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

Project no. FP7 – ICT – 258030

Deliverable D.5.1.2
Serenoa Framework (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]	**[Public]**	**Yes**

Document Information	
Lead Contractor	TID
Editor	F. Javier Caminero
Revision	V1.0 (26/09/2013)
Reviewer 1	ISTI
Reviewer 2	
Approved by	
Project Officer	Michel Lacroix

Contributors	
Partner	**Contributors**
TID	**F. Javier Caminero, Mari Carmen Rodríguez**
CTIC	**Javier Rodríguez Escolar, Cristina González Cachón**
ISTI	**Fabio Paternò, Giuseppe Ghiani, Marco Manca**
SAP	**Safdar Ali**
UCL	**Vivian Motti**
W4	**Nicolas Bodin**
W3C	**Dave Raggett**

Changes			
Version	**Date**	**Author**	**Comments**
0.1	11/09/2013	TID	First integrated draft
0.5	20/09/2013	ISTI	Consolidated version
1.0	26/09/2013	TID	Reviewed version

Executive Summary

The technical and theoretical work developed in WP2 (computational framework), WP3 (languages and methodologies) and WP4 (runtime and authoring tools) is fundamental for the prototyping work which should be carried out in the task 5.2. The main goal of the task 5.2 is to show (through the implementation of several scenarios) the possibilities which the Serenoa Framework offers for the adaptation of Service Front-Ends (SFEs). In this document, the components (e.g. tools, methodologies or engines) which compose the Serenoa approach are briefly described as well as their main role and how they are supposed to be used by the developers participating in the prototyping task. In concrete, these components are:

- *WP2-Computational Framework*: to devise theoretical framework for multi-dimensional context-aware adaptation of SFEs (Service Front-Ends) and the related theoretic/conceptual elements. It includes :
    - The Reference Framework (CARF) and Design Space (CADS) which define the dimensions for context-aware adaptation of SFEs and let developers to explore different adaptation methods.
    - The Ontology for CAA of SFEs (CARFO). It is the formal representation of the different aspects involved in the adaptation process of SFEs. This information is used by the runtime modules and developers in order to get e.g. the characteristics of a specific device.
- *WP3-Models, Languages and Methodology:* to devise the languages/methodologies that support the development of adaptive SFEs. They are:
    - The Advance Service Front-Ends Description Language (ASFE-DL), aims to provide a unified description of the User Interfaces (UIs). The language provides an abstract description of the UI which means that the UI is described without taking into account a specific device or interaction modality.
    - The Advance Adaptation Logic Description Language (AAL-DL) is a high-level declarative description of the adaptation logic for context-sensitive applications. It mainly defines the reaction that the interactive application should perform in order to adapt itself according to a set of contextual events
    - The Agile methodology for developing adaptive SFEs. It is about Software Development Life Cycle (SDLC) for adaptive UIs. Our proposal is based on agile techniques combined with user centred design (UCD).
- *WP4-Algorithms, Runtime and Tools:*
    - The Runtime UI engine is responsible to generate, deploy and execute applications taking into account the Delivery Context. There are several sub-modules which offer different modalities (e.g. mobile web applications, vocal user interfaces, graphical UIs adapted to different screens or high-level avatars).
    - The Adaptation Engine is in charge of gathering the high level description of the front-end and the adaptation rules for a Serenoa application, the context in which it is being used, and the knowledge about adaptation represented in the CARFO ontology, and then indicate what adaptation should be performed.
    - The Runtime infrastructure for the context of use (Context Manager) is able to track and store the changes to the different dimensions of the context. This module is used by the Adaptation Engine in order to retrieve the current state of the context.
    - The Authoring environment composes of Quill (editing of model-based user interface designs based on modern web standards -HTML5, Javascript, etc.-) and an Eclipse plug-in (editing of abstract-level descriptions of SFEs and a Rules Editor for context dependent transformation rules).

Table of Contents

1 Introduction

1.1 Objectives

The main goal of the T5.2 in the Serenoa project is the implementation of several scenarios based on the technology developed. This deliverable presents a description of these components, their roles and explanations about how to use them. Thus, the current document provides a comprehensive documentation of the components of the Serenoa framework as well as how to extend the framework if needed. All this information is aimed to be a valuable information source for the developers community.

1.2 Audience

The following groups are the audience for this document:

a) Developers, who will find here a description of the main components involved in Serenoa adaptation process and a guide for making an effective use of them.
b) Members of the consortium, who use this document as a basis for the discussion about architectural aspects of the Serenoa framework.
c) EC officials that will use the information in this document as an account of the activities taken in the project tasks.

1.3 Related documents

The most relevant documents which are related to this deliverable are:

- *D1.2.1, D1.2.2 -Architectural Specifications (R1, R2),* in which the general architecture of the Serenoa framework is outlined.
- *D2.1.1-CARF and CADS (R1)* and *D2.1.2-CARF and CADS (R2)* which describe CARF and CADS theoretical models. These models give support to developers since they proposed and formalized adaptation dimensions to be taken into account in the implementation of SFEs.
- *D2.2.1-CARFO (R1),* in which the concepts of the ontology and knowledge management in Serenoa are outlined.
- *D2.3.1-CARFO, D2.3.2-CARFO Population (R1, R2),* in which the aforementioned steps in establishing an ontology for adaptation are put forward into a knowledge base usable by other Serenoa components.
- *D3.2.1-ASFE-DL, D3.2.2-ASFE-DL, D3.2.3-ASFE-DL: Semantics, Syntaxes and Stylistics (R1, R2, R3)* in which the fundamentals of the description language for the SFE are explained.
- *D3.3.1-AAL-DL, D3.3.2-AAL-DL: Semantics, Syntaxes and Stylistics (R1, R2),* in which the Serenoa language for rule description is put forward. This language is the one in which adaptation rules produced by the authoring tools are codified.
- *D4.1.1, D4.1.2-Runtime UI Generation Engine (R1, R2),* which describes the different runtimes for Serenoa that are in charge of generating the adapted SFEs.
- *D4.3.1, D4.3.2-Adaptation Engine (R1, R2),* which describes the fundamental of the Adaptation Engine responsible of finding an optimal adaptation strategy regarding the context of the user.
- *D4.4.1, D4.4.2-Context of Use Runtime Infrastructure (R1, R2),* in which the context management infrastructure is presented. It feeds the rest of the components in Serenoa with the current context of the running system.
- *D4.5.1, D4.5.2, D4.5.3-Authoring Environment (R1, R2, R3),* in which work on the Authoring Environment is summarized.
- *D5.2.1, D5.2.2, D5.2.3-Application Prototypes (R1, R2, R3),* in which a description of the prototyped scenarios and their development process are presented.

1.4 Organization of this document

In section 1, 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, a brief overview about architectural aspects of Serenoa framework is given. Section 3 lists the components/modules involved in the Serenoa framework. It gives a short description of the components, but it is mainly focused on presenting their usage (e.g. the available interfaces) and how they interact with the rest of the Serenoa framework. We conclude with some thoughts about the integration in section 4.

2 Architecture overview

This section aims to provide a 'big picture' of the architecture in order to compile a list of the components involved in the Serenoa framework, (i.e. Authoring Tools, Service Repository, Adaptation Engine, Ontology, Runtime Engine, Context Manager, etc.) as well as the languages, the methodology and the computational framework which supports the creation of adaptive Service Front-Ends (SFEs).

The Serenoa architecture is represented in Figure 1.

Authoring Tools are used to support the design of model-based user interfaces. The authoring tools support Serenoa's advance user interface description language (ASFE-DL), as well as the Serenoa language for expressing adaptation rules (AAL-DL). There are two different authoring tools: one based on an Eclipse[1] plug-in and another on a HTML5-based browser application called Quill.

The *Computational Framework* is made up of reference models and an ontology. These reference models (i.e. Context-Aware Design Space -CADS- and Context-Aware Reference Framework -CARF-) are aimed at guiding developers and designers during the complete software life-cycle, listing alternative possibilities for implementing context-aware adaptation and permitting the analysis and comparison of adaptive and adaptable applications. The ontology, based on the reference models, is intended to gather all the knowledge involved in advanced adaptation logic for user interfaces. It is used to inform developers as well as to support adaptation processes at run-time.

Finally, the run-time phase transforms the user interface description and the associated transformation rules into a final user interface. The *Adaptation Engine* determines the optimal adaptation for the current context of use, based upon the context events and adaptation rules. To achieve this goal, the *Context Manager* provides information related to all the possible contextual dimensions (e.g. the user preferences, the environment, social relationships, etc.) The *Runtime Engine* (i.e. RUIGE) generates the final interactive application according to the context. This module is composed of a set of sub-modules which cover several modalities (e.g. mobile web applications, multimodal Web interfaces, avatar-based interaction or desktop business applications).

[1] Eclipse is one of the most widely used Integrated Development Environments (IDEs) in the research and industrial communities for software development http://www.eclipse.org/

