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

Project no. FP7 – ICT – 258030

Deliverable D.4.4.2

Context of Use Runtime Infrastructure (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	ISTI
Editor	ISTI
Revision	1.0 (26/09/2013)
Reviewer 1	CTIC
Reviewer 2	
Approved by	
Project Officer	Michel Lacroix

Contributors	
Partner	Contributors
ISTI	Giuseppe Ghiani
CTIC	Javier Rodríguez, Cristina González

Changes			
Version	Date	Author	Comments
0.1	04/09/2013	ISTI	First draft.
0.2	04/09/2013	CTIC	DDR integration
0.3	20/09/2013	ISTI	Consolidated version
0.4	23/09/2013	CTIC	Reviewed version
1.0	26/09/2013	ISTI	Final version

Executive Summary

This document describes the final version of the Context of Use management, also referred to as Context Manager, in the Serenoa framework.

As discussed in the previous version (D4.4.1), the Context of Use management support consists of several modules for gathering context information, saving and exposing it to other modules/components of the framework (e.g., Adaptation Engine).

The current document focuses on the updates that have been introduced, such as novel functionalities and data modeling strategies, after finalization of the previous version (D4.4.1).

At finalization time of this document, some modules have been integrated with the Context Manager. Such modules include context delegates deployable on user devices, providing context information to the Context Manager, as well as major components of the Serenoa framework. An example of such components is the Adaptation Engine that subscribes to particular events in order to get asynchronous notifications about relevant context changes.

At finalization time of this document, an implementation of the context of use management support is available and has already been exploited by project partners (e.g. remotely accessed and/or integrated in prototypes).

Table of Contents

1 Introduction

1.1 Objectives

This document is the second version of design and implementation of the support for acquiring, updating and distributing Context of Use information across the Serenoa framework architecture. The current version is intended to integrate the previous one, by extending the architecture description with the specifications of the novel functionalities. A discussion on the motivations for adding the described functionalities (that led to the specific implementation choices) is also provided.

1.2 Audience

The audience of this document consists of those subjects, even beyond the current Serenoa project partners, which are interested in applying the Context of Use management architecture, or part of it, introduced in Serenoa.

1.3 Related documents

- *Deliverable D4.4.1 Context of User Runtime Infrastructure (R1)* is the first version of the document.
- *D4.3.2 Adaptation Engine (R2)* tackles runtime adaptations triggered and driven by context related information.
- *D5.2.3 Application Prototypes (R2)* describes how the prototypes implemented over the Serenoa framework (i.e. E-Health, E-Commerce, Warehouse Management) are integrated with the Context Manager.

1.4 Organization of this document

The first section describes objectives and audience. The subsequent part of the document is organised as follows:

- Section 2 provides a brief overview of the state of the art about context management, and mentions how our conceptual/implementation choices build on top of previous proposals.
- Section 3 describes architectural/functional updates with respect to the previous version of the deliverable (e.g. novel event subscription mechanisms, RESTful interface, context delegates). The current DDR implementation is also discussed.
- Section 4 reports on considerations about the current implementation and gives hints for future development.

2 State of the Art

2.1 Context Management in Literature

Context awareness in computing systems has been widely discussed in the past, and various definitions of context information can be found in literature. Some works, e.g. [2] refer to it as:

"*a limited amount of information covering a person's proximate environment*".

This includes user location and available technological resources, environmental aspects (noise/light levels) and social implications. Dey and Abowd [1] instead define context information as:

"*…any information that can be used to characterize the situation of (…) a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*"

2.2 Our Approach with Respect to Previous Proposals

Although the two above cited definitions differ in the amount of information involved (i.e. *limited amount of* vs. *any* information), we believe that there is not incompatibility between them as they can coexist in a single solution. We designed our context manager to combine these two visions: a lower layer managing information as a set of interconnected entities and an external interface allowing to access such information with different abstraction levels.

The undeniable benefit of the low level way of storing information is flexibility. Entities are basic data containers, able to store an arbitrary number of attributes and/or references to other entities. Entities can be created, modified, deleted in real time by processes internal to the context server or residing outside.

One way for creating/updating/deleting context information is to directly manipulate low level entities. The other way is to cope with more abstract resources. The former mechanism, in our proposal, is accessible by sending commands via HTTP POST, the latter is implemented as a set of RESTful services.

Both strategies have pros and cons. Sending *commands via HTTP* is a powerful way to access any stored entity, since a single HTTP POST can embed an arbitrary number of commands (e.g. for modifying as many entities as needed at a time). However, knowledge of the command language as well as entities identifiers are needed to express the needed operation(s). *RESTful services* are less performing, but provide higher level integration mechanisms, e.g. a formal interface definition.

A whole description of such features is provided in the following sections, where benefits and limitations are also discussed.

