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

Project no. FP7 – ICT – 258030

Deliverable D.4.3.2 Adaptation Engine (R2)

Due date of deliverable: 30/09/2013

Actual submission to EC date: 30/09/2013

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

Document Information	
Lead Contractor	TID
Editor	TID
Revision	v.1.0 (19/09/2013)
Reviewer 1	W4
Reviewer 2	
Approved by	TID
Project Officer	Michel Lacroix

Contributors	
Partner	**Contributors**
TID	**Javier Caminero, Mari Carmen Rodríguez**

Changes			
Version	**Date**	**Author**	**Comments**
0.5	09/06/2013	TID	First version of the document
1.0	19/09/2013	TID	Final version to be reviewed
1.1	23/09/2013	TID	Final version

Executive Summary

This document describes the Adaptation Engine for the Serenoa framework. The role of this engine is to select the adaptation logic for the transformation of the abstract user interface (AUI) descriptions of Serenoa applications into concrete UIs (CUI). This process will be informed chiefly by context information such as the device used to host the runtime, the environment in which the application is run and the particular user's preferences. In the document, this description is provided in a two-step process: first a bird's eye view of the components that form the inner architecture of the engine and the design principles used and second, some details of the current implementation of the system, such as the technologies used and the APIs that connect the module to the other components in the Serenoa framework.

Table of Contents

1 Introduction

1.1 Objectives

This deliverable is intended to be an update of the work developed in the Adaptation Engine component. This task had only two deliverables: the previous version, which was released at month 18 of the project and one last document at the end of the project. Hence, we agreed an intermediate update to present the advances and the direction the research is taking. Like the previous one, this is a 'prototype'-classed release and as such, it has to be reviewed in parallel with the software release of the system, which was part of the M24 review.

The Adaptation Engine is one of the core elements of the Serenoa architecture. It is in charge of gathering the high level description of the front-end for a Serenoa application, the context in which it is being used, the knowledge about adaptation represented in the CARFO ontology and the Knowledge Base, and then produce the rules to be performed in the runtime in order to implement the adaptation of the application to the context.

In this document, we present a high level overview of the inner architecture of this subsystem and how it works along with the other components in Serenoa. We also cover the technical details of the software itself, such as the different interfaces and data formats exchanged between the Engine and its associated components in the architecture. We do not intend to provide extensive detail regarding the implementation, but we do describe the design of the modules.

1.2 Audience

The audience for this document is composed of the following groups:

a) Members of the consortium, who will find here a detailed description of the fundamentals of the Adaptation Engine, for their understanding and also for reference when integrating together their own components.
b) Researchers in the relevant fields: adaptation of SFEs, UI theorists, XML transformation practitioners and medium-scale project software engineering.
c) EC officials that will use the information in this document as an account of the activities taken in the project tasks that inform this work.

1.3 Related documents

- D1.1.1 and D1.1.2, which describe the general requirements that the adaptation framework resulting of Serenoa must fulfil. They impact key decisions in the Adaptation Engine.
- D1.2.1 and D1.2.2, in which the general architecture of the Serenoa framework is outlined.
- D2.2.1 and D2.2.2, in which the first concepts of the ontology and knowledge management in Serenoa are outlined.
- D3.1.1, D3.1.2 and D3.1.3, in which the adaptation reference models are summarized. They will inform the strategies followed by the Adaptation Engine for adapting SFEs.
- D3.2.1, D3.2.2 and D3.2.3 in which the fundamentals of the description language for the SFE are first proposed. The AUI level there defined will be used by the Adaptation Engine to provide standardized way of describing SFEs in Serenoa.
- D3.3.1 and D3.3.2, in which the Serenoa language for rule description is put forward. This language will be the one in which adaptation rules produced by the authoring engine or other entities will be codified. These rules will be interpreted by the adaptation engine and changes in the SFE will result.
- D4.1.1 and D4.1.2, which describes the runtimes for Serenoa that will interpret the output of the adaptation process.
- D4.2.1 and D4.2.2, in which the different techniques for adaptation available to the engine are listed and described in detail.
- D4.3.1, which is the previous version of this deliverable group. It presents the theoretical foundations of the Adaptation Engine and some preliminary technical details which will be consolidated in the present document.

- D4.4.1 and D4.4.2, in which the context management infrastructure, which will feed the rest of the components in Serenoa with the current context of the running system, is designed and, crucially for the Adaptation Engine, its output format described.
- D4.5.1, D4.5.2 and D4.5.3, in which the work on the Authoring Environment is summarized.

1.4 Organization of this document

In Chapter 1 of this document, an introduction to the work is presented, along with a summary of the objectives, the target audience and a list of related documents in the Serenoa project. In section 2, we adopt a high level overview of the adaptation problem in Serenoa and come up with the design of the inner modules that are integrated for developing the engine. In addition, we place the adaptation module in the big picture of the project's architecture and give some details about the interconnection with several key modules. Chapter 3 is devoted to describe the implementation of the adaptation task and validate how the functional requirements are being fulfilled. Besides, a practical example which describes the communication protocol between the Adaptation Engine and the avatar-based RUIGE sub-module is presented. We end, in chapter 4, with a conclusion for the work undertaken.

2 Adaptation Engine Framework

The Adaptation Engine is a key element in the Serenoa framework. It serves the main functional purpose of powering abstract-to-concrete transformations of the UI descriptions of Serenoa applications and has to do so in a manner that is consistent with the following elements:

- Context of the application, understood as the set of conditions for the user, the device and the environment present at runtime.
- Knowledge about adaptation, represented in the project in a formalism defined by the CARFO ontology.
- Adaptation rules, defined by the application designer using the authoring tool.
- User preferences and customization options, which may be understood as a different set of rules than authoring ones.

