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

Project no. FP7 – ICT – 258030

Deliverable D.4.3.1 Adaptation 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)

<table>
<thead>
<tr>
<th>Dissemination level</th>
<th>PU</th>
<th>Public</th>
<th>Yes</th>
</tr>
</thead>
</table>

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

<table>
<thead>
<tr>
<th>Role</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lead Contractor</td>
<td>Telefónica I+D</td>
</tr>
<tr>
<td>Editor</td>
<td>F. Javier Caminero Gil</td>
</tr>
<tr>
<td>Revision</td>
<td>v.1.0 (29/02/2012)</td>
</tr>
<tr>
<td>Reviewer 1</td>
<td>Vivian Motti</td>
</tr>
<tr>
<td>Reviewer 2</td>
<td></td>
</tr>
<tr>
<td>Approved by</td>
<td></td>
</tr>
<tr>
<td>Project Officer</td>
<td>Michel Lacroix</td>
</tr>
</tbody>
</table>

### Contributors

<table>
<thead>
<tr>
<th>Partner</th>
<th>Contributors</th>
</tr>
</thead>
<tbody>
<tr>
<td>TID</td>
<td>F. Javier Caminero Gil, Mari Carmen Rodríguez Gancedo</td>
</tr>
<tr>
<td>CTIC</td>
<td>Javier Rodríguez, Cristina González, Ignacio Marín</td>
</tr>
<tr>
<td>UCL</td>
<td>Vivian Motti, Nesrine Mezhoudi</td>
</tr>
<tr>
<td>W3</td>
<td>Dave Raggett</td>
</tr>
</tbody>
</table>

### Changes

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>03/02/2012</td>
<td>TID</td>
<td>ToC version</td>
</tr>
<tr>
<td>0.2</td>
<td>12/02/2012</td>
<td>TID, UCL</td>
<td>Initial contents</td>
</tr>
<tr>
<td>0.3</td>
<td>14/02/2012</td>
<td>TID, UCL</td>
<td>First consolidated version</td>
</tr>
<tr>
<td>0.4</td>
<td>17/02/2012</td>
<td>CTIC, W3</td>
<td>Second contribution by partners</td>
</tr>
<tr>
<td>0.5</td>
<td>28/02/2012</td>
<td>TID, UCL</td>
<td>Final consolidation</td>
</tr>
<tr>
<td>0.6</td>
<td>29/02/2012</td>
<td>UCL</td>
<td>Reviewer Comments sent to TID</td>
</tr>
<tr>
<td>1.0</td>
<td>29/02/2012</td>
<td>TID</td>
<td>Final version</td>
</tr>
</tbody>
</table>
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 descriptions of SERENOA applications into concrete UIs. 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 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 then some detail of the actual implementation of the system, such as the technologies used and the APIs that connect the module to the rest of components in the framework. We also provide some details on the future evolution of this adaptation engine and its co-ordination with related efforts in the SERENOA framework, such as the runtime and the authoring tools.
# Table of Contents

1 Introduction .............................................................................................................................. 5  
1.1 Objectives .......................................................................................................................... 5  
1.2 Audience ........................................................................................................................... 5  
1.3 Related documents ............................................................................................................. 5  
1.4 Organization of this document ......................................................................................... 6  
2 Adaptation Engine Framework .............................................................................................. 7  
2.1 Role and place in the SERENOA architecture ................................................................ 8  
2.2 Relationship with other modules .................................................................................... 10  
  2.2.1 Context Manager .......................................................................................................... 10  
  2.2.2 CARFO ....................................................................................................................... 10  
  2.2.3 Runtime: adapters & generators .................................................................................. 10  
3 Adaptation Engine Implementation ....................................................................................... 12  
  3.1 Lifecycle and adaptation task management ..................................................................... 12  
    3.1.1 Representation format ............................................................................................... 12  
    3.1.2 Task management ..................................................................................................... 12  
    3.1.3 Optimization engine ................................................................................................. 12  
  3.2 Optimization engine .......................................................................................................... 13  
    3.2.1 Support in R1 demonstration (M18) .......................................................................... 13  
    3.2.2 Provision for future ML techniques .......................................................................... 13  
4 Conclusions ............................................................................................................................ 18  
  4.1 Summary ......................................................................................................................... 18  
  4.2 Future Work .................................................................................................................... 18  
  4.3 References ...................................................................................................................... 19  
Acknowledgements .................................................................................................................. 21  
Glossary .................................................................................................................................... 22
1 Introduction