3 Architecture of the Context Management Support

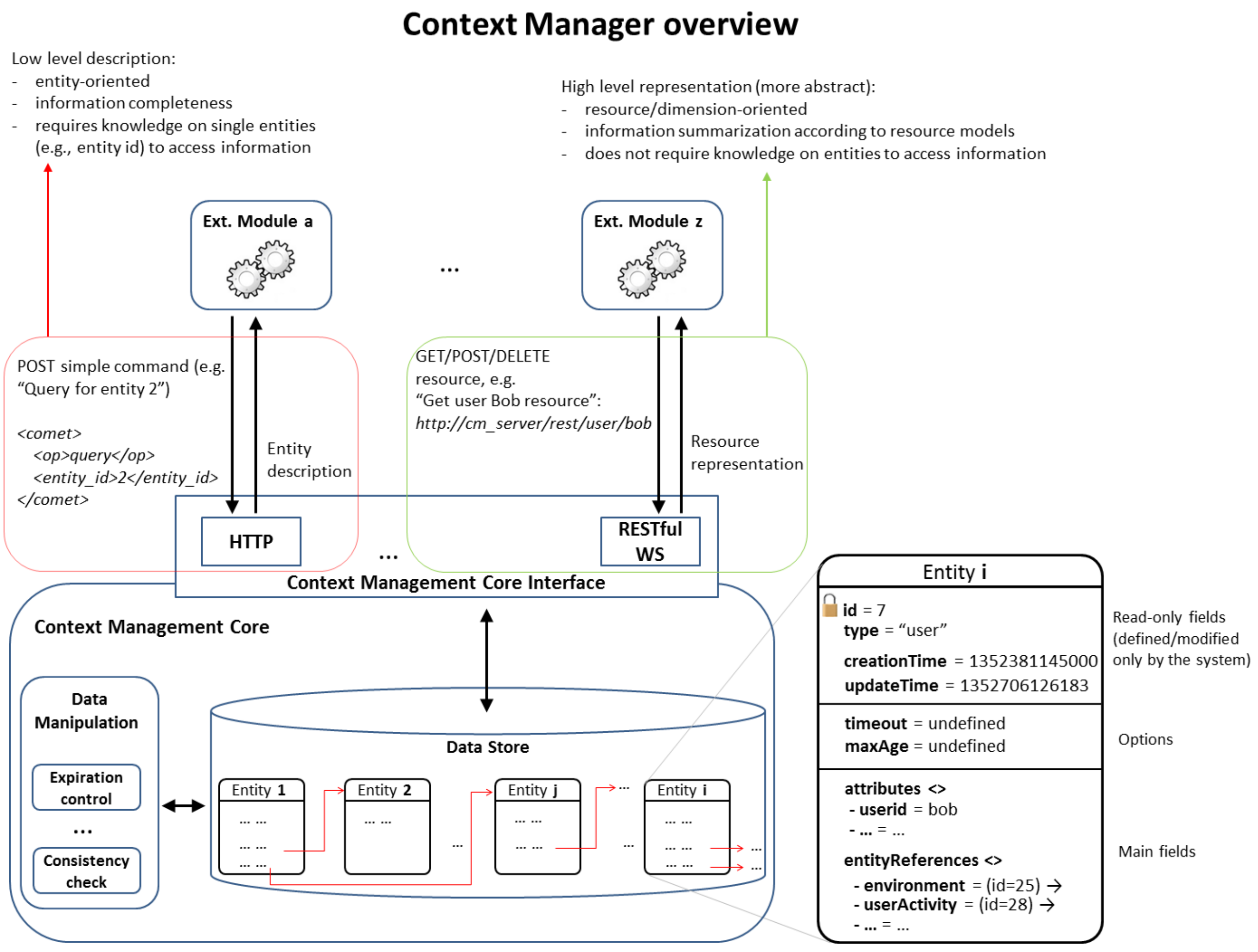

Context Manager overview
Low level description:
- entity-oriented
- information completeness
- requires knowledge on single entities (e.g., entity id) to access information
High level representation (more abstract):
- resource/dimension-oriented
- information summarization according to resource models
- does not require knowledge on entities to access information
Ext. Module a
...
Ext. Module z
POST simple command (e.g. "Query for entity 2")
<comet>
<op>query</op>
<entity_id>2</entity_id>
</comet>
Entity description
GET/POST/DELETE resource, e.g. "Get user Bob resource": http://cm_server/rest/user/bob
Resource representation
HTTP
...
RESTful WS
Context Management Core Interface
Context Management Core
Data Manipulation
Expiration control
...
Consistency check
Data Store
Entity 1
Entity 2
Entity j
Entity i
Entity i
id = 7
type = "user"
creationTime = 1352381145000
updateTime = 1352706126183
timeout = undefined
maxAge = undefined
attributes <>
- userid = bob
- ... = ...
entityReferences <>
- environment = (id=25)
- userActivity = (id=28)
- ... = ...
Read-only fields (defined/modified only by the system)
Options
Main fields

Figure 1. Context Management Core

With respect to the preliminary version, the context management support has been extended with an additional interfacing mechanism to the Context Management Core (CMC). As introduced in the previous version of this document, the CMC allows external modules to gather/store/retrieve information.

The CMC keeps data as a set of (interconnected) entities, i.e. simple data structures defined by an identifier, an array of attributes and/or references to other entities, etc. (please, refer to D4.4.1 for a full entity description). In order to access such data, external modules can rely on two mechanisms with two different abstraction levels, as illustrated in Figure 1.

The most elementary mechanism is implemented by a simple HTTP interface that accepts requests via the POST method. A request contains one or more commands in *comet* language ("comet" stands for COntext ManagemEnT, as described in D4.4.1), that encode an elementary operation to be performed on the Data Store: insertion, update, query, deletion of an entity.

