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

Project no. FP7 – ICT – 258030

Deliverable D.4.1.1
Runtime UI Generation Engine (R1)

Due date of deliverable: 29/02/2012
Actual submission to EC date: 29/02/2012

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)

| Dissemination level | [PU] | [Public] | Yes |

This work is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA (This license is only applied when the deliverable is public).
### Document Information

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Lead Contractor</strong></td>
<td>Javier Rodríguez Escolar</td>
</tr>
<tr>
<td></td>
<td>Cristina González Cachón</td>
</tr>
<tr>
<td></td>
<td>Ignacio Marín Prendes</td>
</tr>
<tr>
<td><strong>Editor</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Revision</strong></td>
<td>28/02/2012</td>
</tr>
<tr>
<td><strong>Reviewer 1</strong></td>
<td>Telefónica I+D</td>
</tr>
<tr>
<td><strong>Reviewer 2</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Approved by</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Project Officer</strong></td>
<td>Mr. Michel Lacroix</td>
</tr>
</tbody>
</table>

### Contributors

<table>
<thead>
<tr>
<th>Partner</th>
<th>Contributors</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTIC</td>
<td>Javier Rodríguez Escolar</td>
</tr>
<tr>
<td></td>
<td>Cristina González Cachón</td>
</tr>
<tr>
<td></td>
<td>Ignacio Marín Prendes</td>
</tr>
<tr>
<td>CNR</td>
<td>Fabio Paternò</td>
</tr>
<tr>
<td></td>
<td>Lucio Davide Spano</td>
</tr>
<tr>
<td></td>
<td>Christian Sisti</td>
</tr>
<tr>
<td>UCL</td>
<td>Vivian Genaro Motti</td>
</tr>
<tr>
<td></td>
<td>Jean Vanderdonckt</td>
</tr>
<tr>
<td></td>
<td>Pascal Beaujeant</td>
</tr>
<tr>
<td>SAP</td>
<td>Jörg Rett</td>
</tr>
<tr>
<td>TID</td>
<td>Francisco Javier Caminero Gil</td>
</tr>
<tr>
<td>W4</td>
<td>Jean-Loup Comeliau</td>
</tr>
<tr>
<td>Version</td>
<td>Date</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>1</td>
<td>16/01/2012</td>
</tr>
<tr>
<td>2</td>
<td>13/02/2012</td>
</tr>
<tr>
<td>3</td>
<td>16/02/2012</td>
</tr>
<tr>
<td>4</td>
<td>23/02/2012</td>
</tr>
<tr>
<td>5</td>
<td>24/02/2012</td>
</tr>
<tr>
<td>6</td>
<td>27/02/2012</td>
</tr>
<tr>
<td>7</td>
<td>28/02/2012</td>
</tr>
</tbody>
</table>
Executive Summary

This deliverable tries to describe the runtime engine of the Serenoa framework (RUIGE, Runtime User Interface Generation Engine). It is composed by sub-modules, each one in charge of generating different application renderers. All sub-modules have an abstract language in common called ASFE-DL to define abstract user interfaces. Then, this definition will be transformed to a different CUI (Concrete User Interface) depending on the sub-module. Such CUI will be used to create the FUI (Final User Interface) and deliver it to a client device to execute the application. The term application is used in this deliverable as a synonym for Service-Front End. Thus, the applications generated by the RUIGE (actually, by any of its sub-modules) are applications which provide a user interface that allows interaction between the user and one or more services deployed remotely. As publishing services by means of different technologies has been sufficiently studied in previous research works, the Serenoa project does not consider this aspect. It focuses on the provision of user interfaces which adapt to the context and which play the role of front-end for those remote services.

In order to help achieving this, a set of rules may also be defined in AAL-DL to manage adaptation at the abstract, concrete and final UI levels. Other modules of the Serenoa framework, such as the Context Manager and the CARFO Ontology, will help RUIGE to make all the adaptations.

At the end of the development of this document, two sub-modules are in an advances development status: one for the creation of mobile UIs (based on the MyMobileWeb technology) and another for the creation of vocal UIs (based on the MARIA technology). An additional RUIGE submodule for desktop (HTML5 web and Java Swing) applications is also prepared for more basic applications, and another one for interaction with an avatar which is starting its development.

However, all these sub-modules and their corresponding interaction modes are just examples of the possibilities of the Serenoa framework. From the expressive power of the ASFE-DL and AAL-DL plus the functionality of other Serenoa modules, such as the Context Manager, provide richer possibilities than those offered by the RUIGE sub-modules under development.
# Table of Contents

1 Introduction .................................................................................................................. 8

1.1 Objectives .................................................................................................................. 8

1.2 Audience .................................................................................................................... 8

1.3 Related documents ..................................................................................................... 8

1.4 Organization of this document .................................................................................. 8

2 General overview of RUIGE .......................................................................................... 9

3 Components of a RUIGE sub-module .......................................................................... 11

4 Runtime Requirements .................................................................................................. 12

4.1 Vocal module ............................................................................................................. 12

4.2 Mobile Web module ................................................................................................. 12

4.3 Avatar Module .......................................................................................................... 12

5 The role of the Runtime in the Serenoa framework .................................................... 14

5.1 Relationship with other modules ............................................................................. 14

5.1.1 CARFO ................................................................................................................. 14

5.1.2 AAL-DL rule system ............................................................................................. 14

6 Description of the modules ............................................................................................ 16

6.1 Mobile Web RUIGE sub-module: MMW-4S ............................................................. 16

6.1.1 UI Transformer ..................................................................................................... 16

6.1.2 UI Generator ........................................................................................................ 16

6.1.3 UI Runtime .......................................................................................................... 17

6.1.4 Demo M18 example ............................................................................................. 17

6.1.5 Maturity Level ...................................................................................................... 19

6.2 MARIA RUIGE sub-module ...................................................................................... 19

6.2.1 UI Transformer ..................................................................................................... 20

6.2.2 UI Generator ........................................................................................................ 22

6.2.3 UI Runtime .......................................................................................................... 24

6.2.4 Demo M18 example ............................................................................................. 24

6.2.5 Maturity Level ...................................................................................................... 29

6.3 UsiXML RUIGE sub-module ..................................................................................... 29

6.3.1 UI Transformer ..................................................................................................... 30

6.3.2 UI Generator ........................................................................................................ 31

6.3.3 UI Runtime .......................................................................................................... 31

6.3.4 Maturity level ....................................................................................................... 31

6.4 Avatar RUIGE sub-module ....................................................................................... 31

6.4.1 UI Transformer ..................................................................................................... 32

6.4.2 UI Generator ........................................................................................................ 32

6.4.3 UI Runtime .......................................................................................................... 33

6.4.4 Example ................................................................................................................ 34
Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>RUIGE architecture</td>
<td>10</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Mobile User Interface</td>
<td>18</td>
</tr>
<tr>
<td>Figure 3</td>
<td>The global adaptation architecture</td>
<td>21</td>
</tr>
<tr>
<td>Figure 4</td>
<td>The adaptation process in detail</td>
<td>21</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Diagram exemplifying the transformations between different granularity levels</td>
<td>30</td>
</tr>
<tr>
<td>Figure 6</td>
<td>SAIBA framework phases</td>
<td>32</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Adaptation example for TID scenario</td>
<td>35</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Adapting the e-Commerce customer web portal for disabilities</td>
<td>37</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Adapting the e-Commerce customer web portal for disabilities</td>
<td>37</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Menu adaptation</td>
<td>38</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Table view adaptation</td>
<td>38</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Adaptation of a graphical user interface to different platforms</td>
<td>39</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Graphical user interface for the wearable computer to directly support the task of picking</td>
<td>40</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Graphical user interface adapting to the user profile</td>
<td>40</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Objectives

This document describes the proposal for the Runtime UI Generation Engine (RUIGE) within the Serenoa project, which will be in charge of consuming all the logical descriptions defining a Serenoa application in order to generate, deploy, and execute it. These descriptions are the CARFO ontology, the ASFE-DL specifying user interfaces at an abstract level and the adaptation rules, expressed in AAL-DL.

As committed in the Description of Work, the RUIGE will allow the execution of Serenoa applications at least in a Web 2.0 Platform (by means of HTML5, CSS, Javascript and AJAX-like approaches), covering both desktop and mobile environments, and vocal UIs (for instance, based on VoiceXML). Multimodal UIs are also in scope for the future.

Note that both application generation and user interface generation expressions are indistinctly used in this document. The Serenoa framework is intended to help developers implementing Service Front-Ends, which means applications that work as a front-end for services. This implies that generating adaptable user interfaces and the mechanisms for navigation between the different screens/views/presentations is the main challenge. The development of services and the way in which they are exposed is out of the scope of the Serenoa project.

1.2 Audience

The audience for this document are the following groups:

a) Members of the consortium, in order to understand the runtime engine of the Serenoa framework: how it works, which is its role, the different modules and their functionality…

b) Developers and developer communities, in order to understand how to create ad-hoc modules fully compliant with RUIGE and the Serenoa framework.

c) The members of the research community dedicated to the creation of context-aware applications, who might learn concepts and techniques from the Serenoa framework and improve it.

d) 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

- **D1.1.2 Requirements analysis** contains the basic requirements for the runtime engine.
- **D1.3.2 Serenoa Roadmap** describing next steps in the evolution of the Serenoa framework.
- **D2.2.1 CARFO (R1) and D2.3.1 CARFO Population**, which describe an Ontology for Multi-Dimensional Adaptation of SFEs and how it is populated. It will be useful for RUIGE in order to guide the transformations to be performed at runtime.
- **D3.2.1 ASFE-DL: Semantics, Syntaxes and Stylistics** describe the abstract language that is used by the runtime.
- **D3.3.1 AAL-DL Semantics, Syntaxes and Stylistics** describe the adaptation rules languages used by the runtime.
- **D4.3.1 Adaptation Engine** describes some adaptations needed on the runtime.