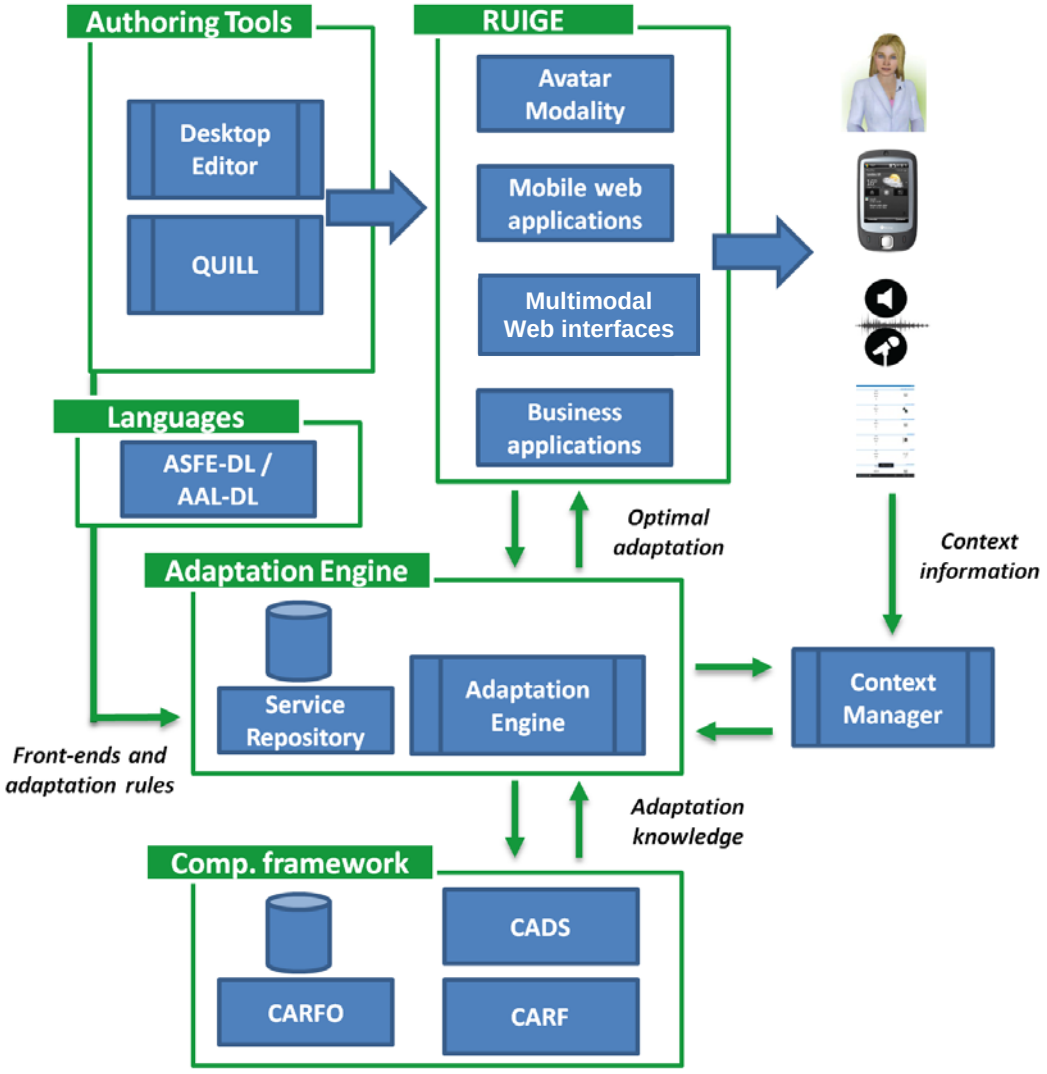

Figure 1: The Serenoa architecture

3 Serenoa framework

3.1 Introduction

Serenoa is aimed at developing a novel, open platform for enabling the creation of context-sensitive service front-ends (SFEs). The Serenoa framework will provide the theoretical and software components (e.g. languages, computational framework, runtime engines, authoring tools, etc.) to support the automatic adaptation of UIs.

The scientific and technical work that results on the development of the Serenoa framework has been carried out in the WP2 to WP4. In this document, and specifically in the next sections, a description of the components of the Serenoa framework is presented. The goal is to create a comprehensive usage manual for developers in the consortium and finally, for the whole developer community.

The necessary collaboration for the integration of the components which are part of the Serenoa framework, the definition of the common interfaces and the compilation of the related documentation has been obtained through various means:

- Bi-weekly conference calls where all partners participate. These calls have been useful to coordinate the project tasks and to reach compromises in the integration work.
- Face-to-face meetings where enough time is saved to integration discussions.
- The project wiki[2]. This collaboration tool provides a common place where partners could contribute and access to useful information regarding the software components.
- Sharepoint platform[3] in order to share deliverables and additional documentation.

Next, a description about the Serenoa framework is presented.

3.2 Computational framework: Context-Aware Reference Framework (CARF)

3.2.1 Description

In order to provide a rigorous basis for applying the adaptation of SFEs in a unified way, there is a need to conduct some fundamental research that systematically analyses existing adaptation techniques. The Reference Framework (CARF) is being elaborated in order to articulate all dimensions, their abstraction levels and concepts relevant to adaptation. The content is, in general, being gathered by performing a systematic analysis of the literature: compiling adaptation techniques, abstracting individual techniques into more general ones whether experience permits and consolidating techniques. The techniques are being structured according to how, what, who, when, where, with respect to what and with which models (based on the CAMELEON Reference Framework (Calvary, et al., 2003)). Finally the techniques are also being generalized when permitted.

The root of the CARF is composed by seven branches, as Figure 2 illustrates, in clockwise sense they are: why, what, who, when, where, to what, and how. These branches are abstract concepts defined as follows:

- **Why:** defines the main goals for the adaptation process, they are mainly expressed in terms of software qualities. For example: *adaptation can be performed aiming to improve the usability levels*.
- **What:** represents the type of resources that is adapted, in general they can be defined as navigational flow, presentation or content. For example: *re-size all the images*.
- **Who:** refers to the actor who triggers and is in charge of each phase of the adaptation process, for instance the end user, the system, or a third party. For example*: in mixed approaches both end users and system collaborate with the adaptation*.
- **When:** represents the state, in which the adaptation process is performed, i.e. it can occur at design time, run time, compilation time. For example: *adaptations performed at run time*.
- **Where:** this branch refers to the virtual location in which the adaptation takes place, i.e. according to the architectural approach adopted it can be at the client, at the proxy, or at the server. For example:

[2] https://wikis.tid.es/serenoa (accessible only with username and password)
[3] https://colabora.tid.es/serenoa (accessible only with username and password)

adaptation performed at the server side.

- **To what:** lists potential context information that justifies and defines the adaptation process, i.e. usually the application resources are subject to adaptation according to the user, the platform, or the environment. For example: *adapting according to colour-blind users*.
- **How:** defines how the adaptation process is performed, by listing possible methods, techniques and strategies for adaptation. For example*: change the font size.*

Figure 2: Reference framework

3.2.2 Use

The main goal of the CARF consists in defining the most relevant concepts for adaptation and to extensively list and present the possibilities regarding its implementation and execution. The CARF can be used before the implementation phase of an application, as an extensive catalogue to guide developers in taking design decisions, or after the implementation phase of an application, to analyse or evaluate the concepts that were taken into account (and consequently identify possibilities for future extensions).

For further information (http://www.serenoa-fp7.eu/deliverables/):

- D2.1.1-CARF and CADS (R1)
- D2.1.2-CARF and CADS (R2)

3.2.3 Integration

We envision the integration of CARF in the Serenoa framework as online tool, in which the stakeholders can submit their specifications and have the graphical representations generated in order to perform the analysis of their applications. This approach also enables stakeholders with different expertise levels to benefit from CARF.

3.3 Computational framework: Context-Aware Design Space (CADS)

3.3.1 Description

The Context-Aware Design Space (CADS) is a theoretical method that provides stakeholders with a tool to support them in the phases of implementation, analysis and evaluation of adaptive and adaptable applications.

The dimensions described below represent the basic structure for the CADS:

- **User Interface Component Granularity:** this dimension defines the levels of abstraction for the UI elements that can be subject to adaptation. Three levels are defined for these dimensions, interactor level, dialog level and total level. Interactors correspond to UI elements (e.g. a combo-box), dialog refers to containers (i.e. a composition of UI elements), and total level refers to the complete user interface. It is worth to note that this dimension is only valid in principle in the context of GUIs.
- **Modality:** this dimension refers to the adaptations that change the modality type for the user interaction, when the same modality is maintained the modality level is classified as intra-modality, when it changes from one type to another it is inter-modality, and when multiple modality types are involved, the adaptation is classified as multi-modality.
- **State Recovery Granularity:** this dimension refers to the application of the adaptation towards the impact to the end user, i.e. if the user is obliged to quit the session and re-start a new one, the state recovery occurs at the session level, if the task is impacted the recovery occurs at the task level, and if just the action itself is impacted, the recovery is said to be at the action level.
- **User Interface Deployment:** this dimension represents how much adaptation has been pre-defined at design-time vs. computed at runtime, thus respectively permitting a static or a dynamic

deployment.

- **User Feedback:** this dimension refers to how the opinion of the user is taken into account, i.e. if the system is adapted, and the user can just accept or reject the adaptation after it has been performed, it can be classified as Post; if she is able to accept it (or reject) before it is applied, it is said to be Pre; evaluations refer to the possibility of the users to provide their feedback to the system, in a numeric (e.g. with a Likert scale) or literally, providing further details about their feedback.
- **Technological Space Coverage:** this dimension refers to the technologies adopted and used by the application, when the same technology is maintained it is classified as intra-technological space, when the technology changes from one to another, it is called inter, and with multiple technologies are possible, it is classified as a multi-technological space adaptation.
- **Existence of a Meta-UI:** it concerns a meta-user interface description that formally represent and handle the adaptation process and also allow users to control, evaluate and evolve it. The possibilities are: no-meta UI, meta-UI without negotiation, meta-UI with negotiation, and plastic meta-UI.
- **Autonomy Levels:** it refers to the level in which adaptation is implemented, i.e. applications with fixed design do not perform adaptation at all, adaptable applications rely on users to trigger and perform the adaptation, adaptive systems rely on the adaptation to be automatically performed, and self-modifying means evolutionary systems able to adapt their own adaptation engines.

For further information (http://www.serenoa-fp7.eu/deliverables/):

- D2.1.1-CARF and CADS (R1)
- D2.1.2-CARF and CADS (R2)

3.3.2 Use

The goal of CADS is helping developers before implementing their applications to be aware of possible dimensions and granularity levels for performing adaptation, and after the implementation to analyse, evaluate and compare these dimensions regarding their respective coverage levels. As such the CADS supports the analysis and the comparison of different applications that execute adaptation and during their complete development life-cycle. The CADS is built as a radar chart (see Figure 3), which is a useful approach to represent multi variable observations with an arbitrary number of variables. In principle, these representations are used for ordinal measurements, however in the CADS case qualitative values are represented with their respective empiric scale associated.

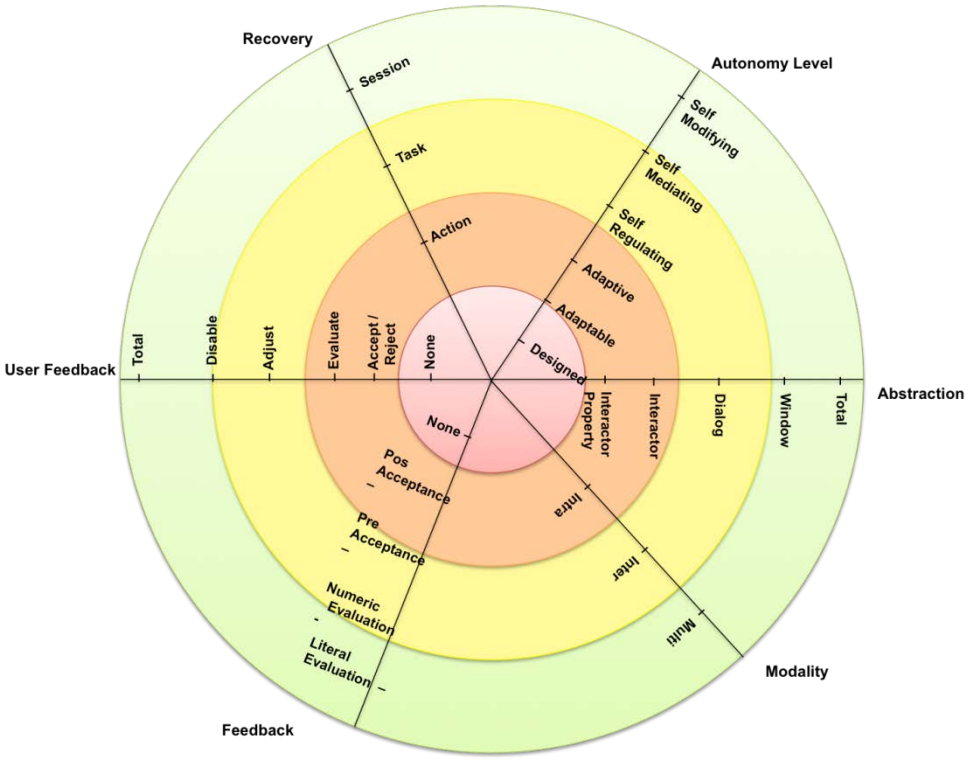

Figure 3: CADS radar chart covering 6 dimensions

Figure 4 presents an application example . In this chart the developer would mark the coverage level for each CADS dimension. The marks are defined according to what was previously considered during the implementation of the application to be analysed and what is currently available. So, for instance if the adaptation regarding the UI component granularity occurs at the interactor level, the axis must be marked (highlighted) until this specific level. This procedure must be repeated for each dimension of the model. As a result the developer will have the applied CADS in which she can easily identify the dimensions that were better explored and the ones that could be also taken into account to extend the application in the future.

