simpli·city

The Road User Information System Of The Future

WP5 – Mobility Services Framework

D5.3.2: Service Registry Prototype

Deliverable Lead: TUV

Contributing Partners: TIE

Delivery Date: 09/2014

Dissemination Level: Public

Version 1.00

This deliverable describes the work carried out during the development of the Service Registry Prototype of the SIMPLI-CITY platform. It specifies how to install and execute the different subcomponents implemented.

This project has received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no 318201

Document Status	
Deliverable Lead	Philipp Hoenisch, TUV
Internal Reviewer 1	Antonio Martínez, Worldline
Internal Reviewer 2	Marina Giordanino, CRF
Type	Deliverable
Work Package	WP5: Mobility Services Framework
ID	D5.3.2: Service Registry Prototype
Due Date	30.09.2014
Delivery Date	24.09.2014
Status	Approved

Document History	
Contributions	V0.1, Philipp Hoenisch, TUV, 03.09.2014, Added document structure. V0.2, Vadim Petrenko: TIE, 08.09.2014, Feedback. V0.3, Philipp Hoenisch, TUV, 03.09.2014, Updated. V0.4, Stefan Schulte, TUV, 10.09.2014, Latest revision. V0.5 Philipp Hoenisch, TUV, 10.09.2014, Integrated comments, updated interface descriptions V0.5.1, Philipp Hoenisch, TUV, 15.09.2014, integrated comments from Marina V0.5.2, Philipp Hoenisch, TUV, 16.09.2014, integrated comments from Toni V0.6, Philipp Hoenisch, TUV, 16.09.2014, ready for review 2 V0.7, Philipp Hoenisch, TUV, 18.09.2014, finalized V1.0, Stefan Schulte, TUV, 24.09.2014, final layout changes
Final Version	September 24th, 2014

Disclaimer

The views represented in this document only reflect the views of the authors and not the views of the European Union. The European Union is not liable for any use that may be made of the information contained in this document.

Furthermore, the information is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose. The user of the information uses it at its sole risk and liability.

Project Partners

Vienna University of Technology (Coordinator), Austria

Ascora GmbH, Germany

TIE Nederland B.V., The Netherlands

Technische Universität Darmstadt, Germany

IBM Research – Ireland
Smarter Cities Technology Centre

Forschungsgesellschaft Mobilität, Austria

Talkamatic AB, Sweden

Atos Worldline, Spain

Centro Ricerche FIAT, Italy

SRM – Retie Mobilità, Italy

Executive Summary

This deliverable describes the work which was carried out during the development of the SIMPLI-CITY Service Registry Prototype. For this purpose, the document starts with introducing the Service Runtime Environment and describes the scope of the Service Registry.

Afterwards, the degree of fulfilment of each requirement to be covered by the component and specified in the Requirements Analysis Report (D2.3) is described.

The final prototype of the Service Registry provides Java and RESTful interfaces to service developers to perform CRUD actions (i.e., create services, update services and their descriptions, read/get services and delete services). In order to evaluate these functionalities, an optional Java Command Line Client is provided. Further, the use of the RESTful interface is demonstrated. In addition to that, the functionality of the Service Watch is described as well as how to make use of its RESTful interfaces.

The last sections of this document describe how potential users (i.e., service developers and administrators) can prepare, install and execute the different parts of the Service Registry. For this purpose, a step-by-step guideline to install and make use of the prototype is provided.

This deliverable D5.3.2 will be superseded by deliverable D5.3.3, which will be the final prototype of the Service Runtime Environment.

http://www.simpli-city.eu/	Copyright © SIMPLI-CITY Project Consortium. All Rights Reserved. Grant Agreement No.: 318201

Table of Contents

1 Introduction

SIMPLI-CITY – The Road User Information System of the Future – is a project funded by the Seventh Framework Programme of the European Commission under Grant Agreement No. 318201. It provides the technological foundation for bringing the “App Revolution” to road users by facilitating data integration, service development, and end user interaction.

Within this document, the prototype of the SIMPLI-CITY Service Registry will be presented. The document accompanies the corresponding software prototype, which is the main content of the deliverable.

