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

**Project no. FP7 – ICT – 258030**

# Deliverable D.4.1.2
# Runtime UI Generation Engine (R2)


**Due date of deliverable: 28/02/2013**

**Actual submission to EC date: 25/02/2013**

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



| Document Information | |
|---|---|
| **Lead Contractor** | |
| **Editor** | Javier Rodríguez Escolar<br>Cristina González Cachón<br>Ignacio Marín Prendes |
| **Revision** | 25/02/2013 |
| **Reviewer 1** | **TID** |
| **Reviewer 2** | |
| **Approved by** | |
| **Project Officer** | Mr. Michel Lacroix |

| Contributors | |
|---|---|
| **Partner** | **Contributors** |
| **CTIC** | **Javier Rodríguez Escolar**<br>**Cristina González Cachón**<br>**Ignacio Marín Prendes** |
| **CNR-ISTI** | **Marco Manca**<br>**Fabio Paternò**<br>**Lucio Davide Spano** |
| **UCL** | **Vivian Genaro Motti**<br>**Pascal Beaujeant**<br>**Nesrine Mezhoudi**<br>**Jean Vanderdonckt** |
| **SAP** | **Jörg Rett** |
| **TID** | **Javier Caminero, Mari Carmen Rodríguez Gancedo** |
| **W4** | **Jean-Loup Comeliau** |

| Changes | | | |
|---|---|---|---|
| **Version** | **Date** | **Author** | **Comments** |
| 1 | 09/01/2013 | CTIC | Table of Contents |
| 2 | 25/01/2013 | CTIC | First version of the deliverable |
| 3 | 08/02/2013 | UCL, SAP, CNR-ISTI, TID, W4 | Contributions from the partners |
| 4 | 13/02/2013 | CTIC | Second version of the deliverable |
| 5 | 20/02/2013 | TID | Review of the document |
| 6 | 25/02/2013 | CTIC | Final version of the deliverable |

# Executive Summary

This deliverable describes the current state of the Runtime User Interface Generation Engine (RUIGE) in a concise manner. RUIGE is intended to generate multi-platform applications from a common abstract description defined in *ASFE-DL*. Moreover, the generated applications are adapted to the delivery context in order to maximize the user experience. To carry out such adaptations at runtime, RUIGE needs to interoperate with other modules of the Serenoa platform, mainly with the Adaptation Engine.

This document provides an update of the previous version of the deliverable (D4.1.1), so it focuses on the progresses made over it. RUIGE has been conceived as a modular system in order to facilitate the creation of different sub-modules specialized in the runtime generation for different platforms or modalities. This deliverable describes the current state of each of the RUIGE sub-modules developed as part of the Serenoa project.

# Table of Contents

# Table of Figures

# 1 Introduction

## 1.1 Objectives

The Runtime User Interface Generation Engine (RUIGE) is in charge of the generation of context-tailored UIs from different abstract descriptions defined in the project. This document aims at describing the architecture of RUIGE and its role within the Serenoa Framework. It intends not only to provide a clear overview of the module, but also to highlight its evolution from the release of the previous version of this deliverable in M18 (D4.1.1).

As introduced in previous deliverables, RUIGE has been designed as a modular system, thus facilitating the creation of different modules, each one intended to generate UIs for a specific set of target devices, platforms or modalities. This document offers a description of all the RUIGE modules developed as part of the Serenoa project, detailing the following aspects: their general purpose, the abstract languages they support as an input, the target platforms they cover, how they interact with the rest of the Serenoa modules, their current state and how they have evolved from M18, the expected state by the end of the project and they availability for the developers community (e.g. open-source license).

## 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 them.
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

- *D1.2.2 Architectural specifications (R2)*
- *D1.3.2 Serenoa Roadmap* describing next steps in the evolution of the Serenoa framework.
- *D3.2.2 ASFE-DL: Semantics, Syntaxes and Stylistics* (R2) describe the abstract language that is used by the runtime.
- *D3.3.1 AAL-DL Semantics, Syntaxes and Stylistics* describes the adaptation rules languages used by the runtime.
- *D4.1.1 Runtime UI Generation Engine (R1)*
- *D4.3.1 Adaptation Engine* describes some adaptations needed on the runtime.
- *D5.1.1 Serenoa Framework (R1)*
- *D5.2.2 Application Prototypes (R1)*

## 1.4 Organization of this document

Chapter 1 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 specifies the role of the runtime inside the whole project and highlights the relation established with the Adaptation Engine. The mission of chapter 4 is to describe the current state of the different runtime sub-modules. To finish the deliverable, Chapter 5 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

RUIGE is one of the most illustrative modules of the Serenoa Framework. It is a modular engine responsible for the creation of final applications adapted to the context starting from a specification of the UI at the Abstract UI level by means of *ASFE-DL*.

RUIGE is composed of various sub-modules, each of them in charge of generating different outputs, such as mobile interfaces, vocal systems, avatar engines, etc. New sub-modules might be added to the architecture in order to support additional interaction modes and target platforms for the same application definition. The existing sub-modules were explained in the last version of this deliverable, D4.1.1, and the changes carried out over them will be explained in Chapter 3.

Each RUIGE sub-module follows several stages in order to generate the final application based on the abstract definition. Each stage is associated to a specific module:

- **Transformer**: component in charge of the process of converting the abstract language (*ASFE-DL*) into a generic language for the representation of the concrete user interface of each sub-module. This language will be the input to the next stage.
- **Generator**: this module analyses the output of the transformer and generates the executable code, which will be used at runtime.
- **Runtime**: this component is intended to provide support for the execution of applications. Note that some sub-modules may require both deployment and execution stages. This component is in charge of deploying the application, if it has not been done before by the generator. For instance, uploading web applications in a web server, deploying an application in a servlet container or even installing (or running) a file in the user device. After the deploy stage, the application is executed by an execution platform; for instance, a web browser.

This modular architecture is shown in Figure 1. The RUIGE modules establish communication with other modules of the Serenoa framework, such as the Adaptation Engine or the back-end services.