In order to further describe the scope of the work and its design constraints, as well as how they will be addressed in this document, let us also examine the requirements for the module as stated in project deliverable D1.1.2:

- **Adaptation is functional [Must-have]**: this is the main global requirement, that is, that our design works for its primary function. In this deliverable we will explain the basic mechanism used for powering the adaptation which relies primarily on transformations applied upon the input abstract UI description until a concrete description is achieved.
- **Connectivity with the Context Manager [Must-have]** and **Connectivity with the Runtime [Must-have]**: these two requirements describe the basic connectivity that the engine should offer for a proper integration in the Serenoa framework. The Context Manager and the Runtime provide the basic input and output information flows that the system needs to perform adaptation. In this document, we will explain how these can be articulated by means of an API and data exchange formats.
- **Able to understand ASFE-DL AUI [Must-have]** and **Able to understand AAL-DL [Must-have]**: in addition to achieving integration, not only pure connectivity is needed, but also representation formats and underlying models that are compatible across modules. ASFE-DL (described in [Serenoa D3.2.3]) and AAL-DL (documented in [Serenoa D3.3.2]) are the project languages for this purpose, the former being used for description of UIs and the later for rules. In this deliverable, we will show how these languages are put to work together for the adaptation task.
- **Able to change its execution parameters based on feedback [Must-have]**: in the project's description of work, it is stated that the Adaptation Engine is not just a deterministic piece of software, but that it is augmented with the result of classic optimization techniques, classic Artificial Intelligence (AI) and Machine Learning (ML). For these techniques to work, it is critical that the adaptation process should not be monolithic, but depending on parameters that may be tweaked by optimization and learning algorithms to control the process. In this document, although we will focus on a first release of the adaptation engine without the bulk of the AI/ML techniques, we will provide the first study of how such a learning system may be used and what parameters should be selected for controlling the process.
- **Scalability and Support for distributed environments [Should-have]** and **Support for multiple deployment architectures [Could-have]**: an often overlooked aspect in research projects is the fact that real-world systems have to cope with load situations and deployment scenarios that are way beyond lab conditions. Thus, we will make an effort in designing an Adaptation Engine with an architecture that allows growing in capacity with the addition of new resources, hence assuring scalability on load. We will also try to provide a solution that is as portable as possible in terms of platform and operating system. In this deliverable, we will describe how these secondary aims might be achieved by implementing a distributed architecture allowing for adaptation tasks to be efficiently routed to adaptation resources.

2.1 Role and place in the Serenoa architecture

Let us first examine the general architecture of the Serenoa framework to introduce the Adaptation Engine in the appropriate context. In Figure 1, we present an updated diagram of the current Serenoa architecture. On the upper left side, the authoring environment is represented. It consists of the collection of tools intended to support the process of development of a SFE prepared to be adapted through the Serenoa framework. Once a proper definition of the interface and the associated adaptation rules have been provided, this information and the current context of use are passed to the Adaptation Engine and stored in the Service Repository. It launches the optimization processes (i.e. classic optimization techniques or AI/ML techniques) needed for the selection of the proper adaptation logic, based on the information abovementioned. Further details about the design and implementation of this component will be given next. The new AAL-DL resulting from this task will be the guideline for the runtime generation.