Another possibility to mark the dimensions consists in colouring (with stronger tones) the region of the circle under the level of interest, however this approach works well only if all the levels correspond to the circles, and besides comparing multiple applications would not be possible with such an approach.

In order to compare two or more applications, developers have two possible approaches: (i) parallel lines can be drawn in different colours, allowing a straightforward comparison; or (ii) an additional model can be used, comparing thus different application of the CADS in parallel. Both approaches permit multiple applications to be compared simultaneously, however for a large number of samples the second approach is recommended since it does not affect the readability of the dimensions' labels.

When the comparison of multiple applications rely on colour to differentiate them, it is necessary to choose then different tones or styles, avoiding thus accessibility issues that may rise in case of colour blind users for example.

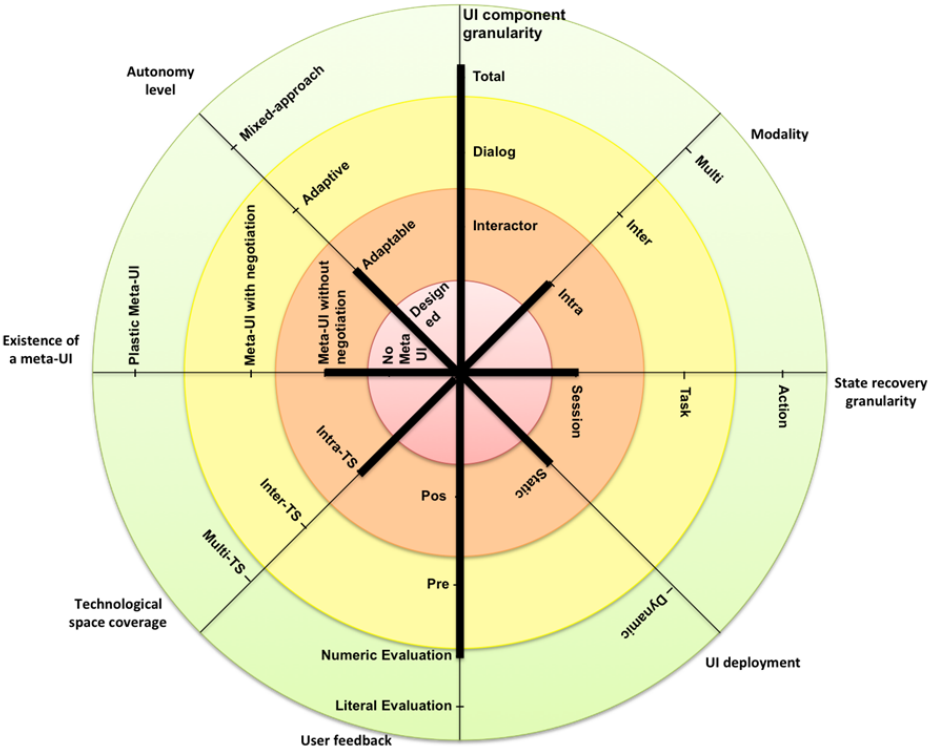

Figure 4: Example of the application of the CADS

3.3.3 Integration

In the same way as CARF, the context-aware design space (CADS) wants to be an online tool, in which the stakeholders can submit their specifications and have the graphical representations generated in order to perform the analysis of their applications.

3.4 Computational framework: CARF Ontology (CARFO)

3.4.1 Description

In the context of the Serenoa project, the CARFO ontology formalizes the concepts and relationships expressed in the Context-Aware Reference Framework (CARF). CARFO enables browsing and search for information relevant to defining and implementing the adaptation process. This is useful throughout all of the phases of an interactive system: design, specification, implementation and evaluation. Thus, CARFO is intended to capture information about the user, the platform, the environment, and other aspects, such as the software application content (structure and presentation). This feature maximizes the amount of contextual information that can be used to accomplish sophisticated adaptation.

Moreover, adaptive Service Front-End systems (SFEs) rely on their own formalism and vocabulary for data representation. Using CARFO as a standardized data model, systems can share and reuse data (described according this ontology) to solve the inherent lack of interoperability among SFEs.

3.4.2 Use

The CARFO ontology is the basis of a live, run-time available element: the CARFO Knowledge Base (CKB). This knowledge base could be used during execution by other modules to extract information such as adaption techniques, devices on which Serenoa applications may run and user information. In order to achieve the adaptation, it is needed not only the definition of CARFO ontology as a domain model, but also its population with useful data (instances).

From the technical point of view, a developer uses an SPARQL endpoint available to query the CKB, in order to obtain the knowledge required for the sophisticated adaptation.

The kind of information that developers may retrieve from the CARFO knowledge based is device

descriptions extracted from several repositories.

For further information (http://www.serenoa-fp7.eu/deliverables/):

- D2.2.1-CARFO (R1)
- D2.3.1 CARFO Population (R1)
- D2.3.2 CARFO and CARFO Population (R2)

3.4.3 Integration

CARFO Knowledge Base is the main source of contextual information, that is well-structured by the CARFO ontology. The CKB can be used as support for advanced adaptation of SFEs for different target platforms. To accomplish this goal, it is well-integrated with other modules in the Serenoa framework:

- *Authoring Tools:* it would be useful that users/developers knew the capabilities (components, modalities, restrictions, etc.) each RUIGE sub-module has. It would favour a good design of the applications and thus, effectively plan how to cover their development needs. CARFO may be queried for authoring tools or any other supportive tool in order to obtain this information.
- *RUIGE sub-modules:* CARFO knowledge could be used by RUIGE sub-modules to provide useful information for the transformations at runtime.

3.5 Languages: ASFE-DL

3.5.1 Description

The Advanced Service Front End Description Language (ASFE-DL) for Serenoa provides a unified description of the User Interfaces (UIs) that are exploited by a user in order to access the functionalities offered by Web Services and similar technologies. The language provides an abstract description of the UI, and two concrete descriptions (graphical and vocal) according to the CAMELEON Framework (Calvary, et al., 2003). The abstract description represents the user interface without taking into account a specific device or interaction modality.

3.5.2 Use

A UI is contained into an *AbstractUIModel*, which is composed of three main parts:

- *DataModel*, which contains the specification of the data types manipulated by the UI.
- *AbstractInteractionUnit(s)*, which contains the specification of the UI structure. Different interaction units are connected through *Connection* elements. An *AbstractInteractionUnit* contains:
    - *AbstractInteractor*s, which define generic interactors. An abstract interactor belongs to one of the following subclasses: *Selection* (for expressing the selection of one or more values in a predefined set of choices), *Edit* (for the input of manually edited values), *OnlyOutput* (for presenting information to the user) and *Control* (for performing actions, such as triggering functionality or submitting data).
    - *AbstractRelationship*s, which represent a generic relationship among the contained interactors. Different types of relationships are modelled in the language, such as *Grouping* (a generic group of interactors), *Ordering* (a set of interactors that has an order among the elements), *Hierarchy* (a set of interactors where different levels of importance can be identified), *Repetition* (a template for a list of interactors to be repeated for representing a dynamic list of items) and *Dependency* (a dependency relation between one interactor or interactor group and a set of N interactors or interactor groups).
- *AbstractDialogModel* contains the specification of the UI behaviour in terms of abstract events and their composition.

In addition two refinements of the abstract ASFE-DL have been carried out: for the graphical modality and for the vocal one.

For further information (http://www.serenoa-fp7.eu/deliverables/):

- D3.2.1-ASFE-DL: Semantics, Syntaxes and Stylistics (R1)

- D3.2.2-ASFE-DL: Semantics, Syntaxes and Stylistics (R2)
- D3.2.3-ASFE-DL: Semantics, Syntaxes and Stylistics (R3)

3.5.3 Integration

The ASFE-DL is a key component in the Serenoa Framework, because it provides the description of the structure and the behaviour of the UI. It is exploited by different modules which implement the adaptation. The ASFE-DL has been provided with XML syntax for different tools to exploit the models.

First of all, the language is the UI definition format used by the Authoring Tools, which help designers and developers to create context-sensitive SFEs. Therefore, all the authoring tools are able to save models developed according to the ASFE-DL syntax and to load them from an XML file.

In addition, the different RUIGE modules start from such description as a starting point for the generation of running application. Indeed, all RUIGE modules are able to transform the abstract model described in ASFE-DL towards a concrete representation, which is exploited for the final UI generation.

Finally, the Adaptation Engine is able to perform UI adaptations that are defined as modification of properties of some ASFE-DL modelling elements. Such kinds of actions are defined into the rules that the Adaptation Engine may select at runtime according to the context state. They are defined through the AAL-DL format (see section 3.6).

3.6 Languages: AAL-DL

3.6.1 Description

The Advanced Adaptation Logic Description Language (AAL-DL) is a high-level declarative description of the adaptation logic for context-sensitive applications. It mainly defines the reaction that the interactive application should perform in order to adapt itself according to a set of contextual events. Such kinds of transformations occur when a specific situation takes place at both the application and the context level. AAL-DL allows the definition of adaptation rules in a format described in section 3.6.2. It is possible to specify first-order rules that, according to the specified situation, directly modify the interface state. In addition, it is also possible to specify second-order rules, which are able to activate other rules if the specified situation occurs.

The language defines the adaptation logic according to an Event-Condition-Action (ECA) format and in its specification refers to other models (e.g. UI or context) in order to describe which elements of the application are affected by the adaptation process. Despite the fact that it has been designed to work with the models exploited in Serenoa, the language is able to include references to models developed outside the project.

3.6.2 Use

An AAL-DL model contains a set of *Rule*s, each one defining a different reaction for a given situation. A *Rule* consists of three parts:

- *Event*, which defines the type of notification that has to be listened in order to trigger the adaptation logic (e.g. a change in light conditions).
- *Condition*, which defines a predicate that has to be verified on the models state in order to perform the adaptation (e.g. whether the user is visiting a specific page).
- *Action*, which is the list of modifications that has to be performed in order to implement the adaptation logic. In such part of the language it is possible to indicate modifications of models that have been specified through the ASFE-DL.

For further information (http://www.serenoa-fp7.eu/deliverables/):

- D3.3.1-AAL-DL: Semantics, Syntaxes and Stylistics (R1)
- D3.3.2-AAL-DL: Semantics, Syntaxes and Stylistics (R2)

3.6.3 Integration

The language contains the information about the adaptation logic, which is exploited by different modules of

the Serenoa Framework and represents the core of the context-sensitive applications.