By definition, the RUIGE module receives the application definition by means of:

- A UI specification in *ASFE-DL* created by the Authoring Tools (see D4.5.1).
- A set of adaptation actions coming from the Adaptation Engine (see D4.3.1). To get the final application, some adaptation rules could be applied expressed in *AAL-DL*. They will guide the adaptation process at design time or even at runtime level of both user interfaces and resources referenced by the UI.
- Additional information coming from the back-end system.

The output of the RUIGE module should include the final code of the application and its corresponding runtime support. This means a set of libraries to carry out the adaptations while the user interacts with the application.

Figure 1 shows the interaction between the RUIGE module and the rest of components of the Serenoa framework. Basically, the authoring tool creates the abstract language for the UI definition by means of *ASFE-DL* and the adaptation rules expressed in *AAL-DL*. To facilitate the recovery of both files by any of the Serenoa components, they are stored in a Service Repository. The Adaptation Engine will retrieve them in order to accomplish its goal. To make the adaptations, RUIGE will query the Adaptation Engine identifying the UI it should adapt in order to get the corresponding actions to perform the adaptation in the appropriate stage.

RUIGE allows application generation at least for rendering both desktop and mobile platforms and for vocal interfaces.

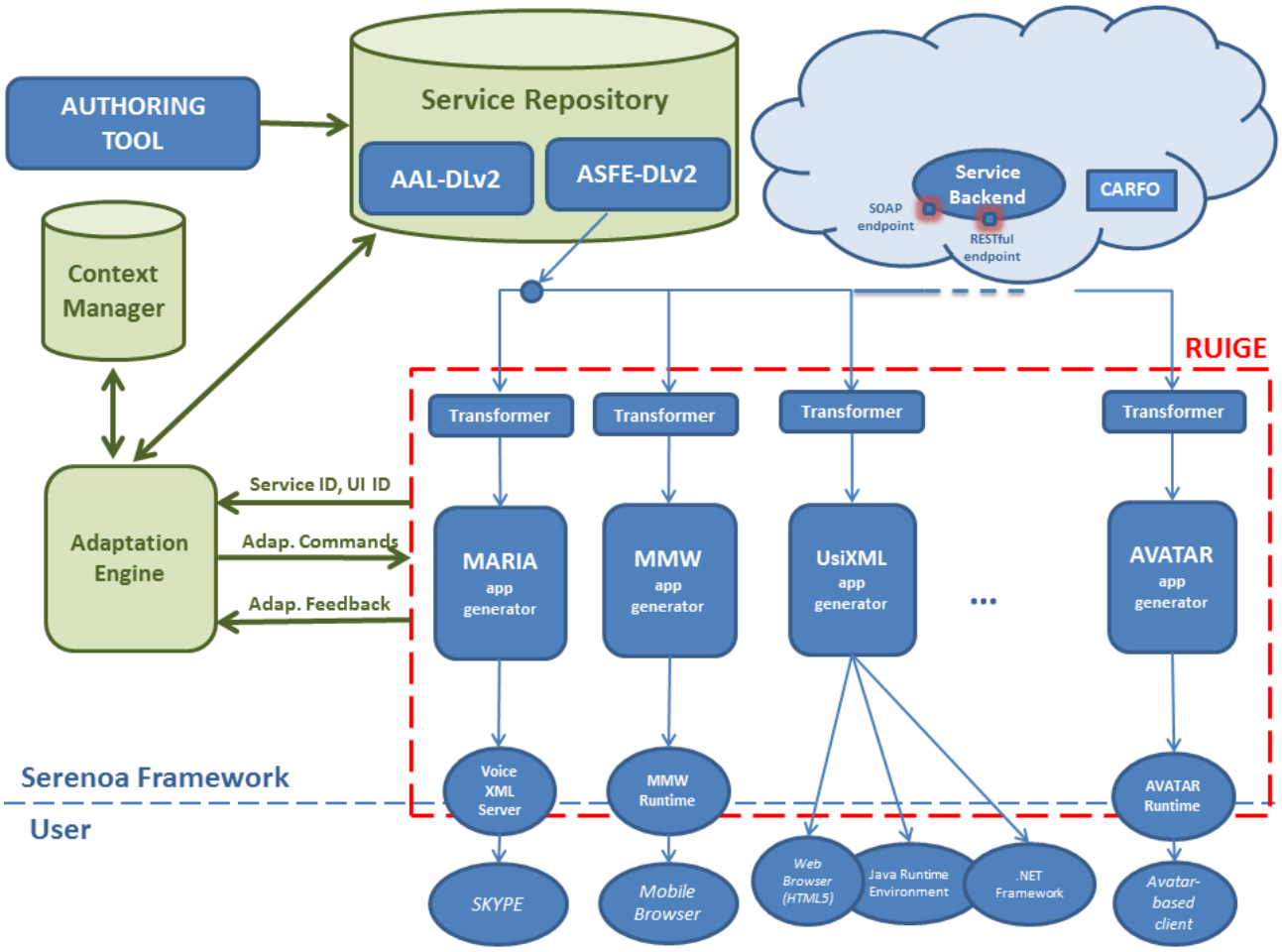


Figure 1. Serenoa Architecture

# 3 The role of RUIGE in the Serenoa Framework

The functionality of RUIGE is made possible by other modules of the Serenoa architecture, such as the CARFO Ontology and the Context Manager. In D4.1.1, the relations between RUIGE and these sub-modules were explained. However, from now on, the Adaptation Engine will assume the responsibility of gathering all the information from the rest of the modules in order to act as an adaptation manager. Consequently, RUIGE will just need to interact with the Adaptation Engine, avoiding the establishment of direct communications with CARFO or the Context Manager. In this chapter, we focus on the interaction between RUIGE and Adaptation Engine, while the interactions between Adaptation Engine and the rest of the modules are explained in D4.3.1.

## 3.1 Relation between the Adaptation Engine and RUIGE

The Adaptation Engine (AE) is the module in charge of collecting the high-level descriptions of the application (UI definition and adaptation rules), the information of the context in which it is being used and the adaptation knowledge in order to produce the adaptation commands. Some of these commands might be used at runtime to adapt the application to the context, while others might be employed at transformation time. The adaptation commands guide the optimization process needed for the selection of the proper adaptation logic and are expressed by means of a reduced version of the *AAL-DL* language.