In general, as already mentioned, entities can be manipulated in real time by processes that may reside inside or outside the Context Management Core. Internal processes are, for instance, the consistency daemon for recovering dangling entity references, and the expiration checker for deleting entities that go on update-timeout or that exceed the absolute time-to-live. Such internally deployed processes would include also the mechanism for allowing the Context Management Core to interface with an instance of Device Description Repository (DDR, see section 3.4). DDR indeed can contain data which could be accessed at runtime by the Context Management Core. These data can include, for instance, a set of capabilities supported by the device and/or browser in use. Indexing could occur through the user-agent of the web browser: a possible strategy is to detect the user-agent at login time (i.e., when the user accesses the platform) and provide it to the Context Manager (e.g., by calling a dedicated RESTful web service for updating the user session entity). The Context Manager would then query the DDR in order to acquire more information about the device in use.

A novel mechanism has been defined aiming to expose data in a higher abstraction level. The goal is to hide to higher layers the entities complexity, i.e. internal structure of the entities as well as the connections among them. The mechanism relies on a RESTful interface and more details are given in the following sections.

3.1 Core Functionalities

3.1.1 RESTful Web services

RESTful Web services are formally defined both in terms of methods and of input/output data by the Web Application Description Language (WADL). Each service allows manipulating a single resource without requiring knowledge of the entity/entities involved. For instance, the service to retrieve the preferences resource of a user, accessible through a URL like “http://cm-server/rest/user/{username}/preferences”, provides the description of all user preferences by hiding the complexity of gathering the entities of type “preference” referred by the user entity. The issue is here related to inefficiency when several generic entities need to be updated at a time. This is because the RESTful interface is pre-defined according to the platform requirements (i.e. the aspects that are likely to be monitored at runtime) in order to model specific resources such as user, preferences, environment, technology, etc. We refer to any module devoted to detect context information as context delegate. A context delegate, upon sensing several context parameters, usually updates the corresponding attributes in the context server. Attributes such as user location coordinates, device battery level, environment noise may refer to diverse entities/resources. If an ad-hoc service is not defined, then the delegate has to invoke the single RESTful services of the three involved resources, thus performing three HTTP requests. It is worth noting that the same updates could technically be done by packing three entity-update commands on a single HTTP POST. To this regard, bandwidth and energy usage optimization should be taken into account as context delegates are typically deployed on mobile devices.

Some of the RESTful services we have so far defined are listed in the following:

- UserEnvironment, for getting/setting the current user environment entity that includes attributes such as noise/light/temperature.
- UserDeviceBattery, for getting/setting the current user device battery entity, defining left percentage and charging status.
- UserLocationBluetoothBeacon, for getting/setting the state (e.g. signal strength) of a Bluetooth beacon detected by the device.
- UserPhysiologicalActivity, for getting/setting the current user respiration and heart rate.

GET method data is encoded in XML. For efficiency purposes, updates (POST) are encoded in JSON rather than in XML. This is done to reduce bandwidth usage, because updates are typically done by context delegates deployed on mobile devices.

We have implemented a number of context delegates for Android devices, that exploit such RESTful interfaces. Their description is given later (section 3.2).

3.1.2 Event subscription

The preliminary subscription mechanism presented in D4.4.1 allowed an external module to subscribe to an entity change. Notification was sent through a TCP message. The main issue of this approach was the inability for the subscriber to specify conditions on a specific attribute, neither belonging to the same entity nor to different entities.

An enhanced subscription mechanism has been defined for a subscriber to be notified only when one or more conditions are met on entity attributes.

Conditions refer to one or more attributes belonging, in general, to any entities. Attributes path is consistent with the context model (e.g., noise level attribute is defines on environment sub-entity of user entity). Notifications are sent via HTTP POST.

Subscription is carried out by forwarding an “event” and a “condition” as parameters of a special *comet* operation labeled “subscribe_event” (which has been added to the set of operations described in D4.4.1, sect. 5.2.1).

An example of subscription to an event is reported in the following XML code fragment. It is worth noting that *event* and *condition* elements syntax is analogous to adaptation rules, structured as event-conditions-actions, as discussed in Serenoa deliverables D3.3.1 and D3.3.2:

```
<comet>
   <op>subscribe_event</op>
   <subscriber_address>http://urano.isti.cnr.it/ae/simulatedServer?user_id=david</subscriber_address>
   <rule priority="0" name="outPromptVocal" id="rule1">
      <event>
         <simple_event   event_name="onEnvironmentNoiseLevelDecreased"   xPath="/user   /environment/@noise_level   "
externalModelId="ctx"/>
      </event>
      <condition operator="and">
         <condition operator="gt">
            <entityReference xPath="/user/environment/@noise_level" externalModelId="ctx"/>
            <constant value="20" type="int"/>
         </condition>
         <condition operator="lt">
            <entityReference xPath="/user/environment/@noise_level" externalModelId="ctx"/>
            <constant value="50" type="int"/>
         </condition>
      </condition>
   </rule>
</comet>
```

3.2 Context delegates

3.2.1 Environment noise and light

Two different delegates are devoted to user environment updating: one for the light and one for the noise level. The former parameter is provided by the device light sensor; the latter is obtained by analysing the audio input amplitude provided by the embedded microphone.