First of all, such logic needs to be defined. This task is assigned to the Authoring Tools, specifically to the Eclipse plug-in, which is able to create models according to the AAL-DL definition and to serialize and de-serialize them using its XML syntax.

Once the rules have been created, they need to be interpreted during the runtime execution of the application. This task is performed by the Adaptation Engine, which reacts to the changes occurring in both the application interface and in the context, selecting the rules that need to be executed and sending the set of updates and modifications to the different RUIGE modules. A more detailed description of this process can be found in section 3.9.

3.7 Agile methodology

3.7.1 Description

Software Development Life Cycle (SDLC) for adaptive UIs (e.g. Serenoa application prototypes), including requirements, scenarios, adaptation use cases, design prototypes and technical implementation, should be agile and user centred. This section describes the fusion of these methodologies applied to the gathering, design and implementation of adaptation rules.

3.7.2 Use

Research projects in general, and particularly Serenoa, are characterized by fast changing requirements where development teams are required to build complex software.

For further information (http://www.serenoa-fp7.eu/deliverables/):

- Deliverable D5.2.2 Application Prototypes (R1)

3.7.3 Integration

The integration of the agile methodology is not mandatory. Those partners who have developed Serenoa application prototypes were free to try the integration of the agile methodology.

3.8 Runtime UI generation engine

3.8.1 Description

The main idea of the Serenoa framework is to provide context-aware user interfaces. One of the key components of this framework is the Runtime User Interface Generation Engine (RUIGE). The goal of this module is to generate, deploy and execute applications taking into account the Delivery Context (DC).

RUIGE has a modular structure and it is composed of different sub-modules, each one in charge of generating applications adapted to the context. Various modules have been defined in Serenoa, although new sub-modules might be added to the architecture in order to support additional interaction modes and target platforms for the same application definition. So far, the defined modules are:

- ***Mobile Web RUIGE module****:* the aim of this module is to generate mobile web applications for a large number of web-enabled mobile devices in the market.
- ***MARIA RUIGE module****:* this module is in charge of generating multimodal Web user interfaces implemented using HTML5, CSS3, Javascript and JSP and Servlets (if access to remote Web services is necessary).
- ***UsiXML RUIGE module****:* it allows generating HTML and Java Swing applications.
- ***Avatar RUIGE module****:* it generates a high-level avatar, rendered using a 3D engine running as an ActiveX component inside a web page and a lower-level one based on HTML5 technology.
- ***Leonardi RUIGE module****:* it is able to produce UIs for different contexts. The generator could provide graphical screens based on a great variety of technologies as swing, AJAX, HTML5, Android or iOS.

The creation of applications involves a series of stages starting from the definition of the application in an

abstract language (ASFE-DL) and transforming it to the Concrete User Interface (CUI) language. Then, CUI is converted to executable or interpreted content and, finally, RUIGE will render the Final User Interface (FUI) at runtime using the previously generated code.

3.8.2 Use

RUIGE is aimed to generate, deploy and execute context- aware applications. This module is able to build applications from the same logical description that defines the application in an abstract way (input) for one or more target platforms (outputs).

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

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

As it was previously commented, it is necessary to introduce some adaptation rules to be considered during the application lifecycle. In order to clarify how RUIGE works in terms of the adaptation process, some rules coming from the Adaptation Engine are defined and applied in some of the RUIGE sub-modules:

- **Adaptation to device features**:

R1: If the device is a tablet, then master-detail presentations will be rendered in one single view.

This rule must be considered during the transformation process when converting from ASFE-DL (AUI) to the concrete description by means of an XSLT. This XSLT generates two different concrete versions description. In the case of tablets, a master-detail view is expressed in just one presentation. Otherwise, in case of mobile devices, the same view is expressed in two different presentations, plus the additional flow logic. In order to carry out the adaptation, the mobile web runtime sends the User-Agent to the Context-Manager to identify which kind of device is accessing to the web.

- **Adaptation to the user profile**

R2: If user is color-blind, then it will use an alternative color palette in the referenced images.

Rule 2 is considered at execution time by the runtime module in order to dynamically adjust the color palette while the user is interacting with the application. The Context Manager handles user profiles in order to recognize special user features as color-blindness.

- **Adaptation to the status of the device**

R3: If the level of the battery is higher than a prefixed threshold, then a video must be shown but if its level is lower, it renders an image.

As in R2, R3 must be considered at execution to dynamically adjust the video/image while the user is interacting with the application. In order to get the level of the battery a context delegate was built.

- **Adaptation to the position of the user**

R4: If the user is far away from the device, then bigger font-size will be shown but if the user is closer, the font-size will be smaller.

R4 must be considered by the generator during the generation process when converting from the concrete decsription to HTML5 + CSS3 + Javascript (FUI). In this case, the adaptation rule is considered in the generation process to determine the Javascript code that should be delivered. However, the adaptation will be carried out in the runtime process when the Javascript code is executed. In order to change the font-size, the frontal-web camera is used by means of webRTC[4]. This API allows calculating the distance between the screen and the user and this value is periodically sent to the server to update the font-size.

For further information (http://www.serenoa-fp7.eu/deliverables/):

- D4.1.1-Runtime UI Generation Engine (R1)
- D4.1.2-Runtime UI Generation Engine (R2)

3.8.3 Integration

The functionality of RUIGE is made possible with the collaboration between the modules of the Serenoa framework, such as the CARFO Ontology, the Adaptation Engine (AE) and the Context Manager.

One of the modules communicating with RUIGE is the Adaptation Engine. It is responsible of gathering all the information from the rest of modules to manage the adaptation process. RUIGEs just need to interact with the AE, avoiding the establishment of direct communications with CARFO or the Context Manager.

There is also a relation between RUIGE and some abstract languages as:

- *ASFE-DL*, the language used to define the AUI
- *AAL-DL*, the language that describes the adaptation rules that need to be applied during the adaptation process.

3.9 Adaptation engine

3.9.1 Description

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 and the adaptation rules for a Serenoa application, the context in which it is being used, and the knowledge about adaptation represented in the CARFO ontology. Finally, it selects the rules to be performed at runtime in order to execute the adaptation process.

The Serenoa framework uses a Service Repository which is in charge of storing the service descriptions (high level definition of the UI and adaptation rules) defined by the developers through the Authoring Tools. This repository is accessed by the Adaptation Engine (see detail in Figure 5) in order to retrieve services information and merge it with the rest of context-aware data (i.e. adaptation knowledge and context information).

[4] WebRTC web site: http://www.webrtc.org/

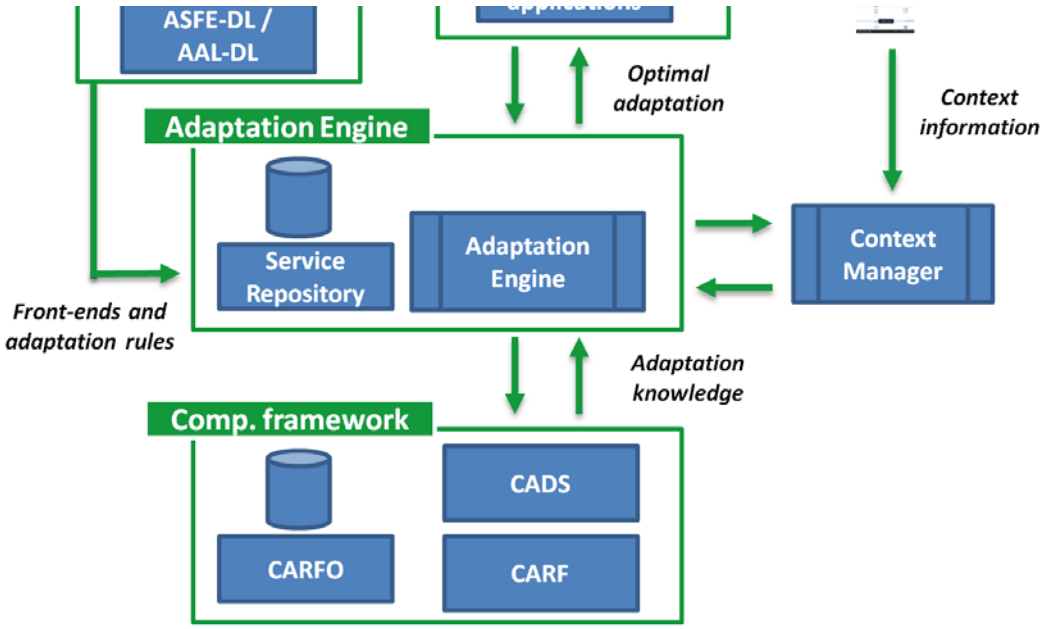

Figure 5: Detail of Serenoa architecture for the Adaptation Engine module

3.9.2 Use

In this section, the Service Repository and the Adaptation Engine are described in terms of functionality for the developers of Serenoa-based prototypes.

3.9.2.1 Service Repository

The Service Repository is based on a REST service in charge of managing a database of the services, their descriptions and the important associated meta-data (developer name, ids, etc.). Specifically, the information which is kept in this service is the following:

- *service_id:* a unique identifier for each service. It is automatically assigned.
- *service_name:* the name of the service.
- *developer_name:* the username of the developer who has uploaded the service description to the repository.
- *asfedl:* the abstract description of the service.
- *aaldl:* the user-defined rules corresponding to the service.

Next a compilation of the most useful resources provided by the Service Repository REST interface is presented. The basepath of the REST service is as follows: http://195.235.93.35:8080/SerenoaAE_M18/rest

Get a service description

It gets the high-level description (i.e. ASFE-DL) and the adaptation rules (i.e. AAL-DL) associated to a service, jointly with the developer name. This resource could be used by the Authoring Tools in case a developer wants to retrieve the version stored in the repository, or also by some runtime modules which want to access to the service description asked for a specific user.

- URL: http://195.235.93.35:8080/SerenoaAE_M18/rest/services/{service_name}
- HTTP method: GET

- Parameters:

Parameters	*Description*
{service_name}	The name of the requested service. Example value: 'Car Rental'

- Response format:

The response follows the following format:

```
<result action="getService">
<service_name>...</service_name>
<username>...</username>
<asfedl>...</asfedl>
<aaldl>...</aaldl>
<code>...</code>
</result>
```

- Sample request:

GET http://195.235.93.35:8080/SerenoaAE_M18/rest/services/CarRental

Create a new service description

It allows creating an instance for a non-existing service and populating it with appropriate information such as the name of the developer or the description of the service. In this case, the resource will be mainly used by the Authoring Tools when a developer wants to load a new implemented project in the Service Repository.

- URL: http://195.235.93.35:8080/SerenoaAE_M18/rest/services/{service_name}
- HTTP method: POST
- Parameters:

Parameters	*Description*
{service_name}	The name of the service which wants to be created. Example value: 'Car Rental'
user_name	The name of the developer. Example value: 'Albert Einstein'
asfe_dl	The abstract description of the new service. Example value: '<abstractUIModel xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"...'
aal_dl	The user-defined rules. Example value: '<rule><event><simple_event event_name="onRender"...'