1.4 Organization of this document

Chapter 0 is an introduction that describes the objectives, the audience and related documents of the deliverable. Chapter 2 outlines a general overview of the runtime of the Serenoa framework. Chapter 3 details the different components in a module of the runtime. Chapter 4 specifies the role of the runtime inside the whole project and the relations established with other modules. The mission of chapter 5 is to describe the requirements of RUIGE. As the RUIGE is composed by different runtime sub-modules, those under development in this project are described in chapter 6. To finish the deliverable, Chapter 7 describes the conclusions of the work achieved in the development of the RUIGE module so far, and the future work to be carried out until the end of the project.
2 General overview of RUIGE

The Runtime User Interface Generation Engine (RUIGE) is the module of the Serenoa Framework responsible for the generation, deployment and execution of applications. In order to achieve this, it must be able to interpret the specification of the UI of the application at the Abstract UI level (ASFE-DL), transform it to the corresponding Concrete UI specification and finally generate the Final UI of the application, execute it, and provide support for runtime adaptations. For that purpose, adaptation rules specified in AAL-DL must be interpreted by RUIGE. These rules may, for instance, imply transformations from ASFE-DL to other user interface description languages at lower abstraction layers (concrete or final, for example), and may also guide run-time adaptations of the user interface. The RUIGE module may also have to communicate with other modules of the Serenoa framework, such as the Context Manager, the CARFO ontology and, of course, with the back-end services to which the application will work as a front-end.

Considering the ambitious goal of the Serenoa framework (a software platform envisioned to help developers in the creation of any kind of applications which adapt to the context, as defined in the CADS and in the CARF [see D2.1.2]) and given the complexity of both those design space and reference framework, the RUIGE has been conceived as a modular engine in which diverse sub-modules allow the generation, deployment and execution of an application for different interaction modes and target technology platforms. In this way, the aim of the project is to demonstrate the framework with an initial set of RUIGE modules but with the intention to allow future developments to be included in the Serenoa framework, thus supporting additional interaction modes and target platforms for the same application.

By definition, a RUIGE module is a piece of software that can receive a Serenoa application definition by means of:

- Its UI definition in ASFE-DL. Such definition may be performed by means of the corresponding Authoring Tools (see D4.5.1).
- A set of adaptation rules expressed in AAL-DL coming from the Adaptation Engine (see D4.3.1). These rules may guide the transformation process from abstract user interface definition to lower UI definition levels (CUI or FUI). Rules expressed in AAL-DL also allow for adaptation at runtime of both the user interface itself and of the resources referenced by the UI – such as media elements.
- Additional relevant information sources for any runtime provided by the Serenoa Framework, such as the CARFO ontology (see D2.2.1) and the Context Manager (see D4.4.1).

After processing the abovementioned inputs, a RUIGE module should be able to provide:

- The final code of the application expressed in the appropriate language (FUI)
- The runtime support, commonly distributed as a set of libraries, for performing on-the-fly adaptations while the user interacts with the application.

From the previous elements, which define a Serenoa application, a RUIGE module must be able to execute the application in a specific amount of target platforms and for a specific amount of interaction modes.

Figure 1 shows the way in which the RUIGE (each of its sub-modules) interacts with other parts of the Serenoa platform. Each sub-module can interpret an ASFE-DL document with a transformer (called adapter in previous deliverables) in order to generate an internal representation of the UI of the application. This internal representation must be a CUI language specific to the technology and interaction modes supported by the corresponding runtime. Such internal representation would be used by the corresponding generator sub-module to create the final code of the application (and, if needed, additional required resources). In some cases, this may mean generating a web application which must be deployed and started in a web server or application server. In some others, it may be an installation file or, more directly, an executable which can be downloaded by users to be installed or run in their devices. Note that the modular architecture of RUIGE is exemplified by means of specific sub-modules in Figure 1. These modules will be further explained on chapter 6.

As indicated in the Description of Work, the RUIGE is intended to render for at least the Web 2.0 Platform (HTML5, SVG, CSS, Javascript, AJAX), covering both desktop and mobile environments, and vocal interfaces (supported by VoiceXML-based technology). So far, the RUIGE allows application generation for both mobile (including tablets) web browsers and vocal interfaces. However, more RUIGE modules are
under development, covering more interaction modes and target platforms – as noted in section 5 of this document.

Figure 1. RUIGE architecture
3 Components of a RUIGE sub-module

As proposed in Figure 1, each RUIGE sub-module must include the same set of components, each of them solving the different stages required to obtain the execution of an application from its definition in ASFE-DL plus additional adaptation rules in AAL-DL. Those stages are defined as:

- **Transformation**: a component called transformer is in charge of the process of converting ASFE-DL into a representation of the user interfaces and additional aspects required to define an application in the format required by the next stage (application generation). Typically, that representation will use a CUI language plus, potentially, additional languages to define other aspects of application definition at an abstraction layer lower than the abstract one. This stage is **mandatory**.

- **Application generation**: a component called application generator analyses the abstract or concrete definition of the application in order to generate the executable or interpretable code which will be used at runtime. Optionally, it might directly interpret either the abstract or the concrete definition of the application and render the application. This stage is **mandatory**.

- **Deployment**: This is an optional step which is required in order to deploy the application before execution. It implies publishing the executable code of the application for users to download it and then execute it. For instance, if the application is a single .exe file, it might be uploaded to a web server, so users can download it. Another example is the case of a web application packaged as a .war file or a similar format, which needs to be deployed in an application server or in a servlet container, depending on the content of the application. This stage is **optional** and it may be automatically performed by the application generator in its last step or even manually, by the developer.

- **Execution**: This is the stage which takes place when the application is executed. It is responsibility of the runtime module and it is obviously **mandatory**. It is performed by an operating system executing machine code, a virtual machine interpreting bytecodes or similar interpretable object code (or compiled just-in-time), a web browser or another execution platform –including the application generation itself, in the case that it directly interprets AUI or CUI.
4 Runtime Requirements

The definition of some requirements is needed for each of the modules in the Serenoa framework (see D1.1.2). In this section, runtime aspects will be described. RUIGE, as it was commented before, is divided into different sub-modules, each of them supporting the following requirements:

4.1 Vocal module

- **Handling CUI descriptions**: the Vocal Transformer takes a CUI for a desktop platform and transforms it into a CUI for the vocal platform.
- **Customisation**: the transformations carried out by the Vocal Transformer can be customised by the end user.
- **Generation from MARIA Vocal CUI**: the Vocal Generator generates a Final UI, starting from a concrete UI description for the vocal platform.
- **Generation from Serenoa Vocal CUI**: The Vocal Generator generates a Final UI, starting from a concrete UI description for the vocal platform.
- **Support from final-to-concrete UI transformation**: the Reverser builds a Concrete UI logical description starting from a web implementation.
- **Support from concrete-to-abstract UI transformation**: the Reverser builds an Abstract UI logical description from a CUI logical description.

4.2 Mobile Web module

- **Transformation from AUI to CUI**: the Mobile Web transformer takes an AUI description and transforms it in a CUI description tailored to mobile Web.
- **Support for Adaptation Rules in the transformation process**: the Mobile Web transformer must take into account the Adaptation Rules defined at the abstract level during the transformation process.
- **Generation from CUI to FUI**: the Mobile Web generator takes a Mobile Web CUI description and generates the appropriate FUI according to the target device.
- **Support for Adaptation Rules in the generation process**: the Mobile Web generator must take into account the Adaptation Rules defined at the concrete level during the transformation process.
- **Support for multi-device generation**: The Mobile Web generator must be able to generate the appropriate contents (mark-up, Javascript, style sheets) for each Delivery Context.
- **Generation performance**: the performance of the generation process should be sufficient to guarantee the simultaneous access of several clients with an acceptable response time.
- **User interfaces usability**: the final User Interfaces generated by this module should be usable.
- **Provide dynamic information to the Context Manager**: each module within RUIGE architecture must provide the convenient mechanisms to provide dynamic information to the Context Manager.
- **Retrieve information from the Context Manager**: each module within RUIGE architecture must provide the appropriate mechanisms to obtain information from the Context Manager, both in synchronous and asynchronous modes. Note that the Context Manager will contain information coming from different external sources, such as DDRs, CARFO, etc.

4.3 Avatar Module

- **Generation from AUI to CUI**: the avatar transformer converts from the AUI language in SERENOA to an intermediate CUI language in which general-use human-avatar dialogue is described.
- **SERENOA Avatar CUI is based upon FML**: the intermediate CUI language for SERENOA should be close to standards in the field. We are targeting the FML (Functional Mark-up Language) although our implementation may not be complete or differ in some aspects.
- **Generation from CUI to FUI**: the Avatar Generator must transform the CUI describing general-use human-avatar dialogue language into one which is suitable for a particular implementation of avatar.
- **SERENOA Avatar FUI is based upon BML**: The FUI language for SERENOA should be close to
work in the field. We want to use the concepts of BML\(^1\) (a preliminary proposal is available).

- **Avatar FUI supports at least two avatar engines**: avatar engines are a very heterogeneous set in terms of functionality and requirements upon the underlying platform. SERENOA should target a CUI that is able at least two FUI implementations for different avatars, crucially differing in hardware requirements, e.g., an avatar engine for desktop and other for mobile.