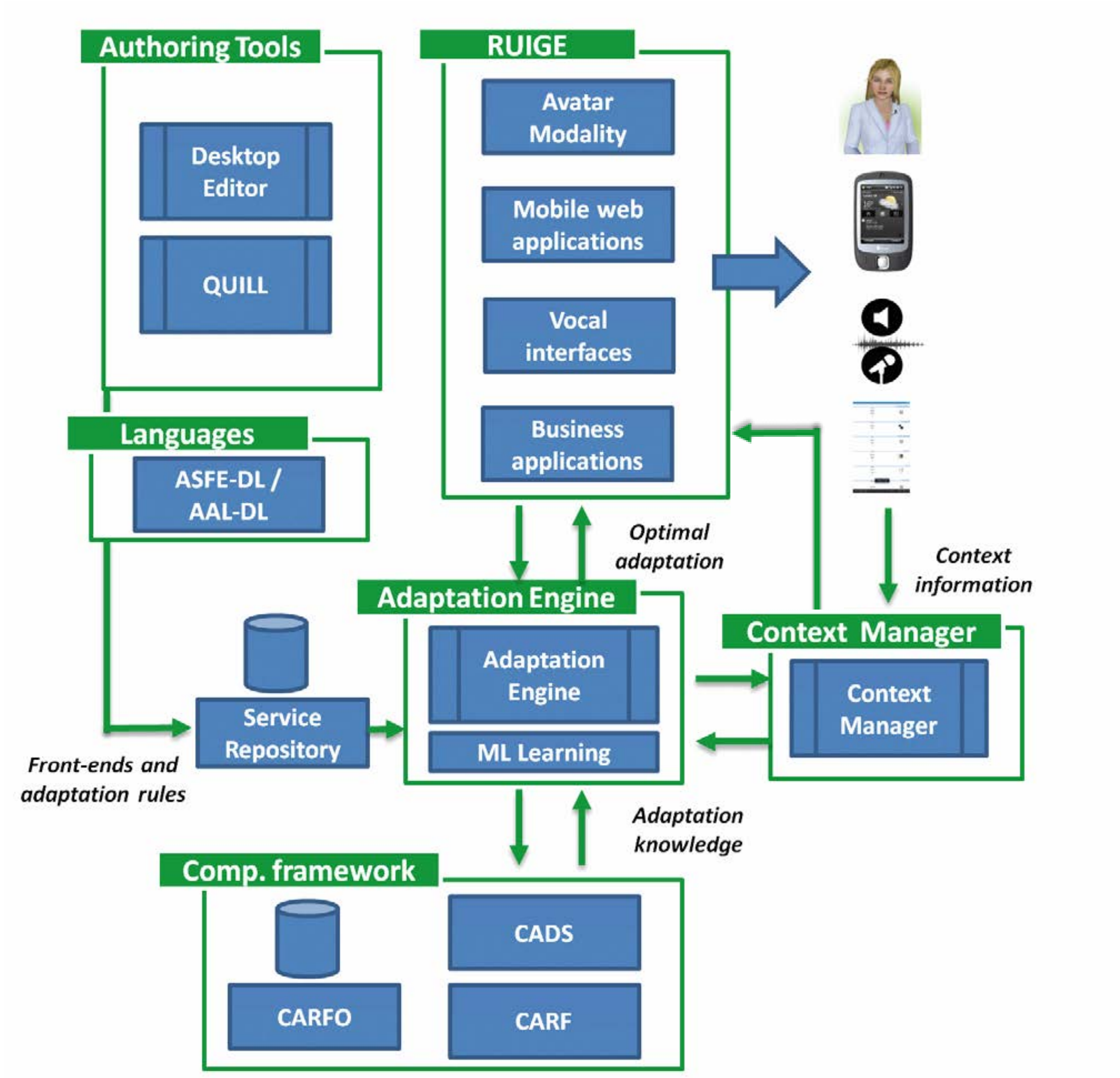

Figure 1 Serenoa architecture

Internally, the data flow in the Adaptation Engine will comply with the diagram in Figure 2, in which the most relevant elements of the problem and software modules involved are also identified.

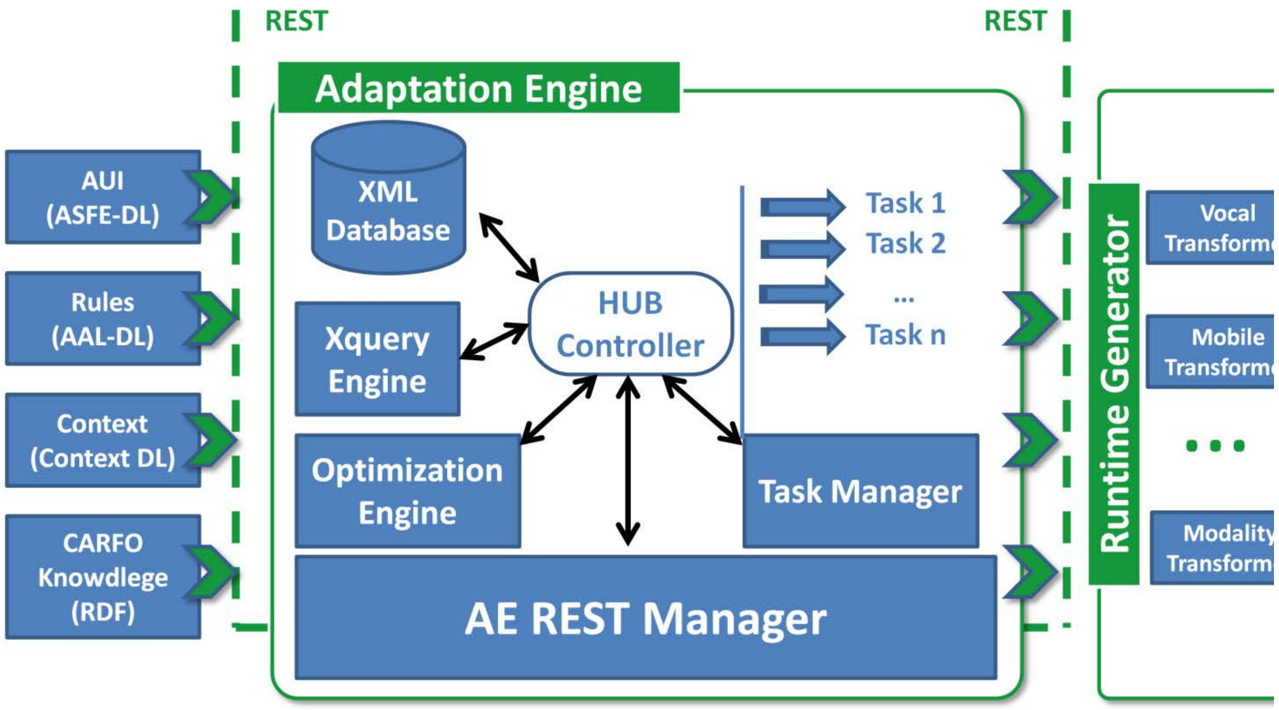

Figure 2 Internal Adaptation Engine functionality

As we can see, there are three main areas in the diagram. The leftmost one represents the four main elements serving as input for the adaptation engine and their representation formats (in parenthesis). They are: the abstract UI; the rules, which can be author- or user-defined; the context of the application and the adaptation knowledge that the system implements, which will be used to give meaning to the concepts implied in the adaptation.

In Figure 2 we can also see how these inputs have their representation formats well-defined: ASFE-DL and AAL-DL as description languages for the UIs and adaptation rules respectively (described in [Serenoa D3.2.3, 2013] and [Serenoa D3.3.2, 2013]), a tailor-made context description language (described in [Serenoa D4.4.2, 2013]) and RDF triples for the exchanges with the knowledge base. These inputs will arrive to the engine via an open API, which in its first implementation is based on RESTful concepts.

Once all this input content arrives to the engine, it is turn for the adaptation process itself to begin. The internal functionality of the engine has been separated into five main components, as it is evident in the figure:

- First of all, the *REST manager*, which specializes in the connection of the engine with the remaining elements in Serenoa, such as the inputs described above.
- An *adaptation task manager*, whose main roles are:
  - Pack incoming data from the inputs as ‘adaptation tasks’, which can be thought as self-contained sets of data waiting for processing.
  - Store the adaptation tasks as whole pieces in a data store (XML Database in Figure 2)
  - Monitor the availability of adaptation resources and, if they are available, send an adaptation task for processing.
  - Control and monitor the lifecycle of the adaptation tasks from the ‘idle’ state to the ‘processing’ and then to the ‘adaptation available’ so results can be communicated to the rest of the Serenoa framework.
- An *XML Database*, whose role is threefold:
  - Storing adaptation tasks prior to their processing (and possibly results prior to their sending to the rest of Serenoa, should the communication resources be limited as well).
  - Storing useful XML adaptation-related data, such as description language metamodels (i.e., ASFE-DL schema definitions), or any other useful data for adaptation.

