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

Project no. FP7 – ICT – 258030

Deliverable D.4.4.1

Context of Use Runtime Infrastructure (R1)

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

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

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

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).
## Context of Use Runtime Infrastructure (R1)

### Document Information

<table>
<thead>
<tr>
<th>Lead Contractor</th>
<th>ISTI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Editor</td>
<td>ISTI</td>
</tr>
<tr>
<td>Revision</td>
<td>v.1.3 (01/03/2012)</td>
</tr>
<tr>
<td>Reviewer 1</td>
<td>Ignacio Marín Prendes (CTIC)</td>
</tr>
<tr>
<td>Reviewer 2</td>
<td>TID</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>ISTI</td>
<td>Giuseppe Ghiani, Fabio Paternò, Carmen Santoro</td>
</tr>
<tr>
<td>CTIC</td>
<td>Ignacio Marín Prendes</td>
</tr>
<tr>
<td>TID</td>
<td>Javier Caminero</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>13/02/2012</td>
<td>ISTI</td>
<td>First draft.</td>
</tr>
<tr>
<td>0.2</td>
<td>17/02/2012</td>
<td>CTIC</td>
<td>State of the art extension, static information definition, DDR.</td>
</tr>
<tr>
<td>0.3</td>
<td>27/02/2012</td>
<td>TID</td>
<td>Adaptation Engine integration.</td>
</tr>
<tr>
<td>1.2</td>
<td>29/02/2012</td>
<td>ISTI</td>
<td>Examples of usage and context model integration</td>
</tr>
<tr>
<td>1.3</td>
<td>02/03/2012</td>
<td>ISTI</td>
<td>Final review</td>
</tr>
</tbody>
</table>
Executive Summary

This document describes the layer that supports the Context of Use management in the Serenoa framework. The layer is referred to as context management support, and consists of several modules. The main one is the Context Management Core, which hosts several sub-modules such as a data store that keeps information and an interface for interaction with other external modules/components. Examples of such external components include both those that provide context information (e.g. context delegates of user devices, DDR, …), and components that make use of context information (e.g. CARFO, Adaptation Engine, …).

It is worth noting that context providers, even if they are external to the Context Management Core, are part of the whole context management support since they actively participate in context information gathering.

At finalization time of this document, the implementation of a preliminary prototype of the context management support has been carried out, allowing a first integration of the Context Management Core with DDR and some adaptation component.
# 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 ............................................................................... 5  
2 State of the art ............................................................................................................ 7  
3 Data classification ...................................................................................................... 9  
3.1 Static information ................................................................................................... 9  
3.2 Dynamic information ............................................................................................ 11  
4 Context Model ........................................................................................................... 12  
5 Architecture of the context management support ............................................... 18  
5.1 General description (overview) ............................................................................. 18  
5.2 Server-side: Context Management Core ............................................................... 18  
5.2.1 Data Store ........................................................................................................ 19  
5.2.2 Context Management Core Interface ............................................................... 20  
5.2.3 Data Store Interface ......................................................................................... 21  
5.2.4 Data Manipulation ......................................................................................... 21  
5.3 Device-side ............................................................................................................. 21  
5.3.1 Device-gathered information ........................................................................... 21  
5.3.2 Context Delegate implementation ................................................................. 21  
5.4 Further external modules ...................................................................................... 22  
5.4.1 DDR ................................................................................................................ 22  
5.4.2 CARFO ............................................................................................................ 25  
5.4.3 Adaptation Engine ........................................................................................... 25  
6 Examples of usage ...................................................................................................... 26  
6.1 Specifications and requirements .......................................................................... 26  
6.2 Examples of command messages ......................................................................... 26  
6.2.1 Insertion .......................................................................................................... 26  
6.2.2 Subscription/Update/Notification .................................................................... 27  
7 Conclusions ................................................................................................................ 29  
7.1 Summary ............................................................................................................... 29  
7.2 Future Work .......................................................................................................... 29  
8 References ............................................................................................................... 30  
Acknowledgements .................................................................................................... 31  
Glossary ....................................................................................................................... 32
1 Introduction

1.1 Objectives
This document is the first version of design and implementation of the layer that will support acquiring, updating and distribution of Context of Use information. Monitoring context information and its evolution allows enabling an Adaptation component to perform at runtime the most appropriate AAL (Advanced Adaptation Logic) rules. Collecting and processing the information related to the Context of Use is thus a fundamental requirement for automatic adaptation of service front-ends.

The context management support of Serenoa runtime is responsible for context information gathering (e.g., user/device position, environment parameters, …), storing (temporarily or permanently), modifying (e.g., through updates). Distribution is also an important task: by distributing information, the context management support is able to provide it on request or automatically, by notifying all the modules that have previously subscribed to updates. The former strategy is synchronous, while the latter is asynchronous and requires specific interfacing mechanisms between the context management support and the external runtime modules.

The discussion about in this document tackles design issues, as well as implementation choices that have been made in order to setup a flexible and robust layer for successful integration with external runtime modules.

1.2 Audience
The main addressees of this document are the partners of the Serenoa project, whose activity is related to modules of the Serenoa framework exploiting Context of Use information. The consortium members that cope with Context of Use information, by working on this first version of the support, will provide substantial feedback for the final version improvements.