\(^1\) [http://www.mindmakers.org/projects/bml-1-0/wiki/Wiki](http://www.mindmakers.org/projects/bml-1-0/wiki/Wiki): Behavioral Mark-up Language definition
5 The role of the Runtime in the Serenoa framework

As commented in Section 2, the RUIGE and all its possible modules are responsible for the generation, deployment and execution of Serenoa applications. However, its functionality depends on other modules of the Serenoa architecture commented in other deliverables of this project.

5.1 Relationship with other modules

5.1.1 CARFO

The CARFO ontology is defined in Serenoa as a knowledge system which is used to define all the relevant concepts defined by the Context-Aware Design Space and the Context-Aware Reference Framework (CADS an CARF, see D2.1.2). More particularly, at the time of writing this version of D4.1.1, it models the Context of Use, providing a common vocabulary for all the modules of Serenoa. In this way, all of them will use the same concepts in the same manner and the same relationships between them. CARFO may also allow certain degrees of reasoning and information queries on knowledge bases built following this ontology.

A clarifying example of the use of the CARFO in Serenoa is the possibility to embed more or less complex grouping/classification/aggregation criteria in SPARQL queries to the Context of Use module. Device grouping is one of the main warhorses in multi-device application development, as stated in the Device Description Structures document [1] released by W3C within the Device Description Working Group. For instance, a SPARQL query might be used to decide whether a specific device is considered a tablet, in order to apply or not one or more adaptation rules. From the same knowledge base built on top of the definitions in the CARFO, different organizations might use their different criteria in the form of distinct SPARQL queries using the same parameter and result types in order, for example, to classify the devices which access their applications and services in a different way. Device classification is one of the most controversial issues in application and content adaptation, as it is tightly coupled to the adaptation approaches followed by each specific adaptation platform and application domain.

5.1.2 AAL-DL rule system

AAL-DL is the language proposed in Serenoa to express adaptation rules. These adaptation rules may be applied:

- To guide adaptations from ASFE-DL to CUI languages. In Figure 1, this is expressed by the arrow from the AAL-DL module to the transformer of each RUIGE sub-module.
- To guide adaptations from CUI languages to FUI languages. In Figure 1, this is represented by the arrow from the AAL-DL module to the app generator of each RUIGE sub-module.
- To guide runtime adaptations. In Figure 1, this is represented by the arrow from the AAL-DL module to the runtime applications. In this case, it is possible that the runtime of each RUIGE sub-module uses a rule based system, but it is also possible that the rules are embedded in the executable code of the application. An additional possibility is the translation of AAL-DL rules into the rule language for existing rule-based system such as Drools, Jess or JRules, so the runtime takes decisions at runtime by means of those systems.

5.1.2.1 Context Manager

The RUIGE, as all the other modules of the Serenoa framework, may benefit from the services offered by the Context Manager module. This module provides information about the Context of Use of the application. In the case of the RUIGE, information obtained from the Context Manager is useful in different stages covered:

- **Transformation**: the transformer may need to solve, for instance, the evaluation of expressions in the antecedent of a rule expressed in AAL-DL. Such antecedent may contain references to static information about the Context of use (namely, static properties in device descriptions) which would be stored by the Context Manager, in order to adapt from ASFE-DL to a CUI or another intermediate definition of an application.
- **Application generation**: the application generator of a RUIGE sub-module may also query static information of the Context of Use, mainly static device descriptions to generate
executable/interpretable code or to interpret an abstract or concrete definition of an application.

- **Execution**: The application may adapt itself to the Context of Use after querying distinct static or dynamic properties stored in the Context Manager, such as device descriptions or user profiles.

The Context Manager shall be queried by means of the vocabulary specified in D4.4.1. via SOAP-based web services, RESTful web services and Web Sockets.

### 5.1.2.2 Backend Services

Being the Serenoa framework a software platform to generate applications, which will act as Service Front-Ends, one of the mandatory pieces of its architecture is services themselves. However, exposing functionality as services is a part of computer science sufficiently covered by both the developer and researcher community. Therefore, this project does not cover how services are created and made available to a Service Front-End.

Considering that there are many software libraries to build clients to access remote services in all the development technologies/languages/platforms, it is not mandatory for a RUIGE module to support a specific set of service consumption technologies. Nevertheless, following the current trends in consumption of remote services, it is recommended that information exchange via both SOAP-based web services and RESTful web services is supported by any RUIGE module.
6 Description of the modules

6.1 Mobile Web RUIGE sub-module: MMW-4S

MyMobileWeb [2] is an open-source project licensed under the LGPL, which has been created in order to help developers to build mobile web applications for the majority of web-enabled mobile devices in the market. Until the beginning of this project, MyMobileWeb was focused on mobile phones and PDAs. The evolution of MyMobileWeb in its convergence with the Serenoa framework will also include tablets in the scope of MMW. MMW is hosted, as a self-contained project, in the Morfeo Project [3], the open-source community that also hosts the Serenoa project.

In order to obtain a deep description of the platform, the best source is the web site of the MyMobileWeb project. This site provides access to the SVN repository containing the source code [4] and the Wiki [5] with the description of the platform and its modules. The Wiki includes a tutorial that helps developers to create their first application, step-by-step and covering all the languages and features of the platform [6].

MMW is implemented in JavaEE. It uses existing many open-source Java components and JSRs and it relies on standards from organizations such as W3C, OMA or the 3GPP. In addition to the usage of standards, several contributors to MMW have contributed to W3C working groups (Mobile Web Best Practices, Device Description and Ubiquitous Web Applications) in the definition of Working Group Notes and Recommendations, and in the creation of Reference Implementations for some Recommendations.

The approach in MMW is to provide developers with a declarative approach so they can specify what they want their applications to do, instead of how to do it – as it happens in imperative approaches. The declarative approach is based on two languages: IDEAL2, which is a higher abstraction language with a set of generic UI components which can be mapped to their equivalents in the different markup languages used by the mobile web browsers in the market. It is augmented by means of a generic CSS language whose features are also mapped to the different CSS features supported by the different markup languages; SCXML is a W3C Recommendation to define finite state machines, which is used in MyMobileWeb to express application flows.

MMW supports various type of context adaptation such as media transcoding on-the-fly to the best format supported by the client device or internationalization support (localizing contents after detecting the language of choice by the user).

This section of the D4.1.1 document is aimed to describe how MyMobileWeb converges with the Serenoa framework by becoming a RUIGE module called MMW-4S (MyMobileWeb For Serenoa).

All the source code has been released as open source and it is available under the Serenoa Morfeo Forge at [8]

6.1.1 UI Transformer

The UI Transformer for MMW is an XSL transformation sheet which converts the description of an application in ASFE-DL to an equivalent description in IDEAL2, the language used in MMW to describe UIs, and SCXML, the language used in MyMobileWeb to express the application flow.

A description of the transformation process will be provided in the next version of this deliverable, D4.1.2. The most up-to-date version of the XSL transformation sheet is available at [9].

6.1.2 UI Generator

The UI Generator for MMW is the part of the platform which creates a set of JSPs that implement the presentations defined in IDEAL2, and a set of Java classes that implement the SCXML state machine representing application flow. Each presentation in IDEAL2 implies the generation of a JSP for each markup language in the market (WML, XHTML Basic/MobileProfile, HTML4 and the new HTML5 developed within Serenoa). As it will be described in 6.1.3, client devices accessing the generated web application at runtime will be served by the appropriate JSP for each presentation after a device identification process based on the analysis of the headers of the HTTP Requests sent from the web browser. The UI generator finally creates a WAR file which can be deployed in a servlet container, such as Apache Tomcat, in order to
execute the web application.

### 6.1.3 UI Runtime

Once the application is deployed at a Servlet container, the MyMobileWeb application is available at a specific URL. The application starts in the state specified as initial in the SCXML state machine. Consequently, the application will serve the presentation corresponding to the IDEAL2 presentation associated to that initial state, by interpreting the JSP corresponding to that presentation which matches the markup supported by the client device. This decision was performed in the past by a Device Description Repository module built on top of the WURFL database but this has been changed in the context of the Serenoa project so the Context of Use module developed in Serenoa is queried.

### 6.1.4 Demo M18 example

The demo will present a mobile interface to show a renting-car agency. It will have the following key features:

- An ASFE-DL definition of the interface (common to all the sub-modules)
- An XSLT that transforms ASFE-DL to CUI, in this case, to IDEAL 2 + SCXML
- Some adaptation rules will be considered. It must be visualized at least, in two different mobile devices, such as a mobile phone and a tablet.
- It will use some static context information in order to adapt the application according to the static capabilities of the device.
- It will use some dynamic context information such as user preferences or geolocation to have a better “on the fly” adaptation
- Connect to a knowledge base to consume data from a Web Service

In Figure 2, a mobile screen diagram of the application is shown. The application tries to imitate the webpage of an agency for renting a car. It will contain some forms to introduce user data and some searching criteria to look for the car you are interested in. This is composed by various screens:

- **Login**: the first presentation to access to the application will be a login page for authentication purposes.
- **Registration**: if the user is not logged in the database of the renting-car agency, she must register in the application. After that, she should login into the system.
- **UserInfo**: This view shows the registration data that the user has filled out on the form. Data can be modified in this section.
- **Preferences**: After being logged, this is the first presentation. It shows a list of preferences to define the look-and-feel of the application as location awareness, ordering criteria, the way of showing the list of agencies,…
- **Search Criteria**: On the third tab, some searching criteria are listed as agency location, car features, etc.
- **Select Agency by list**: Once that the criteria has been defined, a different screen might appear depending on the selected preferences. If the user has chosen the list option, a series of car agency names will be shown.
- **Select Agency by map**: If the user has selected the alternative choice, a map will be rendered pointing the nearest agencies.
- **Select car**: After selecting one of the agencies, a list of cars will appear on the screen.
- **Car details**: This presentation shows the details of the chosen car.
After the explanation of the basic functionalities of the prototype, it is necessary to introduce some adaptation rules to be considered during the application lifecycle. In order to exemplify how MyMW-4S treats adaptation rules we will suppose the following set of rules, as coming from the Adaptation Engine. For the sake of legibility they are expressed in natural language in this document, although they are defined in AAL-DL in the demonstrator prototype:

- **R1**: If the device is a tablet, then master-detail presentations will be rendered in one single view.
- **R2**: If user is colour-blind, then it will use an alternative colour palette in the referenced images.
- **R3**: If the level of the battery is higher than a prefixed threshold, then a video must be shown but if its level is lower, it renders a picture.
- **R4**: If user is dyslexic, then change the type of fonts in the application.