The noise delegate hosts an Android remote service, i.e. implements an external interface, through which external applications can pause/resume noise detection. This is because most devices embed a single microphone that, in the considered scenarios, can be devoted to sense noise environment as well as to get user vocal input, and this can lead to concurrence problems. The external interface we have defined allows, for instance, a vocal application to pause the noise delegate when waiting for user vocal input, thus avoiding conflicts. As soon as input has been acquired, the application resumes the noise delegate.

3.2.2 Bluetooth beacons availability

Bluetooth beacons (e.g. Bluetooth embedded devices, dongles, etc.) available in the environment are detected by a context delegate that provides the context manager with their identifier (name, MAC address) and Received Signal Strength Indicator (RSSI).

3.2.3 Physiological parameters

A context delegate has been developed for interfacing with a physiological monitoring hardware with Bluetooth interface. The physiological monitor is a belt, manufactured by Smartex[1], which can be comfortably worn and senses several parameters (e.g. heart rate, ECG-variability, respiration rate, user motion activity). The physiological context delegate connects to the hardware via Bluetooth, decodes the sensed data and periodically sends the updates to the context server through the UserPhysiologicalActivity service.

3.2.4 Device battery

Battery status (percentage left, charging flag) is detected by a context delegate that queries the operating system and updates the device battery sub-entity of user entity. Updates are posted to the UserDeviceBattery service.

3.3 Limitations and possible improvements

Limitations were found during prototyping and experimenting the Context Manager, but also possible solutions for improvements. One problem is due to inefficiency when several context delegates run at the same time in one device. Each context delegate usually updates one resource, which is accessed by invoking a RESTful service. Each invocation leads the device to make an HTTP request. The amount of HTTP requests thus grows with the number of context delegates. A centralized sensing solution would solve the issue. This could be structured in two ways:

(1) As a set of sensing delegates that connect to a single local daemon instead that to the context server. The daemon packs the samples before sending them to the context server. The local daemon would declare and implement a remote service interface generic enough to be accessed by any context delegate installed on the device. In Android devices, this can be done in AIDL[2].
(2) As a single multi-threaded super-delegate that senses all the parameters and packs the samples before sending them to the context server.

In both cases, a novel functionality is also needed at server-side. At device-side, it is only known the username and the resource to be updated (e.g. user environment), which are used to set up the RESTful URL. The novel functionality should thus accept a set of couples <RESTful URL, current resource representation>.

These technical observations and the suggested improvements are expected to assure shorter response times of the overall system and, thus, a better usability.

3.4 Device Description Repository

As described in [3], device description databases have been named as Device Description Repositories (DDRs) by the World Wide Web Consortium (W3C) through their already extinct Device Independence Working Group. DDRs include device descriptions which contain information known a priori. In this way, a client device can perform a request to a server system and the server can subsequently obtain evidences about the identity of the device. These evidences can be used to query a DDR in order to find out the actual identity of the device and its software and hardware features, thus guarantying an appropriate adaptation according to the device capabilities.

As stated in D4.4.1 (chapter 5.4.1.1), in order to allow interoperability between Serenoa and existing DDR implementations, the Consortium had agreed that any DDR technology to be used by the Serenoa framework would have to expose its functionality by means of the W3C DDR Simple API. However, considering the fine granularity required to perform context-aware adaptations, the licensing terms of most of the existing DDRs and the high costs to manually maintain a DDR, the Serenoa Consortium decided to explore

[1] http://www.smartex.it
[2] http://developer.android.com/guide/components/aidl.html

alternatives to add new device descriptions in an automated way by using automated tests, thus minimizing the need for manual additions. Due to the great difficulties associated to the construction of a universal DDR that covers all the existing device capabilities, imposed by diversity of platforms on the market, we have focused on Web technologies at a first stage.

The prototype developed aims at figuring out if the incoming web browsers support the capabilities depicted in Table 1. Such capabilities cover CSS properties, MediaQueries capabilities, HTML5 features, GPS access, etc.

Web capabilities					
animationTiming	cssBackGround	cssReflections	geoLocation	notifications	webAudioApi
animationTimingInPractice	cssBackground-Standard	cssText	getComputedStyle	offLine	webrtc
applicationCache	cssBorder-image	cssText-Standard	getElementsByClassName	postMessage	webSockets
audioElement	cssColors	cssTransform	hashChange	postMessageOnPractice	webStorage
audioMulti	cssDocType	cssTransitions	hashChangeInPractice	progressEvent	webWorkerBlob
Blob	cssElement	cssUi	history	prompts	webWorkerDataMess
blobConstructing	cssEOT	cssValues	htmlMediaCapture	selector	webWorkerGlobal
Canvas	cssFlexBox	dataset	html5	server-SentDomeventes	webWorkerLocation
canvas2D	cssFlexbox-Standard	datasetAndData	html5FormsInputs	sharedWebWorkers	webWorkerNavigation
canvas3D	cssFontFace	detailsSummary	iframe	svg	webWorkers
canvas3DStandard	cssGeneratedContent	deviceOrientation	iframeSandboxAllowScripts	svgAnimation	xhr2
classList	cssGradients	ecmaScript	iframeSandboxAllowScriptsAllowForms	svgInline	xhr2ArrayBuffer
createObjectUrl	cssImages	fileApi	iframeSandboxSanity	touchEvents	xhr2ArrayBufferResponseType
cssTextStroke	cssMediaQueries	fileReader	indexDb	typedArrays	xhr2Blob
css2-1Selectors	cssMediaQueriesApi	fileSystemFileWriterAPI	input-PlaceHolder	url	xhr2BlobResponseType
css3d	cssMinMax	flexibleBox	jSon	vibration	xhr2DocumentResponseType
css3ImagesStandard	cssOpacity	fontFace	masking	video	xhr2Upload
css3UiStandard	cssOverflow	formData	multiTouch	videoTracks	xhr2Url
cssAnimation	cssOverflow-Standard	fullScreen	navigationTiming	visibilityState	xmlHttpRequest