1.3 Related documents
- D1.1.2 Requirements analysis, defining the basic requirements for the runtime engine for context-sensitive service front-ends (SFEs): the runtime has to be aware of the context and to react to its changes in a continuous way.
- D2.3.1 CARFO Population, describing how the ontology for Multi-Dimensional Adaptation of SFEs is populated. CARFO population is partially performed through the context management support by modules related to the Context of Use runtime, such as the DDR.
- D3.3.1 AAL-DL Semantics, Syntaxes and Stylistics (R1), which introduces the description language for advanced adaptation logic, needed for obtaining context-aware interactive applications.
- D4.3.1 Adaptation Engine, describing some runtime adaptations triggered and driven by context related information.

1.4 Organization of this document
- After the first section that describes objectives and audience, the document is organised as follows: Section 2 provides an overview of previous proposals about context information management. References to works in literature, to previous EU projects and to related standardization activities (i.e. W3C working groups) are given.
- Section 3 introduces the main categories of data managed by the Context of Use runtime in Serenoa framework.
- Section 4 deals with the elements characterizing the context model.
- Section 5 describes the architecture of the context management support as a whole, thus providing
details on the internal sub-modules as well as on the integration strategies with external components.

- Section 6 mentions a few simple use cases for the context management support.
- Section 7 reports on some observations on the current implementation of the context management support, that will drive further developments for the final version.
2 State of the art

Context of use and its monitoring/management has been already investigated as a means for supporting situation-driven information retrieval, adaptation of the application to current situation and thus better exploitation of its resources.

Context widgets were defined in [11] as structures that hide the sensor specifics to the higher layers (i.e. to the application layer). Context widgets manage raw data coming from generators (which get data from sensors) and have a state and a behaviour. Context widgets can even exploit an interpreter, if further information abstraction is needed. The proposed strategy assumes the presence of TCP/IP capability and achieves inter-component communication through exchange of XML-formatted messages via HTTP protocol. A Java implementation of the context toolkit that provides context widgets was created.

A totally distributed architecture for context management is introduced in [12] and evaluated in [7], where the Context Agents embed the functionalities of the context widgets described in [11]. A context agent includes several sub-modules: a client application interface, one or more retrievers, a data source abstraction layer (DSAL). It is worth noting that retrievers and DSAL have similar function to, respectively, generators and interpreter of [11].

The framework in [12, 7] supports communication with client applications by XML-RPC over the HTTP protocol.

One of the most advanced and spread techniques within the community of developers devoted to adaptive software is the usage of Device Description Repositories (DDRs) to access static device features. DDR technology has been covered in depth by the World Wide Web Consortium (W3C), by means of its already closed Device Description Working Group (2005-2008). Although covering only requirements for basic web adaptation, its concepts are extensible in order to create technology supporting more advanced types of adaptation. The Device Description Working Group followed the work of the Device Independence Working Group (2001-2006), which defined the CC/PP Framework and the concept of Delivery Context and other relevant concepts to define requirements and use cases for web adaptation.

The “Device Description Landscapes 1.0” Working Group Note [5] defines a Device Description as “a data structure that describes the characteristics and behaviors of a device. Conceptually, a Device Description provides both knowledge to be able to identify a device from its requests for content, and data about the devices attributes so that content adaptation can be effectively performed”. It is recommended to read this document, as it defines the need for device descriptions, the relevance of its availability all across the adaptation chain (from design-time to application execution) and all the stakeholders, users and providers, in such chain. Although used in distinct documents produced by W3C, the Device Description Repository concept is not defined until the “Device Description Repository Simple API” W3C Recommendation[6], as described below:

“Web content delivered to mobile devices usually benefits from being tailored to take into account a range of factors such as screen size, markup language support and image format support. Such information is stored in Device Description Repositories (DDRs)”

It is important to note, as previously commented, that DDRs (also popularly known as “device databases”) only keep static device descriptions. This is, information about device features which can be known “a priori”. Screen resolution and color depth, CPU model or RAM memory available are typical examples of static device features. As opposed to static, dynamic device features are those who can only be retrieved at runtime, such as screen orientation, battery level or type/quality of network in use.

There are different implementations of Device Description Repositories, most of them have been exploited for Web adaptation. The Wireless Universal Resource File (WURFL) [15] has been the first widely adopted DDR, setting apart from the rest over time because of its open/free nature. Unfortunately, its license has been strongly restricted in August 2011. Different implementations of the WURFL API allow exploiting information from the WURFL XML database for the major languages used in web development (Java, C#,
Python, PHP). Device Atlas [4] is a commercial DDR implementation from mTLD/dotMobi to include device knowledge among their many services for the community of developers devoted to web adaptation. DetectRight [3] is also a major stakeholder in DDR services. In November 2011, the OpenDDR initiative started, providing an XML device with device descriptions and a DDR in Java (promised soon to be available in other environments). Most of them, excluding DetectRight, allow using the DDR Simple API Recommendation. This permits changing the DDR technology used in an adaptive web application while minimizing the development efforts required. The main problem changing one DDR technology by another one, if all of them publish their functionality via the DDR Simple API, is the vocabulary used to express device features and their admissible values. For this reason, in order to cover basic web adaptation, the W3C Device Description Working Group proposed the DDR Core Vocabulary [1], which is used by every DDR Simple API compliant implementation.

Prior to the invention of the DDR Core Vocabulary, the only vocabulary for device description accepted by the major stakeholders in the mobile adaptation research and development field was UAProf [14]. UAProf is a vocabulary proposed by the Open Mobile Alliance in 1999 and used by device manufacturers to announce a device description for each of their mobile devices. However, it does not directly nor easily support some of the desirable requirements in device descriptions to be used within a Device Description Repository, as it will be tackled in sections 3.1 and 5.4.1 in this document.
3 Data classification

The information that can be gathered by the Context of Use module may be classified in two main groups, according to the moment in which such information may be obtained. On the one hand, there is static information, which can be known beforehand and does not usually change its value over time. On the other hand, there is dynamic information that, due to its evolving nature, can only be retrieved at application execution time.