Note that each rule needs to be applied at a different level:

- **R1** must be considered by the transformer during the transformation process when converting from ASFE-DL (AUI) to IDEAL2 + SCXML (CUI). In the demonstration, the transformer has been implemented by means of an XSLT. This XSLT generates two different versions of the IDEAL2 UI description. In the case of tablets, a master-detail view is expressed in just one IDEAL2 presentation. In the case of mobile devices, a master-detail view is expressed in two different IDEAL2 presentations, plus the additional flow logic expressed in SCXML.
- **R4** must be considered by the generator during the generation process when converting from IDEAL2 + SCXML (CUI) to HTML5 + CSS3 (FUI). In this case, the adaptation rule is considered in the generation process to determine the CSS style associated to the rendered fonts.
- **R2** and **R3** must be considered at execution time by the runtime module in order to dynamically adjust the multimedia content. In this case, the adaptation rules have been transformed into rules expressed in DRL [7], so as they can be executed at runtime by using the new rule engine developed as part of Serenoa project. Such rule engine is based on Drools [Ref].

### 6.1.5 Maturity Level

MMW is a previously existing project in use by dozens of companies and organizations around the world. Its evolution in Serenoa implies the modification of some components and the development of new ones, as commented in the following list:

- **Evolution of existing renderers**: the renderers are the software modules in MyMobileWeb in charge of the generation of the different JSPs for each IDEAL2 presentation and supported markup technology. Existing renderers (**HTML4, WML and XHTML Basic/Mobile Profile**) have been tested against a larger set of actual devices in the market and thus refined. All those renderers are considered as **mature** for production applications.
- **Development of an HTML5 rendered based on JQuery Mobile**: Until the beginning of the Serenoa Project, HTML4-based content was delivered to the latest state-of-the-art HTML5 mobile browsers. The exploitation of HTML5 at its maximum level is one of the goals of the MMW RUIGE sub-module. Thus, a new HTML5 renderer has been developed by using the popular JQuery Mobile web development framework. The HTML5 renderer is experimental and will be further developed during the rest of the project. It will open space to incorporate advanced content and application cache, offline applications, advanced multimedia components in the development of multi-device mobile web applications. The consideration of all these features within the usage of models to create final applications is another major challenge to be faced in the rest of the Serenoa project. However, the development of the HTML5 renderer is currently sufficient to support the M18 demo application.
- **Alignment with adaptation based on AAL-DL**: This is under initial development. So far, a minimum consumption of AAL-DL rules is done, for the purpose of the M18 demo. In previous R&D projects, a prototype of MMW using the DROOLS rule engine for runtime transcoding tasks has been carried out so it is expected that AAL-DL to DROOLS conversion is achieved during the rest of the project.

### 6.2 MARIA RUIGE sub-module

The MARIA engine can be exploited through two possible paths in the model-based adaptation process:
One starts with abstract UI, derives the concrete descriptions and then the corresponding implementations (thus it is mainly performed at design time);

The other starts with the desktop implementation, builds the corresponding concrete description through reverse engineering techniques and then adapts it for the vocal platform and generates the implementation for this modality as well (thus it is mainly performed at run-time). Here we describe how generation can be obtained in both cases. We do not describe the reverse engineering tool because it does not seem relevant for a deliverable on user interface generation.

6.2.1 UI Transformer

One of the two paths in the adaptation process in which we use the vocal generator starts with the abstract ASFE-DL. It is transformed in an abstract MARIA description, and then into a concrete vocal description. In this section, we provide a brief description of the transformation between the ASFE-DL language and the MARIA AUI. Both languages are at the same abstraction level according to the Cameleon reference framework. The transformation has been implemented using an XSLT.

The AbstractUIModel is transformed into an Interface element, including the description of the DataModel which is translated to an XSD definition in order to be supported in MARIA.

The AbstractInteractorUnits (AIU) are represented by Presentations. The Connection class (which links two different AIU are mapped to ElementaryConnections if they have no conditions inside, or to ConditionalConnections otherwise.

The AbstractRelationship subclasses are mapped to InteractorComposition elements. More precisely:

- the Repetition is mapped to a Repeater
- the Hierarchy and the Ordering are mapped to a Grouping (respectively with the hierarchy or the ordering flag set to true)
- the Grouping is mapped to his homonymous counterpart in MARIA
- the Dependency is mapped to a Relation

The AbstractInteractors are mapped towards the classes with the same name in MARIA (e.g. SingleChoice, MultipleChoice, OnlyOutput etc.), together with their events.

We exploit a model-based language (MARIA) for performing a transformation that preserves the semantics of the interaction. The framework provides abstract (independent of the interaction modality) and concrete (dependent on the interaction modality but independent of the implementation) languages. Such languages share the same structure with different levels of refinements. An AUI (Abstract User Interface) is composed of a number of presentations, a data model and a set of external functions. Moreover, each presentation contains a number of user interface elements, called interactors, and a number of interactor composition operators. Examples of interactor composition operators are grouping and relations to group/relate different interactors. The interactors are first classified in abstract terms of editing, selection, output and control and may have a number of associated event handlers. While in graphical interfaces the concept of presentation can be easily defined as a set of user interface elements perceivable at a given time (e.g. a page in the Web context), in the case of a vocal interface we consider a presentation as a set of communications between the vocal application and the user that can be considered as a logical unit, e.g. a dialogue supporting the collection of information regarding a user. Examples of interactors are navigators (allow moving from one presentation to another) and description (allow TTS functionality). A grouping can contain both interactors and other composition interactors (such as groupings itself). Another composition concept used to structure the presentation is the relation operator, which defines a kind of relation between two groups of elements, typical example is a form with one group for editing input values and one for controlling them (clearing, sending to the server, …).

Our solution is based on an adaptation server, which provides a number of functionalities (see Figure 3):

- **Reverser**, parses the content of the Web page and the associated style sheets, and builds up a corresponding Desktop Concrete Logical Description (this module is not relevant for this deliverable);
• **Graphical-to-Vocal Adapter**, transforms the Desktop Concrete Logical Description into a Vocal Concrete Logical Description, which is optimized for vocal browsing;

• **VoiceXML Generator**, a VoiceXML implementation is generated from the vocal concrete description so that the final result can be loaded onto a vocal browser for execution.

The work described in this paper focuses principally on the Graphical-to-Vocal Adapter. In the next sections we first describe an adaptation example obtained with our approach, and then we describe our solution for the Graphical-to-Vocal Adaptation process.

Adapting a graphical Web page into a menu-based vocal one poses a number of specific problems. We summarize them into three categories:

- **Content**: some contents of a graphical page translate poorly into the vocal modality. For example, a Web page could contain text not supported by the target Voice Browsers (e.g. Chinese alphabet symbols).

- **Structure**: although the logical description of a graphical page is a tree (good for menu navigation), the depth and the width are typically too large for vocal browsing. So there is a risk of generating a large number of nested menus with little content on the leaves.

- **Menu Items Naming**: it is crucial to find simple meaningful names for activating the menu items that, in some way, anticipate the corresponding node content.

The adaptation is performed after the reverse engineering phase, in which an analysis of the implementation in terms of all the HTML tags and associated CSS files is performed on the Web page considered in order to build the corresponding logical description. The transformation is carried out through various phases (see Figure 4):

- **Pre-Converter**: 1) performs the *Content Optimization* by removing from the graphical logical description the elements that badly reflect into vocal interfaces (e.g., images without alternative text); 2) performs the *Structure Optimization* by recognizing the page components and removing the
unnecessary one (e.g., grouping elements used for only formatting purposes); 3) performs the Calculate Cost computation (see next sections), whose results are used in the Menu Generator phase.

- **Menu Generator:** a new hierarchical structure is generated in order to allow navigating the interface through menus/submenus.
- **Graphical-to-Vocal Mapper:** the graphical interface elements are mapped onto vocal ones having the same semantic effect.

In the graphical-to-vocal mapper the elements of the graphical concrete presentations, as modified in the previous stages, and translates them into corresponding elements of the vocal concrete language that have similar effects.

The presentation title is used to build a suitable introduction sentence to use as the welcome message for the first time the user accesses the application. A vocal presentation may have a number of associated vocal commands. The transformation enables the following commands: previous, to go back to the last menu heard; main menu, to go back (at any moment) to the main menu, next/back: allows the user to navigate back and forth through different parts of the same section; start/stop, pauses the interface at any time and restarts it at the same dialogue position; reprompt, enable repeating the current dialogue; and exit, to close the vocal interface.