1.1 Objectives

This deliverable covers the first release of the SERENOA Adaptation Engine. It is a ‘prototype’-classed release and as such it has to be reviewed in parallel with the first software release of the system, due for M18.

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 the adaptation of the application to the context.

In this document we provide an introductory overview of the state of the art for the fields relevant to the Adaptation Engine and present a high level overview of the inner architecture of this subsystem and how it works along with the rest of components in SERENOA. We also cover the first 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 into the implementation but rather a summarized bird’s eye view of the parts in the system.

1.2 Audience

The audience for this document are 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, in which the general architecture of the SERENOA framework is outlined.
- D2.2.1, in which the first concepts of the ontology and knowledge management in SERENOA are outlined.
- D2.3.1, (produced in parallel to this document) in which the aforementioned first steps in establishing an ontology for adaptation are put forward into a knowledge base usable by the Adaptation Engine.
- D3.1.2, in which the reference models for adaptation are summarized. They will inform the strategies followed by the Adaptation Engine for adapting SFEs.
- D3.2.1 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 in its first version to provide standardized way of describing SFEs in SERENOA.
- D3.3.1, (produced in parallel to this document) 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, (produced in parallel to this document) which describes the runtimes for SERENOA that will interpret the output of the adaptation process.
- D4.2.1, in which the different techniques for adaptation available to the engine are listed and described in detail.
- D4.4.1, (produced in parallel to this deliverable) 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, in which the first work on the Authoring Environment is summarized.

1.4 Organization of this document

In the 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 the related documents of the SERENOA project. In section 2 we adopt a high level overview of the adaptation problem in SERENOA and come up with a preliminary 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 detailing the implementation of the adaptation task. We also discuss the first details of the changes needed to enable machine-learning subsystems to take control of the adaptation in later stages of SERENOA. We end in section 4 with a set of conclusions for the work undertaken and a description of the future work to undertake in this task.
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.
- 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 as the 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 have 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 do 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 achieve integration, not only pure connectivity is needed, but also representation formats and underlying models that are compatible across modules. ASFE-DL (first described in [SerenoaD3.2.1]) and AAL (documented in [SerenoaD3.3.1]) are the project languages for this purpose, the first used for description of UIs and the second 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 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 is not monolithic but depending on parameters that may be tweaked by the 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 based on the one in deliverable D1.2.1. On the left side the authoring environment is represented. It consists on the collection of tools intended to support the process of development of a SFE prepared to be adapted by means of 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. 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 resulting from this task will be the guideline for the runtime generation.

![SERENOA architecture](image)

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.
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 the 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 [D3.2.1] and [D3.3.1]), a tailor-made context description language (described in [D4.4.1]) and RDF triples for the exchanges with the knowledge base. These inputs will arrive to the engine via an open API, which in the first implementations will be based upon 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 will be initially separated into four 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 SERENO, 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 SERENO 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 SERENO, 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.

- A *XQuery Engine* which will support 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 main goal appropriately associating the context information that is gathered from the context of use with appropriate adaptation techniques (see D1.2.1 and D4.2.1 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 to users can access an adapted FUI.

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 nonhierarchical 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 the Figure 2 how we connect our outputs to the SERENOA runtime so adapted results can be presented to the user. This is also planned to be made via a REST interface.