3.1 Static information

Some examples of static information (device description items) are the operating system name and version, device name and version, device maker, screen resolution and color depth, available input/output mechanisms (such as keyboard(s), camera(s) and microphone), available sensors (e.g., accelerometers, compass or GPS), preinstalled web browser name, version and maker, and the different device features which can be known beforehand.

A basic set of static device characteristics required for basic web adaptation has been defined by the W3C Device Description Working Group in the DDR Core Vocabulary. This vocabulary defines two aspects (webBrowser and device—the default aspect) which serve to disambiguate the usage of some properties such as the “vendor”. Following this approach, it defines the following properties and data types to be considered by the W3C DDR Core Vocabulary:

- Web browser vendor and device vendor (String).
- Web browser model and device model (String).
- Web browser version and device version (String).
- Display width for both web browser and device (int).
- Display height for both web browser and device (int).
- Input devices of the device (an enumeration with the possible values keypad, touchScreen, stylus, trackball and clickWheel).
- Markup support of the web browser (enumeration: xhtmlBasic10, xhtmlBasic11 and xhtmlMP10).
- Stylesheet support of the web browser (css10, css21, wcss10).
- Image format support of the web browser (gif87, gif89a, jpeg, png).
- Input mode of the web browser (it uses an enumeration with a large set of possible values, which describe the preferred ways of supporting specific formats for input type="text" fields; this can be done with the 'format' attribute, as a WCSS property or using the 'inputmode' attribute).
- Cookie support of the web browser (boolean).
- Script support of the web browser (ecmascript-MP, for example).

Each of the different DDR technologies propose additional vocabularies in order to express interesting device features for more advanced types of adaptation. In the same way, as the Core Vocabulary is intended to cover only device properties for basic web adaptation, the Serenoa project will also extend the expressivity of the DDR technology of choice with one or more vocabularies. An addition of more static device properties will be proposed in Serenoa as the project advances, depending on the distinct challenges discovered as the Consortium faces the evolution of the Serenoa framework in the last half of the project. A first proposal for addition of static device properties are:

- additional properties required for non-web adaptation, with an initial set of properties described below:
  - Operating system name and model.
  - CPU model and frequency.
  - Available RAM.
  - Available storage capacity.
Type of device, with (for now) three possible values: desktop, tablet or mobile.

An interesting source for more fine-grained information for the DDR technology to be used in the Serenoa project is UAProf, the vocabulary proposed by the OMA which is used by some mobile device manufacturers in order to announce a device description for each of the devices that they bring to the market.

Although UAProf profiles are an interesting source to obtain device descriptions from device makers, it lacks some interesting features in its information model to be directly used in a DDR. This will be covered in section 4.4.1 later in this document. When a device manufacturer decides to offer a UAProf profile for a device, one or more HTTP Request headers are included (typically the x-wap-profile header) in each request submitted by the web browser of the device to a remote server which references the URL of the UAProf.

Unluckily, UAProf has only been intended for the mobile industry, so other devices are out of scope. In addition, major stakeholders in the mobile industry, such as Apple or Microsoft, have never announced device properties by means of a UAProf profile. Thus, this device description technology does not provide a holistic way to obtain device descriptions for all (or, at least, most) of the connected devices in the market.

Static information, as it can be gathered “a priori”, is useful for content and application adaptation, providing static device descriptions for origin-server adaptation, delivery infrastructure, client-side adaptation and design-time. More information on this can be found in the W3C Device Description Landscape [5] Working Group Note. Although focused in the web domain, the concepts are clearly extensible to native applications which play the role of Service Front-Ends. As stated in the Landscape document, static device descriptions (and thus, DDRs) are very important information sources at the time of design of an adaptive application. In addition, the vocabularies that they provide and use can be reused for the antecedents in rule languages, such as the AAL-DL under development in Serenoa.
3.2 Dynamic information

Data that is characterised to change continuously and that is gathered in real-time, are referred to as dynamic information and can be related to user’s tasks, environment, device in use, etc. Such values can be detected by sensors or can be derived from device status (e.g. battery level).

Examples of relevant dynamic information which might be provided from devices include (but are not limited to):

- **Device tilting**, useful to properly adapt the UI to screen orientation (landscape vs portrait layout).
- **Network technology in use** (GPRS/EDGE/UMTS/HSDPA/HSUPA/LTE/WiMax/802.11a/b/g/n/...) and quality of connection (e.g. percentage of signal).
- **Battery level**, useful to apply transformations for minimizing CPU consumption. For instance, still images may be served instead of videos when battery percentage left is below a specific threshold.
- **Location**, taking into account error estimation and available technologies (GPS/Glonass/GSM cell triangulation/WiFi triangulation/... or combinations of them).
- **Orientation**, to help determining whether the user is heading towards a physical location. Orientation can be detected by means of an electronic compass, and might be used to trigger some actions in the application.
- **Acceleration**, detected through one or more accelerometers, indicating which type of physical actions the user is performing (e.g. walking vs running). Acceleration lets infer the chances that the user is paying attention to what is on the screen.
- **Gyroscope data**, helping to guess the position of the device (and also to estimate whether the user is paying attention to the device).
- **Environment light**, which drive adjustment of colour palette, contrast and brightness of the screen.
- **Environment noise**, Microphone input, for instance in order to know whether the user is in a noisy environment and thus use vibration rather than the speakers to get users’ attraction.
- **CPU load, available memory storage**, that indicate whether a specific task can be executed.
- **Running applications**, obtained by querying the device task manager.
- **Web activity**, monitored by logging HTML events through proxy-injected scripts.
4 Context Model

In this section we present the basic features of a first draft of a context model that we have identified so far in Serenoa. The related UML class diagram is shown in Figure 2.