In the style-to-emphasis mappings we are constrained by the limited support of the vocal browsers. Regarding the output-only interactors, we provide the following processing. The text elements are directly mapped onto speech elements. If the text element defines a path to the content, we map this information directly onto the speech element. In the case of audio content, if the ALT attribute is set, we use it to give a description of the audio source. The audio element is mapped onto a prerecorded message vocal interactor for rendering. In the case of images, if the ALT attribute is specified in the source document, then we transform the element image into a speech element that renders the attribute. As in the case of an image, for videos we use the ALT attribute (when present) to vocally render its description, otherwise it is discarded. The audio elements are mapped onto the prerecorded message interactors; if the ALT attribute is available we use it to provide a general description of the audio resource. Data tables are simply mapped into vocal tables.

Regarding the control interactors we provide for the following processing. The text links are mapped onto the vocal link element. The name of the link is used as activation vocal sentence. The sentences that users must say to activate a link will be synthesized with higher pitch in order to let the user recognize it. The image link differs because in this case we want to communicate to the user what to say to activate the link, but we do not have an explicit title for the link. Currently we use the navigator ID as the corresponding vocal command, where the ID is that defined by the developers of the original Web pages. Graphical buttons are mapped onto vocal links, triggered by pronouncing their labels or, if a button is an image, by pronouncing the corresponding ID. If the button is used to submit the values of a form then it is mapped onto a vocal submit. This interactor is automatically enabled when the all the form input fields have been filled.

In the case of graphical interactors supporting selection, they are managed differently depending on whether they support single (radio button, list box, drop down list) or multiple selections (check box, list box), and consequently associated with the single or the multiple vocal selection interactors. In both single and multiple vocal selection, the labels of the graphical choice elements are used to build either the request or the list of accepted inputs.

The edit interactors allow users to enter pieces of information of various types. They raise the issue of the generation and use of grammars for vocal interfaces. For numerical input (which graphically can be provided by spin boxes, track bars and other similar techniques) we set a grammar able to recognize numbers. In case of textual input, we instead introduce two pre-generated grammars: a vocal grammar able to recognize short sentences composed of a well-defined list of words; and a DTMF grammar able to transform the keyboard inputs into words in a way similar to that used to write text messages in mobile phones.

### 6.2.2 UI Generator

The current standard for voice browsing implementation is VoiceXML. Thus, this was the first target implementation language from the vocal logical description. Since VoiceXML is also an XML-based
language, XSL Transformations (XSLT) seemed the most appropriate technology for implementing such transformation. This language provides a number of constructs for creating mappings among elements of two XML-based languages. However, such mappings are not trivial to create because both languages have a structure that provides constraints about where to locate an element. For example, in VoiceXML a vocal output is implemented differently depending on whether it occurs in a form or in a menu. This has been solved using the “XSLT modes”, an XSLT technique to identify which template to use in the transformation when this element occurs. More generally, this mechanism allows the transformation to change template to apply in the mapping depending on the current context.

SGRS\(^2\) grammars format has been used to specify possible inputs, since that it is what the VoiceXML specification supports. The grammars define the set of possible input that the vocal platform is able to recognize. In the generated implementations we use predefined VoiceXML grammars, when possible, such as date, number and currency in the numerical edit element mapping.

In some cases we generate inline grammars, which are grammars defined in the transformation process and directly inserted in the VoiceXML code. This is the case of the link element that can be activated both by vocal or DTMF command thanks to two expressly created grammars (using the parameters defined in the logical description).

Another solution is used in the mapping of the single vocal selection: in this case, we know ‘a priori’ the list of possible inputs, and we can use one VoiceXML <option> element for each input. This is equivalent to using a grammar able to recognize the entire list of user input.

The case of the multiple vocal selection, instead, introduces a problem: we do not know how many choices (in the predefined list) the user will make. In our solution the platform asks all the choices one-by-one and the user must answer yes/no to accept/reject each one. This solution may seem verbose but in practise there may be two situations: few choices, in this case the verbosity is negligible; many choices, in this case the verbosity is an advantage because it will reduce the mnemonic effort of the user.

As mentioned in previous sections, textual edit is similar to the visual concept of editable text box. It is very hard (if not impossible) to have a grammar able to recognize every kind of user input. We prefer to leave it up to the application developer to implement an external-grammar (or to find a pre-built one) that satisfies the possible input (case by case). For example, suppose that the platform asks the name of the user, the developer should build a grammar containing a dictionary of names to provide for recognition.

In the visual context we have some mechanisms to force the input of numbers in a certain range (e.g. a spin box); from the vocal point of view we resolve this problem by carrying out a check, before accepting the user input, using the conditional VoiceXML tags. If the number specified by the user is out of range, we refuse the input and re-prompt the request.

The control interactor command is mapped into a VoiceXML variable that contains the results of the execution of a script that must be defined at presentation level (which corresponds to a VoiceXML document in the implementation). The others control interactors: submit, goto, link, and menu are mapped into corresponding VoiceXML elements. VoiceXML links must be declared externally to the dialogue constructs and thus they are globally activable. Each possible menu choice is transformed into a <choice> VoiceXML element. In this way the grammar for recognizing the user input is automatically provided by the VoiceXML browser.

In the case of textual input the developer has the possibility to specify an appropriate grammar containing the rules for the acceptable inputs. Numerical input can be recognised by predefined grammars. It is also possible to generate in the code the check whether the input satisfies a given range.

In order to provide the possibility to consume web services and to maintain the application state between different presentations, the vocal application needs also a server-side support. Such support is generated starting from the MARIA Vocal CUI instance, through a Java procedure. The result is a set of Java Servlets that implement the runtime behaviour of the different models (data and user interface). The communication between the VoiceXML front-end and the Java is implemented through a set of jsp tags that define the

---

dynamic parts.

The generated servlets are the following:

- **UiManager**: this servlet maintains the state of the interactors properties. If contains methods for changing or updating the value of the specified interactor.
- **DataManager**: this servlet maintains the current state of the data model. The latter is represented by an ad-hoc generated class, which contains one element for each type declared into the data model definition.
- **Web Service Servlets**: for each web service (REST or SOAP based) declared into the ExternalFunction element, the generator analyses its definition and creates methods for accessing each operation. In addition, it creates a method that allows invoking a generic operation, specifying the operation name and the references to the elements containing the input parameters and where to put the invocation results. The references are given in a string format.

All these servlets provide their functionalities through method invocation (server-side) and through HTTP requests (client-side).

The server-side part of the interface behaviour is defined in a jsp that is imported into the VoiceXML front-end. It contains a set of invocations to the aforementioned servlets, which are derived from the EventHandler definition inside the MARI CUI model.

### 6.2.3 UI Runtime

The generated VoiceXML vocal interfaces have been tested with the VOXEO Voice Browser and have passed the validation test integrated in it. The applications have been used through VoIP access to the vocal server. A configuration tool was developed in order to change the parameters of the adaptation process and have a preview of the structure of the generated vocal application. This allows the possibility for developers of vocal applications to control the adaptation process and better tailor it to their specific needs.

### 6.2.4 Demo M18 example

In this section, we discuss the structure of the demo application for the vocal platform. The application is the vocal version of the one described in section 6.1.4. The application has the following features:

- It shares the ASFE-DL abstract model with the desktop and the mobile version.
- The transformation takes into account a set of preferences, in the form of adaptation rules. It is possible for instance to configure the maximum number of items included in a menu or to configure the voice type (male or female).
- The MARI Vocal CUI is transformed to a Voice XML application with a Java Servlet backend, which is able to consume the car rental web service.
- The generated application is able to take into account context information provided by the Serenoa platform such as user location and preferences.

Table 1 shows the application structure. The first diagram is an overview of the vocal menu structure for the whole application. The following ones describe more in detail the interaction of the user with the vocal application. The texts enclosed into the boxes represent the information that is vocally rendered by the application, while the arrow labels are the keywords that the user has to pronounce in order to navigate the application.

The application consists of seven presentations, which are devoted to the following tasks:

- **Login or Registration** for authenticating or registering a user
- **Main menu** for selecting one of the application functionality
- **Preferences** for editing the application preferences
- **User Info** to modify the user’s information
- **Search Info** for specifying which type of car the user wants to rent
- **Submit Request** for having a summary of the request and to submit it to the application server
- **Results** for listing the search result.
Menu Structure

Login or Registration

- Car rental
- Login or Registration
- Main Menu
  - Preferences
  - User Info
  - Search Info
  - Submit Request
  - Results

The following commands are globally defined for the application (the option is available for every menu, also if not explicitly represented):

- Back go back to the previous menu
- Main Menu go back to the main application menu
- Exit quit the application

Login or Registration

- Say login if you have an account, or register for creating a new one
- login
- Please say your username
- Please say your password
- Please spell your username
- Please spell your password
- Your username is ... your password is ... Do you confirm your request?

Main Menu
Submit request

Please choose one of the following options:
- Summary
- Submit request

You want a [colour] car at maximum 0.5 km from [address]. The car has to be provided with [options]. Do you want to proceed?

Yes

Results

Please choose one of the following options:
- Agency 1
- Agency 2
- ...

Car model: [model], Price: [price]

Agency n

This menu is rendered only if choose by agency is true

Select

You selected the following car model: [model], Price: [price]

Do you confirm?