The relation between both modules is based on the adaptation process: the Adaptation Engine calculates the adaptation logic that has to be applied by the RUIGE to perform the optimal adaptation for a given context. Each internal component of a RUIGE sub-module has a particular relation with the Adaptation Engine.

- **Transformer**: component in charge of transforming from the abstract level to the concrete one of each runtime. The input of this module will be formed not only of the *ASFE-DL* but also of a set of adaptation rules that the AE provides. Some of these rules have direct impact in the transformation process while others might be applied at generation time or at runtime.
- **Generator**: it takes the concrete description from the adapter and transforms it to the final representation. As the transformer does, the generator may consider some of the adaptation commands provided by the AE.
- **Runtime**: this module is responsible of the execution of the application. A bidirectional communication is established between both modules. On the one hand, RUIGE runtime must react to the adaptation commands produced by the AE in order to provide support for context variations. On the other one, the RUIGE module may report the feedback of the users to the Adaption rules so it can be taken into account in future adaptations.

Note that the bidirectional communication between RUIGE and AE might occur in two different modes:

- RUIGE may query the AE in order to know how to adapt a specific UI in a given context.
- AE may send adaptation commands to be executed by RUIGE as a result of any context variation. RUIGE must be ready to react to those commands at any time.

# 4 Evolution of the RUIGE sub-modules

## 4.1 Mobile Web RUIGE sub-module: MMW-4S

MMW-4S (*MyMobileWeb for Serenoa*) is an open-source module[1] fully compliant with RUIGE. It is aimed at generating mobile Web applications for multiple platforms and devices. Moreover, MMW-4S is able to interoperate with the Adaptation Engine Module in order to adapt the generated UIs accordingly, thus maximizing the user experience. Figure 2 shows a simplified version of Figure 1, where the different sub-modules of MMW-4S are highlighted in blue colour. The following sub-sections explain the current state of each of these sub-modules, how they have been evolved from M18 and how they will be evolved until the end of the project.

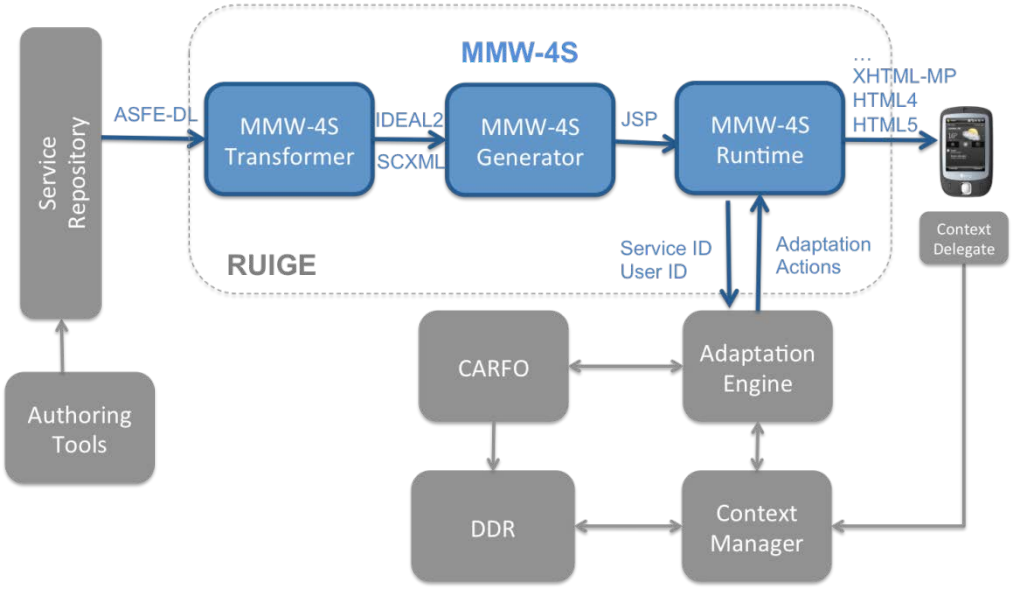


Figure 2. MMW-4S Architecture

### 4.1.1 MMW-4S Transformer

MMW-4S transformer has been implemented as an XSL transformation sheet (*XSLT*) due to the fact that both the input language (*ASFE-DL*) and the output languages (*IDEAL2* and *SCXML*) are *XML*-based. The current version of MMW-4S Transformer supports the last version of *ASFE-DL* available at the time of writing (see D3.2.2 for more details), the last version of *IDEAL2* [1], the draft version of *SCXML* published on 16 May 2008 [2].

The last changes carried out in the Transformer module were intended to provide support for the last specification of *ASFE-DL*. The last major changes introduced in the *ASFE-DL* language which impacts the MMW-4S Transformer module are:

- The introduction of the Back class, a new type of connection. A connection in *ASFE-DL* declares that from a given *AbstractInteractionUnit* (*AIU*) is possible to reach another *AIU*. A Back instance connects the current *AIU* with the previously visited one. If the current *AIU* can be reached from more than one *AIU*, the target of the back connection is different according to the user's navigation. For instance, suppose that the current *AIU* is *C* and it can be reached from both the *A* and *B* AIUs. If the user has visited *C* from *A*, then the back connection target is *A*, while if the user has visited *C* from *B*, the back connection target is *B*. In order to provide support for this functionality, the Transformer produces:
    - An *<a>* element of *IDEAL2* with a *mymw:back* role annotation, which indicates that this link will be rendered as a back navigation component (note that it might be rendered in a

[1] All MMW-4S code has been released as open-source under the LGPLv3 license and it is available at the Morfeo Forge: https://svn.forge.morfeo-project.org/serenoa/trunk/

different manner depending on the target platform):

```
<a id="lback" resourceid="back" role="mymw:back">Back</a>
```

- An *SCXML* custom action called back within the MyMobileWeb namespace. This action[2] is internally managed by MMW-4S in order to allow the user to navigate to previous states avoiding to define explicit code in the flow definition:

```
<mymw:back />
```

- The addition of a role attribute to the AIUs, in order to allow the designer to specify a semantic tag for them. The semantics of the tag are completely up to the interface designer, therefore we do not assign a unique way for interpreting the values. MMW-4S Transformer uses these annotations to create groups of presentations (*Operation Presentations* or *OPs*) related to the same task.