  - Storing engine-related information not necessarily associated directly with adaptation, such as system options, log files and similar data.
- An *XQuery Engine* which supports the operations of querying and analyzing XML data involved in the adaptation process (i.e. ASFE-DL, AAL descriptions or XML-formatted adaptation tasks).
- An *Optimization Engine* which is intended to apply the decision-theoretic optimization techniques as well as AI/ML techniques in order to infer the optimal advance adaptation logic (AAL) in function of the context and user input data. The adaptation logic has as a main goal to appropriately associate the context information that is gathered from the context of use with appropriate adaptation techniques (see [Serenoa D1.2.2, 2012] and [Serenoa D4.2.2, 2012] which provide an extensive catalogue of adaptation techniques). As a result, the RUIGE is provided with this logic and models of different abstraction levels are generated in order for users to access an adapted Final User Interface.

We can also see that these five functional elements are linked together with a 'hub controller' that sits in the middle. This element is used to facilitate communications among modules by means of a common interface based upon a message-passing paradigm that uses XML formatted messages over plain text sockets. This controller provides a loosely multi-hub framework which facilitates a non hierarchical connection between the components. For communication between components, a lightweight communication protocol is used to support components implemented in various programming languages.

Additionally, we can see in the rightmost part of Figure 2 how we connect our outputs to the Serenoa runtime so that adapted results can be presented to the user. This is made via Web Services.

2.2 Relationship with other modules

2.2.1 Context Manager (CM)

Context information lets the Adaptation Engine know under what conditions the user is interacting with the service and in case these conditions change, the Adaptation Engine can adapt the front-end to the new scenario upon request.

The Context of Use infrastructure of the Serenoa framework relies on a distributed management support, whose main part is the Context Management Core. The distribution of the context information is carried out by the Context Management Core and its external interface. This can be executed on request or automatically. The former approach offers to the Serenoa external modules an interface in order to query the state of specific entities. The latter strategy implies that the external modules sign up to the subscription service offered by the Context Manager. Thus, a notification will be sent to the subscriber in case a specific context variable changed.

2.2.2 Context-Aware Reference Framework Ontology (CARFO)

The CARFO ontology is based on the theoretical models of Serenoa (CADS and CARF), and by formalizing the context information, it provides the connection point between the theoretical and practical aspects for adaptation. The context information that is considered corresponds to the most relevant concepts that are used to instantiate the adaptation rules.

2.2.3 Runtime User Interface Generation Engine (RUIGE)

The relationship between RUIGE and the Adaptation Engine is crucial, since the later is the component in charge of calculating the adaptation logic (adaptation rules expressed by means of AAL-DL) that has to be applied by the former in order to perform the optimal adaptations for a given context.

As set out in [Serenoa D4.1.2, 2013], RUIGE has been designed as a modular architecture composed of different sub-modules, such as a vocal runtime, a mobile web runtime, an avatar one, etc. A RUIGE module compliant with Serenoa framework must have three internal modules:

- ***Adapter***: responsible of transforming from the abstract language (ASFE-DL) to a concrete one depending on the runtime. The input of this module will be composed not only of the ASFE-DL itself, but also of a set of adaptation rules coming from the Adaptation Engine. Some of these rules will have a direct impact in the transformation process itself, while others might need to be considered at generation time or even at runtime.

- ***Generator***: it takes the resulting CUI description from the adapter and transforms it to a FUI description. To accomplish such goal, some of the rules produced by the Adaptation Engine must be considered.
- ***Runtime***: this module contains the libraries responsible for the execution of the application. The communication between this module and the Adaptation Engine must be bidirectional. On the one hand, the runtime must be aware of any changes in the adaptation rules and consequently react to them. Note that is especially important to consider on-the-fly changes in the adaptation rules in order to provide instant support for context variations. On the other hand, the runtime module must be able to report the feedback of the users to the Adaptation Engine, so as it will be taken into account in future adaptations.

3 Adaptation Engine Implementation

In this section, we will describe the implementation of the Serenoa Adaptation Engine (AE). Figure 2 displays the modules that compose the Adaptation Engine and the data flows. Next, we will review the development process of each component and, if needed, the different approaches which are being considered.

3.1 HUB Controller

In order to provide a light communication between components, we have developed a Java hub-based message-passing framework using XML formatted messages over plain text sockets. Every component in the system is connected to a repeating hub which broadcasts all messages received to all connected components.

Such an approach allows to employ modules, no matter whether they are Windows, Linux or iOS-based and regardless of their programming language. Each module is wrapped into a connector that provides the messaging API. The connectors have been developed and tested for Java, Perl and Python. Furthermore, the architecture allows to conveniently relocating any computationally demanding modules to a dedicated hardware. It would also permit different modules to be developed independently, and only sharing a common communication protocol (XML based) to collaborate on a specific task.

3.2 Adaptation Engine REST Manager

The release of the Adaptation Engine is interfaced by a REST architecture component. This service is intended to be queried by RUIGE sub-modules when a user asks for a specific service in a concrete context (i.e. platform, environment, preferences, etc.). Thus, the AE REST Manager is prepared to receive the following parameters:

- *service_id:* a unique identifier for each service. It is automatically assigned.
- *user_id:* name of the user who is accessing to the service.

Since the REST Manager will be in charge of serving the new AAL to the RUIGE sub-modules, there will be the options of AAL push or AAL pull. Push refers to AAL results upon being ready are pre-emptively sent to a specific module from RUIGE. It results in quicker responses and can be of use in case of a task prioritization system was considered. The second option is AAL pull. Here, it is the RUIGE's module who asks whether AE has new available logic for adaptation. If the answer is yes, the new AAL is sent to the requester. This will be done using RESTful calls through the AE REST Manager. The AAL pull mechanism is now supported and is described in the next section.