- Response format:

```
<result action='create_service'>
<service_name>...</service_name>
<code>...</code>
</result>
```

- Example request:

  POST http://195.235.93.35:8080/SerenoaAE_M18/rest/services/CarRental

  POST Data: user_name, asfe_dl, aal_dl

Update an existing service description

The resource is intended to edit an existing service description. In case developers want to modify the abstract definition of the interface or the user-defined rules, this resource should be used.

- URL: http://195.235.93.35:8080/SerenoaAE_M18/rest/services/{service_name}
- HTTP method: PUT
- Parameters:

Parameters	*Description*
{service_name}	The name of the service. Example value: 'Car Rental'
asfe_dl	The abstract description of the new service. Example value: '<abstractUIModel xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"...'
aal_dl	The user-defined rules. Example value: '<rule><event><simple_event event_name="onRender"...'

- Response format:

```
<result action="updateService">
<code>...</code>
</result>
```

- Example request:

  PUT http://195.235.93.35:8080/SerenoaAE_M18/rest/services/CarRental

  PUT data: asfe_dl, aal_dl

3.9.2.2 Adaptation Engine

The Adaptation Engine is also interfaced by a REST service. It 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.) Then the Adaptation Engine resolves the most suitable adaptation strategy to be performed over the service front-end, taking into account the context information and the high-level description of the service.

The Adaptation Engine offers the adaptation in function of the user and the platform. These two cases are going to be described next.

Adaptation to users

This resource is intended to get the user's information (via Context Manager) and then interpret the service's logic in order to provide the adaptation strategy.

- URL: http://195.235.93.35:8080/SerenoaAE_M18/rest/service/{service_name}/aIU/{interactor_id}/user/{user_id}
- HTTP method: GET
- Parameters:

Parameters	*Description*
{service_name}	The name of the service. Example value: 'Car Rental'
{interactor_id}	The id corresponding to the element where the interaction is happening.
{user_id}	The name of the user which has accessed to the service.

- Response format:

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

- Example request:

  GET http://195.235.93.35:8080/SerenoaAE_M18/rest/service/CarRental/aIU/resultsList_iu/user/ann

Adaptation to platforms

This resource is intended to get the device features (e.g. screen resolution) and current status (e.g. battery level) and then interpret the service's logic in order to provide the adaptation strategy.

- URL: http://195.235.93.35:8080/SerenoaAE_M18/rest/service/{service_name}/aIU/{interactor_id}/device_category/{device_id}
- HTTP method: GET
- Parameters:

Parameters	*Description*
{service_name}	The name of the service. Example value: 'Car Rental'
{interactor_id}	The id corresponding to the element where the interaction is happening.
{device_id}	The name which describes the category of the device which is being used by the user (e.g. tablet)

- Response format:

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

- Example request:

  GET http://195.235.93.35:8080/SerenoaAE_M18/rest/service/CarRental/aIU/resultsList_iu/device_category/tablet

For further information (http://www.serenoa-fp7.eu/deliverables/):

- D4.3.1-Adaptation Engine (R1)
- D4.3.2-Adaptation Engine (R2)

3.9.3 Integration

The Adaptation Engine module is integrated with several Serenoa components:

- *Authoring Tools:* they should be able to modify and retrieve information about the services. In order to do this, they query the Service Repository service as indicated in section 3.9.2.1.
- *RUIGE sub-modules:* the runtime adaptation pipelines ask the Adaptation Engine module in order to obtain a suitable adaptation to the user's context (see section 3.9.2.2).
- *Context Manager:* the Adaptation Engine gets the context information querying this module. Afterwards, the data is used for deciding the optimal adaptation strategy. Context Manager keeps

updated different context variables such as user preferences, battery level, etc.
- *Ontology:* during the adaptation decision process, the Adaptation Engine handles with concepts that are formalized in this knowledge base. For example, it could be used for describing what kind of platform is a tablet.

The flow of information between these modules is represented in Figure 5.

3.10 Context manager

3.10.1 Description

The Context Manager (CM) includes a set of functionalities for tracking and storing the changes to the different dimensions of the context. This module allows the Adaptation Engine to select the most appropriate adaptation logic to be performed according to the current context state. It has the responsibility to gather, store and update the context-related information, sending notifications to the other modules that may listen to changes of a given context dimension with a publish-subscribe mechanism.

The CM Core module consists of three main components:

- *Data Store*, which contains the information of context entities (see section 3.10.2) and allows the performance of different operations on them, such as inserting a new one, querying the current value, updating the current value, removing one, clearing the current value, subscribing or unsubscribing for value updates.
- *Context Manager Interface*, which allows external modules to invoke the Context Manager functionality through simple HTTP requests (e.g. by posting XML commands) or *RESTfully* (i.e. by invoking RESTful web services).
- *Data Manipulation*, which periodically checks the consistency and the expiration of the stored entities.

Information addition/update is mainly possible by means of specific CM Components, referred to as *Context Delegates*. Such modules are installed on users' devices, gather the information directly from the device and sends the updates to the CM Core.

3.10.2 Use

The CM module stores information according to a specific Context Model. The specification of the context consists of different elements:

- *Users.* For each of them, it is possible to store preferences, tasks, disabilities, position and knowledge.
- *Environment.* It is possible to maintain light and noise conditions.
- *Technology*. It is possible to store the type of the device, the type of connection and browser(s) available.
- *Social relationships*, that can be modelled through the definition of different groups, identified by the list of users belonging to them.

We include here an example of how it is possible to communicate with the Context Manager at runtime. In order to create, modify, cancel or query an entity, the external modules can access the Context Manager by posting (i.e. via HTTP POST request) an XML command message to the HTTP interface (implemented by a Servlet), or by invoking one of the methods of the RESTful Web service interface.

Examples

- The addition of the entity for user "mario" via HTTP can be done by forwarding the following message as POST content (the context message is enclosed on a "comet" tag, where comet stands for COntext ManagemEnT):

```
<comet>
<op>insert</op>
<entity_type>generic_user</entity_type>
<entity_ttl>0</entity_ttl>
```

```
<entity_data>
<user_name>mario</user_name>
<user_location>office</user_location>
</entity_data>
</comet>
```

- The current representation of the previously created/updated entity for user “mario” can be obtained by querying the Context Manager through the message:

  ```
  <comet>
  <op>query</op>
  <entity_id>[id]</entity_id>
  </comet>
  ```

  (where “id” is the entity identifier obtained at entity creation time)

  or through the REST method by performing a GET on:

  http://context-manager-url/rest/user/**mario**

The reason why the same operations might be performed in different manners is that different modules might need to access the Context Manager. Thus, each module would exploit the most convenient way to perform the needed operation. For instance, a module devoted to low level operations (e.g., coping with entities, and thus aware of entities identifiers) might exploit the low level HTTP simple interface to directly query/modify entities. Modules unaware of entities identifiers (e.g., applications that only know e.g. user name) would instead use RESTful web services to add, modify, retrieve resources (which refer to actual entities, but that hide their peculiarities).

For further information (http://www.serenoa-fp7.eu/deliverables/):

- D4.4.1-Context of Use Runtime Infrastructure (R1)
- D4.4.2-Context of Use Runtime Infrastructure (R2)

3.10.3 Integration

The Context Manager (CM in the following) is currently integrated with the Adaptation Engine, which is able to receive updates about events associated with changes of entity values.

Several integrated prototypes, developed by Serenoa partners, already exploit the CM:

In the *Warehouse* multimodal application, developed jointly by SAP and ISTI, interface adaptations are triggered according to the current context of use (e.g. user position with respect to the available shelves, environment noise, etc.). The Warehouse application is Web-based and multimodal, and is interfaced to the Serenoa framework through an intermediate module. Such module, embedded in the application, is able to asynchronously receive updates from the Adaptation Engine. Such updates are sent when it is asynchronously notified about relevant context changes by the CM. The context changes relevance depends on the events the Adaptation Engine has previously subscribed, that are those specified on the adaptation rules associated to the current user/application.

In the *E-Health* application, developed by TID, the CM is used for two reasons: store user's preferences and retrieve user's context information. User properties are dynamically stored into the CM, although some of them may not change over time. For instance, the user is able to set the language of the interface, which is supposed not to change during interaction. But s/he is also able to show or hide the avatar, and this preference may often change over time due to specific conditions.

The prototype also retrieves context information, such as the initial noise level when the user logs in. If the noise level is too high, for instance, the avatar is automatically muted. The E-Health prototype is able to subscribe to the CM in order to receive events when the context changes, and to dynamically adapt the prototype during user session.

The *E-Commerce* Scenario, tackled by W4, implements a runtime integration with the CM. The CM integration is used in two layers of the scenario and communication between the CM and the web :

- The customer front end for people who want to check the catalogue of products, the price and the products availability: based on the user context (desktop, mobile, colour blinded user, visual disabilities), the web application stores relevant information into the context manager to determine which transformation rules should apply
- The employees of the company use the back office application. The employee profile and preferences (preferred language, known disabilities, preferred colours, platform…) are also set within the CM.

On user interactions (login, click on a preference switch button, change of page or heading…), the e-commerce application may update (if applicable) the context manager. When context changes, the application shall query the adaptation engine with the context identification in order to get the adaptations rules which are applicable for this new context. Finally, when one or several rules have changed the graphical user interface is updated to reflect this new contextual rule.

The integration of the CM within the e-commerce scenario is based on the architecture below:

1) Serenoa adaptation rules are stored in the Service Repository
2) When a user (A) accesses the public product catalog, or when an employee (B) connects to the intranet back office application a Context manager is initiated and all information is stored in the CM by using its services.
3) Prior rendering the HTML page, the web application from the e-commerce server establishes a connection to the Application Engine, specify current context, and attempts to fetch one or several applicable adaptation rules.
4) If some rules are found (for instance to apply colour blinded adaptations and use mobile phone adaptations), then the content of the page is modified in order to fulfil requested adaptations.
5) When the user navigates to another page, clics on a configuration button, logs in for instance, context may change, the context Manager is updated with new context data. The web application repeats step 3) to get applicable adaptation rules.