![Figure 2: Representing the context in SERENOA](image)

As can be seen in Figure 2, the specification of the context is composed of a number of elements: users, environment, technology, social relationships.

The specification of the users belonging to a certain context implies the description of the various (1..n) elements of type user, associated. For each user a number of information is modelled: preferences, task, disability, position, knowledge. In order to represent the characteristics of the surrounding, physical environment we are interested in modelling information about light and noise. As for the technological characteristics of the context, they are mainly represented in terms of devices, connectivity and browsers available. The last dimension of the context, the social relationships, are modelled through the definition of a number of groups (of users), where each group is identified through the provision of a number of elements (of type user_id), each one identifying a user belonging to the concerned group.

We have also identified a XML-based language implementing the semantic modelled in the class diagram shown in Figure 3. Below we provide its description.
At the highest level there is the element named *context*, which has type *ContextType* (shown in Figure 3): this type is a sequence of four elements that model the key dimension of a context: users, environment, technology, and social relationships.
The main characteristics of user elements are shown in Figure 4: users are a sequence of 1..n elements of type user. Each user has type named UserType, which is identified through several elements: preferences, task, knowledge, position, disability. Since it is not mandatory to specify all such information the minimum occurrence of each of these elements is 0. For preferences we can specify the particular language, for the task we should specify the name of the current task, regarding the knowledge we specify the level of education of the user. The position is specified in terms of longitude and latitude (both decimal values), regarding the disability, we specify if the user is affected by some disability (e.g. blindness, dyslexia). For the blindness we can specify whether it is complete, partial, or is colour-dependent; for the dyslexia we can specify whether the level is low, medium or high. The specification of this part is shown in Figure 5.
<xs:simpleType name="BlindnessType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="complete"/>
    <xs:enumeration value="partial"/>
    <xs:enumeration value="colour"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="LevelType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="low"/>
    <xs:enumeration value="medium"/>
    <xs:enumeration value="high"/>
  </xs:restriction>
</xs:simpleType>

Figure 5: Specifying BlindnessType and LevelType

The specification of the environment element (which has type named EnvironmentType) consists of a sequence of three elements: light, noise and position. The type of each of these elements has already been described before (see Figure 4 for the PositionType and Figure 5 for the specification of LevelType).

<xs:complexType name="EnvironmentType">
  <xs:sequence>
    <xs:element name="light" type="LevelType" minOccurs="0"/>
    <xs:element name="noise" type="LevelType" minOccurs="0"/>
    <xs:element name="position" type="PositionType" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="id" type="xs:ID" />
  <xs:attribute name="name" type="xs:string" />
</xs:complexType>

Figure 6: Specifying the environment

Regarding the specification of technological characteristics of the context, we can refer to the following Figure 6: it consists of the specification of a sequence of device, connectivity and browser. For each of these three elements we provided specific type definitions. In particular, a device is identified through an ID, a name, its category (e.g. whether it is a tablet, a smartphone, a desktop, etc.), the level of the battery, and its screen resolution (composed of width and height resolutions). The connectivity part identifies which kinds of connectivity we have in the context (e.g. whether wi-fi, Bluetooth,..). The browser part identified the browsers currently available.
The last part of a context definition regards the possible social relationships that could exist in the context. The specification of this part is shown in Figure 7. In our model, social relationships are modelled through the definition of a number (unbounded) of user groups. Each user group has type named GroupType and through it we should model a group of users for which a particular relationships holds (e.g. closeness, friendship...). If we, in turn, consider the definition of this type named UserGroup, this consists of a sequence of elements, where each element identifies a specific user (see the IDREF type of the element named “user_id”). In addition, each group should specify, apart its name/id, also the type of relationship that exist among its users (see closeness, friendship attributes in its specification in Figure 7).
Figure 8: Specifying social relationships

It is worth pointing out that this context model is just a first draft, which has been identified taking into account the current requirements considered at the current stage of the project (for instance, the current definition of context has been exploited for the specification of the example of adaptation rules defined in D3.3.1). Further updates and improvements will be done to this specification of the context in a next version of this deliverable (D4.4.2, planned for M36).
5 Architecture of the context management support

5.1 General description (overview)

The context management support is an interactive deposit, accessible to external modules and devices, able to store information and to provide it upon request. Any kind of information can be modelled so as to be manipulated by the context management: for instance, the fact that a user is using a particular device to access a specific web page, or that s/he is crossing a certain district might be stored and updated in real time.

The context management support, as a whole, provides mechanisms for treating context information. Figure 9 shows the general architecture.

5.2 Server-side: Context Management Core

The Context Management Core is the coordinator unit of the context management support and provides mechanisms for gathering/storing/retrieving information as a set of entities. Its main components are: a Data Store, a Data Manipulation module and a set of interfaces for data exchange with external modules/devices. An overview of the Context Management Core structure is depicted in Figure 10.
5.2.1 Data Store

The structure that actually contains context information in the form of entity instances and that allows to perform operations on entities is referred to as Data Store.

