abstraction level, market/device penetration and approach to syntax, semantics and stylistics. Here, we will briefly present some illustrative cases, with less emphasis on completeness than on the spotting of key aspects that may be useful for the subsequent definition of SERENOA-DL in this report.

Browser-based

The World Wide Web can be understood as an application based upon the Internet which features a UI-DL of its own (HTML[1]) to describe the linked hypertext documents that compose it. Thus, by definition, HTML is probably one the most used standardized UI-DL in the world. Its syntax and stylistics are based on the Standard Generalized Mark-up Language (SGML) and its semantics are the different elements that comprise hypertext documents on the web. In its initial versions, the layout, behaviour and look and feel aspects were tightly wound, an aspect that was later amended through the introduction of a separate language, CSS[2], for stylistic and look and feel aspects and ECMAscript/Javascript[3] as a *de facto* underlying language to introduce more complex behaviours in an otherwise static UI for the upcoming HTML5.

Concern over the years about making HTML more extensible and interoperable with other mark-up technologies and tools, as well as the need for supporting "alternate ways of accessing the Internet" [4] came to a first result in the proposal of XHTML, defined by W3C as a "reformulation of HTML4 document types as applications of XML 1.0". Syntactic and stylistic changes were introduced such as the enforcement of the well-formedness of the mark-up code and the use of all-lowercase characters in mark-up, respectively. The semantics of the HTML UI description were left largely intact for cross-compatibility.

Due to differing browser implementations though, cross-platform and browser capabilities for web applications proved to be an elusive target. In an attempt to bridge this gap and to bring developers true cross-browser environments upon which to seamlessly build applications, a large number of frameworks thrived after 2005 providing a uniform programming API to developers to target most major browsers, in addition to easing access to features that previously required considerable developer effort such as the manipulation of HTML documents commonly grouped under the DHTML (for Dynamic) umbrella term. These very often provided functionality through AJAX[5] techniques that combined Javascript with asynchronous server calls to improve the responsiveness of websites and a better user experience. Among the

[1] http://www.w3.org/MarkUp/draft-ietf-iiir-html-01.txt : HTML 1.2 standard
[2] http://www.w3.org/Style/CSS/ : CSS home page with links to its specifications
[3] http://www.ecmascript.org/ : ECMAScript, the standardized version of Javascript
[4] http://www.w3.org/TR/xhtml1/ : 'What is XHTML?' World Wide Web Consortium
[5] http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications : AJAX: A new approach to web applications

most successful of these are Dojo[6], the Google Web Toolkit[7] and above all jQuery[8], which is currently on the rise and used in nearly a majority of the top websites[9].

With the appearance of the web as an application platform in the late 90s, a number of non-HTML based UI toolkits were put forward as alternative means to build websites with advanced functionality not present in standard HTML. Adobe Flash[10] became the most ubiquitous[11] of these and soon other alternatives followed such as JavaFX[12] and Microsoft Silverlight[13]. Recent development in the standards field and developer interest in cross-platform functionality seem to indicate that HTML5 technologies have the advantage for the future and are slowly progressing towards becoming the standard Rich Internet Application (RIA) environment for the coming years. However, the study of these older RIA technologies might provide some insight useful for SERENOA, as they have been pioneering in the wide introduction of topics such as UI-DLs (e.g. Silverlight's XAML[14]), full featured and powerful authoring tools in widespread usage (e.g., Adobe Flash Professional[15]) and some cross platform compatibility, mainly across the browser-desktop boundary (e.g., Adobe Air[16] and Silverlight out-of-browser applications[17]).

However, this functionality is built on top of standards rather than being an integral part of them.

Widget Toolkits

These are often part of the UI description process and in many cases are intermingled extensively with the UI-DL itself, while in most cases (especially in XML or other mark-up based approaches) they are just a target for the last-step, final UI rendering. A widget toolkit is in essence a set of concrete UI building blocks (e.g., scroll bars, icons, sliders) along with a defined syntax for associating them together using super structures and containers (e.g., panels, layout primitives) and lastly a programming model associated such as event-driven programming. The look and feel part of the final UI description can be pre-defined or changed by means of a decoupling of interactors and rendering details. The process of applying a definite look and feel to an instantiation of a widget toolkit application is called 'theming' or 'skinning'.

A majority of the most popular toolkits are proprietary and platform dependent, although a small number of them have been released using open standards. Well known proprietary widget toolkits include the following:

- Microsoft Foundation Classes[18], often known for the acronym MFC, a low level widget toolkit typically used to program native Windows applications.
- Abstract Window Toolkit (AWT)[19] and Swing[20], the two initial approaches at providing the Java Platform with a window management toolkit
- Motif[21], one of the first standardized cross-platform toolkits, originally used for UNIX systems and then ported to other POSIX operating systems.
- Tk[22], a simple, straightforward widget toolkit originally used as Tcl-only toolkit, then progressed to cross-platform and cross-language.

---

[6] http://dojotoolkit.org/ : The Dojo Toolkit
[7] http://code.google.com/intl/en-US/webtoolkit/ : Google Web Toolkit
[8] http://jquery.com/ : jQuery toolkit
[9] http://trends.builtwith.com/javascript/JQuery : jQuery usage statistics
[10] http://www.adobe.com/products/flashplayer/features/index.html : Adobe Flash overview
[11] http://www.adobe.com/products/player_census/flashplayer/ : Flash Player penetration in PC market
[12] http://javafx.com/ : JavaFX
[13] http://www.microsoft.com/silverlight/ : Microsoft Silverlight
[14] http://msdn.microsoft.com/en-us/library/ms752059.aspx : Microsoft XAML UI-DL overview
[15] http://www.adobe.com/products/flash.html : Adobe Flash CS5
[16] http://www.adobe.com/products/air/ : Adobe Air
[17] http://www.silverlight.net/learn/overview/out-of-browser-applications/out-of-browser-applications-(silverlight-quickstart) : Silverlight out of browser applications overview
[18] http://msdn.microsoft.com/en-us/library/d06h2x6e(VS.90).aspx : Microsoft Foundation Classes development reference
[19] http://java.sun.com/products/jdk/awt/ : Java AWT
[20] http://java.sun.com/products/jfc/tsc/articles/architecture/ : Java Swing
[21] http://www.openmotif.org/ : Openmotif, lead development organization of the Motif toolkit
[22] http://www.tcl.tk/ : Tk graphical UI toolkit

- GTK+[23] and Qt[24], the current leading free software widget toolkits with extensive cross-platform capabilities and industry adoption. Both are used in industry- and community-backed applications and have extensive platform and programming language support, ranging from mobile to workstations. There are also authoring solutions available for both, such as Glade[25] for GTK+ and Qt Creator[26] for Qt.

Cross-platform general purpose UI-DLs

Over the years, a great number of general purpose UI-DLs has been put forward. They differ in a great number of features: some have been put forward by academic researchers while others are the product of industry consortia, or even single commercial players. A comprehensive overview of the field is beyond the scope of this document; thus we will focus on a few particular ones that are representative of well-known families, or maybe offer features or teachings that may be considered of interest for SERENOA.

- ***UIML***[27]: This is an XML-based meta language intended to be used to define UI elements in declarative terms and in an abstract manner. The language development started in 1997 by a consortium of academic and industrial partners and over time, its control transitioned to the Organization for the Advancement of Structured Information Standards (OASIS), an international body specializing in the standardization of XML-based technologies, which now controls the UIML specification and is in charge of developing standard versions of the language[28].

  The language defines a set of UI building blocks, or interactors, means to group them together and also ways to control the behaviour of the components and bindings to user events. The concrete details of the rendering or styling are separated from the actual definition of the elements that conform to the UI. The language intends to be very wide-ranging, in essence being able to cover the majority of the device families in use today.

  Although UIML allows a multi-platform description of UIs, there is limited commonality between the platform-specific descriptions when platform specific vocabularies are used. This means that the UI designer will have to create separate UIs for each platform using its own vocabulary. Recall that the vocabulary is defined to be a set of UI elements with associated properties and behaviour. Limited commonality is not a shortcoming of UIML itself, but a result of the inherent differences between platforms with varying form factors.

- ***XUL***[29]: The *X*ML *U*ser interface *L*anguage is a general purpose mark-up DL developed by the Mozilla Foundation and primarily used in their open source Firefox Browser. Rather than defined by a formal specification, it is driven by the reference implementation made by the open-sourced Gecko rendering engine. The target of the language is desktop applications and it has a strong emphasis in providing developers with metaphors and syntactic elements very close to those of web languages such as dynamic HTML. Cross platform compatibility is good and several successful open source applications (such as the Songbird[30] media player) have used XUL to define their UIs, although the majority of development efforts based upon it relate to the Firefox browser and its extensions.