Next / previous

Main menu

Car rental confirmed

Table 1: Vocal application structure
6.2.5 Maturity Level

The solution is not able to manage content such as Flash or Java applets. The support for Javascript is in progress. In general, the results depend on the original content. If the original content is developed according to the accessibility W3C guidelines for Web applications then the results are positive.

6.3 UsiXML RUIGE sub-module

UsiXML (User Interface eXtended Markup Language) is a declarative language that enhances the interface modeling by the addition of context driven capabilities. It is an XML-compliant markup language for UI description in multiple contexts of use (e.g. graphic, auditory or multimodal interfaces). It currently supports device independency, platform independency, and modality independency. UsiXML project is funded by ITEA 2 from the Eureka project 3674.

UsiXML counts with many complementary applications, such as: a task model editor, a domain model editor, a context model editor, an abstract user interface editor, and a concrete user interface model editor.

To refine the scope of UsiXML, it does not describe the low-level details of elements involved in the various modalities, such as operating system attributes, events, and primitives, it cannot be rendered nor executed by its own: it relies on an implementation in any third-party rendering engine, and it does not support all attributes, events, and primitives of all widgets existing in nearly all toolkits. Instead, it is intended to support a common subset of them.

UsiXML semantics is defined by means of a UML-compliant meta-model diagram, its syntax is defined by XML Schemas, and its stylistics is decomposed in different notations according to the underlying model, for instance the stylistic of the task model is based on the CTT (ConcurTaskTree) notation. All the project documentation is available online on its website.

In the context of Serenoa project and UsiXML language, we rely in the previous experience with UIDLs and edition tools, to identify possibilities for creating an additional RUIGE sub-module. As such, for this moment although there is no concrete module implemented, we believe that it could be feasible and interesting to integrate the experience with UsiXML to contribute in this sense.

Therefore, this section focuses in explaining the existing tools, transformations, architectural approaches and how they could be possibly integrate to Serenoa as an additional RUIGE sub-module.

The current transformation process for UsiXML involves 7 meta models, whose definition is specified with Ecore. These 7 meta models can be classified as Linear and Transversal.

The Linear transformations include meta models that belong to specific abstraction layers (according to the Cameleon Reference Framework). They are:

- **Task and Domain Meta Models**: describing the computational independent model of the architecture (i.e. Task and Concept layer);
- The **Abstract User Interface Meta Model**: describing the platform independent model of the architecture;
- The **Concrete User Interface Meta Model**: describing platform specific model of the architecture.

The Transversal meta models are in charge of relating models across different abstraction layers. They include:

- The **Context of Use Meta Model**: describing different context situations where sets of entities are executed;
- The **Mapping Meta Model**: relating model elements that belong to different abstraction layers in order to enable model traceability;
- The **Marking Meta Model**: describing how the transformation rules should be applied for a specific model.

Currently the UsiXML project counts with the following tools to support the creation and edition of models:

- **UsiEditor**: a Java Swing Editor to create and edit Task Models, Domain Models, AUI Models and CUI models, with a code editor, a technical code viewer (to create valid documents and to set proper attributes) and a more user-friendly viewer (with drag-and-drop functionalities to handle and arrange UI elements);
- **UsiTransform**: a web service to support the transformation of Task Models and Domain Models to AUI Models, and from AUI Models to CUI Models;
- **UsiRender**: a web service to render the CUI Models to FUI.

Given the definitions and background information presented above, we detail below current possibilities for the UsiXML RUIGE Module.

### 6.3.1 UI Transformer

Many different technologies have been proposed and can be applied to define and execute transformations between different languages. Examples of such languages include: XSLT, ATL and QVT. Currently, UsiXML implementation relies on ATL to execute transformations. Regarding the transformation of the ASFE-DL documents to UsiXML we can equally adopt the XSL technology, which is a W3C standard.

Besides transforming from ASFE-DL to UsiXML, further transformations are planned. The diagram illustrated in Figure 5 depicts the transformations and the relationships between the meta models of UsiXML. In this architectural approach, three transformations are defined:

- **The CIM to PIM transformation**: that takes as parameters a task model, a domain model, a context model, a mapping model, and a marking model; and then as a result the Abstract User Interface model is generated.
- **The PIM to PSM transformation**: that takes as parameter an abstract user interface model, a context model, a mapping model and a marking model. The result of the transformation is a concrete user interface model. The concrete user interface model employs XWT which is interpreted by the
Window Builder Eclipse plugin.

- The PSM to ISM transformation: it is inherent to the Window Builder Eclipse plugin. The Plugin generates the HTML and the SWING/SWT implementations from this specification.

All the models for UsiXML are defined in XMI (XML Model Interchange Language).

### 6.3.2 UI Generator

The current version of the UsiXML language support the generation of UI for different contexts, as such it must support the generation of UIs considering different technologies. For example, generating HTML and Java Swing versions.

In order to transform a CUI model into a FUI model, UsiXML has available a Java Swing application that relies on ATL and XML. It is capable of generating HTML5 and JavaSwing versions.

Besides this, UsiXML adopts also Ecore models, that jointly with XWT and an Eclipse plugin, support the generation of the UIs.

### 6.3.3 UI Runtime

For the execution at runtime, according to the technology in which the FUI was generated a specific approach must be applied. For instance, regarding the HTML document or the Java Swing version, the runtime must be respectively performed with the browser itself or the java compiler.

### 6.3.4 Maturity level

The UsiXML experience provides knowledge to support the definitions and execution of different transformation types. The editors proposed for the context of UsiXML, may also be in the future used in the context of Serenoa to support the creation and edition of models. These tools can also be useful to support the transformation between ASFE-DL and UsiXML.

The transformation between ASFE-DL and UsiXML needs to be specified. Thus, after achieving a stable version of the definitions the mapping between both can be refined. The same is also valid for the AAL.

Additional technologies can also be considered for generating FUI.

### 6.4 Avatar RUIGE sub-module

Regarding conversational applications based on avatars the adaptation problem is twofold. Firstly an adapted version of the ECA (Embodied Conversational Agent) has to be planned in function of the context. A typical example could be the graceful degradation of an avatar representation in devices with low video processing capabilities. Secondly a key issue is the definition of how to perform the synchronization/coordination of the verbal and nonverbal message [10] that could also be affected by the context conditions (e.g.: a more empathic attitude is advisable in case of difficult interaction conditions). In fact, the latter issue represents a hot research area, and a most noteworthy effort is that behind the SAIBA (Situation, Agent, Intention, Behaviour and Animation) framework [11] to define and standardize ECA verbal and gestural communication.

SAIBA framework establishes the general planning stages and knowledge structures that are involved in the creation of an embodied conversational system. In order to do that, three questions are proposed (see Figure 6):

- What is the communicative intent? That is the intent planning, where the goals of the communication are defined.
- What multimodal behaviours would carry out this intent? Or the behaviour specification for the planning intent. The gestures are selected from a collection of gestures (‘Gesticon’).
- How these multimodal behaviours could be realized? Finally, the specific parameters to control the avatar component in order to perform the actions (gestures, voice, etc.)
Next in this section we propose a SAIBA-like modular structure to define communication acts with verbal and nonverbal elements, and generate the suitable context-aware front-end for an avatar-based application.

6.4.1 UI Transformer

This XML Transformation component for avatars will deal with the core task of converting avatar descriptions in an abstract language to others in concrete languages. The concrete language, output of the UI Transformer, is equivalent to that FML proposed in the SAIBA framework. So it will describe the communicative intent, leaving the low-level details of the avatar implementation to the UI Generator.

XML transformation, querying and selection languages will be used, such as XSLT, XQuery and XPath. These tools may be re-used as well for further processing of XML data, such as the rules in AAL-DL, context descriptions and knowledge coming from CARFO.

Next actions for this module include devising a series of strategies to convert SERENOVA ASFE-DL AUI descriptions, which describe general service front-ends, into dialogue-driven interfaces. In the first approximation, the problem is very similar to the one posed by the AUI to vocal adapter described for MARI A in section 6.2.1: abstract repetition, grouping and hierarchical operators will be transformed to menu-based vocal interactions. Coordination actions with the responsible of the MARI A transformer will be performed so that the solutions remain compatible to a certain degree and to prevent overlapping work.

Additionally and exclusively for the avatar adapter, in this transformer we will augment this navigational abstract description of the application with specific mark-up signifying the communicational intent of the interaction of the avatar for each element in the navigational tree. This will be expressed in the terms standardized by FML [12] for communicative elements such as posture, gesturing and eye gazing of the avatar. FML allows us to formulate expressive descriptions of these in abstract terms that can be then realised in the following Generators and Runtime modules.

It is important to remark that we do not plan to fully implement the capabilities of FML in our transformer and our approach may not perfectly align FML and the SERENOVA CUI for the avatar. For example, FML covers gesture-text synchronization and fine descriptions for the elements that conform the gestures, but in the SERENOVA Avatar CUI we plan to describe that there are gestures of a certain class, while the details will be resolved further down the pipeline when more context information about the device, environment and user is incorporated into the adaptation.

6.4.2 UI Generator

The UI Generator is responsible for creating, from the concrete language, the final implementation of the verbal and non-verbal behaviour as well as the different execution profiles (e.g.: a web browser, a mobile, etc.) Several conversational resources as repositories of available gestures or a set of natural language AIML
d files could be referred.

Taking on where the transformer left, the Generator fleshes out the basic description of the dialogue structure and avatar specifics (gestures, postures, eye gazing elements) from a basic description to a FUI implementation that can be directly executed by the avatar runtimes. In simple terms, this would be analogous to transforming ‘the avatar greets the user and makes a gesture with the hand’ to ‘the avatar says

---

4 http://www.mindmakers.org/projects/saiba/wiki
5 http://www.alicebot.org/aiml.html
“Hello, master” at t=1.2 seconds while it performs a triple repeated wave gesture with the hand starting at t=0 seconds with a speed of 1 waving per second’.