An entity represents the basic information unit that can be stored on the context management core, and is characterized by the following fields:

- **Id.** A unique string identifying the entity within the context management core. The id is assigned by the context management core at creation time, and cannot be modified.
- **Type.** A string that describes the kind of data modelled by the entity (e.g. “location”, “temperature”, “current task”, …). The type cannot be modified.
- **Creation time.** Date and time of creation of the entity (milliseconds since January 1, 1970, 00:00:00 GMT). The creation time is assigned by the context management core at creation time, and cannot be modified.
- **Update time.** Date and time of the last modification of the entity (milliseconds since January 1, 1970, 00:00:00 GMT). It is initially equal to the creation time, and is modified by the context management core in case of update or reset of the entity data.
- **Time-to-live.** The amount of time (milliseconds) after which the entity expires (zero if it never expires). The time-to-live cannot be modified.
- **Subscriptions.** The list of network addresses (i.e. IPs and ports) that have subscribed to modifications of the entity. The subscriptions list is modified by the context management core each time a device/module subscribes (or unsubscribes) to the entity.
- **Data.** A set of correspondences “attribute_name”-“attribute_value” that describe the entity. As additional capability (on-going implementation) an attribute value could consist even of another entity.

An operation is a functionality of the Context Management Core that can be invoked from outside. The following basic operations have been implemented so far:
• **Insert (Entity)**. Adds a new entity on the context management core. A type and a data field must be specified in order to create the entity. Returns an entity_id.
• **Query (entity_id)**. Asks the context management core for the current definition of the entity. Returns the description of the entity.
• **Update (entity_id, entity_data)**. Merges the data of the entity with the provided data.
• **Remove (entity_id)**. Removes the entity.
• **Reset (entity_id)**. Clears the data of the entity.
• **Subscribe (entity_id, network_address)**. Creates a subscription on entity updates for the specified network address (IP-port). After the subscription, a message is sent back to the subscriber, for every modification of the entity (update, reset, remove), that specifies the current entity description. Returns a subscription_id.
• **Unsubscribe (subscription_id)**. Removes the subscription.

It is worth pointing out that, in general, the Data Store is able to keep any kind of information (see Figure 11), as far as it is modelled as one or more entities. Thus, information manageable by the Context Management Core may include user profiles (e.g. preferences), user position or environment parameters.

![Figure 11. Example of information hierarchy in the Data Store.](image)

**5.2.2 Context Management Core Interface**

A set of methods for interfacing with the Context Management Core is implemented in the Context Management Core Interface. A HTTP interface has been already implemented that supports most operations, including the **Subscribe** (the notification mechanism is provided via TCP).
5.2.3 Data Store Interface

The conversion between XML-formatted command messages (coming from the Context Management Core Interface) and actual operations on the Data Store (e.g., creation, modification, deletion of entities) is performed by the Data Store Interface. This module is also devoted to raise notification events when relevant changes in the Data Store occur (e.g., when an entity with some subscription is updated). The notification is sent back through the Context Management Core Interface.

5.2.4 Data Manipulation

The stored entities are periodically checked. The semantic control performed by the Data Manipulation module removes the “expired” entities and manipulates inconsistent entities (e.g., by removing references to other external entities that do not exist anymore).

5.3 Device-side

User devices are steadily increasing in performance and capabilities, including more and more sensing resources. Modern devices often embed hardware components for detecting user position, intensity of environment light, and other parameters. This data can be collected in real time and combined with additional information regarding the activity of the user (e.g., the list of running applications). Thus, user device is a potential source of interesting information regarding the context of use.

5.3.1 Device-gathered information

As already mentioned, various data can be obtained by the device in use.

In order to obtain data and forward the related information to the Context Management Core, some software is needed in the user device. For instance, a smartphone device can run a process in background that monitors the environment light intensity and reports it periodically (e.g., every few seconds) to the Context Management Core. At the same time, the Web activity could be monitored by means of scripts injected in the navigated pages by a proxy. The software that extracts data, processes it and interfaces to the Context Management Core is referred to as Context Delegate.

More detail on the Context Delegate is provided in the following subsection.

5.3.2 Context Delegate implementation

A Context Delegate can be implemented and structured in different ways, according to the device capabilities and to other constraints or requirements.

The Context Delegate for monitoring user position would typically be a lightweight stand-alone application that reads the GPS serial stream, extracts the terrestrial coordinates from the NMEA sentences and forwards them to the Context Management Core.

The Context Delegate for logging Web activity, instead of being deployed as stand-alone application on the device, could be “injected” in the navigated Web page as JavaScript code. Script injection is done by a dedicated proxy server through which the user accesses the Web (the proxy would be part of Serenoa runtime). Injected scripts, besides detecting relevant events (e.g., number of clicks), would forward logged information to the Context Management Core.

New information is actually sent to the Context Management Core by insert or update messages. The connection between Context Delegate and Context Management Core can occur, for instance, over HTTP. In the case of stand-alone Context Delegate, the connection would be managed by the library of the development environment. In the case of JavaScript Context Delegate, a simple Ajax call is done (and the same-origin policy is respected as the proxy lies in the same domain of the Context Management Core).
5.4 Further external modules

External modules can interface to the Context Management Core directly or indirectly.

In the direct interfacing the external modules have to actively support the Context Management Core Interface and the Data Store operations (e.g. performing inserts, updates, queries in order to provide/get information to/from the Context Management Core).

In the indirect interfacing the external modules passively exploit an intermediate module, that could be considered analogous to the Context Delegate. The intermediate module, deployed somewhere, actually performs the information exchange by querying the Context Management Core and updating the external module, or vice versa.