### 2.2 Relationship with other modules

#### 2.2.1 Context Manager

The current state and the evolution of the context are crucial for the adaptation process. Context information lets the Adaptation Engine know under what conditions the user is to interact with the service and in case these conditions change, the Adaptation Engine can adapt the front-end to the new scenario.

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. At the moment a HTTP interface supports all these operations (excluding the notification, which is given via TCP), but the implementation of other methods is being carried out (i.e., Web Sockets for HTML5 or Web Service).

More detail about the Context of Use infrastructure is given in D4.4.1.

#### 2.2.2 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: adapters & generators

The relationship between RUIGE and the Adaptation Engine is crucial, since the latter 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 D4.1.1, 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 Rules, so as it will be taken into account in future adaptations.
3 Adaptation Engine Implementation

In this section, we will describe the current implementation plans for the SERENOA Adaptation Engine (AE). After the design work documented in section 2, we are going to pay attention to implementation details related with some of the modules which have been pointed out before.

3.1 Lifecycle and adaptation task management

The goal of this section is to describe the flow of an adaptation task from its creation to the generation of a new logic for adaptation. Besides we will give some details about the implementation of the modules involved in this process.

3.1.1 Representation format

The adaptation process starts when a new request arrives to the AE Rest Manager. The request consists on some of the following four inputs:

- Abstract User Interface, defined using ASFE-DL. The first steps towards the description language are presented in “D3.2.1 ASFE-DL (R1)”;  
- The author- or user-defined rules, described by AAL-DL. The characteristics of this language will be initially described in “D3.3.1 AAL-DL: Semantics, Syntaxes and Stylistics (R1)”;  
- The context of the application provided by the Context of Use module which could provide static or dynamic information (real-time);  
- The adaptation knowledge that the system implements, which will be used to apply transformations of one type or the other.

3.1.2 Task management

The Task Manager gathers all requests of adaptation and generates a list of adaptation tasks to be executed. The information which describes each adaptation task is 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 + 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\(^1\) 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\(^2\) which is a W3C sanctioned query language.

The execution of the 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.

3.1.3 Optimization engine

Once the adaptation task is launched, it is time for the Optimization engine to perform the selection of the most suitable advance adaptation logic (AAL), taking into account the information related with the context of use. To that end ML techniques will be adopted, using feedback from the user or data from past user interactions. In section 3.2.2 some examples of ML techniques applied to adaptation tasks are described.

Finally, the REST Manager will be in charge of serving the new AAL to the RUIGE. To that end 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 quickest 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

\(^1\) [http://exist.sourceforge.net](http://exist.sourceforge.net) eXist project page  
\(^2\) [http://www.w3.org/TR/xquery](http://www.w3.org/TR/xquery) W3C Recommendation
requester. This is all done using RESTful calls through the AE REST Manager.

3.2 Optimization engine

The Optimization engine will be responsible of 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 going to follow the next steps:

- Firstly, we are giving priority to the connection and integration of the AE component within the SERENOA architecture. In section 2 we put the AE into the scene and describe its relations with the rest of SERENOA modules.
- Then, a very basic decision strategy (rule-based) for an optimal AAL will be implemented to complete the integration necessary for the M18 demonstration.
- Later, our goal will be focus on the exploration of different optimization and reasoning methods such as decision-theoretic optimization (through cost functions which maximize different parameters related with the UI usability) and AI/ML techniques. A special attention will be given to the latter approach due to its promising results in the field (see section 3.2.2.2).

In the following sections an introduction about the working line which will guide the R1 demonstration is given. Later, a whole section is devoted to an overview of the most relevant approaches for Machine Learning techniques, and how they could be applied to reach the goals of the AE.