Table 1 - Web capabilities considered

In order to carry out the detection of these properties, we have implemented a JavaScript test suite to be executed in Web browsers. This approach facilitates test executions in comparison with native probes, since users can run the test suite without installing any additional software apart from the Web browser itself.

The test suite has been implemented by using JQuery, a powerful JavaScript library, and QUnit, a JavaScript unit testing framework commonly used in combination with JQuery. QUnit provides useful assertion methods to determine whether a test is successful or not and also offers the possibility of establishing groups of tests.

Figure 2 shows a very basic test that checks if the web browser supports the HTML canvas element.

```
test("Canvas", function()
{
   var canvas = document.createElement("canvas");

   assert( "getContext" in canvas, "canvas getContext supported" , "canvas");
   assert( "toDataURL" in canvas, "canvas toDataURL supported" , "canvas");

   var CanvasRenderingContext2D = window.CanvasRenderingContext2D,
   context = canvas.getContext("2d");

   assert( ! ! CanvasRenderingContext2D, "CanvasRenderingContext2D supported", "canvas"
);
   assert(   context   instanceof   CanvasRenderingContext2D,   "context   instanceof
CanvasRenderingContext2D", "canvas" );
   assert( ! ! context.fillText, "2D Text supported", "canvas" );
}
);
```

Figure 2 - Canvas test

In this case the evaluation of the canvas capability implies to check the support of other five sub-properties related to the canvas element: *getContext*, *toDataURL*, *CanvasRenderingContext2D*, *context instanceof*

CanvasRenderingContext2D and *2D Text*.

In our DDR prototype, all the information related to the test suite is stored in a NoSQL database (CouchDB). CouchdDB is an open-source database that stores JSON documents and uses JavaScript at server-side as query language. Moreover, CouchDB allows us to expose all the information by means of HTTP/REST services. Both test definitions and test executions are stored in CouchDB as JSON documents and are accessible by means of REST services using the basic HTTP operations: POST, GET, PUT and DELETE.

Figure 3 shows the schema of a test definition, as it is stored in the CouchDB database. In this case the *"doc_type"* attribute is *"TestDefinition"* and it contains a set of *tags* that characterizes the test. Note that this document includes the test itself (media files, HTML5, CSS and JavaScript code) as part of the *attachments* of the JSON document.

```
{
   "_id": xxx,
   "_rev": xxx,
   "doc_type" = "TestDefinition",
   "tags": ["xx","xx",...],
   "vocab_name": xxx,
   "description": xxx,
   "links": ["xxx","xxx",...],
   "name": "xxxx",
   "_attachments": {
       "xxxx.xx": {
           "content_type": xxx,
           "revpos": xx,
           "digest": xx,
           "length": xx,
           "stub": xx
       }
   }
}
```

Figure 3 - Test definition example

In order to facilitate the execution of tests, we have created a *CouchApp* that dynamically generates the tests according to users' requests. A *CouchApp* is an application written in HTML5 and JavaScript that can be served directly to the browser from *CouchDB*, without any other software in the stack.

Figure 4 shows the schema of a JSON document that represents a *test execution*. In this case the "doc_type" attribute is "test_execution" and it contains navigator features and the results of the test (name, test URI and whether the test has succeeded or not).