### 4.1.2 MMW-4S Generator

The UI Generator of MMW-4S is the part of the platform in charge of generating *JSPs* (JavaServer Pages) from *IDEAL2* documents. Each UI defined in *IDEAL2* implies the generation of a specific JSP for each markup language in the market (*WML*, *XHTML Basic*/*Mobile Profile*, *HTML4* and *HTML5*). Note that the generated *JSPs* contain server-side code that will be executed at runtime (see 4.1.3). Minor changes have been performed in this module in order to improve the generation of the set of JSPs that guide the execution of the *HTLM5* Rendering Engine at runtime.

### 4.1.3 MMW-4S Runtime

This module is intended to provide the most appropriate Final User Interfaces at runtime. Whenever a client device requests a specific Web page to the server, the corresponding *JSP* (see 4.1.2) will be executed. Note that *JSPs* are translated into *Java Servlets* at runtime. Consequently, each web page access will result in the invocation of server-side code containing the logic that generates the adapted Web content dynamically.

In order to obtain the most appropriate Web contents for each delivery context, MMW-4S runtime libraries query the Adaptation Engine and render the final UI in accordance to the resulting adaptation commands. The major changes carried out in this module from M18 are:

- Improvements in the Rendering Engine to support the generation of more *IDEAL2* UI components by means of *JQMobile*. The current state of the implementation supports the following subset of *IDEAL2* elements: *ui, body, section, div, menu, a, input, inputDate, secret, submit, select, select1, item, label, value, submit, footer image, media, map, placemark.*
- Integration with the Adaptation Engine to carry out advanced adaptation actions.

## 4.2 MARIA RUIGE sub-module

The MARIA RUIGE sub-module is composed of a transformer, a generator and a runtime according to the RUIGE architecture. Figure 3 summarizes the architecture highlighting the MARIA sub-modules. This RUIGE module is able to subscribe to the Adaptation Engine and to receive from this external module the adaptation actions which describe how to adapt the interface to the context of use.

[2] http://forge.morfeo-project.org/wiki_en/index.php/SCXML_Flow_Engine_4_0#.3Cmymw:back.3E

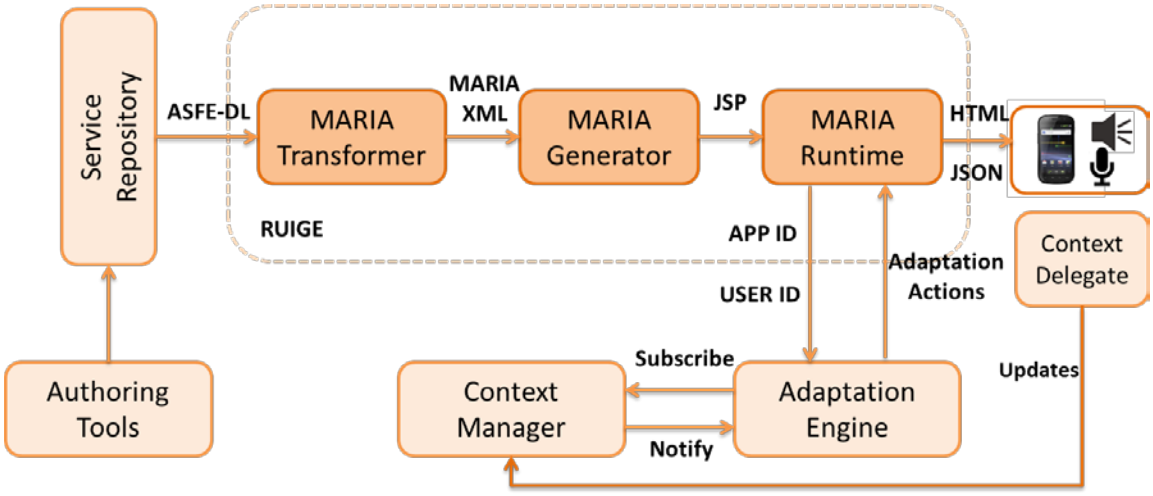


Figure 3. MARIA Architecture

### 4.2.1 MARIA transformer

The MARIA transformer is implemented through an *XSLT* stylesheet and aims to transform the Serenoa abstract description (*ASFE-DL*) to MARIA notation. We decided to exploit *XSLT* transformation because *ASFE-DL* and MARIA are both *XML*-based languages. Various *XSLT* transformations from MARIA *AUI* to various concrete languages have been developed for different platforms. In the last year we have particularly focused on the transformation for the MARIA Multimodal concrete description that will be the input for the generator module. This concrete language describes in an implementation language independent way the possibility of obtaining user interfaces that combine, in different ways, graphical and vocal interactions using CARE (Complementarity, Assignment, Redundancy, Equivalence) properties associated with the various user interface elements.

### 4.2.2 MARIA Generator

The MARIA Multimodal Generator is also implemented with an *XSLT* stylesheet and produces JSP implementations combining graphical vocal interfaces in Web environments.

For this purpose, we developed a *Chrome* extension (for desktop devices) and an *Android* app (for mobile devices) able to interpret the *HTML* generated and which call Google APIs for Automatic Speech Recognition (ASR) and Text-To-Speech (TTS) synthesis. In order to understand when to activate the functions that access the vocal Google libraries, the generator considers the indications of the CARE properties and annotates each user interface element accordingly with a specific *CSS* class if it contains a vocal part: *tts* for the output elements; *asr* for the input parts of interaction elements. The generated elements are marked with these classes because the multimodal application/extension uses them to identify all the graphical elements having an associated vocal part.