3.2.1 Support in R1 demonstration (M18)

Regarding the M18 demo, it is worth mentioning the demonstration wants to cope with a multiple-selection interface for a car rental service. In order to do that a rule-based approach has been selected. In [Nichols et al., 2002] a similar approach has been taken in order to generate UIs based on context information (e.g. UIs to control different devices). In our case the preliminary scenarios are intended to show the capabilities and be applied some kind of rules as following:

- S1: The user is allowed to enter the car rental information on several devices, but with one modality only (i.e. by means of a GUI).
  Two examples of rules include:
  - R1.1: If the device is a smartphone, then re-size the content of the view
  - R1.2: If the device is a smartphone, then change resolution of videos
- S2: The user is allowed to enter the same information, but with another interaction modality (e.g. through a vocal interface: “I’d like a grey van car with diesel from … to …”).
  - R2.1: If the UI is vocal, then remove images that cannot be rendered vocally
- S3: The user is allowed to enter a part of the information with one modality (e.g., the personal information is done once in a GUI) while the rest is updated at any rental time with another modality (e.g., the preferences in vocal mode, through the phone or embedded car browser).
  - R3.1: If the interactor is marked as vocal by the user, then the interactor changes its modality type

3.2.2 Provision for future ML techniques

3.2.2.1 Machine Learning strategies overview

Works in the domain of machine learning (ML) resulted in the development of an extensive list of algorithms. Typically, learning in these algorithms is accomplished by searching through a space of possible hypotheses to find an acceptable generalization of a concept. However, ML algorithms vary in their goals, learning strategies, the knowledge representation languages they employ and the type of training data they use [Smith and Blandford, 2003]. Such algorithms have been applied for different domains, such as medical diagnosis and natural processing languages, and some of them were already successfully applied for context-aware adaptation, like Bayesian networks and Decision trees. The items below describe 8 examples of machine learning algorithms applied to support context-aware adaptation:

- **Bayesian network.** A Bayesian network is a directed acyclic graph in which nodes represent
propositional variables and arcs represent dependencies. The leaf nodes represent observable propositions. Subsequent node’s value is a function of the values of the preceding nodes it depends upon [Jensen, 1996]. They offer a clear method for the compilation and representation of coherences, dependencies and independencies of objects in a probability network [Lorenz et al., 2000]. The Lumiere [Horvitz et al., 1998] system tries to predict user needs through use of use of Bayesian models. The system observes system events (such as mouse movement) and application input (e.g., text in an input field) to predict user needs. The system’s architecture is based on a standalone software control system, which gives feedback on observed events to the user. The system is specifically crafted for Microsoft Office applications and requires use of user profiles to operate. The control system responds based on pre-defined policies, e.g., responds directly to ‘trigger’ events, or collects a number of events before responding. This system offers help to user’s of Microsoft Office and does not offer any adaptation of GUls to devices or any mobility of GUls.

- **Cluster.** As a result of a clustering, components are in some way related to each other and as such they represent a concept. Thus the content of a cluster is assumed to represent positive examples of a concept and anything beyond the boundary of the cluster is taken to be non-representative of that concept [Smith and Blandford, 2003]. In the example of MLTutor there is a learning process that includes clustering in its first phase: it is utilized to find inherent patterns within the hypertext pages browsed by a user. To find these inherent classifications within the hypertext a simple conceptual clustering algorithm can be used [Hutchinson, 1994]. By applying this method, the need for initial “hand-crafted” stereotypical profiles is eliminated as well as any additional input from the user.

- **Decision tree.** A decision tree represents and classifies data. It resembles the tree structure consisting of a set of nodes and a set of directed edges that connect the nodes. The internal nodes stand for questions; the edges answers to those questions and the leaf nodes represent the final answer – also called decision. A decision tree is employed in ContactFinder [Krulwich and Burkey, 1995], InfoFinder [Krulwich and Burkey, 1996], and Lifestyle Finder [Krulwich, 1997] to find and recommend contact, documents and lifestyles [Gomez, J. 2009]. Decision tree algorithms accept a training set of positive and negative instances based on attributes of a concept. This set must be presented before the learning commences. Top down induction of decision trees is an approach to decision tree building, in which classification starts from a root node and proceeds to generate sub trees, until leaf nodes are created. It is possible to categorize conjunctive and disjunctive descriptions of concepts with decision trees, and if-then rules can easily be lifted from the trees. Typically in decision tree, learning instances are represented by a fixed set of attributes. Mitchell (1997) states that decision tree learning is well suited when attributes take on a small number of disjoint values. It maps well to the attribute-encoding scheme employed by MLTutor that is based on the presence or not of keywords in hypertext pages. This suggests that a decision tree building algorithm is a suitable candidate for rule inductions [Smith and Blandford, 2007].