5.4.1 DDR

The DDR is the basic module in Serenoa to store static device descriptions which can drive content and application adaptation both at design-time and at runtime. In order to allow interoperability between Serenoa and existing DDR implementations (as mentioned in section 2), the Consortium has agreed that any DDR technology to be used by the Serenoa framework will have to expose its functionality by means of the W3C DDR Simple API.

The mechanism of creation of new vocabularies opens the space for new device descriptions for novel adaptation strategies and technologies. In this way, the usage of any RUIGE module based on other types of adaptation different to basic web adaptation will imply the enhancement of the DDR technology of choice with new vocabularies. For instance, the RUIGE module for mobile web applications, based on the existing MyMobileWeb project, will make use of the MyMobileWeb vocabulary.

The utilization of a DDR will be exemplified with the mobile web RUIGE module in order to clarify how to enhance an existing DDR to comply with the expressivity needs of any RUIGE module.

5.4.1.1 Integrating an existing DDR technology

The MyMobileWeb RUIGE module currently uses a WURFL DDR. This DDR is based on an API which provides access to an XML database with device descriptions more fine-grained than the W3C Core Vocabulary. The WURFL XML file provides a list of <device> elements with all the information for the different devices in the market. However, it uses a fallback mechanism that provides device hierarchization in the form of a logical tree. This tree provides more generic device descriptions in the root and intermediate nodes in the tree, and device descriptions for actual devices in the final nodes. This mechanism of device grouping allows device classification in families by the underlying technology (typically, those devices sharing the same browser or operating system, or the same version of the browser or the operating system) and saving space so only those device properties which differ between a new device model and a previous model in the same device family need to be annotated. More information on this can be found in ScentiaMobile’s Online Contributors Help web page [13]. Any user of WURFL (respecting the license) may add new properties to a device or to a device family and may also patch the WURFL database with new devices. Therefore, the technology allows the addition of information useful for new adaptation mechanisms and technologies. MyMobileWeb uses some XML files which describe the mapping from the WURFL vocabulary to the W3C DDR Core Vocabulary and the MyMobileWeb Vocabulary. It must be noted that MyMobileWeb uses a version of the WURFL XML database and API prior to the more restrictive changes in August 2011 which end up in a dual Affero and commercial license. Moreover, WURFL does not support the W3C DDR Simple API but a project in the Morfeo Project community (DDR-RI, http://forge.morfeo-project.org/projects/ddr-ri) provides a connector to export the functionalities of WURFL by means of this W3C Recommendation. Considering the licensing restrictions in WURFL, the OpenDDR technology can be interesting for the developers of Serenoa applications, as it follows a very similar approach but under an LGPL license for the OpenDDR API and an Open Database License for the OpenDDR XML database. Vocabulary extension may also be done in the Device Atlas DDR, but one of the most expensive commercial licenses is required. However, by means of the usage of the W3C DDR Simple API, the Serenoa framework intends to provide the developer community with the possibility to select among the various DDR
technologies available: WURFL (old license), OpenDDR and DeviceAtlas.

5.4.1.2 How to use the DDR Simple API

The central concept which serves as basis for the DDR Simple API is the concept of Vocabulary. Before defining it, it is necessary to define two other concepts on which the Vocabulary relies. Firstly, an Aspect of the Delivery Context typically represents a category of hardware or software that participates in delivering a Web experience (in general, delivering access to a service). "Browser", "proxy" and "device" are all examples of Aspects. Aspects can also represent things other than hardware and software, such as end-users and mobile network operators, although it is not very common to represent these entities in the Aspects considered in a DDR. Secondly, a Property is a characteristic of an Aspect that can affect the Web experience. In general, it can affect the way in which access to a remote service is provided. “Screen Width”, “Image Formats Supported”, “Color Depth” and “Audio Codecs Supported” are all Properties. Properties have names and data types for their values (boolean, int and so on). Note that the normative DDR Simple API specification is done in Java so value types for the different items in a vocabulary are Java types. A Property may have an “unknown value” when the actual value is not known to the DDR implementation, and in this case the exists() method of the PropertyValue interface must return false.

One of the most relevant features in the DDR Simple API is that Property names and Aspect names are namespaced to allow independent naming and evolution of sets of Properties. In the DDR Simple API, Aspects share the same namespace as the properties with which they are associated.

After all these previous concepts, a Vocabulary is defined as a set of Properties and the Aspects with which the Properties are associated. Vocabularies should declare a default Aspect for each Property (the Aspect to be used when using the Vocabulary in an abbreviated convention that does not specify the Aspect).

The principal interface of the DDR Simple API is the Service interface, which contains the factory methods for creating instances of the interfaces discussed below. Using methods of Service, the caller supplies Evidence representing the Delivery Context and an indication of the Properties of interest. These methods return PropertyValue objects which can then be queried to reveal the values of the Properties of interest.

The rest of the interfaces are:

- Evidence, to add, query or retrieve an evidence, using an HTTP Header as a parameter –the method to add also needs the value for the corresponding Header).
-PropertyName, to get the name of a Property and its associated namespace.
-PropertyRef, which allows getting a Property name, an Aspect name and the associated namespace.
-PropertyValue, which models a PropertyRef together with its value. Values may be empty, in which case the method exists returns false. An attempt to query an empty value causes a ValueException as does an attempt to query a value with an incompatible accessor method (string as float, for example). It provides PropertyRef and PropertyValue retrieval, confirmation of the availability of a specific Property.
-PropertyValue, which provides access to a set of PropertyValue objects.