1.1 SIMPLI-CITY Project Overview

Analogously to the “App Revolution”, SIMPLI-CITY adds a “software layer” to the hardware-driven “product” mobility. SIMPLI-CITY will take advantage of the great success of mobile apps that are currently being provided for systems such as Android, iOS, or Windows Phone. These apps have created new opportunities and even business models by making it possible for developers to produce new apps on top of the mobile device infrastructure. Many of the most advanced and innovative apps have been developed by players formerly not involved in the mobile software market. Hence, SIMPLI-CITY will support third party developers to efficiently realise and sell their mobility-related service and app ideas by a range of methods and tools, including the Mobility Services and App Marketplaces.

In order to foster the wide usage of those services, a holistic framework is needed which structures and bundles potential services that could deliver data from various sources to road user information systems. SIMPLI-CITY will provide such a framework by facilitating the following main project results:

- Mobility Services Framework: A next-generation European Wide Service Platform (EWSP) allowing the creation of mobility-related services as well as the creation of corresponding apps. This will enable third party providers to produce a wide range of interoperable, value-added services, and apps for drivers and other road users.
- Mobility-related Data as a Service: The integration of various, heterogeneous data sources like sensors, cooperative systems, telematics, open data repositories, people-centric sensing, and media data streams, which can be modelled, accessed, and integrated in a unified way.
- Personal Mobility Assistant: An end user assistant that allows road users to make use of the information provided by apps and to interact with them in a non-distracting way – based on a speech recognition approach. New apps can be integrated into the Personal Mobility Assistant in order to extend its functionalities for individual needs.

To achieve its goals, SIMPLI-CITY conducts original research and applies technologies from the fields of Ubiquitous Computing, Big Data, Media Streaming, the Semantic Web, the Internet of Things, the Internet of Services, and Human-Computer Interaction. For more information, please refer to the project website at http://www.simpli-city.eu.

1.2 Deliverable Purpose, Scope and Context

The purpose of this document is to provide the means to use the final prototype of the SIMPLI-CITY Service Registry and exploit its functionalities. For this, the scope and requirements of the Service Registry prototype, the requirements and preparations for administrators and developers, an installation and usage guide is provided.

The final Service Registry prototype is the outcome of the discussions and implementation work done in project months 12 to 24. It provides the final implementation of the functionalities of the Service Registry as defined within SIMPLI-CITY deliverables D3.2.1 (Functional Specification), and D3.2.2 (Technical Specification). This deliverable D5.3.2 will be superseded by deliverable D5.3.3, which will provide the final prototype of the Service Runtime Environment, although, the functionality of the Service Registry can be seen as finished and will not be extended. Any further minor change (if any) will be described in deliverable D5.3.3.

1.3 Document Status and Target Audience

This document is listed in the Description of Work (DoW) as “Public”. It provides the means to exploit the functionalities of the SIMPLI-CITY Service Registry as defined in deliverable D3.2.2 (Technical Specification).

While the document primarily is aimed at the project partners, this public deliverable can also be useful for the wider scientific and industrial community. This includes other publicly funded projects, which may be interested in collaboration activities.

1.4 Abbreviations and Glossary

A definition of common terms and roles related to the realization of SIMPLI-CITY as well as a list of abbreviations is available in the supplementary document “Supplement: Abbreviations and Glossary”, which is provided in addition to this deliverable.

Further information can be found at http://www.simpli-city.eu.

1.5 Document Structure

This deliverable is broken down into the following sections:

Section 1 provides an introduction for this deliverable including a general overview of the project, and outlines the purpose, scope, context, status, and target audience of this deliverable.

Section 2 provides an overview of the scope and relationship of the prototype, showing where the Service Registry fits into the Service Runtime Environment and the overall SIMPLI-CITY Mobility Services Framework. Furthermore, an assessment of the requirements covered by this prototype is given.

Section 3 presents the requirements and preparations to be done by software developers if they want to make use of the Service Registry within the Service Runtime Environment.

Section 4 states information about the installation and deployment of the provided software package.

Section 5 describes how software developers can use the provided functionalities.

http://www.simpli-city.eu/	Copyright © SIMPLI-CITY Project Consortium. All Rights Reserved. Grant Agreement No.: 318201

Section 6 discusses any limitations of the Service Registry prototype.

Finally, Section 7 provides a summary of the document.