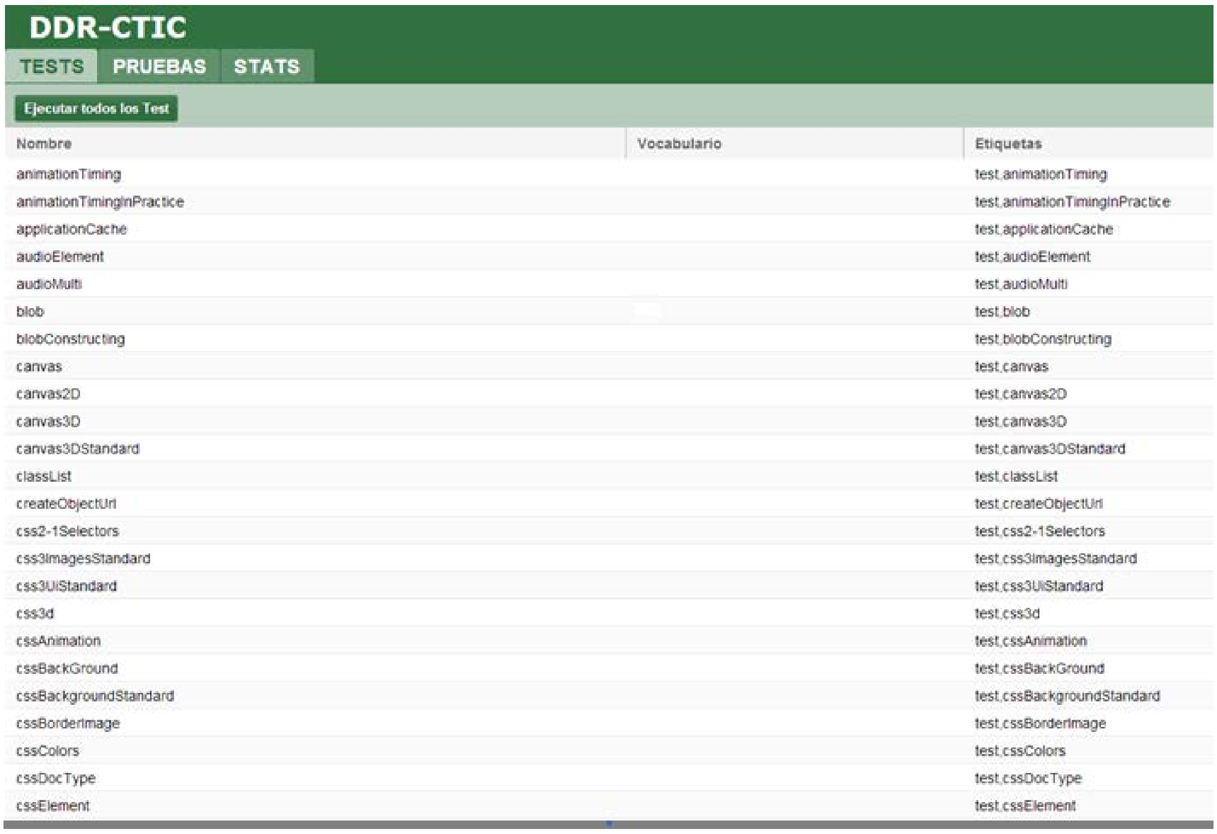
```
{
   "_id": xxx,
   "_rev": xxx,
   "doc_type" = "test_execution",
   "date" = xxxxx,
   "navigator" = {
   "appCodeName" = xxx,
   "appName" = xxx,
   "appVersion" = xxx,
   "platform" = xxx,
   "product" = xxx,
   "productSub" = xxx,
   "userAgent" = xxx,
   "vendor" = xxx,
   "vendorSub" = xxx
 },
 "results" = [
   {
       "name" = "xxxx",
        "test" = "test_url?rev=xxxx",
        "passed" = true|false


   },
   {
       "name" = "xxxx",
        "test" = "test_url?rev=xxxx",
        "passed" = true|false

   }......
 ]
}
```

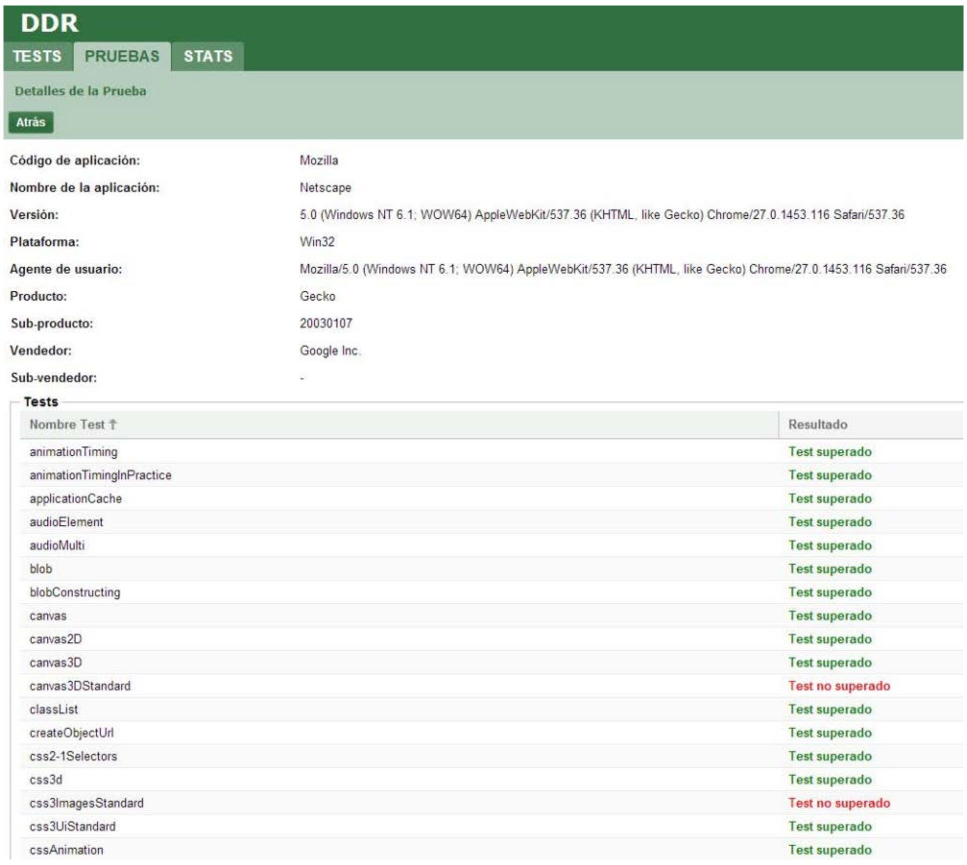
Figure 4 - Test execution example

Finally, a web interface has been developed using ExtJS library in order to allow final users to interact with the DDR, visualizing test definitions (Figure 5), analysing test executions (Figure 6) and visualizing statistical graphs about the tests (Figure 7).