This section does not intend to provide a detailed description of the API, which is formally defined in the corresponding W3C Recommendation. For the sake of brevity, some examples included in the OpenDDR documentation (which have their analogous usage in any W3C DDR Simple API compliant implementation) are provided below in order to help understanding such API. Code Snippet 1 provides access to the OpenDDR functionality in an HTTP Servlet. It uses the HTTP User-Agent as the identification evidence passed to the OpenDDR and asks for the vendor, model, display width and display height of the device.

```java
// Creation of a Service by calling the corresponding factory
Service identificationService = null;
Properties initializationProperties = new Properties();
// PropertyRefs to check
PropertyRef vendorRef;
PropertyRef modelRef;
```
PropertyRef displayWidthRef;
PropertyRef displayHeightRef;

// Code to fill Properties in its initial stage –implementation dependant and avoided for brevity; it usually implies loading configuration files and initial resources
// Code to initialize access to the Service –which is also implementation dependant
identificationService = ServiceFactory.newService("org.openddr.simpleapi.oddr.ODDRService",
initializationProperties.getProperty(ODDRService.ODDR_VOCABULARY_IRI), initializationProperties);

try {
    vendorRef = identificationService.newPropertyRef("vendor");
    modelRef = identificationService.newPropertyRef("model");
    displayWidthRef = identificationService.newPropertyRef("displayWidth");
    displayHeightRef = identificationService.newPropertyRef("displayHeight");
} catch (NameException ex) {
    throw new RuntimeException(ex);
}

PropertyRef[] propertyRefs = new PropertyRef[] {vendorRef, modelRef, displayWidthRef, displayHeightRef};
Evidence e = new ODDRHTTPEvidence();
e.put("User-Agent", ((HttpServletRequest)request).getHeader("User-Agent"));

try {
    PropertyValues propertyValues = identificationService.getPropertyValues(e, propertyRefs);
    PropertyValue vendor = propertyValues.getValue(vendorRef);
    PropertyValue model = propertyValues.getValue(modelRef);
    PropertyValue displayWidth = propertyValues.getValue(displayWidthRef);
    PropertyValue displayHeight = propertyValues.getValue(displayHeightRef);

    if (vendor.exists() && model.exists() && displayWidth.exists() && displayHeight.exists()) {
        ((HttpServletRequest)request).setAttribute("vendor", vendor.getString());
        ((HttpServletRequest)request).setAttribute("model", model.getString());
        ((HttpServletRequest)request).setAttribute("displayWidth", displayWidth.getInteger());
        ((HttpServletRequest)request).setAttribute("displayHeight", displayHeight.getInteger());
    }
} catch (Exception ex) {
    throw new RuntimeException(ex);
}

Code Snippet 1: Example of usage of the DDR Simple API using the OpenDDR

5.4.1.3 DDR exposure under a RESTful API

The previously commented DDR-R1 project, which provides a wrapper for WURFL, so its functionality is exposed by means of the DDR Simple API, also provides a RESTful interface for the WURFL repository. This allows the allocation of a specific machine to host a DDR instance and that device identification queries may be consumed remotely. Documentation about this RESTful API is reported on [2]. More documentation about the usage of MyMobileWeb with different device description providers is available at[8].

5.4.1.4 Need for a more complete DDR technology

Finally, considering the fine granularity required for a DDR providing the expressivity needs to support the adaptation levels which Serenoa intends to cover and the licensing terms of most of the existing DDR technologies, the Serenoa Consortium agrees that a new DDR technology is needed. The Consortium has asked for the inclusion of the development of a new DDR within the project. After the amendment of the Description of Work of the Serenoa Project received in January 2012, which includes the acceptance of such request, CTIC will start the development of a novel DDR covering not only the requirements expected for traditional DRRs, as indicated in the W3C DDR Requirements Working Group Note (http://www.w3.org/TR/DDR-requirements/), but also:

- Use more device identification evidences to access device descriptions. So far, DDR technology is mainly used for the delivery of web content and applications. Thus, the main identification elements for a device are the User Agent HTTP Request header and the x-wap-profile HTTP Request Header (which provides the URI to the UAProf profile of the device). Additional identification elements are needed, so that native applications can send some identification of the device. In this way, the DDR will be able to have a complete description of the device and mechanisms to identify the device both
from a web browser through a HTTP Request and from a native client request.

- Obtain new device descriptions in an automated way or by using automated tests, thus minimizing the need for manual additions.
- Avoid walled gardens by means of a distributed architecture which automatically shares new device descriptions with all the endpoints/deployments of the DDR.
- Provide all this information for free, as long as consumers also produce information. Device descriptions should be a commodity for the developer community. Efforts should be focused on creating good adaptive applications rather than in handling device description information.
- Consider device properties not treated in current DDR technologies, such as CPU, memory, storage, a fine-grained description of operating systems/APIs/SDKs and other properties.
- Support the concept of provenance, so DDR consumers will only get information from trusted sources.
- Consider that device classification, one of the warhorses in the device description arena, is not only to be included in the information model but also supported in the queries to the device database, so different device description consumers can use distinct device classification criteria.
- Consider throughput requirements: the information model in the DDR must allow serving thousands of requests per second, in order to be used by popular complex systems.
- Consider minimization of memory fingerprint, so the information model is compressed as much as possible (perhaps providing each DDR deployment strictly with the view of the model they actually need and/or using caching mechanisms).
- Consider additional requirements which may arise in the development of the DDR.

An initial approach to the development of the DDR, and more particularly to the information model, has been provided in [9]. The paper also discusses why the information model in UAProf is not sufficient for such a DDR.