2 Prototype Scope and Requirements Coverage

2.1 Service Registry – General Information

The Service Registry stores software service artifacts along with their description in order to be accessible by other services or apps running on the Personal Mobility Assistant (PMA). Since apps are not running inside the Service Runtime Environment, the Service Registry provides RESTful interfaces next to the Java interfaces for internal services.

These interfaces provide the means of querying the Service Registry for services according to a particular filter. Further, the Service Registry also offers Java and RESTful interfaces to create, update or delete services including their description to software developers. Thus, developers can deploy their services in the Service Runtime Environment, update the service description or the whole service, or delete everything if not needed anymore. Furthermore, the functionality for retrieving service descriptions by a filter was implemented.

Beside of this, the Service Registry also provides the means of creating, updating or deleting licenses for a particular service, creating and updating a service watch and retrieving historical service watch data.

Figure 1 shows the location of the Service Registry (blue) in the SIMPLI-CITY Global Architecture. For the full Global Architecture, refer to deliverable D3.1.

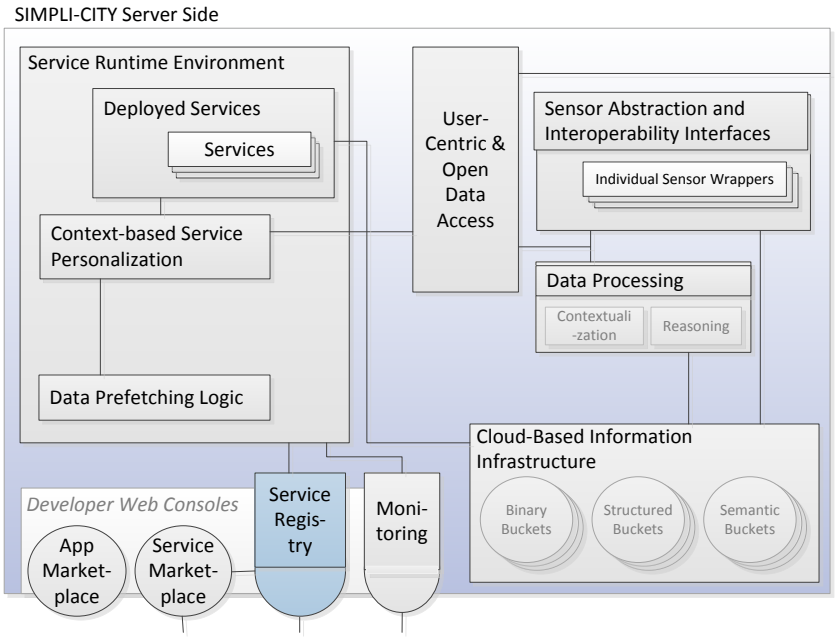

Figure 1: Location of the Service Registry in the SIMPLI-CITY Global Architecture