### 4.2.3 MARIA Runtime

The MARIA Runtime integrates the communication with the Adaptation Engine and the Context Manager. The run-time subscribes itself to the Adaptation Engine sending the user and service (application) identifier. Then, the Adaptation Engine subscribes to the Context Manager for being notified of the relevant events. The Adaptation Engine receives events detected by the Context Manager, and checks whether some adaptation rules should be triggered. In the positive case, the Adaptation Engine sends the action part of the XML rule to the run-time, which transforms the actions in an equivalent *JSON* object (since *JSON* is easier to manipulate through *Javascript* in client side) and updates the final user interface to perform the associated modifications. In multimodal adaptation, the action part of the rules changes the CARE values of the interface elements adding or removing the graphical or the vocal modalities. An adaptation script, inside the generated final user interface, receives the action part of the rule in *JSON* format and interprets the received CARE values by adding or removing *CSS* class (as it was aforementioned) of the interface elements in order to adapt the interface.

We have developed a Multimodal Car Rent prototype that uses the MARIA RUIGE module to adapt the UI adding or removing interaction modalities accordingly to the adaptation rules.

### 4.2.4 Warehouse picking prototype based the MARIA Runtime

The first release of the intelligent picking prototype (Deliverable D5.2.2 Application Prototypes (R1)) used an internal representation of the adaptation engine and retrieved the context information (e.g. level of experience of the user) directly. For its second release, the prototype will communicate with the Adaptation Engine as shown in Figure 3. Thus, the prototype will not use the context information directly but the Adaptation Engine takes care of selecting the appropriate action considering the state of the context.

It was decided to exploit the existing MARIA RUIGE as its focus on multimodal interaction fits well with the characteristics of the intelligent picking prototype. The prototype uses a combination of visual and vocal interaction modes while the degree of multimodality depends on the state of the context (e.g. in a noisy area only visual interaction will be used). Additionally, the MARIA generator is focused on web environments which fits well in the architecture of the prototype. The prototype exploits a browser-based solution using a smartphone as the platform on the client side. The Head-Mounted Display (HMD), which allows the effect of augmented reality, is in the basic version merely used as a monitor displaying (i.e. duplicating) the smartphone's desktop.

## 4.3 UsiXML RUIGE sub-module

The UsiXML module consists in a proposal for generating final user interfaces that are adapted according to the context. As the other modules of RUIGE, it receives as input information the description of the application in *ASFE-DL* format and through the application of different transformations generates the final UI compatible with the UsiXML language. As mentioned in the previous releases of this deliverable, the UsiXML module is a proposal, since *ASFE-DL* still did not reach a consolidated and standard version, it is likely that further modifications and adjustments may be needed in order to implement the proposal. Besides this, there are minor modifications that apply to this module, thus this section is a recall of the previous contents presented.

The transformations within UsiXML context can be executed in two directions: *Linear* transformations are responsible for specializing or reifying the models, changing their abstraction level within the same context, and *Transversal* transformations act between two different contexts but within the same level of abstraction, changing thus specific aspects of the models according to the specificities of the new target context of use.

UsiXML covers different contextual dimensions, therefore the context information may belong to user, platform or environment, covering the adaptation process in general. The final goal is to reach applications that are both device-independent and modality-independent.

### 4.3.1 UI Transformer

According to the D4.1.1, transformations in this module can rely on mapping technologies such as *XSLT*, ATL, or QVT. Aiming at compatibility and reuse, ATL is preferred at this stage of development. The three transformations envisaged include:

- From *CIM* (Context-Independent Model) to *PIM* (Platform-Independent Model): based on more generic specifications of the UI, and models like task model, domain model, context model, a mapping and a marking model, the Abstract User Interface (*AUI*) is generated, reaching a specification that is still at this stage device-independent.
- From *PIM* to *PSM* (Platform-Specific Model): based on the model specification that was generated in the previous stage, a new specification that targets the new context of use is generated. At this stage the new device is considered, and appropriate changes are applied in the model generating a Concrete User Interface (*CUI*) specification that is compliant with the target platform. So far, *XWT* has been employed in this process (interpreted by a Windows Builder Eclipse plugin).
- From *PSM* to *ISM* (Implementation-Specific Model): this transformation takes into account the particularities of the system of the user and configuration of the device in order to generate the Final User Interface (*FUI*). At this stage, the plugin is able to generate *HTML* and *SWING/SWT* implementations.

It is worth to note that *XMI* (XML Model Interchange Language) has been used in all models for UsiXML.

### 4.3.2 UI Generator

Although UsiXML targets at different dimensions of the context, the RUIGE sub-module and the description provided mainly focus on specificities of the platform and device, for instance by considering different technologies and languages in order to generate the *FUI* specification (e.g. *HTML* and Java Swing versions).

For now, the generation of the *FUI* relies on a Java Swing application and uses *ATL* and *XML* for transformations. Such technologies are complemented by the use of Ecore models, *XWT* and an Eclipse plugin, all supporting the different processes during the generation of the final UIs.

### 4.3.3 UI Runtime

As previously mentioned in the D4.1.1, the execution at runtime varies according to the technology used to generate the *FUI*. Specific approaches are applicable in each case: for the *HTML* document or the Java Swing version, the runtime must be respectively performed with the browser itself or the java compiler.

### 4.3.4 Application

In the example of the car rental application, the transformations specified in the previous sections would (Figure 4):

- First, take into account models describing the: tasks, domain, context, mappings, markings to generate the *AUI*
- Then, with the *AUI* and the target context specification generates the *CUI* (by means of *XWT* transformations)
- Finally, based on the generated CUI, more specific transformations would lead to the generation of the *FUI*, in *HTML5* or *SWING* according to the context.