- **Genetic Algorithms.** The flexibility of these algorithms allows them to handle any kind of heuristic or domain dependent knowledge. Ranking schemes handle the situation as if another data is collected. In addition to that, other methods can be used to solve optimization problems. However, GA is best fit for adaptation due to its ease of use in highly constraint problems. Besides this, GA is scalable to higher dimensions. This is important because in more complex systems feature space can grow exponentially with the number of features [Acay, 2004].

- **Markov Models.** Techniques derived from Markov models have been extensively used for predicting the action a user will take next, given the sequence of actions she has already performed. For this type of problems, Markov models are represented by three parameters \(< A, S, T >\), where \(A\) is the set of all possible actions that can be performed by the user; \(S\) is the set of all possible states for which the Markov model is built; and \(T\) is a \(\mid S \times \mid A\) Transition Probability Matrix (TPM), where each entry \(t_{ij}\) corresponds to the probability of performing the action \(j\) when the process is in state \(i\). The state-space of the Markov model depends on the number of previous actions used in predicting the next action. The simplest Markov model predicts the next action by only looking at the last action performed by the user. In this model, also known as the first-order Markov model, each action that can be performed by a user corresponds to a state in the model. A somewhat more complicated model computes the predictions by looking at the last two actions performed by the user. This is called the second-order Markov model, and its states correspond to all possible pairs of actions that can be performed in sequence. This approach is generalized to the Kth-order Markov model, which
computes the predictions by looking at the last K actions performed by the user, leading to a state-space that contains all possible sequences of K actions [Deshpande and Karypis, 2004]. Markov-based models provide better data utilization while retaining satisfying prediction rates [Deshpande and Karypis, 2004]. One example of an approach based on Markov model consists in the Predictions of the user behavior [Deshpande & Karypis, 2004]. The technique perform data mining, seeking to analyze web page navigation path based on server logs, similarity of web page content, linking structure of web pages, and user goals [Mitrovic, Royo and Mena, 2009].

- **Markov Chain**: Markov models have also been used to analyze web navigation behavior of users. A user's web link transition on a particular website can be modeled using first- or second-order Markov models and can be used to make predictions regarding future navigation and to personalize the web page for an individual user. Mitrovic et al. (2005) advocate using Markov-based models; specifically in their prototype they use the Longest Repeating Subsequence (LRS) method. A longest repeating subsequence is the longest repeating sequence of items (e.g. user tasks) where the number of consecutive items repeats more than some threshold T (T usually equals one). Statistical models such as LRS can be beneficial for predicting user actions. However, there are two major drawbacks to such models: 1) in order to predict next actions, training data must be supplied before the first use, and 2) poor quality training data can potentially divert users from using preferred application paths.

- **Neural Networks**: Seo and Zhang (2000) investigate software entities that are standalone, desktop applications. Such entities monitor the use of a particular web browser application; it collects information from the web pages opened by the user, information about the user such as time to read a page, scrolling up and down and so forth. The ‘learning agent’ then compiles a user profile and guides future web browsing by assessing relevance of web content to the user profile. This assessment is performed using a multi-layer neural network. Artificial neural network is an interconnected group of artificial neurons. It is composed of input, processing elements (containing mathematical or computational model for information processing) and output nodes [Hepner et al., 1990]. Boone uses such a network to represent and process feature vectors of uses such a network to represent and process feature vectors of messages to filter irrelevant emails [Gomez, 2009].