2.2 Scope of this Prototype

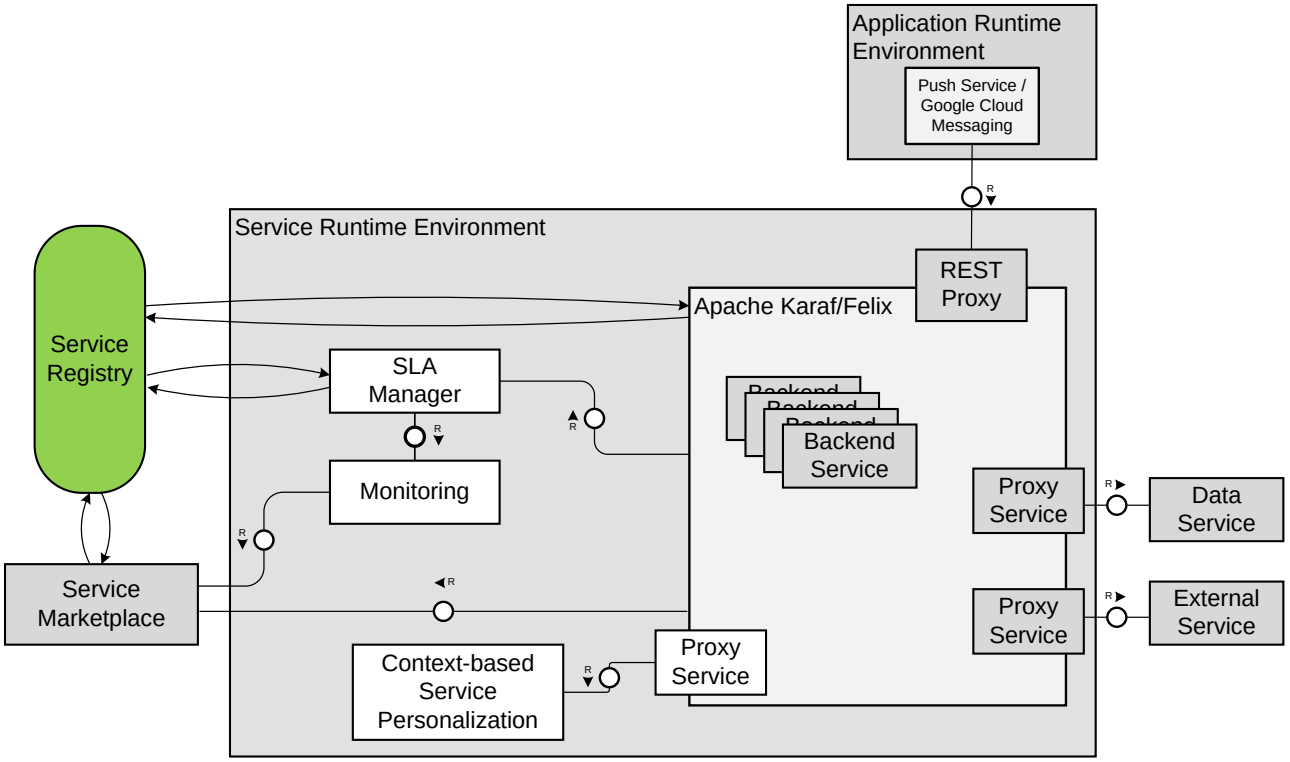

Figure 2: Scope of the Service Registry and the Service Runtime Environment

Figure 2 depicts the Service Registry and its relation to the Service Runtime Environment. For that, the subcomponents that are covered within this deliverable are highlighted. Note, that this figure shows the Service Runtime Environment and also other parts of the SIMPLI-CITY Global Architecture which are not developed within this component. For further information on the Service Runtime Environment, please refer to the upcoming D5.3.3.

The status of the implementation is shown using the following colour codes:

- Green: Fully implemented
- Orange: Partially implemented
- White/Grey: No implementation so far, or not part of this deliverable