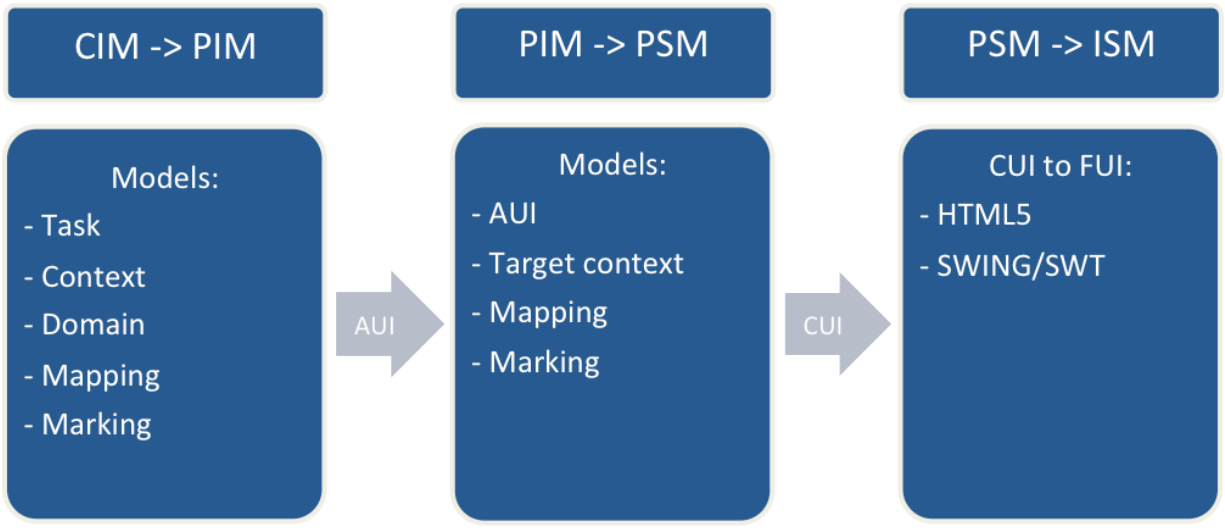


**Figure 4. Architectural approach for UsiXML RUIGE module transformations**

## 4.4 Avatar RUIGE sub-module

Minor changes have been introduced in the Avatar RUIGE sub-module. This sub-module is structured as SAIBA framework [3] and it proposes to generate the best suitable context-aware front-end for an avatar-based application. Some libraries have been analyzed and we have taken a decision on which one to use. The final demo planning has also been changed from the previous version, but the use case remains the same.

### 4.4.1 UI Transformer

The UI Transformer outputs the Serenoa Avatar CUI, which is based on the *FML* language [4] proposed by SAIBA framework from the abstract descriptions of the avatar. We are not going to fully implement the capabilities of *FML* in our transformer and our approach may not perfectly align *FML* and the Serenoa Avatar CUI for the avatar. For instance, *FML* covers gesture-text synchronization and fine descriptions for the elements that conforms the gestures, but the SERENOA Avatar CUI only describes a generic gesture. The details about gestures will be resolved later, when more context information about the device, environment and user will be incorporated into the adaptation process.

### 4.4.2 UI Generator

The UI Generator incorporates the remaining elements of *FML* that have not been included in the aforementioned transformation process. In order to achieve this goal, the results of the transformer are converted to our own *BML-like* language that will be used for the different target platforms. It is important that, while we are using the same *FUI* language for all the possible target platforms (see section 4.4.3), different descriptions are produced in accordance to the context. For instance, if we know that our target runtime belongs to a reduced profile family (e.g., a mobile phone with no generative Text-to-Speech capabilities but just recordings of the possible utterances by the avatar), we would not send the *'Hello master' TTS* element but an identifier of the appropriate available pre-recorded sound instead.

### 4.4.3 UI Runtime

The 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 in case of a mobile device platform. 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.

In case the runtime environment is an Internet Explorer browser, a Haptek [5] player will be used. The other avatar runtime that Serenoa will feature is internally developed 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 taking advantage of its <video> tag, and thus will be also available for desktop environments and will be developed with extensibility in mind.

### 4.4.4 Example

The final demo due by the end of the project has been refined and will include a conversational avatar. This section is going to describe an example based on the *eHealth* prototype. 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 an 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: Jane is logged into the application. RUIGE will generate the UI following the adaptation rules based on Jane's context and preferences.
- R2: If the device is a mobile, RUIGE will adapt the UI to be displayed in such device, i.e., the avatar will be rendered as a sequence of images or pieces of videos.
- R3: If the noise conditions get worse, then the avatar stops speaking and only communicates by text.

Then, as it is shown in Figure 5, the adaptation process performed by Serenoa framework would be as follows: Jane starts using the desktop application, at home, rendered following Jane's preferences (R1). When she is outside and she tries to access to the *eHealth* assistant using her smartphone (R2), the Runtime

module is in charge of degrading the avatar presentation and presenting the sequence-of-images version. Afterwards, the Context Manager warns about a high level of noise (R3), so it has no sense to keep the avatar voice feature. Then, the Runtime module is responsible of automatically stopping this feature.

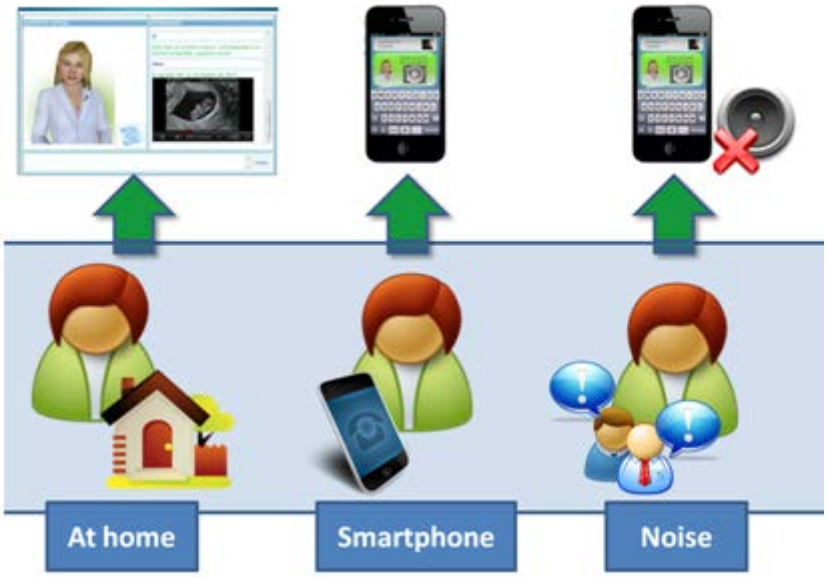


Figure 5. Adaptation example for the eHealth scenario

## 4.5 Leonardi RUIGE sub-module

The LEONARDI RUIGE's architecture and sub-modules is similar to the other Serenoa RUIGEs. It is depicted in Figure 6. It includes a transformer, a generator and a runtime.