In terms of the SAIBA framework, in this stage we incorporate the remaining elements of FML that haven’t been expressed in the transformer and then convert the results to our own BML-like language that will be used for the different runtimes. It is important that, while we are targeting to use the same FUI language for all our possible runtimes (see section 6.4.3), in this generator module different descriptions are produced in accordance to context: in the ‘simple’ example above, if we knew that our target runtime is of a reduced profile (e.g., a mobile phone with no generative Text-To-Speech capabilities but just recordings of the possible utterances by the avatar), we wouldn’t send the ‘Hello master’ TTS element but an identifier of the appropriate available pre-recorded sound.

For this adaptation to be performed, we will characterise our runtimes according to profiles, indicating their degree of compliance with several features of avatar engines, such as the ability to do gestures or Text-to-Speech on demand versus pre-recorded gestures and utterances, the possibility of changing the avatar’s appearance and so on. The FUI description will be specific for a certain kind of profile rather than for a specific runtime: two mobile runtimes may use the same FUI description if they share similar features and thus similar functionality.

This characterization of runtimes by ‘profiles’ will allow us to establish different levels of compliance with features of avatar systems that can crucially be ordered as progressively more compliant. This will be useful for example if we need to deliver an application to two devices simultaneously, as we would be able to find a common profile that both systems are compliant with.

6.4.3 UI Runtime

UI Generator will provide different execution profiles depending on the performance conditions. For example, an ActiveX component in case of the runtime environment is an Internet Explorer browser, or a sequence of images or videos for mobile devices. The UI Runtime is in charge of interpreting the FUI description of avatar actions and actually rendering an avatar that acts in accordance to them.

We are planning to release for SERENOYA at least two avatar runtimes with different profiles (for a description of the Runtime Profiles see section 6.4.2 above). The most complex will be the ‘desktop’ avatar, rendered by means of a 3D engine running as an ActiveX component inside a web page. This will be our highest standard of functionality as it will feature all possible options in an avatar system, such as quality-on-demand TTS (via SAPI), fine-tunable movements and facial expressions and fully customizable appearance of the avatar. This profile is using the commercial Haptek engine.

Haptek provides an environment for the design and implementation of 3D characters. There are mainly two ways for integrating Haptek avatars in your applications: through the Haptek player within a web browser and through an ActiveX control which could be embed in your applications using typical programming environments (i.e., Microsoft Visual Studio). The former integration method, which is the most suitable to the goals of the project, supports a JavaScript API that is used to control the avatar within the browser. The core channel of the communication is a function called SendText() that basically sends plain text to the Haptek player. As parameters of this function Haptek HyperText6 commands are used. The Haptek HyperText commands are able to play a new utterance, perform a specific gesture, or even load an entire new scene (loading new character, collection of gestures, etc.) Underlying this high level performance Haptek engine is controlled by a Switch Engine which is a kind of state machine. Then, a switch is composed by a collection of states and paths to transition from one state to another. In turn each state has parameters coupled with associated intensities and durations. Finally, the parameters represent a low-level scalar property of the avatar, such as the rotation of the wrist or the amplitude of the smiling expression.

In addition to the engine, a proprietary wrapper by TID is used to translate between the FUI and the internal switch engine of the Haptek ECA.

The other avatar runtime that SERENOYA will feature will be internally developed by TID and will be a lower-specified one, but able to run in less powerful devices (i.e., mobile phones, tablets). The

implementation will be based upon HTML5 technologies and thus will be also available for desktop and will be developed with extensibility in mind.

This runtime is still in the design phase, but we plan to take advantage of CSS3 animations and the HTML5 Canvas element to program an avatar engine that is able to minimally display text with utterances to the user and perform basic gestures. Existing efforts and helper libraries such as Sencha Animator\(^7\) will be explored and used if suitable.

An additional avatar runtime profile may be served by a video-based one, using HTML5 video. Video chunks for predefined dialogue fragments could be produced, and put together in one single video file or else in multitude of video files accessible to the client application. The playback of these fragments may be done programmatically using the capabilities in the `<video>` tag of HTML5. More complex techniques are possible with the manipulation of the HTML5 Canvas defined by the `<video>` tag. For example, chroma key\(^8\) for transparent backgrounds, colour tracking\(^9\) and shapes and appearance of other HTML objects inside the video frame and others. The usage of third party media libraries, such as popcorn.js\(^{10}\) or jPlayer\(^{11}\) for jQuery allows for easy integration of advanced functionality in the video controls, i.e., launching custom events for synchronization of activities in the web applications.

Other possible future development line could be geared towards studying the possibility of incorporating real time, in-browser 3D rendering for fluid and expressive animations\(^{12}\). However, due to the greatly increased demand in incorporating 3D rendering, this may be out of scope for SERENOA.

Text-to-Speech is still the main missing link towards a fully featured avatar engine entirely in HTML5. There are some on-going efforts that we will monitor during the remainder of the project such as speak.js by Syntensity\(^{13}\) and an API for Chrome\(^{14}\).

### 6.4.4 Example

The demo example planned to M18 review does not include a conversational avatar. So, this section is going to describe, taking into account the scope of this deliverable, an example based on the prototype TID is responsible of. In short, the aim of the abovementioned prototype is to help patients in the management of their health information. Further details about the service are available in “D5.2.1 Application Prototypes (Requirements and Design)”.  

In order to show the adaptation process in the RUIGE a use case scenario is proposed:

“Jane is at home checking her last blood test results, using eHealth desktop application and taking advantage of the avatar-based assistant. She receives a call and she has to leave. While she is on the bus, she decides to continue reviewing the information, but this time using her smartphone. Unfortunately the noise conditions are terrible inside the bus.”

In accordance with this scenario the adaptation rules that could be applied are:

- **R1**: If the device is a mobile, then the avatar will be rendered as a sequence of images.
- **R2**: If the noise conditions get worse, then the avatar stops speaking and only communicates by text.
- **R3**: If the performance of the application is getting worse, the avatar changes her attitude to a more empathic one.

Then, as it is shown in Figure 7, the adaptation process performed by SERENOA framework would be as follows: Jane starts using the desktop application, at home. When she is outside and she tries to access to the eHealth assistant using her smartphone (R1), the Runtime module is in charge to degrade the avatar

---

\(^7\) [http://www.sencha.com/products/animator](http://www.sencha.com/products/animator)

\(^8\) [http://www.tinkernut.com/2011/03/22/awesome-html5-video-trick](http://www.tinkernut.com/2011/03/22/awesome-html5-video-trick)

\(^9\) [http://anavallasuiza.com/popcorn](http://anavallasuiza.com/popcorn)

\(^10\) [http://popcornjs.org](http://popcornjs.org)

\(^11\) [http://jplayer.org](http://jplayer.org)

\(^12\) [http://stickmanventures.com/labs/demo/webgl-threejs-morph-target](http://stickmanventures.com/labs/demo/webgl-threejs-morph-target)

\(^13\) [http://syntensity.com/static/espeak.html](http://syntensity.com/static/espeak.html)

presentation and present the sequence-of-images version. Afterwards the Context Manager warns about a high level of noise (R2), the avatar was speaking her answers but now it has no sense. So, again Runtime module is responsible to automatically stop the avatar voice feature. Finally, and due to the hard conditions for the interaction inside the bus, a more empathic attitude of the avatar (R3) is advisable in order to control the frustration of the user. In this case, the Generator module changes the avatar’s mood trying to make friendlier the interaction.

![Figure 7. Adaptation example for TID scenario](image)

6.4.5 Maturity Level

Up to now most of the efforts have been put in the specification of the architecture for supporting the Avatar engine flow and the needs for FML-like concrete language. UI Generator has been also partially developed making possible the translation between a preliminary FML-like language and the final language for the Haptek runtime. Since the avatar component is not due to the M18 demo, in the following months we plan to deal with the following tasks:

- Integration of XSLT to transform the abstract UI definition in ASFE-DL to FML-like, taking into account the AAL-DL rules.
- Evolution of the UI Generator in order to support the final version of the FML-like language. It also should generate the rest of the execution profiles (as it was presented in the abovementioned demo example).
- Development of the alternative avatar engines (i.e., based on sequence of images, or videos).

6.5 Leonardi RUIGE sub-module

The Leonardi RUIGE sub-module is the environment that is used for implementing W4’s e-commerce scenario. It is based on the Leonardi framework, a J2EE, model based environment that automatically produces a concrete user interface by interpreting the model on the fly. Leonardi is distributed under the open source GPL V2 license and is the property of W4.

The Leonardi framework includes two main modules:

- **The Leonardi Composer**, which is a graphical environment that allows developers to create the “business model”. The resulting model is a set of hierarchical XML files.
- **The Leonardi Engine** (aka the W4 Engine), a Java based program that interprets the model to create the screens dynamically, feeding them with the configured data source, as the user interacts with the
system. The role of this component is to transform dynamically the AUI into a CUI.

A Leonardi model is a logical representation of the business world. It includes a description of the data structure of business objects, using classes, typed attributes and relationships. The model also includes, on top of the data structure, a GUI layer that is directly linked to the modelled objects and allows automating the production of user screens. This layer consists of an AUI with:

- The navigation graph, which describes how to access screens and the actions that are available for end-users, with optional directives to specialized Java code for implementing the business logic
- The types of views used to visualize data (tables, trees, forms, maps, etc.) and associated layouts
- A set of resources and graphical components, such as icons, images, fonts, skins and strings

By using Java, the developer can always specialize the UI appearance and behaviour offered by default by the framework, or implement rules that are too sophisticated to be expressed simply by configuration.