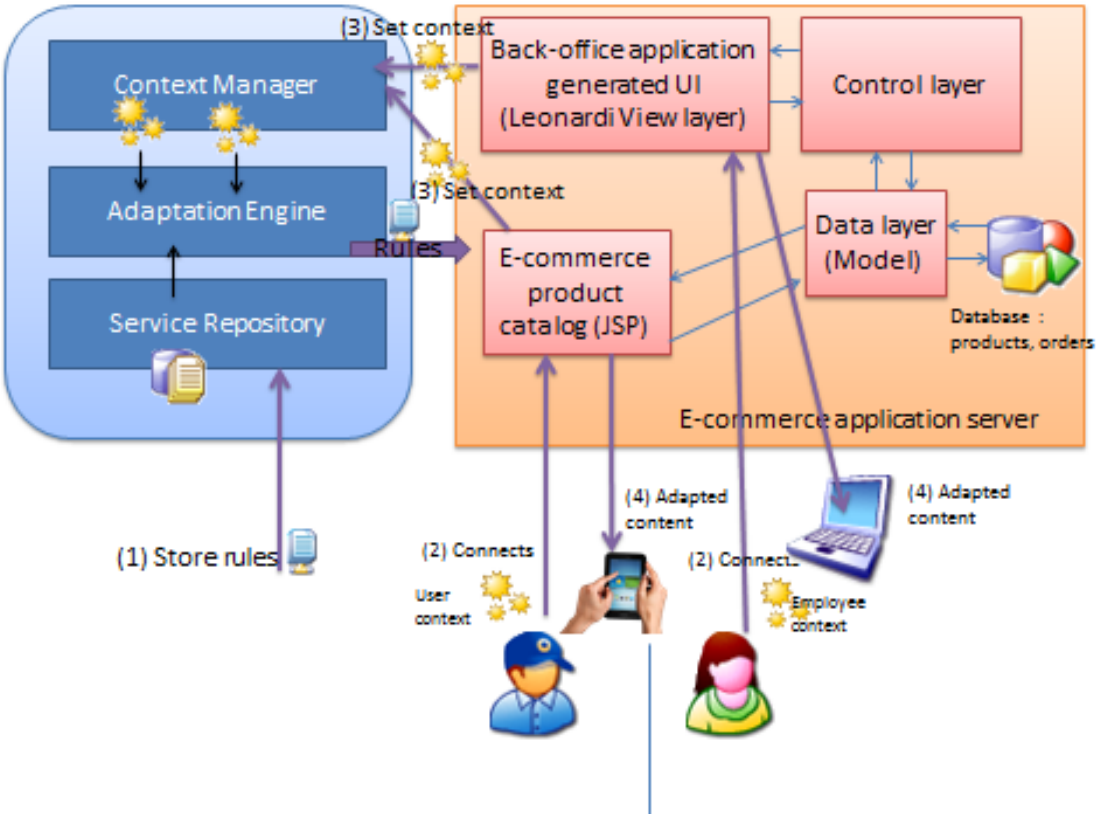

Figure 6 E-Commerce scenario architecture

3.11 Authoring tools: Quill

An authoring tool helps the prototype developers, engineers and web authors to easily create context-sensitive SFEs for different platforms, which may also use different interaction modalities, e.g. graphics, voice, touch etc. The authoring tools are based on widely used development environments i.e. Eclipse, and an HTML5 browser-based thin client application. The authoring tools have been developed in a way that they are understandable for non-expert programmers, and fulfil some additional key success factors. Amongst these factors, the usability of the graphical interface, the availability on multiple platforms and the support for concurrent work of several users (engineers and web authors may work as a team) appear as very important requirements. The authoring tools provide support for editing not only the model-based descriptions at abstract, concrete and final levels, but also the context-dependent transformations rules.

The current section and the next one will cover the description, integration and use of the editing tools which have been developed in Serenoa project.

3.11.1 Description

Quill is an experimental design tool intended for distributed editing of model-based user interface designs. It consists of a web client implemented in HTML5 that runs in modern web browsers, and a server implemented in Node.js. The HTML5 *canvas* element is used for graphical models and WebSockets for communication with the server.

A server-based design-time adaptation engine allows designers to work at different levels of abstraction, synchronising changes in a cooperative workflow (see Figure 8 and Figure 9).

3.11.2 Use

Quill organizes work into projects, and holds the project data on a Web server. Multiple people can work concurrently on the same project, in an analogous way as Google Docs. Each person can only view and edit

one layer of the CAMELEON reference framework at a time. A novel feature of Quill is the ability to work top-down or bottom-up. Previous work on model-based UI design tools has focused on top-down generation of models, but required manual intervention to propagate changes at lower-levels upwards in the abstraction stack defined by the Cameleon reference framework. This has perhaps unsurprisingly proved to be technically very challenging, and we have been exploring how to realize this as a rule-based design assistant using constraint propagation as a means to limit search through the design space to solutions that match the constraints provided by the human designers at each level of abstraction.

As an example, a designer could choose to work at the concrete UI layer on models for a desktop interface. The server-based design assistant will then automatically update the models at the abstract UI layer, and from there propagate design changes to models at the concrete UI layer for other chosen target platforms such as mobile or TV. The adaptation rules can add tasks to Quill's Design Agenda, for the human designers to deal with. This is necessary when the adaptation process requires human intervention. Quill has a modular design that separates out the authoring user interface for each layer, and makes it easy to switch the visualization used at each layer of abstraction. Layout algorithms are used to adapt the visualization to the author's browser window size, and authors working on the same project will see their own visualization.