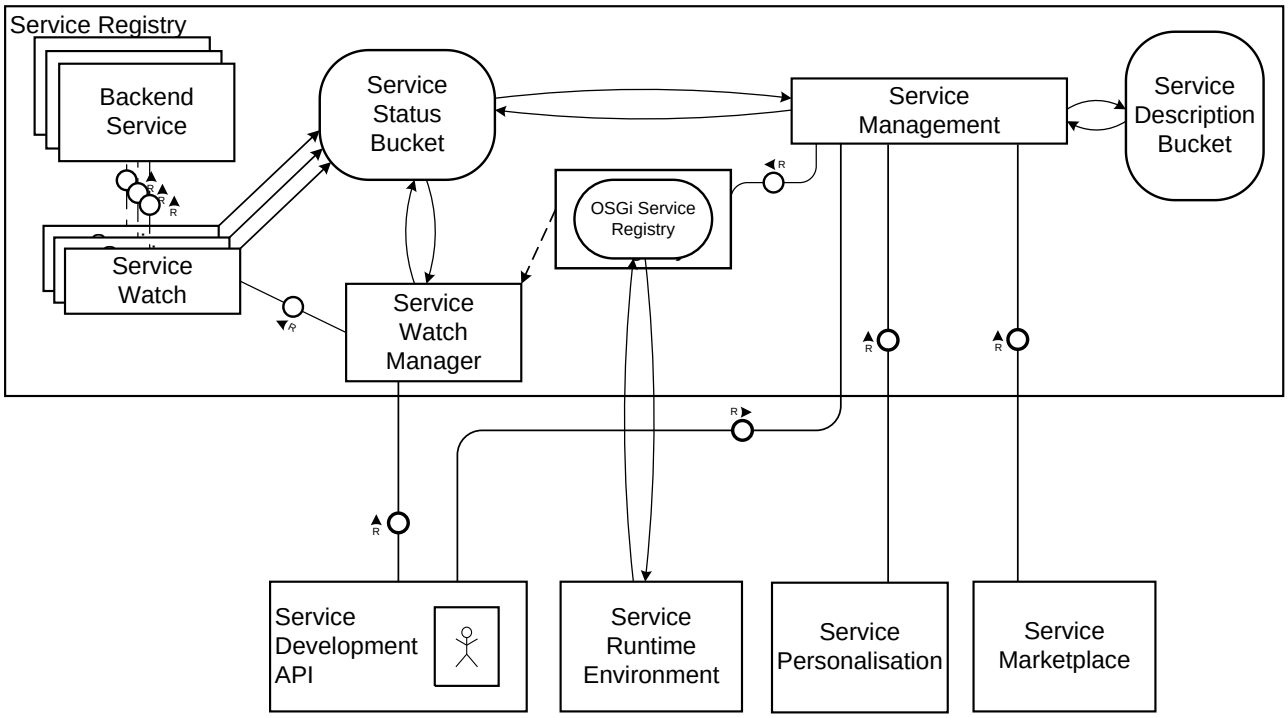

Figure 3: Service Registry Details

Figure 3 shows the Service Registry and its subcomponents more detailed. In the following subsections, the scope of the single subcomponents (as depicted in Figure 3) will be further discussed. For the Functional Specification and Technical Specification of these subcomponents, refer to SIMPLI-CITY deliverables D3.2.1 and D3.2.2, respectively.

2.2.1 Service Management

The Service Management subcomponent offers the central API functionalities to all external components either via plain Java or RESTful interfaces, e.g., the methods to create, delete, or update service artifacts and service descriptions within the OSGi Service Registry. The service descriptions are stored within the Service Description Bucket, which is a XML database located on the same machine as the Service Runtime Environment. Last but not least, it provides any external component with the means to access data from the Service Status Bucket.

2.2.2 OSGi Service Registry

The OSGi Service Registry is the basic subcomponent of the SIMPLI-CITY Service Registry and actually offers all registry and repository functionalities necessary for the operation of services within the SIMPLI-CITY Service Runtime Environment, i.e., the means to store services and look them up in order to invoke them. For this, the OSGi Service Registry offers an internal data storage (not depicted in Figure 3) as well as the according methods.

2.2.3 Service Description Bucket

The Service Description Bucket is a supplemental data storage for service-related data that cannot be stored within the OSGi Service Registry, i.e., the detailed service

description in form of XML. For that, an XML database is deployed which provides the means to search for Service Descriptions by XPath queries.

2.2.4 Service Watch Manager, Service Watch and Service Status Bucket

The Service Watch Manager and the Service Watch offer the possibility to regularly poll data services in order to make sure if a service is still alive. A watch can be set up for a particular data service through the Service Development API, by naming the data service to be watched, the time interval for the polls, and the storage lifetime of the data from these polls.

This data is stored in the Service Status Bucket and can be accessed by other components through the Service Management API, e.g., in order to provide a service developer with live data about the availability of a particular data source. The Service Status Bucket is a database realized through the means of the Cloud-based Information Infrastructure (see deliverable D4.2).

2.3 Covered Requirements

This section describes the degree of fulfilment of the requirements to be covered by Service Runtime Environment and specified in the Requirements Analysis document (D2.3) and the Functional Specification (D3.2.1).

Table 1: Requirements Related to Service Registry and their Degree of Fulfilment

Requirement	Degree of Fulfilment	Comment
Must Have Requirements		
U148: Service registration	100%	This functionality is fully implemented, tested and working.
U149: Service extension and modification U150: Service control	100%	This functionality is fully implemented, tested and working, i.e., it is possible to update or delete services.
U151: Service versioning	75%	This functionality has been partly implemented within this prototype. The missing parts will be implemented within the SRE deliverable D5.3.3. In more details, the REST Proxy has still to be extended allowing the selection of a particular version of a service.

http://www.simpli-city.eu/ Copyright © SIMPLI-CITY Project Consortium. All Rights Reserved. Grant Agreement No.: 318201