3.2.1 Description

The AE REST Manager has been implemented using Jersey[1] and serves the following resources.

Get optimal adaptation

- *HTTP method:* GET
- *URL:* http://195.235.93.35:8080/SerenoaAE/rest/service/{service_id}?user_id={user_id}
- *Parameters:*

Parameters	*Description*
{service_id}	The name of the service. Example value: 'Car Rental'
{user_id}	The name of the user which has accessed to the service.

- *Response formats:* XML

```
<result action="getAdaptation">
```

[1] Jersey is Sun's production quality reference implementation for JSR 311: JAX-RS: The Java API for RESTful Web Services. Available at: http://jersey.java.net/

```
<aal>
<action>
... (action to be performed for the adaptation)
</action>
</aal>
</result>
```

- *Example request:*

  GET http://195.235.93.35:8080/SerenoaAE/rest/service/CarRental?user_id=ann

3.3 Optimization engine

The Optimization engine is the responsible for selecting the optimal adaptation logic (AAL) in order to adapt the front-end for each context of use. It is worth highlighting the challenge which represents this task. Several works have been focused in this area [Benyon, 1993; Gajos et al., 2010] and in this project, it will be part of thorough research. To that end, we are following the next planning:

- Firstly, we have connected and integrated the AE component within the Serenoa architecture (see section 2 for more details about the role of the AE in the Serenoa framework).
- Then, a rule engine has been implemented in order to understand and apply the logic expressed by the rule language (i.e. AAL-DL). In section 3.3.1, the current support of this ruler is presented.

In the following sections, an explanation about the current support of the optimization engine is detailed.

3.3.1 Rules support

The AAL-DL is a high-level description language intended to express declaratively advanced adaptation logic (AAL). Such adaptation logic should defines the transformations affecting the interactive application when some specific situations occur both at the context level (e.g. an entity of the context changes its state), and in the interactive application (e.g. an UI event is triggered). The rules in the AAL-DL are expressed through an ECA-based format (Event, Condition, Action):

- The *Event* part of the rule should describe the event whose occurrence triggers the evaluation of the rule. At this moment the event part is not considered directly. Instead the event is presumed as fulfilled when the AE receives a request for an optimal adaptation.
- The *Condition* part is represented by a Boolean condition that has to be satisfied in order to execute the associated rule action(s). The optimization engine is able to take into account the following attributes:
  - *externalModelId* attribute, which specifies the id of an external model. The model *ctxModel* is supported and refers to the Context Manager as external information source.
  - *xPath*, which points to the concerned referred model. In case of the *ctxModel*, to the entity which is queried in the CM. For example:
    - */user/preferences/Serenoa/@DeviceCategory*

      entity (user) - entity (preferences_Serenoa) - attribute (DeviceCategory)
    - */user/deviceBattery/@batteryLevel*

      entity (user) - entity (deviceBattery) - attribute (batteryLevel)
  - *constant* tag defines a constant value with a specified type. Current support includes string (i.e. xs:string) and integer (i.e. xs:integer).

  Note: *Nested conditions* (with one additional level) are also supported. For example:

```
<condition operator="or">
        <condition operator="eq">
                <constant value="smartphone" type="xs:string"/>
                <entityReference xPath="/user/preferences/Serenoa/@DeviceCategory"
                externalModelId="ctxModel"/>
        </condition>
        <condition operator="eq">
                <constant value="tablet" type="xs:string"/>
                <entityReference xPath="/user/preferences/Serenoa/@DeviceCategory"
                externalModelId="ctxModel"/>
        </condition>
        <condition operator="eq">
                <constant value="cellphone" type="xs:string"/>
                <entityReference xPath="/user/preferences/Serenoa/@DeviceCategory"
                externalModelId="ctxModel"/>
        </condition>
</condition>
```

- *Action* part, specifying the action of the rule. In general, the action part will directly specify the transformation/adaptation to be executed. This information is sent back, if the condition is fulfilled, to the RUIGE sub-module which requested it.

3.4 Task management

The Task management includes three components: Task Manager, XML Database and XQuery Engine. The Task Manager is intended to gather all requests of adaptation and generates a list of adaptation tasks to be executed. The information which describes each adaptation task is then stored as a whole piece in the XML Database. We choose to use XML as the most widely used net-ready general knowledge representation format. We could have selected a Relational Database with an SQL approach, but we expect that front-ends adapted by Serenoa frameworks very often will be used within a browser. Then, XML representation format seemed the most natural choice. The XML Database is settled in eXist[2] to get a better integration with the rest of the elements in AE and better licensing conditions (eXist is a free software project).

At this point, it is worth mentioning the existence of a module intended to query XML data (XQuery Engine) that could be used to construct complex queries upon which to base the extraction of information from the inputs or the adaptation tasks. It is based on XQuery[3] which is a W3C sanctioned query language.

The execution of adaptation tasks is based on the batch job paradigm: when the Task Manager detects that adaptation resources are available (e.g., because system load allows a new adaptation task to be created), it picks a new adaptation task from the database and sends it to the Optimization engine.

3.5 Validation of the requirements

Here, we will examine the fulfilment of the requirements identified for the Adaptation Engine (see [Serenoa D1.1.2, 2012]). The requirements are as follows:

Adaptation is functional [Must-have]:

- The engine serves its main purpose, empowering the Runtime Engine through the most suitable adaptation logic in function of the active context.

[2] http://exist.sourceforge.net eXist project page
[3] http://www.w3.org/TR/xquery W3C Recommendation

- *Validation Criterion:* The system is able to select the optimal AAL based on the user context.

Connectivity with the Context Manager [Must-have]:

- The Adaptation Engine should be able to interact with the Context Manager and gather the active context from it when needed.
- *Validation Criterion:* There is a shared understanding of the interchange formats and a software API that connects both modules.