- **Rule Induction**: In the example of MLTutor there is a learning process that includes rule induction in its second phase: a rule induction procedure is employed to generate rules. These rules describe the concept of cluster membership. This process works on the basis that the components of a cluster are in some way related to each other and represent a concept. The information concealed in clusters was initially interpreted by the ID3 algorithm [Quinlan, 1986] to create a set of rules which can be viewed as dynamically created user profiles. The rules generated by this phase are used to search for other hypertext pages within the original population of pages, classified as belonging to the concept. The rule induction process generates attribute-based rules defining cluster membership. These rules are used to search the attribute database for other hypertext pages within the original population of pages classified by the rules as belonging to the concept [Smith and Blandford, 2003].

Although some adaptation rules can be straightforwardly selected and applied (e.g. if one specific condition is satisfied then apply one specific adaptation technique), there are many contexts of use that are more complex and that require more advanced logics regarding inference processes and reasoning in order to be successfully handled. As such, in the context of SERENOVA we rely on machine learning algorithms in order to deal with adaptation considering multiple context information, conditions, and techniques. The examples mentioned above represent scenarios in which machine learning algorithms have been successfully applied to perform context-aware adaptation. Therefore, in the context of Serenoa we investigate such applications in order to gather some knowledge in this domain, and to take better decisions for the Adaptation engine.

### 3.2.2.2 Applying Machine Learning in SERENOVA

Machine learning is one of the study fields of the artificial intelligence [Russell et al., 2002]. Machine learning aims at analyzing some facts and organizing them according to their direction or their relations. The notion of action/reaction relating to the contextual change can also be considered. Machine learning has many different potential applications [Kolski et al., 2004]. A decision tree [Murthy, 1998] for instance, can be built during the generation of the user interface. And by placing it as an XML-file in the level of the
distributed code permits to improve the decision rules in the decision trees, this allows the process of machine learning at runtime. Moreover, the access to the knowledge base and the modifications on the rules can be facilitated by representing the decision trees in XML format.

The system, based on machine learning techniques, learns from examples, for instance the examples can be reactions to changes in context of use. The system generally, can at runtime, record user reactions, such as modification of colors, window resize and UI widgets re-organization, and these reactions can be then associated with the occurred contextual changes. The system learns a new rule that can be applied in similar case, or modify an already existed one [Hilbert, 2000]. For example, reset the interface background to grey when the luminosity increases to 75%. In other words, the context capture process sends contextual information and additional information concerning the activities and the user reactions under certain conditions. New knowledge can be detected by comparing the changes at the contextual level to the reactions repeated by the user. Extracted knowledge can also be evaluated and validated in order to improve and to update the knowledge bases.

The use of Machine Learning techniques will provide the required infrastructure that permits the adaptation engine to learn. The three context-aware adaptation scenarios presented below illustrate concrete examples of how the ‘learning’ process can occur (they were previously reported in the D1.2.1).

- By definition, the **Decision Trees** allow the implementation of adaptation rules. So, considering a scenario in which the content subject to adaptation is in text format and the context information being taken into account is the user profile concerning visual impairments, three rules can be inferred to illustrate it: if the user has myopia the size of the text increases, if user is color blind the contrast between background color and text color must be adequate, and if user has astigmatism the readability must be increased (e.g. by increasing the spacing between letters, by improving the contrast level and by changing the font type). However, it is possible that straightforward rules may not be adequate for the ‘actual’ context of use, for instance because the user may already have available assistive technologies to support his or her tasks. So, considering a user with high level of myopia who also uses an assistive technology that is able to increase the size of all interface contents, the application of an automatic rule that increases the font size of texts will not be appropriate. The same occurs to the other scenarios when the assistive technology is involved. Thus, the application needs to allow users to provide their feedback, so once the rule is applied the user must be able to accept or reject it, or even classify or evaluate it according to some pre-defined criteria (e.g. accuracy level). And for the learning process, a certain number of rejections in a row must be considered in order to modify the rule based on the feedback provided by the user and the actual context of use (e.g. for this scenario the user has the visual impairment, and also an assistive technology that supports the interaction).