- ***XAML***[31]: Put forward by Microsoft, it is a general purpose UI-DL which targets a much more concrete implementation than efforts such as UIML or UsiXML, roughly corresponding to the Concrete UI level of the Cameleon hierarchy. It is based on web standards such as XML, XPath and DOM and integrates descriptions of interactors, composition of interactors, stylings, animations and

[23] http://www.gtk.org/ ; GTK+ Project
[24] http://qt.nokia.com/ : Qt cross platform widget toolkit
[25] http://glade.gnome.org/ : Glade GTK+ Authoring tool
[26] http://qt.nokia.com/products/developer-tools : Qt Creator Authoring Tool
[27] http://www.uiml.org : UIML website
[28] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=uiml : OASIS User Interface Markup Language TC
[29] https://developer.mozilla.org/En/XUL : Mozilla XUL
[30] http://getsongbird.com/ : Songbird media player
[31] http://msdn.microsoft.com/en-us/library/ms752059.aspx : XAML Overview

transformation, and complex behaviour with an event-driven model. This all can be done at the same syntactic level, with XAML files able to implement all of the features of an application, or conversely XAML can be used for the description of the UI and the other elements defined programmatically in separate application segments written in other languages (mainly Javascript and .NET languages such as C#, Visual Basic , ironPython and Mono). The official Microsoft specification currently targets as a rendering subsystem either way Microsoft Silverlight for web-based applications or the Windows Presentation Foundation subsystem in the Windows OS. Unofficial implementations such as Moonlight[32] target other platforms.

Microsoft also provides an authoring tool for XAML applications, called Microsoft Blend. It allows developers and designers to build the UI of applications, but it is usually combined with the Microsoft proprietary development platform, Visual Studio .NET, for the business logic as the support for such logic inside Blend is very limited beyond from simple event-driven behaviours.

The language specification is open and has been published by Microsoft. However, it has drawn criticism from the community for the alleged side effect of boosting Windows as a development platform for web applications, as support for other OSs and architectures is not complete and less optimal

After this preliminary overview of the existing solutions for UI description and cross-platform deployment, let us now present briefly the Cameleon hierarchy of UI description levels, which is key for the understanding of the UI-DL languages present in SERENOA's consortium portfolio, whose detailed descriptions will follow in this section 2.

2.1 Cameleon framework UI levels

The FP5 Cameleon project[33], whose aim was to support the definition and development of context-sensitive software systems, had as one of its key results the definition of a conceptual framework to ease the design of such systems for multi-platform environments. This key result, further explained in [Calvary2002], was the cornerstone of some research efforts that followed, such as the UI-DL languages MARIA and UsiXML that will be further discussed in this section. Here we will provide a short summarized explanation of one of the most significant parts of the framework, the four-level hierarchy of UI implementations, according to their level of concreteness.

The framework structures the development life cycle into four levels of abstraction, from the task specification to the running interface:

- **Task & Concepts (T&C):** corresponds to the computational-independent model in MDE, it describes the various tasks to be carried out and the domain-oriented concepts as they are required by these tasks to be performed. These objects are instances of classes representing the concepts manipulated. It considers the logical activities that need to be performed in order to reach the users' goals. Often, it is represented hierarchically along with indications of temporal relations among them and their associated attributes.
- **Abstract UI (AUI):** corresponds to the platform-independent model (PIM) in MDA; it is an expression of the UI in terms of interaction spaces (or presentation units), independently of which interactors are available and independently of the modality of interaction (graphical, vocal, haptic). An interaction space is a grouping unit that supports the execution of a set of logically connected tasks. It groups subtasks according to various criteria (e.g., task model, structural patterns, cognitive load analysis, semantic relationships identification), a navigation scheme between the interaction spaces and selects Abstract Interaction Objects (AIOs) for each concept so that they are independent of any context of use. An AUI abstracts a CUI into a UI definition that is independent of any modality of interaction (e.g., graphical interaction, vocal interaction). An AUI can also be considered as a canonical expression of the rendering of the domain concepts and tasks in a way that is independent from any modality of interaction.

[32] http://www.mono-project.com/Moonlight : Moonlight, an open source implementation of Silverlight for Linux.
[33] http://giove.isti.cnr.it/projects/cameleon/goals.html : FP5 Cameleon Project website.

- **Concrete UI (CUI):** corresponds to the platform-specific model (PSM) in MDA, it expresses the UI in terms of concrete interactors, that are dependent of the type of platform and media available, and it has a number of attributes to define more concretely how it should be perceived by the user; it concretizes an abstract UI for a given context of use into Concrete Interaction Objects (CIOs) so as to define widgets layout and interface navigation. This process abstracts a FUI (Final UI) into a UI definition that is independent of any computing platform.
- **Final UI (FUI):** corresponds to the code level in MDA, it consists of source code, in any programming language or markup language, it can be interpreted or compiled, it is the operational UI i.e. any UI running on a particular computing platform either by interpretation (e.g., through a Web browser) or by execution (e.g., after compilation of code in an interactive development environment). Given that a piece of code may not always be rendered in the same manner depending on the software environment (virtual machine, browser), we consider two sublevels of the FUI: i) the source code and ii) the running interface.

Thanks to the four abstraction levels, it is possible to establish mappings between instances and objects found at the different levels and to develop transformations that find abstractions or reifications or combination [Limbourg et al., 2004a; Vanderdonckt et al., 2004]. Moreover, there can be also a relationship of translation between models at the same level of abstraction, but conceived for different contexts of use (i.e. from source to target context of use).

2.2 UsiXML

The USer Interface EXtensible Markup Language (UsiXML) is an XML-compliant mark-up language to describe user interfaces for multiple contexts and different modalities. UsiXML allows also non-developers to use the language to describe user interfaces, mainly because the elements of the UI can be described at a high level, regardless of the platform of use.

The UsiXML language is currently being submitted to a standardization plan to the W3C.

UsiXML structures the development of UI in the four abstraction levels mentioned in section 2.1. The transformation between models is done using ATL and currently, editors are being developed and published to ease the implementation and the transformations between these models.

UsiXML provides flexibility to the user, who can choose to begin the sketching of the interface at any level. In order to provide this flexibility, the meta-models for each abstraction layer must be independent. However, at any level all the information necessary to build a final UI, must be available.

Each model can contain more or less details according to its abstraction level, for example the behaviour defined at the abstract UI level is also defined at the concrete level, but it may consider details that are specific for graphical UI for instance.

2.2.1 Metamodel

The metamodel of UsiXML for Abstract User Interfaces is composed by *AbstractUIModel* and by a set of *AbstractInteractionUnit* that defines every abstract object. These interaction units may own an *AbstractRelationship* describing the way the different interaction units of the model are related. Furthermore, each interaction unit may have one or more *AbstractListeners*. These listeners allow defining the dynamics of the model and the way the interaction units will react to the different events.