Connectivity with the Runtime [Must-have]:

- The engine is able to reach and interact with the Runtime module of the Serenoa platform.
- *Validation Criterion:* A set of data exchange formats and APIs are defined among Adaptation Engine and Runtime and there is a successful test run.

Able to understand ASFE-DL AUI [Must-have]:

- The engine is able to read ASFE-DL (described in [Serenoa D3.2.3]) files that express AUI descriptions and build internal representations that can be then interpreted.
- *Validation Criterion:* The engine implements an ASFE-DL parser and converts AUI elements to an internal representation format.

Able to understand AAL-DL [Must-have]:

- The engine can read AAL-DL (documented in [Serenoa D3.3.2]) descriptions and interpret them.
- *Validation Criterion:* The engine implements an AAL-DL parser which converts rules to an internal representation format.

Able to change its execution parameters based on feedback [Must-have]:

- The Adaptation Engine should be able ultimately to work using AI and ML techniques and also external feedback coming from the end user directly or through evaluation of prototypes.
- *Validation Criterion:* There are parameters that can be changed to modify adaptation options, accessible in programming, deployment and execution times.

Scalability and Support for distributed environments [SHOULD]:

- The Adaptation Engine for a commercial-grade Serenoa application may operate in very high load scenarios with strict performance requirements.
- *Validation Criterion:* The engine should be able to work in a distributed fashion, able to cope with high load situations by adding more adaptation resources.

Support for multiple deployment architectures [Could-have]:

- The Adaptation Engine may be used in different deployment scenarios including various typical Operating Systems such as Windows, Unix and Solaris.
- *Validation Criterion:* The system is at least able to run on two different platforms.

The implementation of the Adaptation Engine is represented in the Table 1.

Module	Adaptation Engine		
Partners	TID		
State of requirement implementation	Completed	In Progress	To be done
Adaptation is functional	X		
Connectivity with the Context Manager	X		
Connectivity with the Runtime	X		
Able to understand ASFE-DL AUI	X		
Able to understand AAL-DL	X		
Able to change its execution parameters based on feedback	X		
Scalability and Support for distributed environments	X		
Support for multiple deployment architectures	X		

Table 1 Validation of the AE requirements

3.6 Use case example

This section shows how the AE supports the adaptation of SFEs. To that end, the SARA service, which is the TID prototype included in the T5.2, is going to be used as the application scenario. The SARA project is aimed at self-monitoring chronic disease patients in the form of a (Windows based) tablet PC.

Figure 3 depicts the workflow of the adaptation process. Thus, the steps are as follows:

1. The patient requests the SARA self-monitoring service. The UI gets the user's credentials and sends them to a Java Servlet component which validates the user's access rights.
2. The RUIGE asks to the Adaptation Engine for suitable adaptations according to the current service (*service_id*) and the specific user context (*user_id*).
3. The AE retrieves the service description (i.e. ASFE-DL and AAL-DL) from the Service Repository. These descriptions are previously stored by means of the Authoring Tools. This service warehouse is interfaced using REST architecture (further details in [Serenoa D5.1.2, 2013])
4. The AE interprets the AAL rules according to section 3.3.1 and, if needed, requests the context information to the Context Manager. This data is sent from the Context Delegate, running on the client, through a Proxy in order to avoid cross-site HTTP requests[4].
5. The AE returns the suitable adaptation based on the fulfilment of the AAL rules based on the context information, retrieved from the CM.
6. The adaptation strategy is communicated to the avatar sub-module through the HUB. The RUIGE avatar sub-module consists of three main modules, namely: the transformer, the generator and the runtime.
    a. The *transformer* is in charge of conveying the intention for the avatar to express something. It receives the text associated to the patient's input/interaction (currently the text is obtained through a 'static' chatbot[5] which has been introduced in this component) and links the most suitable intent to be expressed by the avatar. A sample of a message sent to the HUB by the avatar transformer is:

[4] HTTP access control (CORS). Link: https://developer.mozilla.org/en/docs/HTTP_access_control
[5] Alice Chatbot. Link: http://www.alicebot.org/

```
<message zone_id = "defatulZone" sender = "bot" message_id = "messageId"
why_id = "whyId">
      <payload>
            <ECAOutput
                  application  = 'SARA'
                  embodiment   = 'samuela'
                  emphasize    = 'medium'
                  perform      = 'RECOGNISE'
                  affect       = 'neutral'
                  gesture      = 'main'
                  text         = 'text' />
      </payload>
</message>
```

b. The *generator* is responsible of mapping the functional description of the avatar behaviour into low-level features which could be rendered in avatar engines (i.e. Haptek engine and image/video adapted engine). In this case the message sent to the HUB would be like this:

```
<message zone_id = "defatulZone" sender = "transformer" message_id  =
"messageId" why_id = "whyId">
      <payload>
            <INTERNAL_ECA_CONTROLER
                  application  = 'SARA'
                  embodiment   = 'samuela'
                  emphasize    = 'medium'
                  perform      = 'RECOGNISE'
                  affect       = 'neutral'
                  gesture      = 'main'
                  text         = 'text’>
                  <sequence>
                  |0#SetSwitchIntensity [switch= expMouthHappy f0= 0.3
                  t= 0.4]|0#SetSwitchIntensity [switch= expMouthSad f0=
                  0.0 t= 0.4]|0#SetSwitchIntensity [switch= expBrowsSad
                  f0= 0.0 t= 0.4]|0#SetSwitchIntensity [switch=
                  expMouthMad f0= 0.0 t= 0.4]|0#SetSwitchIntensity…
                  </sequence>
            </INTERNAL_ECA_CONTROLER>
      </payload>
</message>
```

c. The *runtime* will prepare and send the required information to the runtime environment (i.e. the browser) in order to visualize the avatar interaction as it has been previously configured.