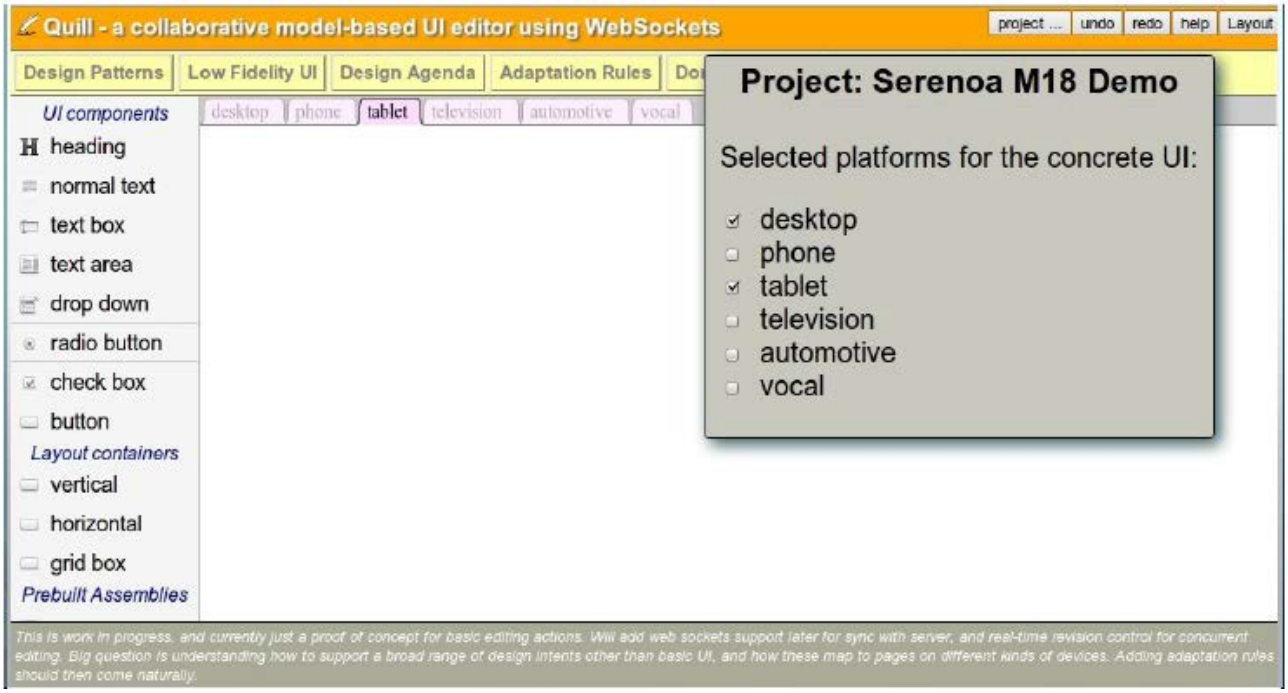

Figure 7Selection of target platforms for a project

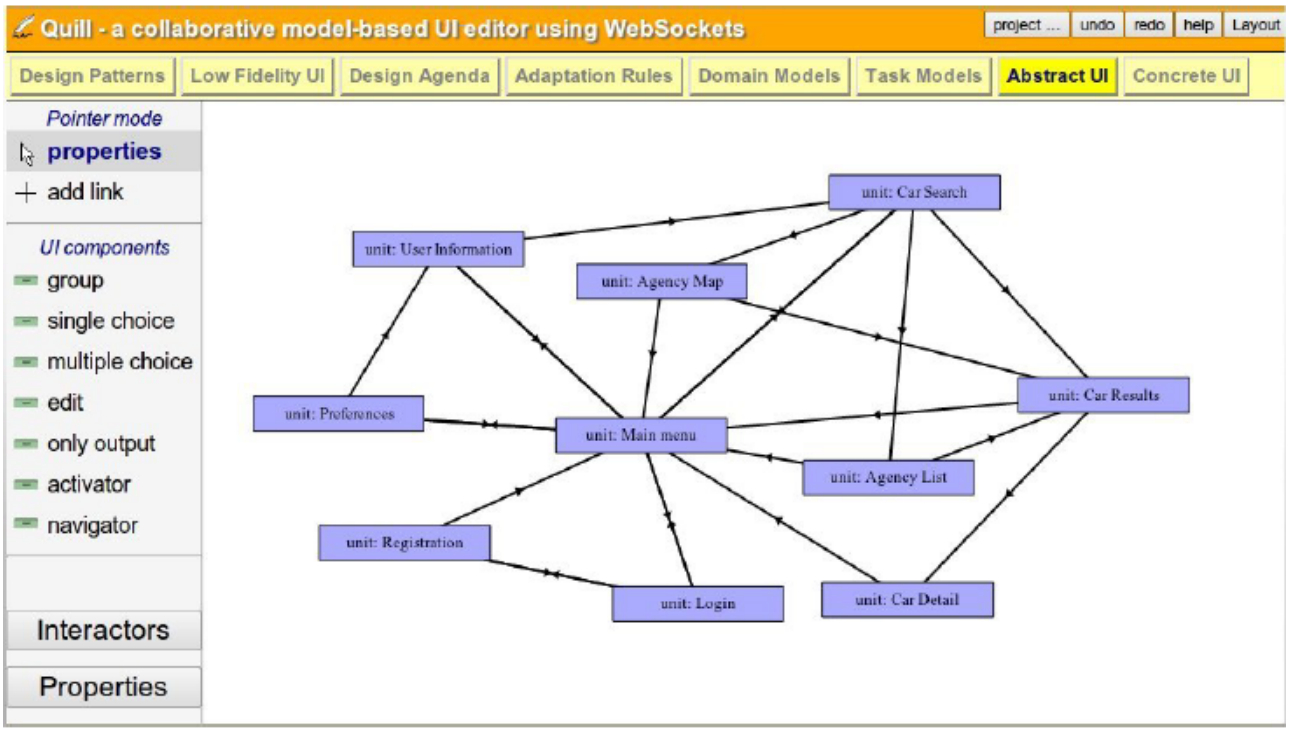

Figure 8: Abstract UI edition with Quill

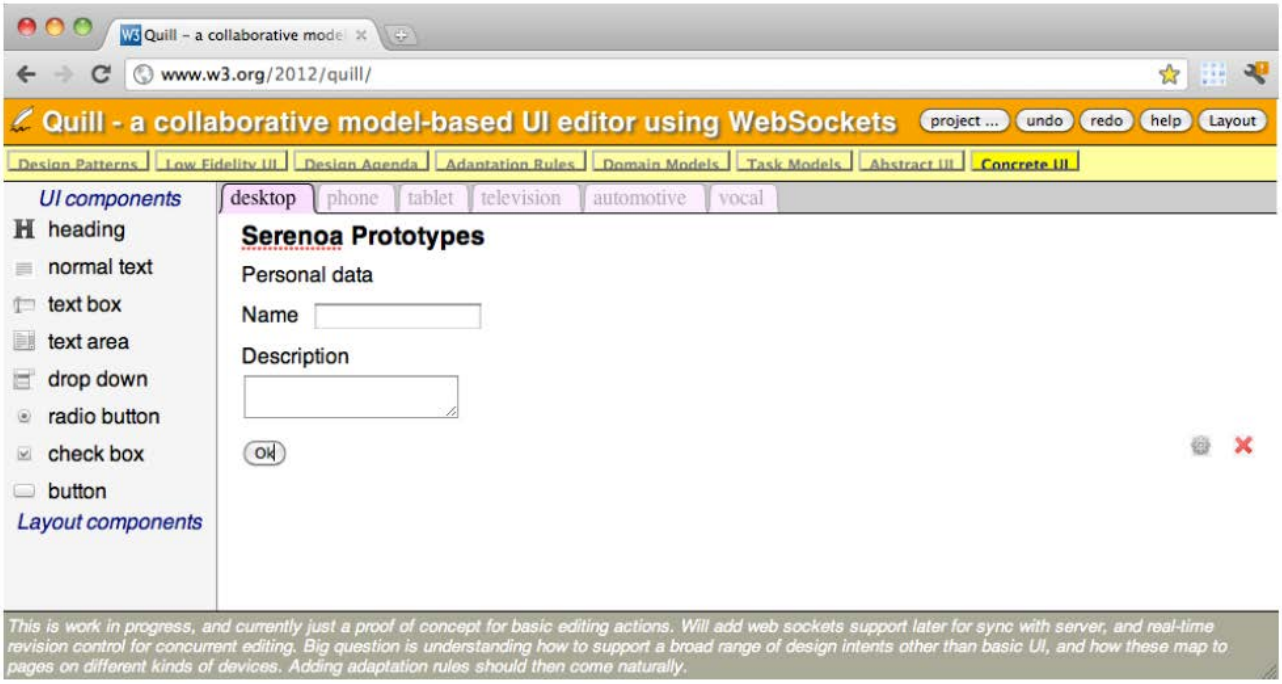

Figure 9: Concrete UI editing with Quill

3.11.3 Integration

Quill is work in progress and a long way from a finished product. The challenges for synchronizing changes at different levels of abstraction have turned out to be much harder than initially anticipated and provide opportunities for further study in future work on model-based UI design. This necessitates finding real-world examples and applying knowledge engineering techniques to discover the rules needed for generating design solutions. Current work is focusing on understanding how to apply constraints to reduce search through the design space. One technique of interest is the use of abduction where missing facts can be inferred if given

relationships across different levels of abstraction are to hold true. There can be multiple solutions depending upon the *ab initio* facts. The XML formats chosen for Serenoa have proved to be cumbersome, and this has prompted the use of more concise text formats and object models inspired by W3C's use of WebIDL (Web Interface Definition Language). An early snapshot of the Quill demo can be found at http://www.w3.org/2012/quill/.

3.12 Authoring tools: Eclipse plug-in

3.12.1 Description

The implementation of the Eclipse plug-in supports the creation and editing of the abstract-level description of a UI model with the help of a *Service Editor*, and supports the creation of context-dependent transformation rules with the help of *Rules Editor*. This document briefly describes the Service front-end Editor.

3.12.2 Use

The ASFE-DL Editor is designed to have support for UI models editing and creation in both code and graphical view. The code view is displayed simply in an XML Editor as the model languages are based on XML as underlying language. The Graphical editor is developed using GEF which internally uses MCV architectural pattern and provides technology to create rich graphical editors and views for Eclipse Workbench UI. The schemas and model-based description of UIs provided by partners helped in designing the graphical part of model-based UI editor. The graphical part is designed to have a tool palette which contains elements list for creating UI models. The element list is divided into three categories as Connections, Grouping and Events (a structure defined for Abstract Interaction Unit). The drawing area is there to drag and drop elements from palette and displays the AIU in a graphical representation (in the form of boxes). This helps the developer to easily and quickly develop the Abstract UI model for Adaptive SFE applications. The editor can also display the Outline view in the form of tree structure and provides displaying and easy navigation feature for the AUI drawn in the drawing area. The property view shows the properties associated with each element that can be set in the fields. The miniature view of the drawing area can be seen giving the idea of graphical part when editing in the XML editor. The graphical editor also provides features like: (Selection, moving, resizing) of elements, Actions (like Redo, Undo, Delete, Zoom), keyboard shortcuts, context menu, drag and drop, cut and paste. Both XML editor and Graphical editors are synchronizes to reflect changes made at any time in their respective views.

Architecture

After studying the schema files for context dependent rules, a basic object model is created that will be used as a parent class for all model elements (*Node*). This class contains the basic properties that all derived classes are able to use. Next, a class derived from Node is written, being the topmost class in the hierarchy (*Abstract UI Model*). This class serves as a wrapper/container for all other elements. Based on the schema that defines the hierarchy of elements, each class is written defining its child/parent relations with other elements and its attributes. Each object in the model needs to be associated with a *Figure* (draw2d) and an *EditPart* (an *EditPart* is an object that will link its model object and its visual representation). A *Factory* is needed to manage the EditParts. A Factory is a class that will handle appropriate object creation (the EditPart) depending on what we want to obtain, no matter what is the object class.

GEF is based on MVC architecture. It means there is a distinction between model (elements) and view (hierarchical boxes). The Controller is the referee in the middle. It drives the view depending on the model and it modifies the model depending on actions carried on the view. Interaction with the box causes the controller to execute commands. These commands modify the model. Subsequently, the model informs the view that it has changed. Finally, the view can update itself.

Features of Graphical Editor Part

In the following we describe the features provided/supported by the graphical rules editor that facilitates the developers to rapidly develop context dependent rules.

Drawing Area

The drawing area contains and displays elements that are dragged from the palette and dropped inside it. Since the root element of the ASFE-DL file is "Abstract UI Model", the container (e.g. the canvas) is named the same. The elements are displayed in the form of a figure created using Draw2d which is a layout and rendering toolkit for displaying graphics on an SWT Canvas. The elements present inside the drawing area can be viewed in the *Outline* with hierarchical structure and can be saved to reflect changes in the ASFE-DL file. The canvas can be zoomed in and out using zoom feature.

Palette

A palette feature is added to the editor to deal with graphical edition and element addition. Tools are added to the palette to manipulate the Abstract UI Model and insert UI definitions into it. Palette is the part of Service Editor which lists the elements needed to create AUIs. The palette used in this graphical editor part is the Flyout palette which is an interactive palette having features of collapsing, palette drawer to categories the elements in folders and pin that folder containing elements. The elements are added to the palette after a careful study of the ASFE-DL schemas. Each element represents a class which is a model defining its attributes, parent, and children. The elements from the palette can be dragged and dropped in the drawing area.

Miniature View

A miniature view of the Abstract UI Model is added in the outline view. The miniature view displays a small graphical view of AUI drawn on a *canvas*. It is very practical when using zoom functions to focus on UIs. It also gives an idea of synchronization between XML editor and Graphical Editor.

Property View

The property view displays attributes associated with each element defined in the model. The values of these attributes can be set in the fields that are then updated in the model. The refreshed view is shown to the user with the XML editor elements updated as well.

Outline

This feature is a special Eclipse view called *Outline*. It is another view type for the graphical editor. It is added to show the figures (boxes) as a tree, providing easy navigation through elements.

Interaction with Elements: Select the Box, Move it, Resize it

GEF provides a visual representation of an object model. In graphical editor, the interaction with figures (boxes) can be done by selecting, moving and resizing them. In Service Editor, the topmost element is Abstract UI Model, which contains a list of AUIs including Connections, Grouping and Events. This feature allows a developer to move AUIs in the Abstract UI Model and Connections, Grouping and Events in their AUIs. It is also possible to select a box (even several with CTRL key), and to move and resize it.

Redo, Undo, Delete

The Redo, Undo, Delete support in the editor is a management functionality added to the figures (boxes) providing addition and removal of elements, handled fully by GEF. A toolbar is added in the editor that contains the redo, undo, and delete action buttons. A developer can delete the whole Rule, or just an element.

Zoom

It is a simple feature for zooming in the Rules (boxes) in the drawing area. Zooming is an action which is added to the toolbar just like Undo, Redo and Delete. It is a visual action without any impact in the model.

Keyboard Shortcuts

The majority of developers find keyboard shortcuts as a very useful feature. The graphical rules editor associates key presses to the actions by mapping keys (e.g., + and – for zooming, *Del* for removal, as well as *Ctrl-z* for *Undo* and *Ctrl-y* for *Redo*).

Context menu

A context menu bound to the right mouse click is added to the graphical editor in order to support *Undo*,

Redo and *Delete* actions to the menu.

Drag & Drop

The Drag & Drop feature provides the possibility to create Connections, Grouping and Events graphically by clicking and drawing these entities. It is user-friendly and is a must-have feature available in every so-called modern graphical editor. It allows the developer to select (drag) an element from the palette and put (drop) in the drawing area.

Cut & Paste

Additionally, there is another must-have functionality in every respectable graphical editor is the cut/paste feature. It provides fast editing and creation of AUI. The entire AUI can be copied and pasted within the drawing area along with all its attributes.

-

3.12.3 Integration

The implementation of the ASFE-DL Service UI editor supports Model-based description of UIs at Abstract-Level. The model language for Concrete- and Final UIs are at under development stage and will be provided by Serenoa project partners. The extension with abstraction of Concrete- and Final UIs will be the future work of ASFE-DL Service UI editor. On the other hand, a remote service repository is integrated (via REST interface) in the authoring tool to support the CRUD (Create, Read, Update, and Delete) operations on the service definitions and adaptation rules. Figure 10 shows how the authoring tools, being an integral part of *Design-Time*, play a key role in the overall Serenoa architecture.