Nombre	Vocabulario	Etiquetas
animationTiming		test animationTiming
animationTimingInPractice		test animationTimingInPractice
applicationCache		test applicationCache
audioElement		test audioElement
audioMulti		test audioMulti
blob		test blob
blobConstructing		test blobConstructing
canvas		test canvas
canvas2D		test canvas2D
canvas3D		test canvas3D
canvas3DStandard		test canvas3DStandard
classList		test classList
createObjectUrl		test createObjectUrl
css2-1Selectors		test css2-1Selectors
css3ImagesStandard		test css3ImagesStandard
css3UiStandard		test css3UiStandard
css3d		test css3d
cssAnimation		test cssAnimation
cssBackGround		test cssBackGround
cssBackgroundStandard		test cssBackgroundStandard
cssBorderImage		test cssBorderImage
cssColors		test cssColors
cssDocType		test cssDocType
cssElement		test cssElement

Figure 5 - Test definition visualization via Web interface


DDR

TESTS PRUEBAS STATS

Detalles de la Prueba

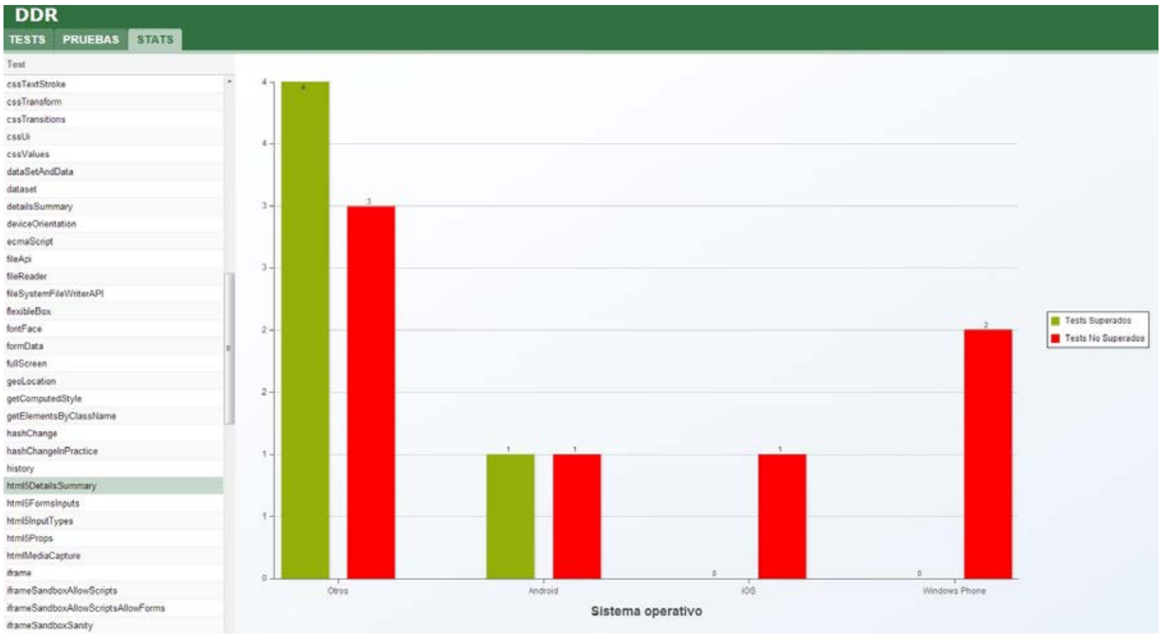
Atrás

Código de aplicación:	Mozilla
Nombre de la aplicación:	Netscape
Versión:	5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/27.0.1453.116 Safari/537.36
Plataforma:	Win32
Agente de usuario:	Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/27.0.1453.116 Safari/537.36
Producto:	Gecko
Sub-producto:	20030107
Vendedor:	Google Inc.
Sub-vendedor:	-

Tests

Nombre Test ↑	Resultado
animationTiming	Test superado
animationTimingInPractice	Test superado
applicationCache	Test superado
audioElement	Test superado
audioMulti	Test superado
blob	Test superado
blobConstructing	Test superado
canvas	Test superado
canvas2D	Test superado
canvas3D	Test superado
canvas3DStandard	Test no superado
classList	Test superado
createObjectUrl	Test superado
css2-1Selectors	Test superado
css3d	Test superado
css3ImagesStandard	Test no superado
css3UiStandard	Test superado
cssAnimation	Test superado

Figure 6 - Test execution visualization via Web interface


DDR

TESTS PRUEBAS STATS

Test
cssTextStroke
cssTransform
cssTransitions
cssUi
cssValues
dataSetAndData
dataset
detailsSummary
deviceOrientation
ecmaScript
fileApi
fileReader
fileSystemFileWriterAPI
flexibleBox
fontFace
formData
fullScreen
geoLocation
getComputedStyle
getElementsByClassName
hashChange
hashChangeInPractice
history
html5DetailsSummary
html5FormsInputs
html5InputTypes
html5Props
htmlMediaCapture
iframe
iframeSandboxAllowScripts
iframeSandboxAllowScriptsAllowForms
iframeSandboxSanity

Figure 7 - Statistics about text executions

The overall architecture of the aforementioned prototype corresponds to the scheme depicted in Figure 8. Note that all the information in the Device Description Repository is available for the Context Manager through a REST interface.