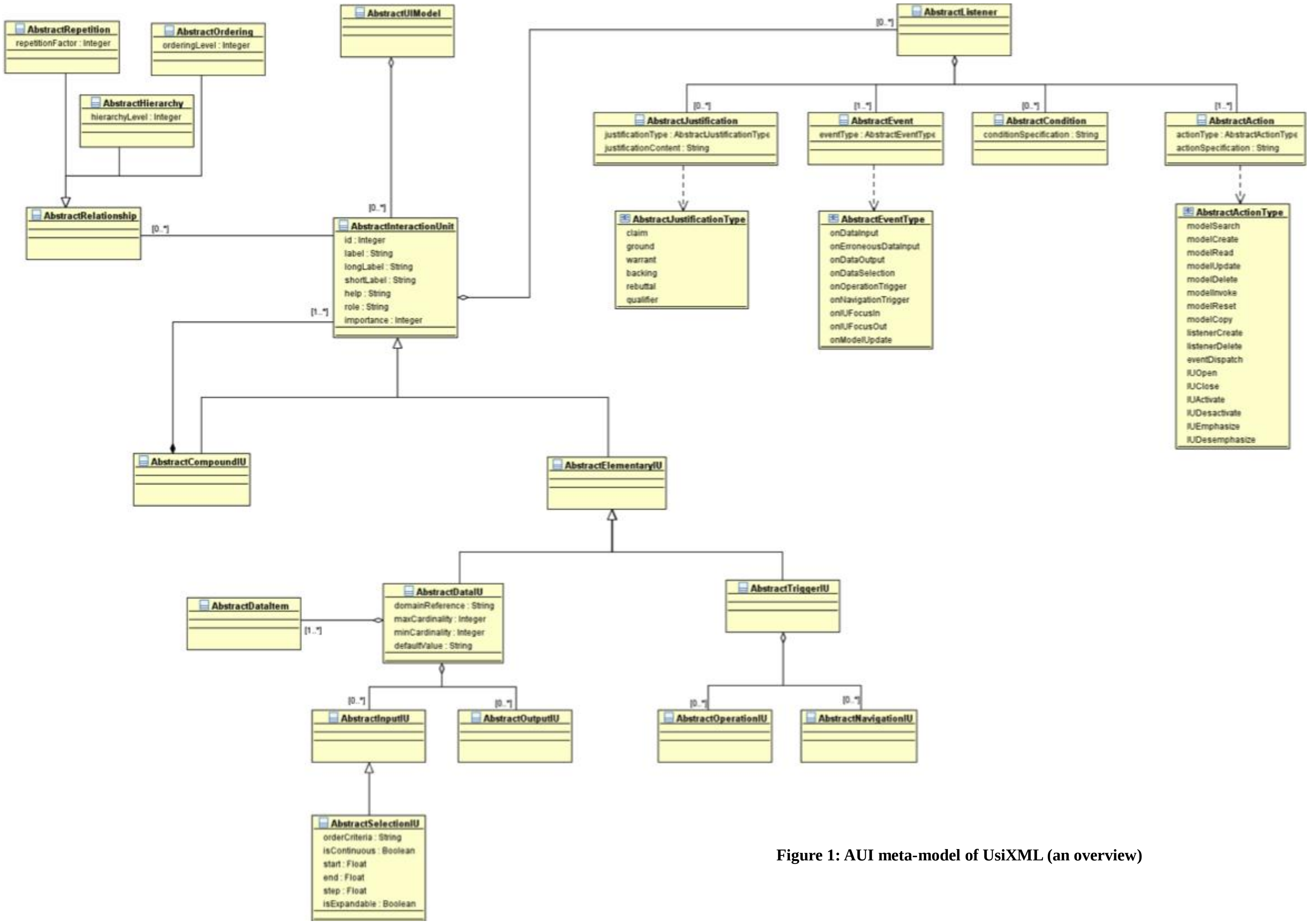

Figure 1: AUI meta-model of UsiXML (an overview)

An Abstract *model* describes the units, relationships and listeners of a UI, independently of concrete interaction units available in target contexts of use. An *interaction unit* is the main entity of the model, composed by every abstract object. The *abstract relationships* define how the units are related. And *abstract listeners* define the behaviour of the units. All these concepts are further detailed with specific meta-models.

2.2.2 Transformations

Example of transformations in UsiXML:

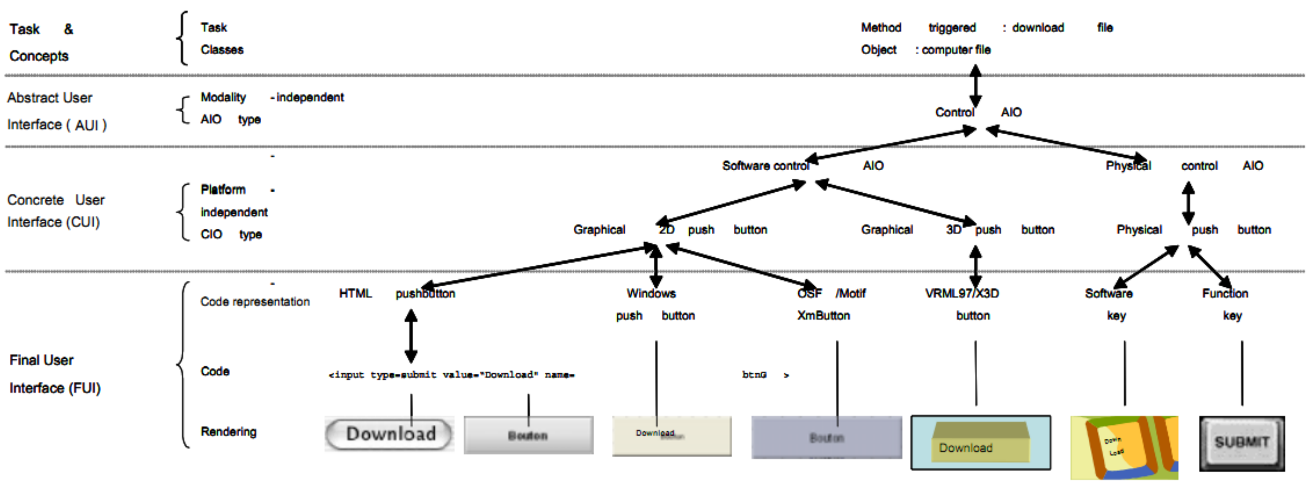

Figure 2. UsiXML Transformations [Source: Limbourg2004b].

Figure 2 illustrates possible transformations between the abstraction levels; the application domain considers a download button. The same task is specified at different levels of details according to the abstraction level considered. The context of use is essential to decide which final UI will be rendered.

Meta-models were defined to implement UsiXML. These meta-models follow several principles to assure a high-quality level such as: completeness, consistency, separation of concerns and extensibility.

The meta-models elaborated for the UI include: task model, context model, mapping model, domain model, AUI model, CUI model, transformation model and workflow model. The main entities of the abstract UI model are *model*, *interaction unit*, *relationship* and *listener*; and the main entities of the concrete UI model are *model*, *interaction unit*, *style* and *listener*.

The Concrete User Interface (CUI) corresponds to the platform-specific-model in MDA; it expresses the UI concerning concrete interaction units, which are dependent of the platform type (and also available media). The main entities of the CUI model for UsiXML are *model*, *interaction unit*, *concrete style* and *concrete listener*.

2.2.3 UsiXML Applications

UsiXML has been applied to many different domains, among which we highlight adaptation of user interfaces [Desruelle2011] [Honold2011] and accessibility [Serra2011] [Bottoni2011] [Kaklanis2011].

There is a set of tools that support UsiXML, for instance: FlashiXML is a translator to Flash; SketchiXML is an editor to draw and sketch UI; and idealXML is a tool for task analysis.

2.3 MARIA

MARIA (Model-based language for Interactive Applications), is a universal, declarative, multiple abstraction-level, XML-based language for modelling interactive applications in ubiquitous environments. For designers of multi-device user interfaces, one advantage of using a multi-layer description for specifying UIs is that they do not have to learn all the details of the many possible implementation languages supported by the various devices, but they can reason in abstract terms without being tied to a particular UI modality or, even worse, implementation language. In this way, they can better focus on the semantics of the interaction, namely what the intended goal of the interaction is, regardless of the details and specificities of the particular environment considered.

MARIA allows describing not only the presentation aspects, but also the interactive behaviour. For this purpose it has various features:

> ***Data Model***. The interface definition contains description of the data types that are manipulated by the user interface. The interactors (which are the elementary abstract UI objects) can be bound with elements of the data model, which means that, at runtime, modifying the state of an interactor will change also the value of the bound data element and vice-versa. This mechanism allows the modelling of correlation between UI elements, conditional layout, conditional connections between presentations, input values format. The data model is defined using the standard XML Schema Definition constructs.
>
> ***Generic Back End***. The interface definition contains a set of *ExternalFunctions* declarations, which represents functionalities exploited by the UI but implemented by a generic application back-end support (e.g. web services, code libraries, databases etc.). One declaration contains the signature of the external function that specifies its name and its input/output parameters.
>
> ***Event Model***. Each interactor definition has a number of associated events that allow the specification of UI reaction triggered by the user interaction. Two different classes of events have been identified: the Property Change Events that specify the value change of a property in the UI or in the data model (with an optional precondition), and the Activation Events that can be raised by activators and are intended to specify the execution of some application functionalities (e.g. invoking an external function).
>
> ***Dialog Model***. The dialog model contains constructs for specifying the dynamic behaviour of a presentation, specifying what events can be triggered at a given time. The dialog expressions are connected using CTT operators in order to define their temporal relationships.
>
> ***Continuous update of fields***. It is possible to specify that a given field should be periodically updated invoking an external function.
>
> ***Dynamic Set of User Interface Elements***. The language contains constructs for specifying partial presentation updates (dynamically changing the content of entire groupings) and the possibility to specify a conditional navigation between presentations.

This set of new features allows having already, at the abstract level, a model of the user interface that is not tied to layout details, but it is complete enough for reasoning on how UI supports both the user interaction and the application back end.

The Abstract User Interface (AUI) level describes a UI by just referring to the semantics of the interaction, without considering a particular device capability, interaction modality or implementation technology.