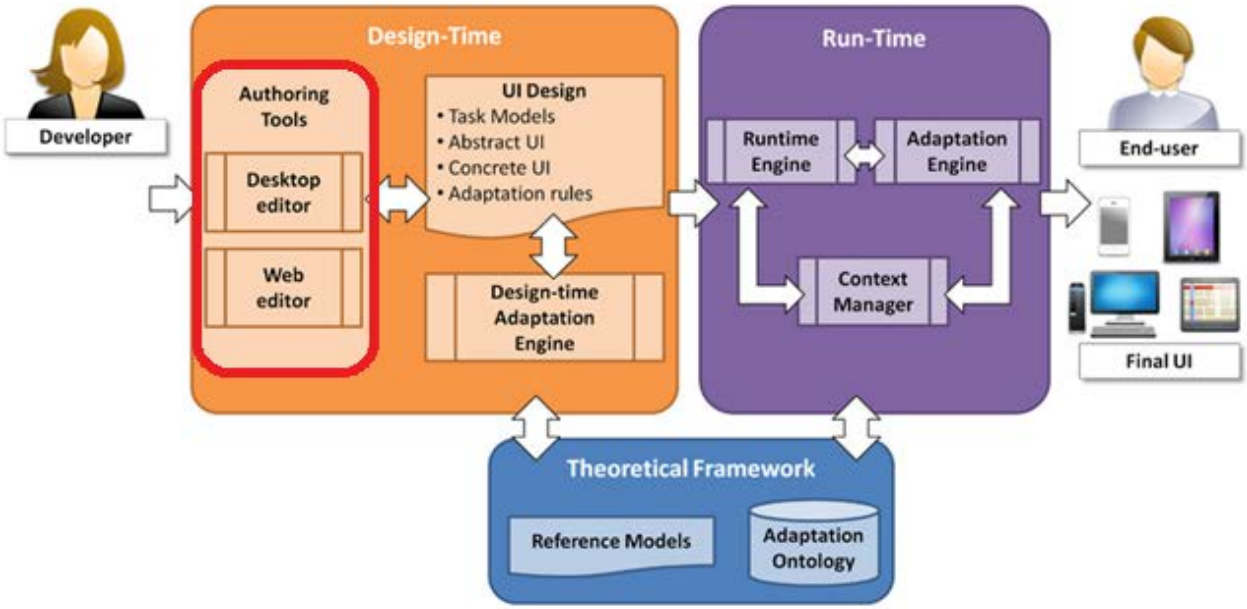

Figure 10. Design-Time Authoring Tools in overall Serenoa Architecture

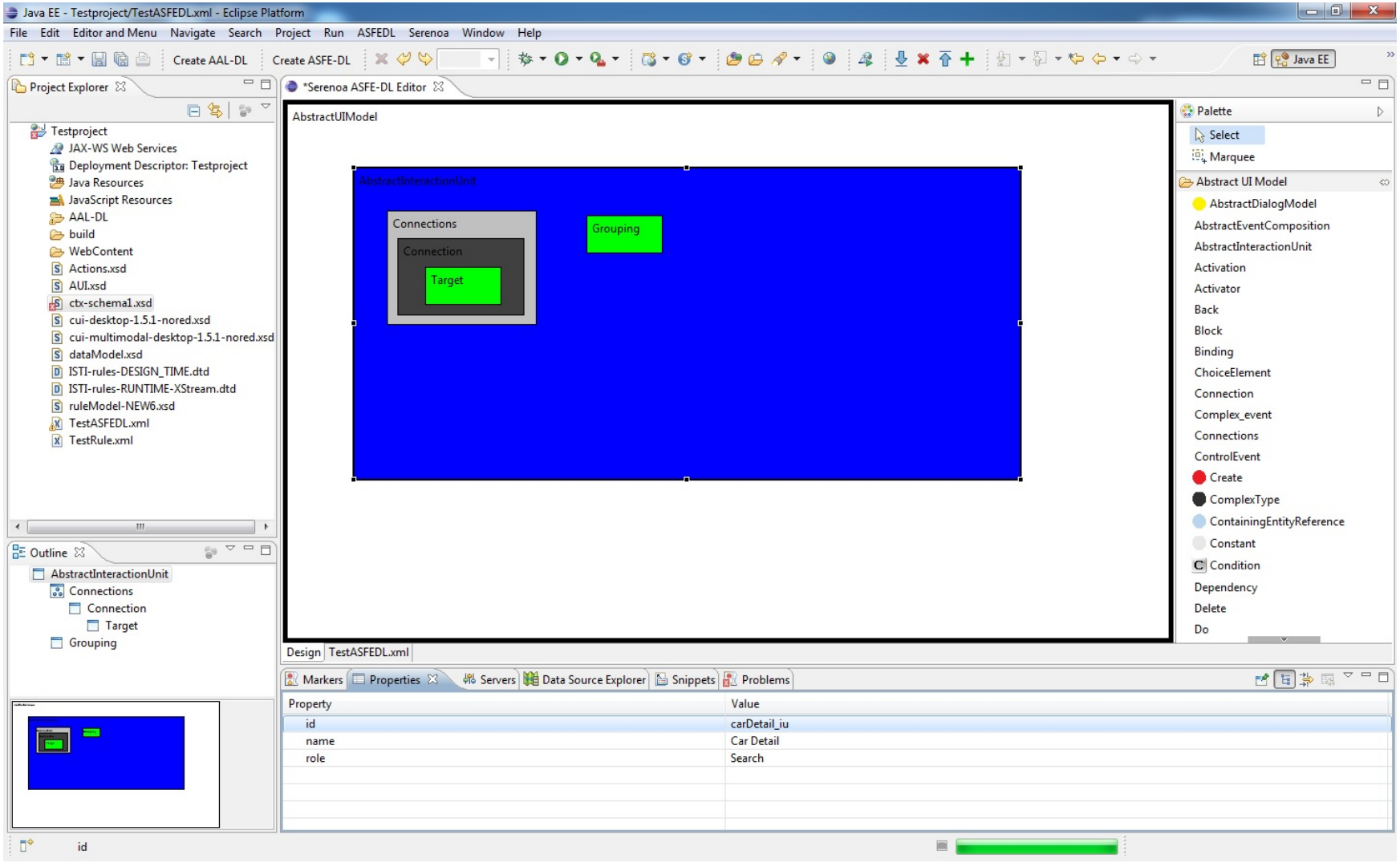

Figure 11. A Design View of ASFE-DL Service Editor

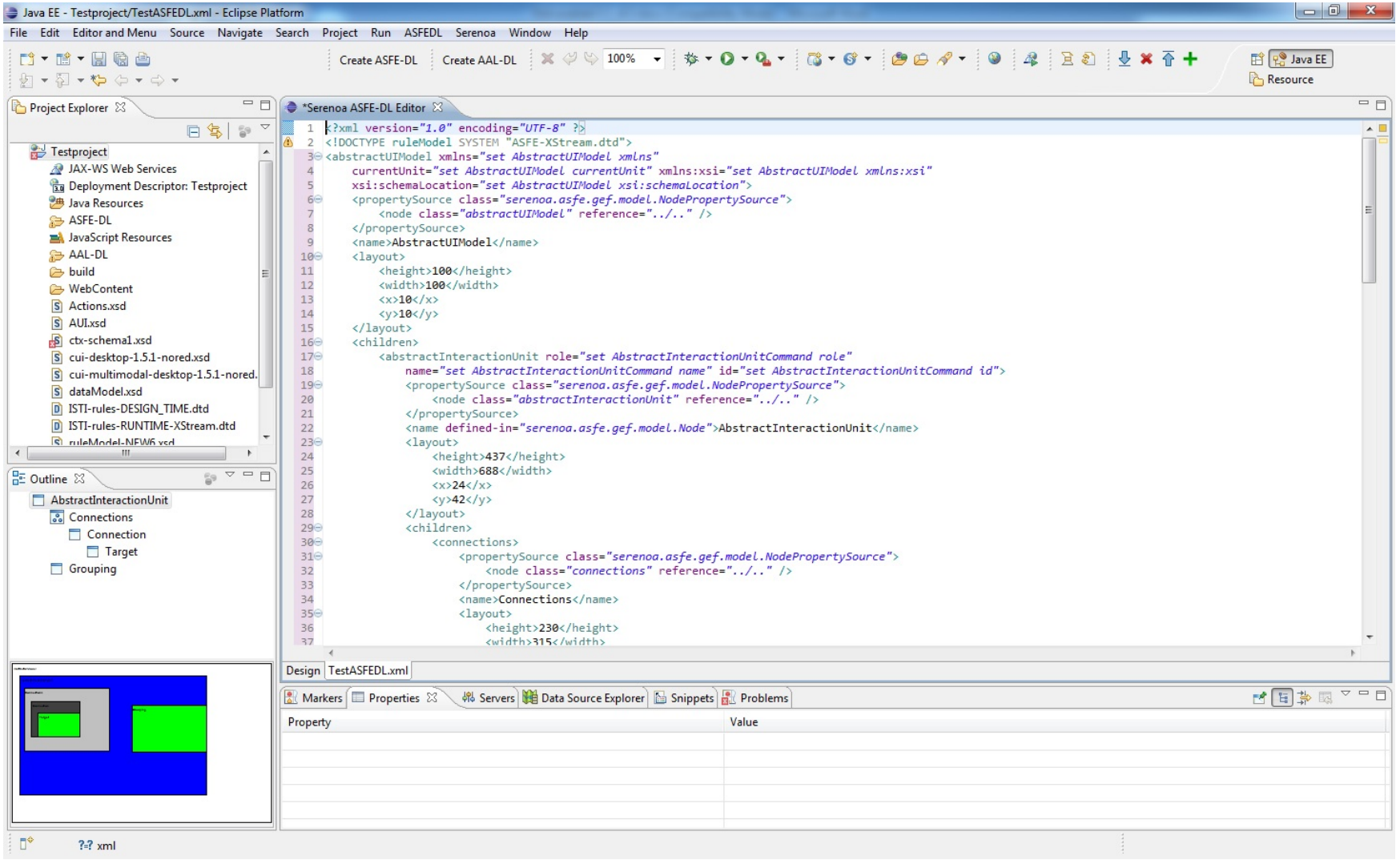

Figure 12. The Source/Code View of ASFE-DL Service Editor

4 Conclusions

In this deliverable we have described the collection of components which are included in the Serenoa framework, namely: Context-Aware Reference Framework (CARF), Context-Aware Design Space (CADS), CARF Ontology (CARFO), ASFE-DL, AAL-DL, Agile methodology, Runtime UI generation engine (RUIGE), Adaptation engine, Context manager, Quill editor and Eclipse plug-in.

Details about their role in the Serenoa framework, their use and the interaction between them have been presented. In a nutshell the Authoring Tools (i.e. Quill and Eclipse plug-in) are intended to support the editing of the service front-ends description (using ASFE-DL for the description of UIs and AAL-DL for the adaptation rules). The descriptions are sent to the Service Repository hosted in the Adaptation Engine (AE) and there, they are accessible through a REST interface. When a user requests a service, the Runtime Engine (RUIGE) asks the AE for the most suitable adaptation strategy to serve the requested service taking into account the user's context. The information about the context is retrieved by the AE from the Context Manager which is receiving updates from the user's device and other sensors as well as storing the user's profile/preferences. Once the RUIGE has the optimal adaptation strategy, it applies the needed transformations and serves the new context-aware front-end to the user. All these processes are fostered by the theoretical and methodological works available in the Serenoa framework, namely: the reference framework (CARF), the design space (CADS), the ontology (CARFO) and the Agile-UCD methodology promoted by the consortium.

5 References

Abras, C., Maloney-Krichmar, D., & Preece, J. (2004). User-centered design. *Bainbridge, W. Encyclopedia of Human-Computer Interaction. Thousand Oaks: Sage Publications, Citeseer*.

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., & Vanderdonckt, J. (2003). A Unifying Reference Framework for multi-target user interfaces. *Interacting with Computers, 15*(3), 289-308.

Schwaber, K., & Beedle, M. (2001). *Agile software development with Scrum* (Vol. 18). Prentice Hall.

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/