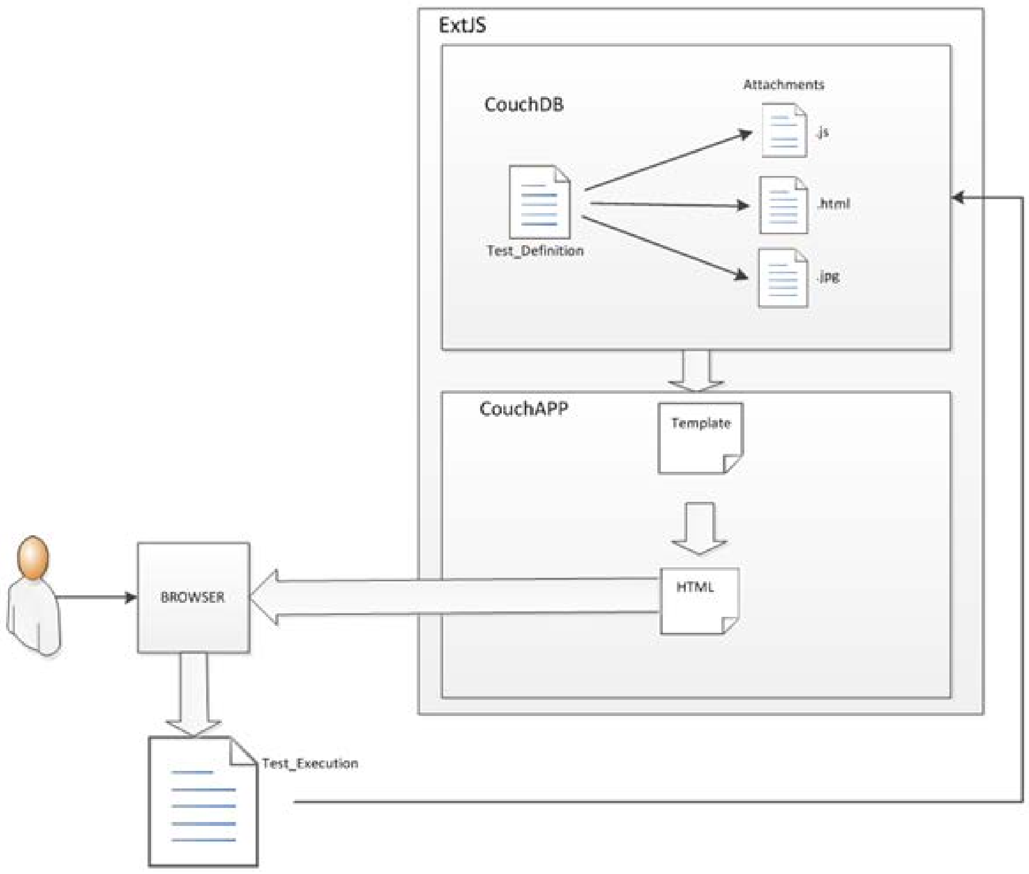


Figure 8 - Overall architecture of the DDR prototype

4 Conclusions

4.1 Summary

This deliverable describes the current implementation of the runtime infrastructure for management of Context of Use. The final version of the Context Management Core is provided, describing the capabilities as well as its external interface (e.g. the RESTful one). The Context Delegates implemented so far for mobile devices are presented.

The final version of the DDR is also discussed, including its powerful interfacing mechanisms which will allow stronger integration with the Context Management Core in future releases.

Some considerations are also drawn that will allow to further improve the support, especially from the performance point of view.

4.2 Future Work

Further developments will improve the performance, making the Context Management support more efficient. Enhancements might be done in context event detection strategies: novel aspects, such as temporal relationships among simple events, could be modelled. This would lead to the detection of more complex events, for better gathering and modelling context of use information.

5 References

1. Dey, A.K. and Abowd, G.D. Towards a better understanding of context and context-awareness. In Proc. Workshop on the What, Who, Where, When and How of Context-Awareness, ACM Press (2000).
2. Schilit, B., Adams, N. and Want, R. Context-Aware Computing Applications. In Proc. WMCSA '94, First Workshop on Mobile Computing Systems and Applications. IEEE Computer Society Washington, DC, 1994, 85-90.
3. Quiroga J., Rodríguez J., Berrueta D., Gutiérrez N., Marín I., Campos A. From UAProf towards a Universal Device Description Repository. In: Proceedings of the 3rd International Conference on Mobile Computing, Applications, and Services (MobiCASE'2011), Los Angeles, CA, October 24-27, 2011.

Acknowledgements

- TELEFÓNICA INVESTIGACIÓN Y DESARROLLO, http://www.tid.es
- UNIVERSITE CATHOLIQUE DE LOUVAIN, http://www.uclouvain.be
- ISTI, http://hiis.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/