- In the case of using **Bayesian Networks**, a sequence of interactions can be used to infer an adaptation rule based on probabilities. The rules can be generalized by analyzing a defined context, such as: a group of users with similar profiles (e.g. in an e-learning environment, students of a given course) and similar behavior regarding their interaction sequence. So, for instance, if the user performed the task A, and then task B, and followed by the task C, and this sequence of interaction was repeated by the majority (or a certain percentage) of the users of the system, the system is able to infer the interaction steps (for this defined context), establish a network, and adapt its parameters. Thus, the interface is able to and must provide an easier access to this sequence of tasks, according to the interaction previously recorded and analyzed, and mainly if the original navigation structure does not have this task sequence easily accessible. Therefore, the navigation structure may be improved according to the users interaction sequence, and this can be implemented by, for instance, suggesting the next task based on the probability that it is generally accessed in a given context of use, or even by modifying the items order in a menu.

- Another technique of Machine Learning that can be implemented to support Adaptation is **Markov Models**. Based on statistical analysis, it permits to define a sequence of states in which the user ‘will be’. So, for instance, given a set of adaptation techniques to modify images, the system can learn from the users the probabilities that a technique (or a set of them) is applied after other specific ones regarding certain contexts (like the image and device type). Once the system has this information it is possible to abstract the most likely sequence, and then perform it in an automated way (for
instancing by grouping adaptation techniques in a method). However, the automation of adaptation processes may also disturb the user, making him confused, especially if the result is different from the expected one. Thus, it is necessary to use it carefully. First, to abstract the sequence in an efficient manner, by, for instance, considering a sufficient number of samples, aiming to provide a more efficient result. Besides, it is also necessary to allow users to provide their feedback, either by accepting or rejecting the automatic adaptation, or by classifying it as adequate or not, according to pre-defined criteria. The user feedback can also be used in the model to adjust the probabilities of the links between each ‘state’ (i.e. adaptation technique).

Another technique that seemed to be promising for context-aware adaptation is Fuzzy Logic. Since the context information not always can have a numeric interpretation, it is large, sometimes uncertain or vague, the fuzzy theory helps to deal with it in a more flexible and adaptive manner. An example of Fuzzy theory for context-aware adaptation was implemented by Cao et al. (2005).

The Learning process takes into account what the user does or prefers in order to: create and edit rules (establishing their parameters), define probabilities, and adjust the parameters of the implemented techniques (trees, networks and models).

For the Adaptation Engine, we intend to further investigate the applications of Machine Learning, to take more appropriate decisions concerning algorithms and scenarios of use, in order to develop and apply such techniques. The implementation of these techniques will be reported in more details in the second release of the D4.2.1 (Algorithms for Advanced Adaptation Logic).
4 Conclusions

4.1 Summary

This document reports the current status 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 to infer the optimal advance adaptation logic (AAL), making use of ML techniques.

Finally some touches have been arisen for the current implementation of the AE. Firstly a description of the ‘adaptation tasks’ lifecycle has shown how the AE inputs generates 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.

4.2 Future Work

Once the Adaptation Engine Framework has been devised and the architecture is being implemented, the future work will be focus in the next challenges:

- Integrate effectively the Adaptation Engine with the rest of SERENOA modules. The communication interfaces have been conceived for the integration between the different components and the M18 demo will be useful for make it real.
- Integrate the updated definition of the SERENOA languages (i.e., ASFE-DL, AAL-DL). In case of AAL a new deliverable is due to M18 (“D3.3.1 AAL-DL: Semantics, Syntaxes and Stylistics (R1)”).
- Improve the mechanisms implied in the ‘adaptation tasks’ management in order to create an AE scalable and efficient.
- Research into the possibilities that ML techniques offer in the adaptation process.
4.3 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