The LEONARDI RUIGE is used to implement the *eCommerce* prototype, which is already operational and integrated with the Serenoa adaptation engine. The prototype is intended to illustrate the end-to-end reuse of Serenoa modules: authoring tools, *ASFE-DL*, *AAL-DL*, Adaptation Engine, RUIGE, Context Manager, CARFO...

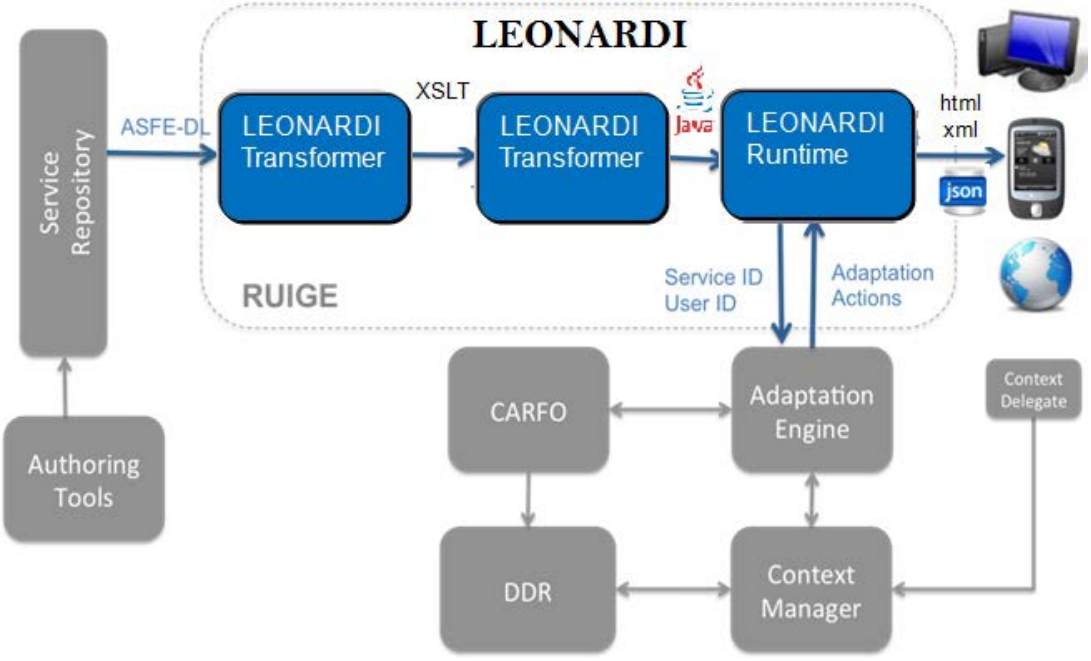


Figure 6. LEONARDI RUIGE Overview

### 4.5.1 LEONARDI Transformer

The LEONARDI Transformer takes as an input the Serenoa abstract description of the UI (*ASFE-DL*) and converts it into a LEONARDI-DL of the associated application. Both languages are *XML* based, so the intent is to rely on some mechanism such as *XSLT* to perform this transformation.

This module has been started but is not yet operational, since we are waiting for a stable definition of the *ASFE-DL* and for the availability of the Eclipse based Authoring tool that will facilitate the production of *ASFE-DL* files. Its implementation is independent on the other integration tasks.

Therefore, currently, the starting point of the *eCommerce* scenario is a LEONARDI description of the application (Version 8.9), which feeds, as an input, the LEONARDI Generator.

### 4.5.2 LEONARDI Generator

The LEONARDI Generator is implemented and already operational, as illustrated by the current status of the *eCommerce* scenario. It uses as an input a LEONARDI model, using the LEONARDI-DL as syntax. This mechanism allows to benefit as much as possible of the LEONARDI environment for Serenoa purposes.

It relies on the LEONARDI execution engine to interpret, on the fly, the application model. The LEONARDI engine is implemented in the Java technology. It is used as a service by fat clients (Java applications), typically used as desktop applications, and deployed as a Servlet for web based applications (including mobile applications).

### 4.5.3 LEONARDI Runtime

The LEONARDI runtime takes advantage of the LEONARDI technology and integrates the communication with the Serenoa Adaptation Engine and the Context Manager. For a specific user and context, the application queries the Adaptation Engine for the best adaptation rule based on the user's profile and context providing user ID and service ID), and then effectively adapts the UI for the specific platform based on the response returned by the Adaptation Engine.

The LEONARDI Runtime is completed by specific modules for mobility, as illustrated by Figure 7. These two modules, named "*iOS Player*" and "*Android Player*", are installed on *Android* and *iOS* smartphones or tablets. The LEONARDI runtime uses different protocols to communicate with different platforms: *JSON* for native applications installed on *Android* or *iOS* platforms, *HTTP* for web browsers and *XML*-based for *Java* applications.