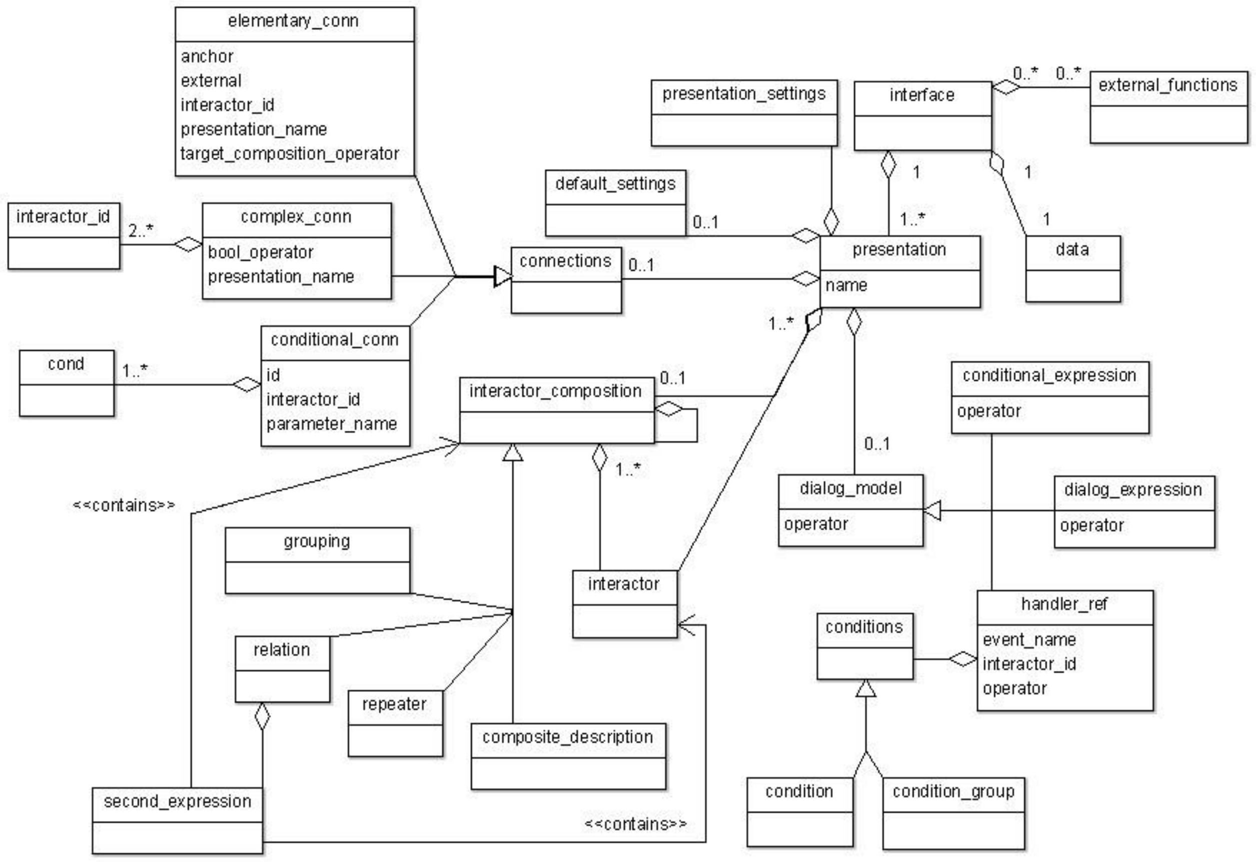

Figure 3: MARIA AUI metamodel

Figure 3 shows the main elements of the abstract user interface metamodel (some details have been omitted for clarity). As can be seen, an interface is composed of one data model (abstract data types managed by the interface model and whose definition generally exploits XSD schema), a list (namely: one or more) presentations that groups model elements and 0..n external functions.

A presentation is basically a set of model elements which are presented to the user at the same time. Such model elements are of two types: *interactor* or *interactor_composition*. The former represents every type of user interaction object, the latter groups together elements that have a logical relationship.

Each presentation has an attribute name, and is composed of a number of possible connections (namely, connections with other presentation), elementary interactors, and/or interactor compositions. Moreover, the presentation is also associated with a dialog model which provides information about the possible sequence of interactions that can occur within the presentation itself.

A dialog model can be represented either by a reference to a handler (*handler_ref*), a conditional expression or a dialog expression. A *handler_ref* has to specify a number of conditions, and in addition has a number of attributes: *event_name* (which has string type and is optional), *interactor_id* (also having string type and being optional), and an additional attribute, operator, which represents a CTT operator. Indeed, the dynamic behavior of the events, and the associated handlers, is specified using the CTT temporal operators (for example, concurrency or mutually exclusive choices, or sequentiality, etc.). When an event occurs, it produces a set of effects (such as performing operations, calling services, etc.) and can change the set of currently enabled events (for example, an event occurring on an interactor can affect the behavior of another interactor, by disabling the availability of an event associated to another interactor). The dialog model can also be used to describe parallel interaction between user and interface.

A connection indicates what the next active presentation will be when a given navigation is triggered. It can be either an elementary connection (which basically occurs with a static target presentation), a complex connection (when Boolean operators compose several connections and then the connection has a complex target), or a conditional connection (when specific conditions are associated with it and therefore the connection has a conditional target).

An elementary connection has a number of attributes: anchor and external, which are both optional Boolean values, *interactor_id*, which is a required identifier of the interactor which connects the current presentation with another (a navigator or an activator); *presentation_name* (the required, target presentation identifier), *target_composition_operator*, which, if present, specifies where to load the *target_presentation_name* inside the *target_presentation*.

Regarding interactor compositions, they can be of four types: *grouping*, *relation*, *repeater*, *composite_description*.

A grouping is a type of interactor composition used when a logic composition of interactors is needed. Therefore, grouping basically represents a generic group of interactor elements.

A relation is an interactor composition which expresses a relation between an interactor (or an interactor composition) and other N interactors (or interactor compositions).

A *composite_description* represents a group aimed to present contents through a mixture of *only_output* elements (namely: description/object/feedback/alarm) with a navigator elements, while a repeater is used to repeat the content according to data retrieved from a generic data source.

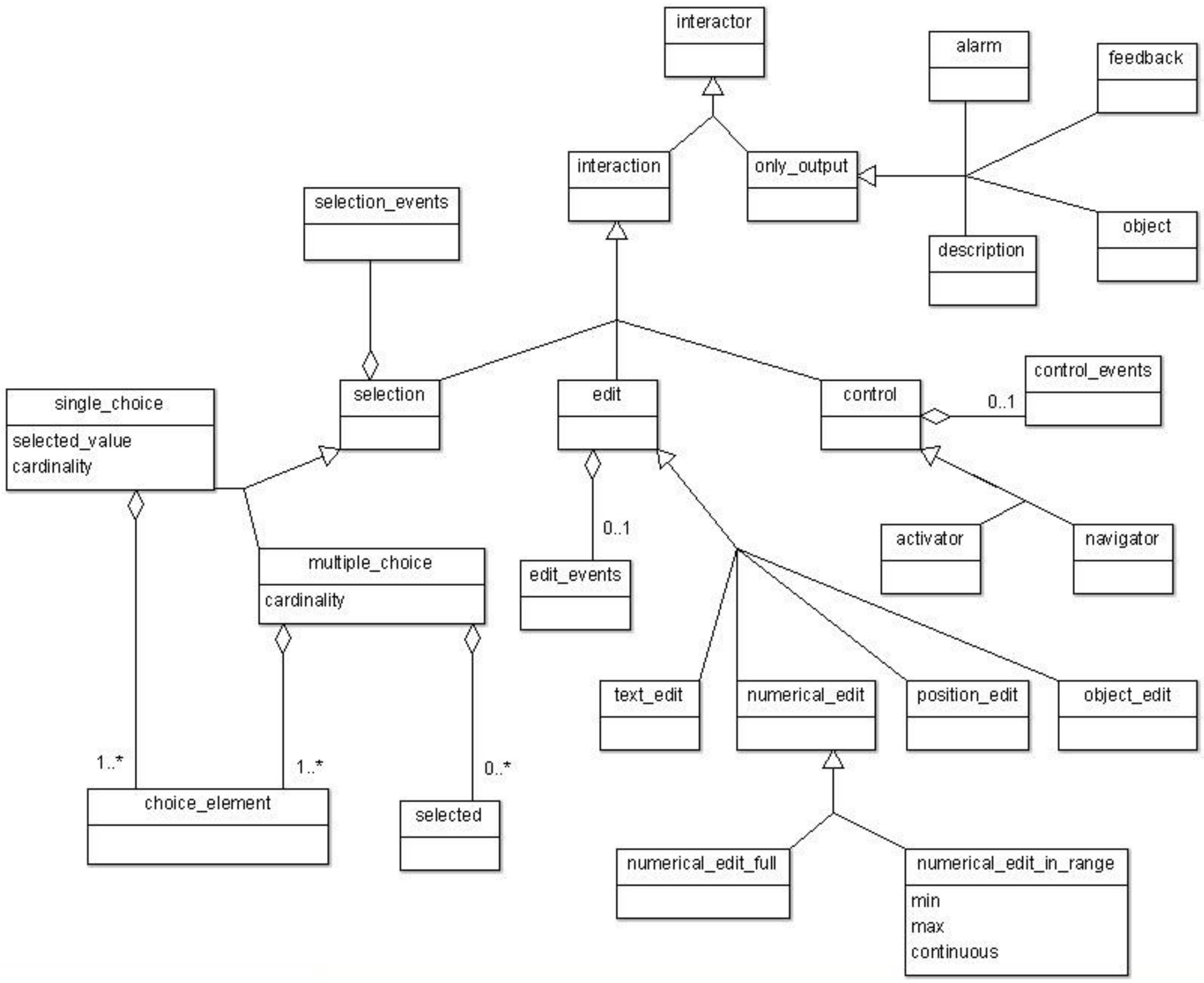

Figure 4. The specification of the abstract interactor element in MARIA

Figure 3 shows the main elements of the AUI meta-model through its class diagram, while Figure 4 shows the specification of the hierarchy of the interactor element.