### 5.4.1.5 Integration with CARFO. Usage for CARFO population

As initially said in this section, the DDR is the basic piece for static device information. The DDR may also be a source for the population of the Context of Use module of the CARFO ontology. This dual approach allows to:

- Meet throughput requirements, by directly using a DDR implementation.
- Meet expressivity requirements, by expressing device descriptions including the description of the relationships between different device features and device classes and instances using the ontology. This approach is covered in the D2.3.1 deliverable (CARFO population).

### 5.4.2 CARFO

Semantic relationships among concepts related to the context of use can be defined by the CARFO module. The context management support is able to gather dynamic information, and to make it available to other modules. For instance, the CARFO might exploit dynamic information provided by the context management support in order to identify relevant semantic relations. At the same time, the Context Management Core can extract information from the CARFO via SPARQL.

### 5.4.3 Adaptation Engine

The information exposed by the Context Management Core Interface will be among the inputs for the Adaptation Engine (AE). The Context Management Core will inform the AE about parameters such as device in use, lighting conditions, current user preferences, etc. So far, a REST manager has been implemented, via an open API, as input interface in the AE. The AE is then able to gather, through a delegate or by using direct interfacing to the Context Management Core, all the context information necessary to select the most suitable adaptation process to be performed.
6 Examples of usage

6.1 Specifications and requirements
The Context Management Core is implemented as a set of Java classes deployed on a Web server. The external interface (i.e. the interface used for the operations) is a Java Servlet.

In order to access the current version of the context management support it is sufficient that a device/module is able to perform HTTP POST calls. The call is addressed to the URL of the Java Servlet, while the only HTTP parameter is the command description.

The command description is a XML string which format varies depending on the operation required.

Messages sent back by the context management (e.g. responses about commands or notifications about subscriptions) are also formatted as XML documents.

On the preliminary implementation of the context management support, notifications are send as TCP messages to the subscriber.

6.2 Examples of command messages

6.2.1 Insertion
Insertion of a newly created entity on the Data Store is done by specifying the entity type, the time to live (zero for eternal entities) and the initial data. The following command message defines the creation of a generic_user entity for user “Mario” that is currently located at the “office”. The entity has not any expiration date. Each 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 response message for the previous insertion is the following:

```
<comet>
    <response>
        <text>
            The entity with id=5 has been successfully updated, and the new representation is reported in the following:
        </text>
    </response>
</comet>
```
A subscription to the entity with id=5, performed by a device/module with IP=146.48.83.144 that requests to be subsequently notified through the port 33333, is defined as:

```
<comet>
  <op>subscribe</op>
  <entity_id>5</entity_id>
  <subscriber_address>146.48.83.144:33333</subscriber_address>
</comet>
```

The subscriber receives a subscription confirmation as a response such as:

```
<comet>
  <response>
    <subscription_id>0</subscription_id>
  </response>
</comet>
```

The following update of the data field is performed to inform the runtime that “mario” has moved to his car:

```
<comet>
  <op>update</op>
  <entity_id>5</entity_id>
  <entity_data>
    <user_name>mario</user_name>
    <user_location>car</user_location>
  </entity_data>
</comet>
```
As soon as the update message is received by the context management support, the subscriber of “mario” entity is notified by the following message:

```xml
<comet>
    <op>notify</op>
    <subscription_id>0</subscription_id>
    <entity_event>updated</entity_event>
    <entity_id>5</entity_id>
    <entity_type>generic</entity_type>
    <entity_creation_time>1327998940063</entity_creation_time>
    <entity_update_time>1330434619250</entity_update_time>
    <entity_data>
        <user_name>researcher1</user_name>
        <user_location>car</user_location>
    </entity_data>
</comet>
```
7 Conclusions

7.1 Summary

This deliverable describes the requirements and the current design for the implementation of the Context of Use runtime infrastructure. First, state of the art on context information management is discussed. Then, the context management support of the Serenoa framework is described as a whole, defining the capabilities of the Context Management Core as well as the specifications of its external interface. Taking into account the specifications provided in this document, it is possible to exploit the context management support for gathering, storing and circulating context information within the Serenoa framework.

7.2 Future Work

As a result of this deliverable, actual integration of the Context Management Core into the Serenoa framework will be possible.

Context delegates will be implemented for specific platforms, especially for mobile devices (e.g., Android, iPhone, …). Context delegates will be aimed to gather dynamic information such as values coming from embedded sensors, and will turn user devices into context providers.

Further developments will enhance the Context Management Core Interface with a Web Service with proper description, a Web Socket support (for HTML 5 applications) and a pure TCP interface. Such an interfacing flexibility of the Context of Use infrastructure will facilitate integration with additional components of the Serenoa framework.

The context model mentioned in this deliverable will be tackled in the next version of the CARFO deliverable (D2.2.2), and will be reflected and enriched by using OWL.

Enhancements of interfacing strategies will enable a future reuse within other platforms. Technical tests will be carried out in order to validate the infrastructure.
8 References

1. DDR Core Vocabulary, [http://www.w3.org/TR/ddr-core-vocabulary/](http://www.w3.org/TR/ddr-core-vocabulary/)
3. DetectRight, [http://www.detectright.com](http://www.detectright.com)
5. Device Description Landscape [http://www.w3.org/TR/dd-landscape](http://www.w3.org/TR/dd-landscape)
6. Device Description Repository Simple API, W3C Recommendation, [http://www.w3.org/TR/DDR-Simple-API/](http://www.w3.org/TR/DDR-Simple-API/)
13. ScentiaMobile’s Online Contributors Help web page, [https://db.scientiamobile.com/static/top.htm](https://db.scientiamobile.com/static/top.htm)
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

- http://serenoa.morfeo-project.org/glossary-of-terms