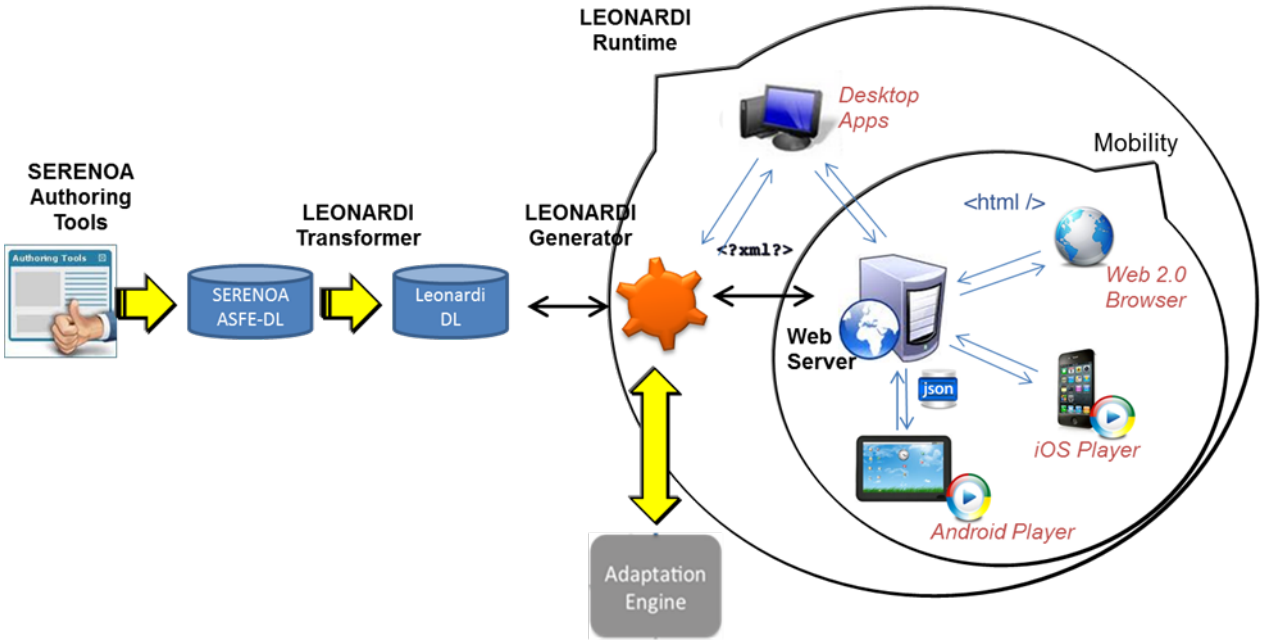


Figure 7. LEONARDI Runtime details

# 5 Conclusions

## 5.1 Summary

RUIGE module is in charge of the generation of different applications adapted to the context and to different platforms. The starting point is an abstract language (*ASFE-DL*) that defines the Abstract User Interface and a set of rules defined by means of *AAL-DL*. RUIGE takes the abstract definitions and creates adapted applications. To get the final application, it queries the Adaptation Engine to retrieve any useful information to carry out the adaptation process. Subsequently, the Adaptation Engine will communicate with the rest of the modules involved in the Serenoa framework: the Context Manager, CARFO, etc.

Finally, RUIGE is composed of different sub-modules trying to cover a great amount of target devices and modalities. It has been designed to facilitate the creation of new sub-modules. The RUIGE sub-modules developed within the Serenoa project are able to produce: mobile web, vocal applications, avatar and *eCommerce* applications. All of them are able to generate, at least, a basic application. Due to the fact that the project is still at M30 and some of the Serenoa components are not in the final version yet, these RUIGE sub-modules might be modified in order to adapt to them.

## 5.2 Future Work

The final version of this deliverable is released at month 30 but the project ends at month 36. This will cause some minor changes on each of the RUIGE sub-modules in order to adapt to other Serenoa components that will be released later, such as the *ASFE-DL*, the *AAL-DL*, the Adaptation Engine, etc.

In spite of the fact that RUIGE is composed of four sub-modules, the modular architecture allows to add any other sub-module to create context-aware *SFEs*, following a set of minimum requirements to become a RUIGE module.

There are also some plans for each one of the RUIGE sub-modules:

- ***MMW-4S RUIGE sub-module***:
  - To adapt the MMW-4S Transformer to be fully compliant with the final Abstract User Interface Model that will be produced in the W3C Model-Based User Interface Working Group.
  - At the time of writing this document, there is no *CUI* version of *ASFE-DL* tailored to mobile platforms (see D3.2.2). In case new specifications are finally defined at the *CUI* level, a future step will be to adapt MyMW-4S to use such specifications.
  - Improve the generation of *HTML5* applications taking into account the feedback received from the developers' community.
- ***MARIA RUIGE sub-module***:
  - To improve the support of the adaptation process by supporting more complex adaptation rules able even to change the structure of the user interface (not only rules that indicates updates of any UI properties).
  - To apply the MARIA environment to the Warehouse prototype.
- ***UsiXML RUIGE sub-module***:
  - To adjust the definitions of potential transformations between *ASFE-DL* and UsiXML assuring the synchronization between them.
  - To refine the definitions currently specified based on specific application scenarios.
  - To test and validate the proposed approaches.
- ***AVATAR RUIGE sub-module***:
  - Avatar RUIGE sub-module should be extended to be reactive to user's context changes while the user is logged or using the service, i. e. changes in user's environment like noise level.
  - Modification of the UI to be rendered in mobile devices, such as Android mobile phones or tablets.
  - Generation of more sequences of images mapping all the gestures that the avatar can render.
- ***LEONARDI RUIGE sub-module***:
  - To complete the LEONARDI Tranformer module.

- Improvement in UI rendering based on adaptation rules.
- To optimize the mobile players components for a better user experience on smartphones and tablets.

A further possible solution for runtime user interface generation will be the engine that is being developed as part of the Quill browser based authoring tool, see D4.5.1. The reasoning engine is being developed in *JavaScript* as a constraint-based expert system that generates the concrete UI for target platforms from the models for the domain, tasks and context of use. The engine will run in the cloud under *Node.JS*. Note that this is very much work in progress and won't be completed until near the end of the Serenoa project. Quill explores the design space looking for solutions that match the constraints provided by the designer and the context of use. At runtime, the end user could provide additional constraints, perhaps even overriding the designer's constraints. The engine uses constraint propagation to reduce the size of the search space and is capable of providing explanations when a solution can't be found. Further work would be needed on the means for end-users to express constraints.

# 6 References

[1] IDEAL2, (http://files.morfeo-project.org/mymobileweb/public/specs/ideal2/
[2] SCXML, http://www.w3.org/TR/2008/WD-scxml-20080516/
[3] Kopp, S.; Krenn, B.; Marsella, S.; Marshall, A.; Pelachaud, C.; Pirker, H.; Thórisson, K. & Vilhjálmsson, H.lmsson, H. (2006), Towards a common framework for multimodal generation: The behavior markup language, in 'Intelligent Virtual Agents', pp. 205—217
[4] Heylen, D.; Kopp, S.; Marsella, S.; Pelachaud, C. & Vilhjálmsson, H. The next step towards a function markup language Intelligent Virtual Agents, 2008, 270-280
[5] http://www.haptek.com/developers/HaptekGuide4/HaptekHyperText/htref/html/index.html

# Acknowledgements


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

# Glossary

http://www.serenoa-fp7.eu/glossary-of-terms/