An interactor (see Figure 4) can be either an interaction object or an only output object. The first one can be one of the following types: selection, edit, control, interactive description, depending on the type of activity the user is supposed to carry out through such objects.

An *only_output interactor* can be object, description, feedback, alarm, text, depending on the supposed information that the application provides to the user through this interactor. It represents information that is submitted to the user, not affected by user actions. It can be a text, a *Description* that represents different types of media, an *Alarm*, a *Feedback* or a generic *Object*.

According to the intended semantics, an interactive element belongs to one the following subtypes: *selection*, *edit* or *control*.

- The *selection* object enables the user to select one or more values among the elements of a predefined list. It contains the selected value and the information about the list cardinality. According to the number of values that can be selected, the interactor can be a *single_choice* or a *multiple_choice*. Each of them has an attribute specifying the cardinality of the possible choices (in addition, the *single_choice* element has a further attribute, *selected_value*, for identifying whether an option element has been selected or not). The *multiple_choice* element has associated a list of elements of type selected for specifying the option items currently selected. It is worth pointing out that further refinement of each of these objects can be done only by specifying some platform-dependent characteristics, therefore it is specified at the concrete level whether the refinement element on a desktop platform will be eg a radiobutton, a list, etc.
- The *edit* object enables the user to manually edit the object represented by the interactor, which can be text (*text_edit*), a number (*numerical_edit*), a position (*position_edit*) or a generic object (*object_edit*). The *numerical_edit* object can be further refined into two possible elements: *numerical_edit_full* and *numerical_edit_in_range*. The latter element is characterised by the fact that a specific interval is considered for the values (it can be refined into e.g. a *spinbox* element). This interactor element has three peculiar attributes: *min* (the minimum value in the considered range), *max* (the maximum value), and *continuous* (which specifies whether the range is continuous or discrete).
- The *control* object enables the user to switch between presentations (*navigator*) or to activate UI functionalities (*activator*).

It is worth pointing out that all the interaction objects have associated events: the events differ depending on the type of object they are associated with. Indeed, we have *edit_events*, *selection_events* and *control_events*.

2.4 IDEAL2

IDEAL2 [Cantera2010] (Interface DEscription Authoring Language, version 2) is an XML-based language for the declarative description of device-independent user interfaces. This language is modular, extensible and standard-compliant, based on different W3C standards such as XForms 1.1 [XFORMS], DISelect 1 [DISELECT] or XML Events [XMLEvents].

IDEAL2 is modality dependant, oriented to graphical interfaces, and has been specifically designed to create mobile web interfaces. For this reason, it cannot be considered at the abstract level (AUI), but at the concrete level (CUI). It provides abstractions over the final language (FUI) that makes it suitable for defining UIs without handling the specific features of the target devices. IDEAL2 defines abstract components, which can be rendered in different ways by depending on the terminal capabilities and the Web browser features identified at runtime. For instance, a *select1* user interface component is intended to allow the user to select one item among several options. It might be rendered as a drop down list, a set of radio buttons or a navigation list with hyperlinks.

Using IDEAL2, developers concentrate their efforts on describing the structure of the user interfaces instead of programming how they should be rendered in different delivery contexts. In this way, they do not need to be worried about the device capabilities, and the different markup languages, scripting code or CSS properties it supports.

Aspects related to the Look & Feel and to the flow definition are out of the scope of IDEAL2. The Look & Feel and the layout could be specified by means of CSS2 [CSS2] and some extensions, while the flow description could be defined by using other declarative languages such as SCXML [SCXML].

Tool support

MyMobileWeb [MyMobileWeb] is an open source, standards-based software framework that simplifies the rapid development of mobile web applications and portals. *MyMobileWeb* uses IDEAL2 as the authoring language in which developers declaratively define UIs in a device independent manner. *MyMobileWeb* software platform is in charge of the automatic adaptation of such UI descriptions to the target Delivery Context (browser, device, network, location, etc.) offering a harmonized user experience.

Example

Figure 5 illustrates a UI description defined by means of IDEAL2 language. It is part of an example application, which is supposed to provide information about the Spanish soccer league. In this case, the UI represents a menu for selecting a specific soccer club.

The *resources* element includes references to external resources such as application icons or references to the CSS style sheets. Note that the *expr* attribute allows the conditional inclusion of elements based on the evaluation of delivery context properties.

The *header* element is intended to act as a container for the head elements. In this case, it contains an *include* element which is used to specify the inclusion of contents which have been defined in other presentation in order to avoid code duplication. In a similar way, the *footer* tag references contents which has been defined in other presentation by using the *include* element.

The main *section* of the UI represents a *menu* that renders information coming from the application context. The developer is in charge of storing all the information related to the soccer clubs inside the application context under the corresponding key. The most important attribute for rendering context information is *repeat-nodeset*. In this example, this attribute takes the value *clubList* which is a context variable containing the items (clubs) to be displayed by the menu. In this menu, the developer aims to show the club name, an associated image and a link to the details page for each team.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE ideal2>

<ideal id="index" title="My Club">

 <resources>
  <link id="icon" rel="shortcut icon" expr="!mymw:belongsTo('iPhone')" type="image/x-icon" href="${myFavIcon}" />
  <link id="iconIPhone" rel="apple-touch-icon" expr="mymw:belongsTo('iPhone')" href="${myFavIcon}" />
  <link rel="stylesheet" id="soccerStyle" href="soccer.css" />
 </resources>

 <ui>
  <body>
   <header id="header">

     <include content="Common/generic/common/header" />

   </header>

   <section id="main">

    <div id="p1" class="common title.common" title="My Club">
     <menu id="myMenu" ref="club" class="clubs center">
        <a id="header" repeat-nodeset="clubList" src="${clubList.current.image}"
           href="${clubList.current.href}">
          ${clubList.current.name}
        </a>
     </menu>
    </div>

   </section>

   <footer id="footer">

    <include content="Common/generic/common/footer" />
    <separator class="line" />
    <include content="Common/generic/common/powered" />