The use of Leonardi provides a double level of abstraction to the user by offering connectors, that allow connection to physical data and display managers, which allow UI rendering on a variety of different presentation environments and platforms.

With Leonardi, there are different ways of performing adaptation work:

- Automatically, by relying on the Leonardi engine that performs automatically UI adaptation based on the context of use (i.e. platform, language, screen size etc.)
- By configuration of the model, by setting specific resources for the different contexts of use

6.5.1 UI Transformer

At this stage, the exact implementation of the UI transformer for the W4 engine still needs to be decided.

It is very likely that applications using the AAL-DL will be converted into XML-based Leonardi models by using tools such as XSLT or XPath.

6.5.2 UI Generator

Leonardi is able to produce UIs for different contexts, by relying on different display managers. The UI generator is enabled through specialized components called display managers, that produce graphical screens based on a variety of different rendering technologies (e.g., Swing, Ajax/DHTML, HTML5/jQuery, Android or iOS).

Therefore, the same application is designed once for all but can be used on different platforms with no extra work. Of course, a finer grain adaptation work is always possible.

6.5.3 UI Runtime

An executable Leonardi application consists of the following components:

- The model itself, which is a logical, XML-based representation of the functional world, with the structure of the data and the definition of GUI components to be used
- Custom resources, including strings, fonts, colours, dictionaries, properties, skins, images, icons...
- The Leonardi engine itself, which is invoked as soon as a user connects to the application, and that dynamically produces the screens as they are requested by the end-user

6.5.4 Example

The e-Commerce scenario is used to illustrate how the W4 engine may consider different rules to adapt the UI based on different contexts of use, typically defined by user profiles (language, disabilities...) and platforms (desktops, smartphones or tablets).

Below are two screenshots showing how a few rules expressed in the SERENO AAL-DL will adapt the UI based on the context in the customer web portal. Here, the context is defined by setting the accessibility mode to on or off.
Some examples of how a same application is adapting the UI on either a desktop or a smartphone, are shown in the next figures. The same concepts will be used on the e-Commerce scenario for the employee portal.
6.5.5 Maturity level
In inception stage.

6.6 Warehouse RUIGE sub-module
This section describes the current state of selecting the appropriate runtime engine for the intelligent picking prototype already presented in D5.2.1 ‘Application prototypes’. The selection of an appropriate runtime engine depends on the fitting of the currently developed engines within Serenoa with the already well defined scenario and requirements of the warehouse picking process.

As presented in D5.2.1 the warehouse prototype will be built upon these wearable and mobile technologies, smart environments and the data from ERP systems. The goal is a UI which can be adapted to this context applied to a logistics task.

6.6.1 UI Transformer
As mentioned above, the intelligent picking prototype will deal with different devices like:
   a) a wearable computer with head mounted display and
   b) a smart phone or touch pad

One technical solution is to base both tapes of devices on Web-browsers. This would provide a high level of platform independency but limit the capabilities of the devices to those of the browser technology. If this solution is chosen nevertheless, the UI Transformer module should perform a conversion from ASFE-DL to HTML using an XSL transformation.

6.6.2 UI Generator
The technical solution based on Web-browsers also requires the generation of additional objects like CSS and JavaScript. Current technologies which realize user interfaces based on HTML also require additional
libraries like JQuery. This decision also affects the downward compatibility as some older browser might not be able to support those libraries. Additionally, as one of the devices will be a mobile one, it needs to be considered if mobile versions of those libraries should be used, as is the case for JQuery mobile.

The UI Generator modules currently developed within Serenoa, like the MARIA app generator, will be the preferred choice to realize the generation of UIs.

### 6.6.3 UI Runtime

The UI Runtime needs to interact with the backend system from which the data of the ERP system is retrieved. Meanwhile, the consumption of such data has been simplified by providing RESTful interfaces like the OData service. Besides this, a feedback from the environment on the current state should also support the task of picking. The context server should receive information from devices which recognize the activity (e.g. Microsoft Kinect, Laser Range Finders, and Accelerometers) or the location (GPS) of the user. A Javascript module would then need to query this information. Finally, advanced interaction modes like gestures and voice should be supported.

Some of the UI Runtime modules currently developed within Serenoa support a vocal interface, like e.g. the generated VoiceXML vocal interfaces of the Maria RUIGE sub-module. Again, the appropriateness of these solutions needs to be investigated.

### 6.6.4 Example

In the current state of the prototype, the picker stands directly in front of the shelf for the picking, but, in future, it will be leveraged with the advanced navigational and interactional capabilities by attaching RFID tags with the shelves and their boxes, so that the picker could be guided with the shortest path from his standing position to the desired shelf in large warehouses in order to save time.

The current client is GUI based with the fixed font size. However, the users having a weak eye sight want the GUI with bigger fonts; hence in future, the GUI will be developed in a way so that it could adapt according to the preferences of the user.

In order to complete an order task, the user currently uses two buttons (forward, reverse) input device in order to navigate between the steps. However, two more interactional modalities will be developed for the future version, namely gesture and speech-based. In noisy environments, gestures based interaction is definitely advantageous.

Concerning the adaptation to the platform, we are currently targeting three devices, a stationary (desktop) PC, the above mentioned wearable computer and a common mobile device (Smartphone or iPad). The GUIs which hold the information about the list of orders to be processed by the picker are shown in Figure 12.
The third platform, the wearable device has a UI which is designed for an efficient processing of the already assigned orders.

The graphical user interface for the wearable computer, as shown in Figure 13, remains with a focus on simplicity. The important aspects of adaptation are the user profile, e.g. images of the items are desired, the environment; the shelves have a colour code or the shelves have intelligent sensors and the features of the platform; a speech and a gesture interface are available.

The adaptation of the UI to the user profile due to the preferred language, the level of expertise and the knowledge about the location are shown in Figure 13.

The user interface adapts to the user profile. In Figure 14, two alternatives are shown:

a) The list of orders adapts to the preferred language and to the level of expertise, adding additional help buttons.

b) The UI offers an additional navigation functionality as the user does not know the location.

For the navigation with a wearable computer a UI a real-time navigation, as known from automobiles can be developed.
6.6.5 Maturity level

The preferred choice for the runtime engine for the intelligent picking prototype is to use the currently developed RUIGE sub-modules within Serenoa. Thus, the maturity level can be derived directly from the sections on MMW-4S, UsiXML and Maria.
7 Conclusions

7.1 Summary

This deliverable exposes the Serenoa runtime architecture (RUIGE) that will be in charge of the generation, deployment and runtime execution of Service Front Ends specified through abstract languages, such as ASFE-DL and AAL-DL. This module plays a key role within the Serenoa framework, since it is the final executor responsible for carrying out the creation and adaptation of SFEs for different target platforms. In order to accomplish such an ambitious goal it has to gather together information coming from different modules of the Serenoa framework: Context Manager, Authoring Tools, Adaptation Engine, etc.

Taking into account the great variety of target platforms that might be considered in the creation of SFEs, and conscious of the limited resources available during the lifecycle of Serenoa project, the consortium has decided to provide a modular architecture for RUIGE. The main benefit of the proposed design is that we envisage Serenoa’s RUIGE as a reference runtime platform for the creation of context-aware SFEs. Following this idea, RUIGE establish a set of minimum requirements for a piece of software to become a RUIGE module. In the current stage of the project those requirements are not very strict, since the status of the rest of the components (both languages and modules) is not definitive. However, they will be refined and sharpened by providing specific interfaces during the evolution of the project.

Last but not least, the Serenoa framework provides a set of RUIGE implementations intended to cover a great amount of target devices and coping with different modalities (graphical and vocal). In what regards to graphical UIs, the first version of an open source RUIGE module able to render mobile Web 2.0 applications (HTML5, SVG, CSS, Javascript, AJAX) has been released. Regarding vocal UIs, a specific RUIGE module producing VoiceXML has been developed. Additional RUIGE sub-modules under development have also been described and they will be commented in more depth in the R2 of this document (D4.1.2).

7.2 Future Work

So far, some general aspects to be considered during the following months have been identified for those sub-modules which are in a more advanced status:

- Support migratory UIs, which are UIs able to adapt to different interaction resources and preserve task continuity across various devices. For this, the concept of session and formats/protocols to describe user session needs to be defined. This will prove interoperability among RUIGE sub-modules. The avatar demo itself requires support for session exchange/migratory UIs.
- Support for totally or partially offline applications, as described for instance in HTML5.

In what regards to the mobile Web RUIGE module (MMW-4S):

- Improve the integration with higher abstraction layers to define UIs (mainly, the abstract) by evolving the UI Transformer.
- Improve MMW’s dialog model to consider advanced context-aware adaptations.
- Support better HTML5 application generation.

Some future steps to be considered in the evolution of MARI A RUIGE module:

- Improve support for web services access using both WSDL and WADL
- Improve support for script use in vocal implementations

Regarding the Avatar RUIGE module:

- Define and integrate transformation capabilities in order to obtain a CUI description from ASFE-DL.
- Update the Avatar Generator in order to support other execution profiles beyond Haptek interface.
- Develop the new avatar-based runtime for mobiles and similar devices which don’t support Haptek runtime. Due to these limitations this basic runtime will be composed by sequence of images or videos.
8 References


Acknowledgements

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