7. Finally, the RUIGE serves the adapted service (i.e. language, avatar behaviour, etc.) to the patient.

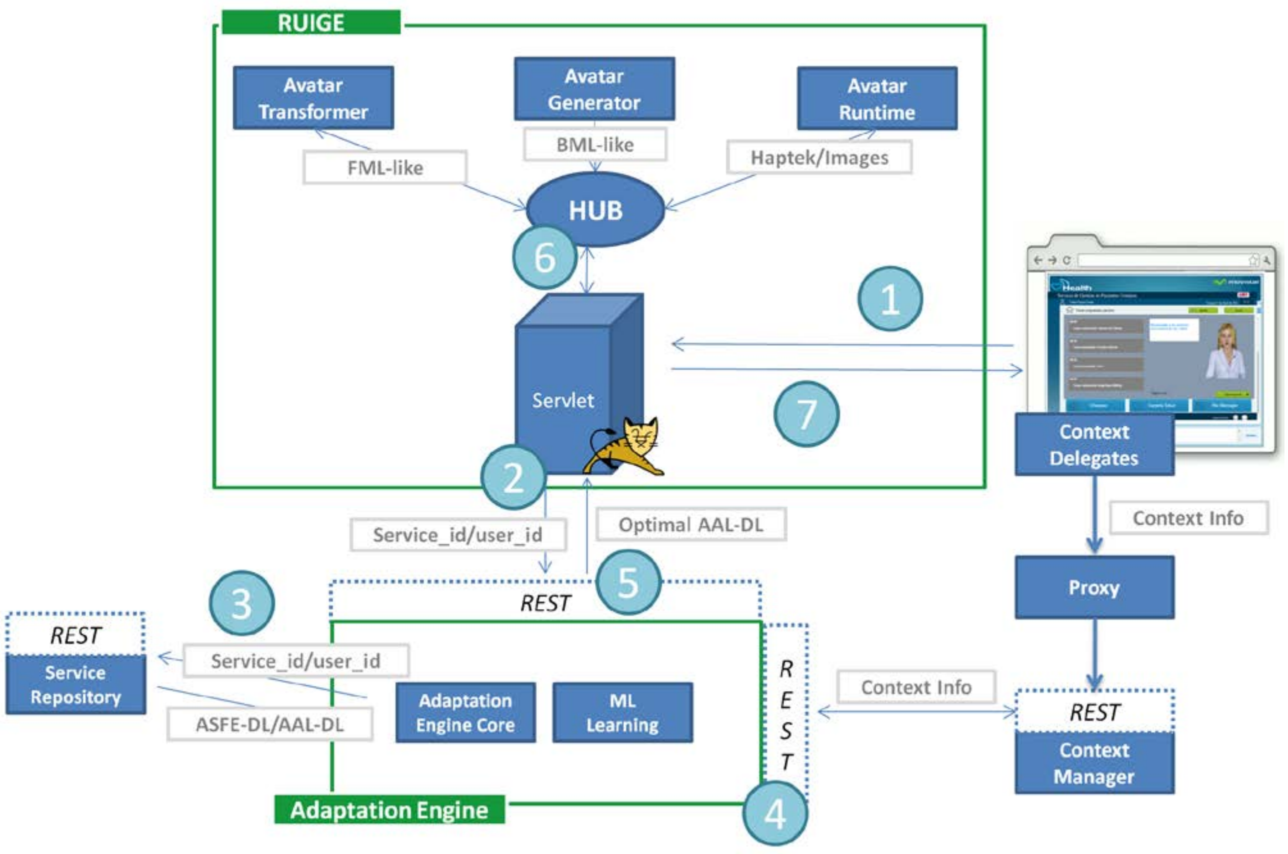

Figure 3 The SARA project workflow

4 Conclusions

This document reports the state of the development of the Adaptation Engine (AE). The AE is intended to power the adaptation process selecting the adaptation logic which is most suitable in function of: the context of the application, the knowledge about adaptation (CARFO ontology), the user-defined rules and the definition associated to the front-end.

The role of the AE has been defined as well as the different modules which compose it, namely:

- *REST Manager*, which is the interface for the rest of Serenoa components
- *Task Manager*, which stores the incoming data as ‘adaptation tasks’ in order to be executed as the resources are available. Besides it monitors and controls throughout the execution process.
- *XML Database*, which keeps the adaptation task prior to their processing and any other useful XML data (e.g. description language metamodels –ASFE-DL-).
- *XQuery Engine*, which is able to analyze XML data and extract information from it.
- *Optimization Engine*, which is in charge of inferring the optimal advance adaptation logic (AAL), making use of ML techniques.

Finally the final implementation of the AE has been analyzed. Firstly, a description of the ‘adaptation tasks’ lifecycle has shown how the AE inputs generate an adaptation task and through the Optimization Engine a new AAL is found. Secondly, a collection of ML techniques and their use for reacting to changes in context of use has been given. Finally, a validation of the requirements identified at the beginning of the project has been carried out.

5 References

Acay, L. D. (2004). Adaptive user interfaces in complex supervisory tasks (Doctoral dissertation, Oklahoma State University).

Benyon, D. (1993). *Adaptive systems: a solution to usability problems*, User Modeling and User-Adapted Interaction 3(1), 65-87.

Cao, J., Xing, N., Chan A. T. S., Feng Y., Jin B., (2005) Service Adaptation Using Fuzzy Theory in Context-Aware Mobile Computing Middleware, Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, p.496-501, August 17-19, 2005 [doi>10.1109/RTCSA.2005.93]

Deshpande, M. and Karypis, G. (2000) Selective markov models for predicting web-page accesses. Technical report, University of Minnesota Technical Report 00-056, 2000.

Deshpande, M. and Karypis, G. (2004) Selective Markov Models For Predicting Web Page Accesses. Acm Transactions On Internet Technology (Toit), 4(2):163–184, 2004.