   </footer>
  </body>
 </ui>

</ideal>
```

Figure 5. IDEAL2 UI description

The above UI description (completed with the addition of CSS style sheets) might be rendered by an adaptation framework in different ways, depending on the delivery context. Figure 6 shows how *MyMobileWeb* renders the UI defined in Figure 5 on different devices.

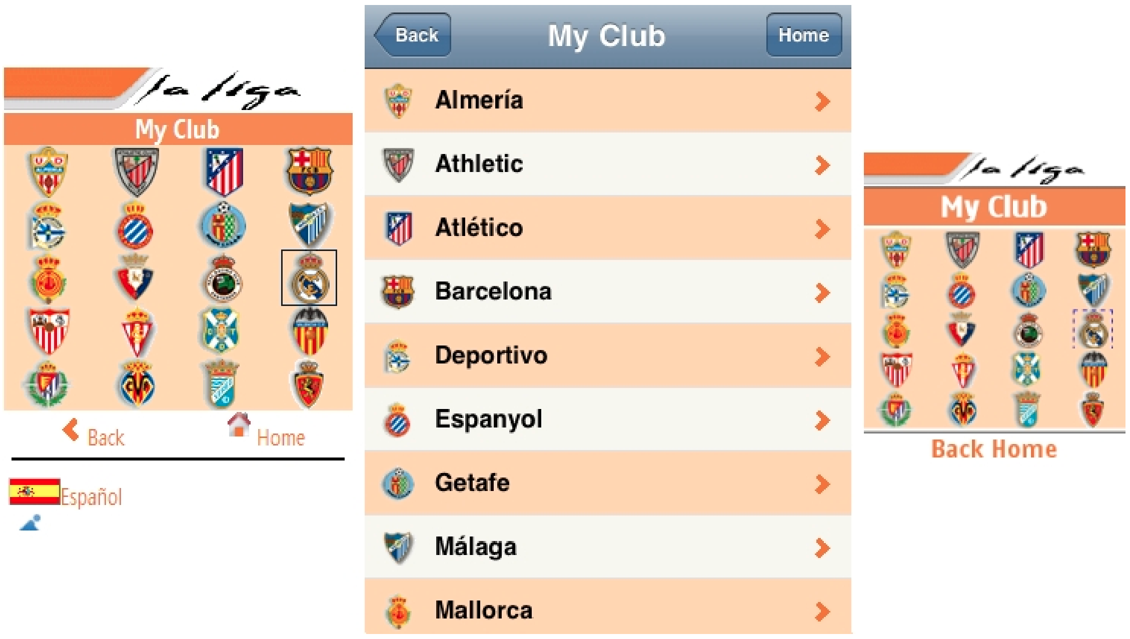

Figure 6: An IDEAL2 UI description rendered by MyMobileWeb on different devices

3 ASFE-DL Abstract UI Level

The Abstract User Interface (corresponding to the Platform-Independent-Model – PIM in MDE) expresses the UI regarding its presentation units, independently of the interactors available and of the modality of interaction (graphical, vocal, haptic). A presentation unit groups a set of logically connected interactors.

3.1 Metamodel

In this section, we describe the first version of the Advanced Service Front-End Description Language (ASFE-DL). This language will enable the development and authoring of SFEs in accordance with the Reference Model described in SERENOA's deliverable D3.1.1. The interfaces modelled with such language will be adapted to the context exploiting the rules defined through the Advanced Adaptation Logic Description Language (AAL-DL), which will be defined in deliverable D3.3.1.

In this first version, we focused on the creation of an AUI meta-model, starting from the CNR and UCL experience respectively in MARIA [Paternò2009] and UsiXML [UsiXML]. The idea is to create a unified and more complete language, combining the strengths of the two languages, unifying concepts and adding new features that will allow this language to meet the SERENOA requirements and to go beyond the state of the art in this field.

In order to achieve this goal, we analysed the metamodels of both languages. Afterwards, we have identified an initial set of concepts than can represent a first basis for ASFE-DL (the Serenoa AUI language) to be further refined in the next future within the project. By analyzing the AUI metamodels of MARIA and UsiXML, a number of situations were handled in order to come up with a preliminary shared proposal for the ASFE-DL.

In some cases, there were concepts directly corresponding to each other between metamodels (apart from slightly different names used in the two originated languages). For instance, the original UsiXML elements of types *AbstractOrdering*, *AbstractHierarchy* and *AbstractRepetition* (corresponding to for MARIA's *Ordering*, *Hierarchy*, and *Repeater* entities) have been basically kept in the shared SERENOA proposal respectively as *Ordering*, *Hierarchy*, and *Repetition*. The same occurred for the UsiXML *AbstractRelationship* concept, which was easily identified as modelling the same concept as *Interactor_composition* existing in MARIA language (eventually, we decided to keep the UsiXML naming and to then have the *AbstractRelationship* element in the ASFE-DL language). Furthermore, both languages supports a clear separation of concerns between the UI specification and the description of data handled in it, by including a specific element for handling data: this was called *DomainModel* in UsiXML and *DataModel* in Maria (the latter name was eventually selected for the ASFE-DL) .However, in other situations, the correspondence between both languages/models was not so direct. For instance, this is the case when the same concept is modelled in a slightly different way, as for the event objects. Indeed, in UsiXML the different types of events are not connected to specific types of interactors, while in MARIA this is the case. Therefore, in the ASFE-DL, we preferred to keep the MARIA approach as it enables a more controlled handling of the specific events associated to each particular type of abstract interactor.

Finally, there were concepts appearing in just one metamodel, which we judged useful to bring in the ASFE-DL. For instance, it was the case of the *Connection* elements (existing in MARIA language), which allow specifying how it is possible to move from one abstract interaction unit to another one.

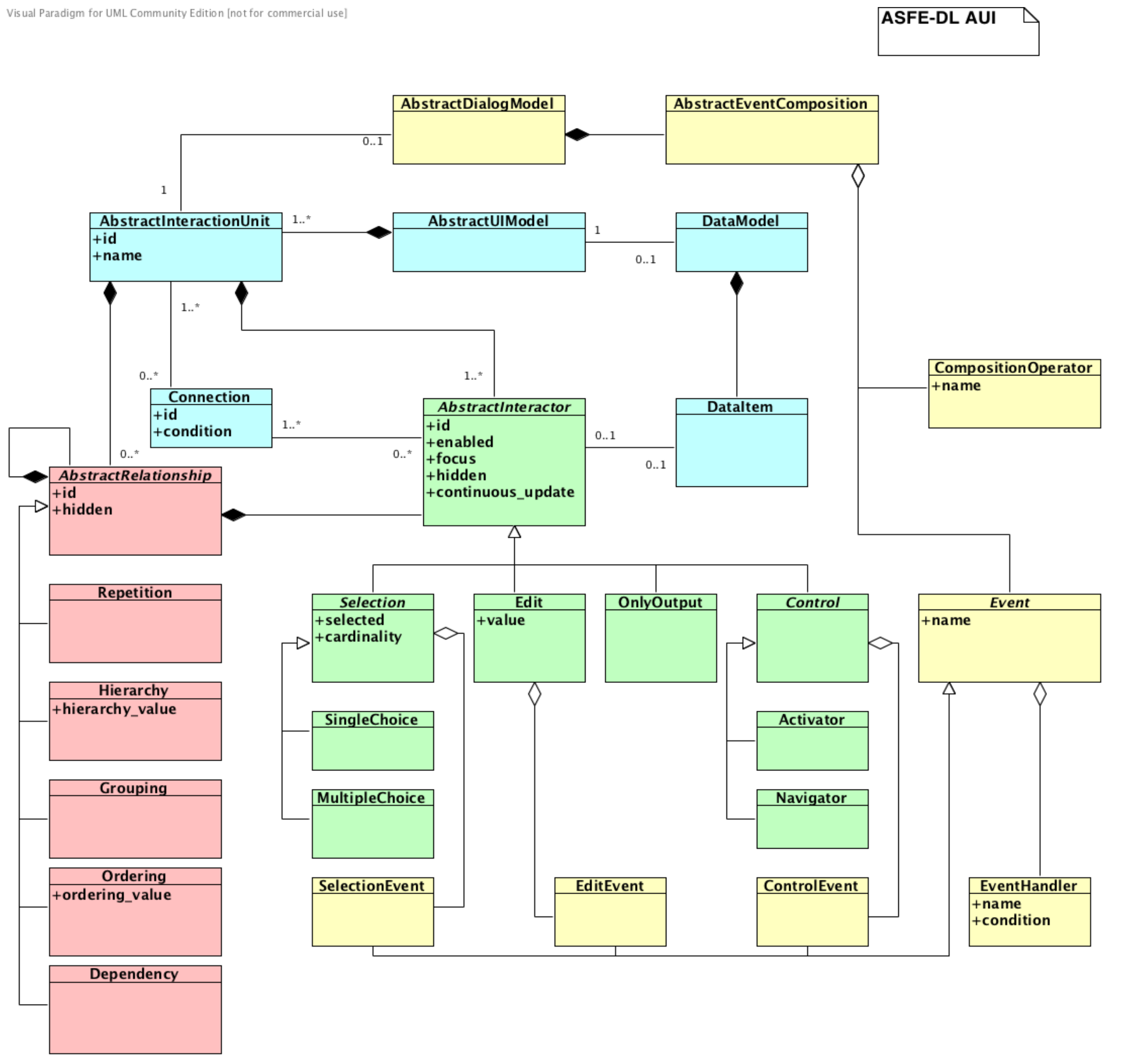

Figure 7: ASFE-DL AUI metamodel (UML View)

Figure 7 shows the UML class diagram for this preliminary version of the ASFE-DL at the abstract level. We used different colours in order to highlight different parts of the metamodel: sky-blue for the main structure of the interface, green for the interactor hierarchy, red for the classes that model the relationships between interactors and yellow for the classes that model the UI behaviour.

The class that represents an abstract user interface model is called *AbstractUIModel*. It contains the entire specification of both the UI structure and behaviour. The interface contains an optional *DataModel*, which defines the set of objects and variables. The data model contains all the data types that are manipulated by the UI in order to allow interaction with the user. The composition relation between the *DataItem* and the *DataModel* class represents such collection of data types. A *DataItem* indeed represents a single data type element.

An *AbstractUIModel* consists of a composition of *AbstractInteractionUnits*. Each instance of this class represents a part of an application user interface that should be presented to the user at once. Each *AbstractInteractionUnit* has two attributes: a unique identifier (id attribute) and a name (which is the name of the abstract interaction unit).

An *AbstractInteractionUnit* is composed of *AbstractInteractors* and *AbstractRelationships*. The first one among these *AbstractInteractionUnits* will be the starting point for interaction.

It is possible to model the navigation among the different abstract interaction units defining instances of the *Connection* class. A connection (which has a unique identifier, id attribute) specifies the target abstract interaction unit and the interactor (or interactors) that triggers such navigation. It is also possible for the connection to specifiy more than one target interaction unit. In this case, the specification will contain also the condition (see the related attribute in the specification of the Connection element) for the dynamic selection of the target among the specified set of abstract interaction units.

The abstract class *AbstractInteractor* defines a generic interactor, which represents a generic user interface object. Its subclasses create a partition, which defines a set of different categories according to the interaction semantics. These categories are:

- *Selection*. This abstract class of interactors represents objects that allow the user to select one or more values from a predefined set of choices. The two refinements *SingleChoice* and *MultipleChoice* represent respectively interactors that allow the selection of only one value among the choices, and interactors that allow the selection of more than one value. The UI objects having such a type have a number of attributes: selected, for identifying the selected value; and cardinality, for specifying the cardinality of the value set.
- *Edit*. This class represents interactors that allow the input of manually edited values. Such an object has the value attribute to identify the currently edited object.
- *OnlyOutput*. This class represents interactors that show information to the user.
- *Control*. This abstract class represents interactors that allow the user to perform actions, submit data or navigate the user interface. The refinements *Activator* and *Navigator* distinguish the objects mainly devoted to performing actions from the ones to perform UI navigation.

Each *AbstractInteractor* can be optionally connected to a *DataItem*, specifying that it represents the referenced data item in the user interface. Each *AbstractInteractor* has also a number of attributes:

- *Id*: the unique interactor's identifier;
- *Enabled*: a Boolean value specifying whether the interactor can respond or not to the user interaction;
- *Focus*: a Boolean value specifying whether the user is actually interacting with this object;
- *Hidden*: if it is true, the interactor is not presented to the user;
- *Continuous_update*: a value specifying whether the content can be continuously updated.

The interactors usually have logical connections that allow the user to focus on different UI parts according to the task he is performing. The *AbstractRelationship* abstract class represents the base type for all the possible logical connections among the interactors. An abstract relationship consists of a composition of other abstract relationships and abstract interactors. It is possible to specify the following relationship types:

- *Grouping*: this class represents a generic group of interactors, which share a logical connection
- *Ordering*: this class represents a set of interactors that have an ordering relation among the elements. There also is *the ordering_value* attribute, which represents the value for an ordered presentation of the content
- *Hierarchy*: this class represents a set of interactors where different levels of importance can be identified. There also is the *hierarchy_value* attribute, which is used for providing a hierarchical presentation of the content.
- *Repetition*: this class represents a template for a list of interactors that have to be repeated in order to represent a dynamic list of items, coming from a data source.
- *Dependency*: A dependency is used when we want to model a dependency relation between 1 interactor/interactor group and other N interactors/interactor groups.

An *AbstractRelationship* element has a number of attributes:

- *Id*: the interactor identifier
- *Hidden*: true if the element is not presented to the user

The *AbstractDialogModel* class defines the behaviour of the UI. It contains a composition of a set of *AbstractEventCompositions*, which consist of Events, connected using *CompositionOperators*. Each *CompositionOperator* has an identifier, determining the type of composition that can occur on sets of events.

Such compositions define the set of events observed by the UI in a given state. Example of event compositions can be e.g. concurrency, sequential enabling, etc. *Event* is the abstract class that models all abstract asynchronous notifications about the changes on the state of the interactors or user inputs.

To each generic *Event,* it is possible to associate a list of *EventHandlers*, which specifies all the actions that have to be performed as a reaction to a certain event. Each *EventHandler* has its own name, and can be also guarded by a condition, which defines constraints for the reactions.

The generic event class is refined into *SelectionEvent*, *EditEvent* and *ControlEvent* classes:

- The *SelectionEvent* is associated to the selection abstract interactors and represents the notifications about the change of the selected value (or values) performed by the user.
- The *EditEvent* notifies the changes performed by the user through the associated edit abstract interactor.
- The *ControlEvents* notifies the activation of some UI functionalities (e.g. executing some script in a web page), or the switching between presentations, respectively through the activator or navigator abstract UI interactors.

3.2 Authoring Tool Support

In task 4.5, we develop authoring environments and their accompanying analysis tools that will simplify the development of adaptive SFEs for all the above-described layers, and will make it easier and efficient for designers and programmers to design, build and deploy adaptive SFEs using the languages, i.e. ASFE-DL provided by this work package**.** We plan to develop a set of tools in the form of plug-ins of some widely used development tool, such as Eclipse, as well as an HTML5 browser based tool to facilitate web based authoring.

The authoring tools will help the designers, engineers and web authors to easily create context-sensitive SFEs for different platforms, which may also use different interaction modalities, e.g. graphics, voice, touch etc. The authoring tools will provide support for editing not only the model-based descriptions at both abstract and concrete levels, but also the context-dependent transformations rules.

Figure 8 shows how the authoring tools fit into the overall SERENOA architecture at different stages of UI development, starting from AUI to CUI. The authoring environment facilitates the developers to fully utilize the CARFO ontology in order to craft not only the desired adaptation rules, but also the AUI description as well, using the respective description languages that will be developed in this work package 3

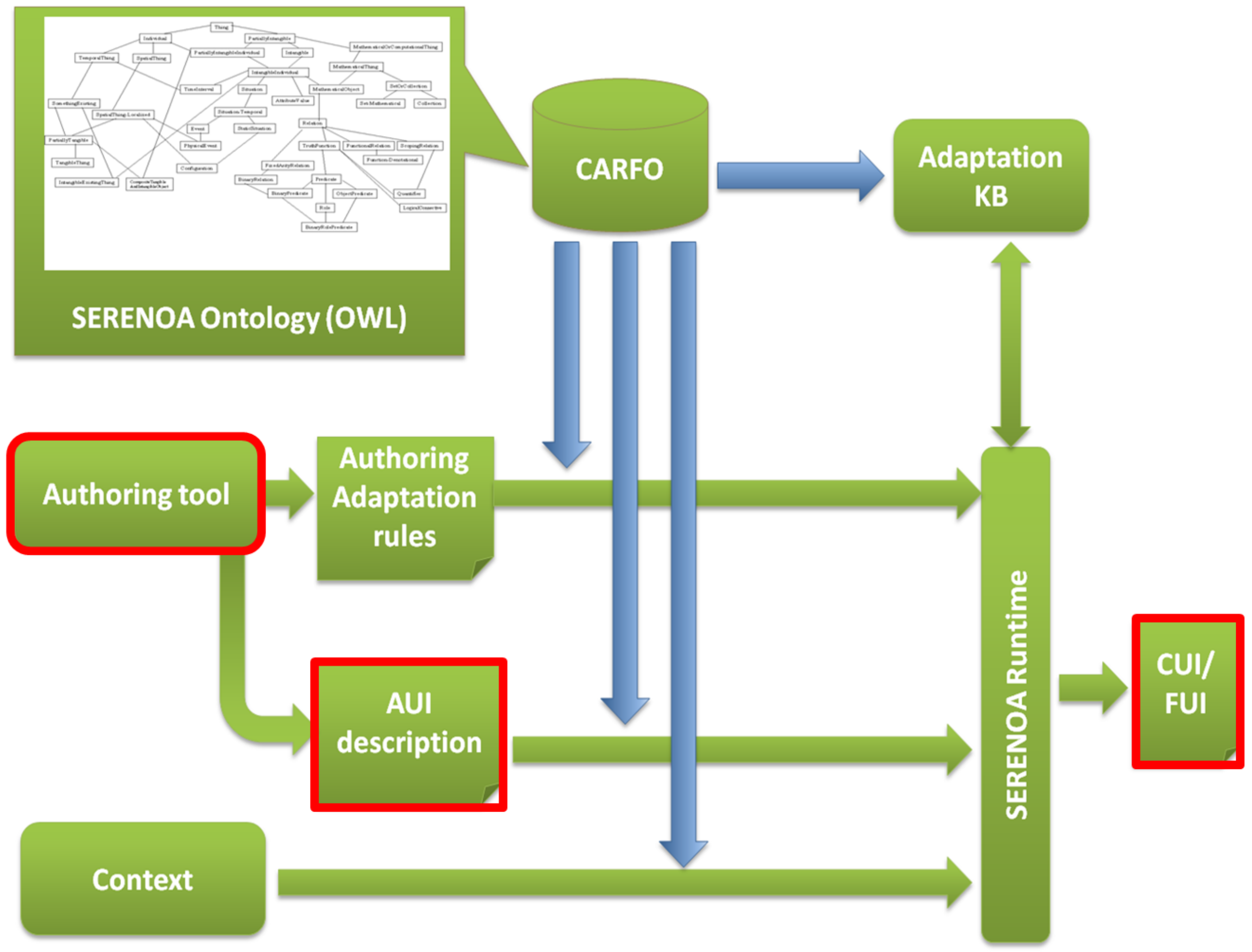

Figure 8: Adaptation Data Flow. Blue lines represent concept definitions while green lines represent exchanged streams of data

4 Conclusion

4.1 Summary

In the ‘big picture’ of the SERENOA project, the need to define an adequate reference language upon which to base the rest of the adaptation work is crucial and hence consumes resources in T3.2 all through the duration of the project. During the first months of this task, work has been focused on laying out solid foundations for the language in itself, but also keeping track of the activity in related areas of the project, such as the theoretical basis in deliverables D2.1.1 and D3.1.1, the ontology work in D2.2.1 and also more real-world issues such as the architecture for the system (D1.2.1) and the first forays into the design of SERENOA prototypes as described in D5.2.1. It is our goal to deliver a UI-DL that will allow enough expressivity to build relevant applications, but not at the expense of making a custom language with little or no practical use beyond the realms of the project – for that regard we have also made contact with the standardization activities running in SERENOA and it is envisaged that this will continue and parts of the work of this task will be subject to a standardization process. This is further explained in section 4.3.

To fulfil these aims, we have chosen to start small and lay out the most general framework we can at this stage, using knowledge from past projects and languages, as outlined in the State of the Art section, especially from the languages that members of the consortium have authored or co-authored. By leveraging on their joint experience and past work, we hope to come up with a UI-DL which is more complete and relevant than its ancestors. Besides, this reuse of existing software tools was one of the concerns mentioned by the members of the SERENOA Industrial Advisory Board (see D6.3.1) during the meeting that was held in Paris on September 17th 2011. It was then explicitly advised to reuse work done on the UsiXML project as much as possible.

In addition to this preliminary study of the state of the art and current consortium portfolio of languages, we have begun to specify our own SERENOA-DL. In this document, we have specified its core concepts at the Abstract UI level. This specification will be complemented in the future with the upper levels of the Cameleon taxonomy.

4.2 Future work