Gajos, K., Weld, D. and Wobbrock, J. (2010). *Automatically generating personalized user interfaces with Supple*. Artificial Intelligence 174 (12-13), 910-950.

Gómez, J.M. and Tran, T.: 2009, A Survey on Approaches to Adaptation on the Web. In: Lytras, M.D., Ordóñez de Pablos, P. (Eds.), Emerging Topics and Technologies in Information Systems. pp. 136-152, IGI Global Publishing, Hershey. In: Miltiadis D. Lytras (Ed.) (The American College of Greece, Greece).

Hilbert D. M. and Redmiles D. F. Extracting usability information from user interface events. ACM Comput. Surv. 32, 4 Dec., pp. 384-421, 2000.

Horvitz E, Breese J, Heckerman D, Hovel D, Rommelse K (1998) The lumiere project: bayesian user modeling for infer- ring the goals and needs of software users. In Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelli- gence, Morgan Kaufmann, San Francisco, pp 256–265

Hutchinson A. (1994). Algorithmic Learning. Oxford University Press Inc, New York.

Jensen, F. V. (1996). An Introduction to Bayesian Networks. New York: Springer.

Kolski C., Grislin-le Strugeon E., “A review of intelligent human-machine interfaces in the light of the ARCH model”. International Journal of Human-Computer Interaction, 10, pp. 193-231, 1998.

Krulwich, B. and Burkey, C., (1995). Contact Finder: Extracting indications of expertise and answering questions with referrals. Submitted to The 1995 International Conference on Knowledge Discovery and Data Mining.

Krulwich, B., and Burkey, C. (1996). Learning User Information Interests through Extraction of Semantically Significant Phrases. In: Proceedings of the AAAI Spring Symposium on Machine Learning in Information Access. Stanford, CA.

Lorenz, A., Oppermann, R., and Zimmermann, A. (2000) Adaptive and Context-Aware Systems: A Survey. Fraunhofer Institute for Applied Information Technology

Mitchell, T. (1997). Machine Learning, McGraw Hill. ISBN 0-07-042807-7, p.2.

Mitrovic, N., Royo, J.A. and Mena, E. (2005) Adaptive user interfaces based on mobile agents: Monitoring the behavior of users in a wireless environment. In: Symposium on Ubiquitous Computing and Ambient Intelligence, Spain. Thomson Paraninfo, 2005.

Mitrovic, N., Royo, A. & Mena, E., (2009). Adaptive Graphical User Interfaces: An Approach Based on Mobile Agents. Networks, p.1-36.

Murthy S. K., “Automatic construction of decision trees from data: A multi-disciplinary survey,” Data Mining and Knowledge Discovery, vol. 2, no. 4, pp. 345–389, 1998.

Nichols, J.; Myers, B.; Higgins, M.; Hughes, J.; Harris, T.; Rosenfeld, R. and Pignol, M. (2002), Generating remote control interfaces for complex appliances, *in* 'Proceedings of the 15th annual ACM symposium on

User interface software and technology', pp. 161-170.

Russell S. J. and Norvig P. “Artificial Intelligence: A Modern Approach” (2nd Edition), Prentice Hall, 2003.

Seo and Zhang, 2000 [Seo and Zhang, 2000] Young-Woo Seo and Byoung-Tak Zhang, Learning user's preferences by analyzing web browsing behaviors, In Proceedings of ACM International Conference on Autonomous Agents (Agents-2000), pp. 381-387, Barcelona, Spain, 2000.

Seo and Zhang, 2000 [Seo and Zhang, 2000] Young-Woo Seo and Byoung-Tak Zhang, A reinforcement learning agent for personalized information filtering, In Proceedings of ACM International Conference on Intelligent User Interface (IUI-2000), pp. 248-251, New Orleans, LA, January, 2000.

Serenoa D1.1.2 (2012). *D1.1.2 Requirements Analysis (R2)*. FP7 EU-funded Serenoa project.

Serenoa D1.2.2 (2012). *D1.2.2 Architectural Specifications (R2)*. FP7 EU-funded Serenoa project.

Serenoa D3.2.2 (2012). *D3.2.2 ASFE-DL: Semantics, Syntaxes and Stylistics (R2)*. FP7 EU-funded Serenoa project. Available online at http://www.serenoa-fp7.eu/wp-content/uploads/2012/10/SERENOA_D3.2.2.pdf

Serenoa D3.3.1 (2012). *D3.3.1 AAL-DL: Semantics, Syntaxes and Stylistics (R1)*. FP7 EU-funded Serenoa project. Available online at http://www.serenoa-fp7.eu/wp-content/uploads/2012/07/SERENOA_D3.3.1.pdf

Serenoa D4.1.1 (2012). *D4.1.1 Runtime UI Generation Engine (R1)*. FP7 EU-funded Serenoa project. Available online at http://www.serenoa-fp7.eu/wp-content/uploads/2012/07/SERENOA_D4.1.1.pdf

Serenoa D4.2.2 (2012). *D4.2.2 Algorithm library for AAL (R2)*. FP7 EU-funded Serenoa project. Available online at http://www.serenoa-fp7.eu/wp-content/uploads/2012/09/SERENOA_D4.2.2.pdf

Serenoa D4.4.1 (2012). *D4.4.1 Context of Use Runtime Infrastructure (R1)*. FP7 EU-funded Serenoa project. Available online at http://www.serenoa-fp7.eu/wp-content/uploads/2012/07/SERENOA_D4.4.1.pdf

Serenoa D5.1.1 (2012). *D5.1.1 Serenoa Framework (R1)*. FP7 EU-funded Serenoa project.

Smith, A. S. G., & Blandford, A. (2003). MLTutor: An Application of Machine Learning Algorithms for an Adaptive Web-based Information System. International Journal of Artificial Intelligence in Education. 13(2-4), 233-260. Available online at http://www.cogs.susx.ac.uk/ijaied/abstracts/Vol_13/smith.html.

Acknowledgements

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

Glossary

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