Work in T3.2 will continue for the remaining part of the project. We will continue to specify our SERENOA-DL in more concrete terms: the AUI specification here contained will be complemented with CUI and FUI descriptions that will be progressively geared towards particular platforms and implementations. In order to remain consistent with regards to the adaptation aims of the project, work will be done in parallel with the rest of the development of the SERENOA framework, especially taking into account the following:

- The Advanced Adaptation Logic Description Language (AAL-DL) to be specified in task T3.3. This language will be used to specify the adaptations that will be applied upon the UI-DL for the realization of UIs in applications under specific context constraints. Coordination work with this task will be necessary so that the two description languages grow in parallel and share as much semantic, syntactic and stylistic features as possible.
- The runtime implementation, which will have to combine the aforementioned UI-DL and AAL-DL to produce actual views for the SFE.
- The target prototypes themselves, whose requirements will be key to evaluating the progress of the SERENOA-DL in terms of completeness and expressivity.
- Last but not least, we will continue surveying the industrial and research communities for similar approaches to UI-DL. These will be explored and their features incorporated to SERENOA if they are appropriate, so that the SERENOA-DL remains as much at the cutting edge of technology as possible.

4.3 Standardization

This report covers work that we expect to submit for standardization at the World Wide Web Consortium (W3C) with a view to encouraging inter-operable formats between model-based user interface design tools, thereby increasing competition between such tools, and a larger potential market as the benefits of model-based UI design is brought to a wider audience. This is especially valuable in the context of an increasing variety of devices on which applications need to be supported. Model-based UI design makes for a better separation of concerns for developing such applications, allowing designers to focus on what they are best at, and at the same time making it easier to test early versions of designs for their usability.

W3C is in the process of launching a new working group focusing on model-based UI, and at the time of writing this report, has just completed the Advisory Committee review of the proposed working group charter. Despite many years of research, model-based approaches to UI design are still at an early stage of deployment in industry, and standards are needed to encourage investment in authoring tools. This report covers work by a number of existing model-based authoring projects, and describes a synthesis of the best ideas based upon the pooled expertise across SERENOA partners. The next step will be to reach out to authoring tool vendors, and corporate and other organizations, with a shared interest in UI design to ensure their involvement in the standards process.

5 References

[Bottoni2011] Bottoni, P., Borgia, F., Buccarella, D., Capuano, D., De Marsico, M., Labella, A., Levialdi, S.: "Issues in Model-Driven Development of Interfaces for Deaf People", Proceedings of the Workshop on Software Support for User Interface Description Language (UIDL 2011), Lisbon, Portugal, September 2011

[Calvary2002] The CAMELEON Reference Framework, G. Calvary, J. Coutaz, D. Thevenin, L. Bouillon, M. Florins, Q. Limbourg, N. Souchon, J. Vanderdonckt, L.Marucci, F.Paternò, and C.Santoro, CAMELEON Project, September 2002.

[Cantera2010] IDEAL2 Core language. José M. Cantera, C. Rodriguez, José L. Díaz. MyMobileWeb Working Draft, 31 December 2010. Available at http://files.morfeo-project.org/mymobileweb/public/specs/ideal2/ideal2-20101231/. Latest version: https://files.morfeo-project.org/mymobileweb/public/specs/ideal2.

[CSS2] Cascading Style Sheets, level 2 (CSS2) Specification. Ian Jacobs; et al. W3C Recommendation, 11 April 2008. URL: http://www.w3.org/TR/2008/REC-CSS2-20080411. Last version: http://www.w3.org/TR/CSS2/.

[Desruelle2011] Desruelle, H., Blomme, D., Gionis, G., Gielen, F.: "Adaptive User Interface Support for Ubiquitous Computing Environments", Proceedings of the Workshop on Software Support for User Interface Description Language (UIDL 2011), Lisbon, Portugal, September 2011

[DISelect] Content Selection for Device Independence (DISelect) 1.0, R. Merrick, R. Lewis, M. Froumentin. W3C Work in Progress, 29 June 2010. Available at: http://www.w3.org/TR/2010/NOTE-cselection-20100629/. Latest version: http://www.w3.org/TR/cselection/.

[Honold2011] Honold, F., Poguntke, M.,Schüssel, F., Weber, M.: "Adaptive Dialogue Management and UIDL-based Interactive Applications", Proceedings of the Workshop on Software Support for User Interface Description Language (UIDL 2011), Lisbon, Portugal, September 2011

[Kaklanis2011] Kaklanis, N., Moustakas, K., Tzovaras, D.: "An Extension of UsiXML Enabling the Detailed Description of Users Including Elderly and Disabled", Proceedings of the Workshop on Software Support for User Interface Description Language (UIDL 2011), Lisbon, Portugal, September 2011

[Limbourg2004] Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Florins, M., UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence, in Proceedings of Workshop on Device Independent Web Engineering DIWE'04(Munich, 26-27 July 2004), M. Lauff (Ed.), Munich, 2004.

[Limbourg2004a] Limbourg, Q.,Vanderdonckt, J., Michotte, B., Bouillon, L., Víctor López Jaquero, UsiXML: a Language Supporting Multi-Path Development of User Interfaces, Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems, EHCI-DSVIS'2004 (Hamburg, July 11-13, 2004), Lecture Notes in Computer Science, Vol. 3425, Springer-Verlag, Berlin, 2005, pp. 200-220

[MyMobileWeb] MyMobileWeb framework. URL: http://mymobileweb.morfeo-project.org/.

[Paternò2009] F. Paternò, C. Santoro, L.D. Spano, "MARIA: A Universal Language for Service-Oriented Applications in Ubiquitous Environment", ACM Transactions on Computer-Human Interaction, Vol.16, N.4, November 2009, pp.19:1-19:30, ACM Press.

[SCXML] State Chart XML (SCXML): State Machine Notation for Control Abstraction , J. Barnett, K. Reifenrath, M. Helbing, N. Rosenthal, R. Auburn, R. Akolkar, D. C. Burnett, J. Carter, S. McGlashan, T. Lager, M. Bodell, R. A. Hosn, T. V. Raman. W3C Working Draft, 26 April 2011. Available at http://www.w3.org/TR/2011/WD-scxml-20110426/. Latest version: http://www.w3.org/TR/scxml/.

[Serra2011] Serra, A., Navarro, A., Naranjo, J.-C: "UsiXML Extension for Avatar Simulation Interacting within Accessible Scenarios", Proceedings of the Workshop on Software Support for User Interface Description Language (UIDL 2011), Lisbon, Portugal, September 2011

[UsiXML] UsiXML website: www.usixml.org

[Vanderdonckt2004] Vanderdonckt, J., Limbourg, Q., Michotte, B., Bouillon, L., Trevisan, D., Florins, M., UsiXML: a User Interface Description Language for Specifying Multimodal User Interfaces, in Proc. of W3C Workshop on Multimodal Interaction WMI'2004 (Sophia Antipolis, 19-20 July 2004).

[XForms] XForms 1.1. J. Boyer. W3C Recommendation, 20 October 2009. Available at: http://www.w3.org/TR/2009/REC-xforms-20091020/. Latest version: http://www.w3.org/TR/xforms11/.

[XMLEvents] XML Events. Shane McCarron et al. W3C Recommendation, 14 October 2003. Available at: http://www.w3.org/TR/2003/REC-xml-events-20031014/. Latest version: http://www.w3.org/TR/xml-events2/.

Acknowledgements

- TELEFÓNICA INVESTIGACIÓN Y DESARROLLO, http://www.tid.es
- UNIVERSITE CATHOLIQUE DE LOUVAIN, http://www.uclouvain.be
- ISTI, http://giove.isti.cnr.it
- SAP AG, http://www.sap.com
- GEIE ERCIM, http://www.ercim.eu
- W4, http://w4global.com
- FUNDACION CTIC http://www.fundacionctic.org

Glossary

A SERENOA-wide glossary of terms can be found online at:

http://serenoa.morfeo-project.org/glossary-of-terms