U152: SLA support	100%	This functionality is fully implemented. However, the SLA compliance check is not done within the Service Registry and remains to the Service Monitor which is part of the SRE and will be implemented in D5.3.3.
U161: Data services (P1) U162: Backend services (P2)	100%	This functionality covers basic CRUD methods for Data and Backend services and was fully implemented.
U179: Service Metadata (P1)	100%	This functionality is fully implemented, i.e., it is possible to lookup services based on their metadata, e.g., the service description.
Should Have Requirements		
U92: Secure access to system U93: Third party access to the system	0%	The secure access, especially for third parties, was not part of the Service Registry.
U125: Open interfaces U126: Openness of the system U127: Extensibility	100%	The interfaces of this prototype are well defined and documented and therefore the SIMPLI-CITY Service Runtime Environment is easily extendible.
U153: Usage of an official SLA standard (P4) U154: Simple SLA description standard (P2)	0%	This requirement was not covered within the Service Registry as an official standard (e.g., WLSA[1]), would be too extensive and would go beyond the constraints of this prototype.
U166: Service Developer Signature (P1)	100%	This functionality was covered within this prototype, i.e., it is possible to define a service owner (alias developer) within the Service Description.

[1] http://domino.watson.ibm.com/library/cyberdig.nsf/papers/CDEDB79080F59EE285256C5900654839/$File/RC22456.pdf

http://www.simpli-city.eu/ Copyright © SIMPLI-CITY Project Consortium. All Rights Reserved. Grant Agreement No.: 318201

Could Have Requirements		
U101: Backwards compatibility of services (P2)	50%	Although the Service Registry already provides the possibility to distinguish between different versions of services, this functionality remains to be fully implemented within the Service Registry, i.e., within the REST Proxy.
U182: Service crash reports (P4)	75%	As the Service Watch is not an active component, it is not able to notify an interested observer if a data service is not responding anymore. However, a Service Monitor will be implemented within the SRE in deliverable D5.3.3.

http://www.simpli-city.eu/ Copyright © SIMPLI-CITY Project Consortium. All Rights Reserved. Grant Agreement No.: 318201

3 Preparations

Since the Service Registry is an integral part of the SIMPLI-CITY Service Runtime Environment, it cannot be run outside it. This section describes the information about what potential users (both administrators and software developers) need to prepare in order to use the functionalities of the delivered prototype.

The server side part of the Service Runtime Environment (and therefore the Service Registry) will be executed by administrators of the SIMPLI-CITY Mobility Services Framework.

3.1 Server Side (System Administrators)

In order to make use of the Service Registry prototype and the Service Runtime Environment, Oracle's Java SE Development Kit 7 and Apache Maven 3.0.5 need to be installed on the system. In addition, port 8080 has to be available for the Service Runtime Environment as this port will be used for the Service Registry. If this port is not available, the Service Runtime Environment will not work properly.

Furthermore, the Service Runtime Environment requires a valid internet connection; otherwise, some of the services may fail.

The prototype was tested on a Linux machine running Ubuntu 12.04, Java(TM) SE Runtime Environment (build 1.7.0_51-b13) and Apache Maven 3.0.5.

In addition, it is required that the environment variable JAVA_HOME is defined as per Oracle guidelines.[2]

[2] http://docs.oracle.com/cd/E19182-01/820-7851/inst_cli_jdk_javahome_t/index.html

http://www.simpli-city.eu/	Copyright © SIMPLI-CITY Project Consortium. All Rights Reserved. Grant Agreement No.: 318201

4 Installation (Deployment)

While Section 3 provides information about the technical preparation before the installation of the software, this section provides guidelines on how to install and deploy the Service Runtime Environment and the Service Registry. The following files can be found next to this deliverable:

- service-runtime-environment.zip: The SRE itself
- dataservices-0.0.1-SNAPSHOT.kar: Data services which are needed to test the Service Watch
- vienna-service.jar & vienna-service.xml: An example service including description

The following provides a manual installation for a Unix-based system. The test environment was Ubuntu 12.04.

1. Unpack the provided archive file (service-runtime-environment.zip) and change into the resulting folder using a terminal window.
2. Navigate into the “bin” folder
3. Type into the terminal: “./karaf”. This will start the Service Runtime Environment. If the output is similar to Figure 4, the Service Runtime Environment should be up and running.

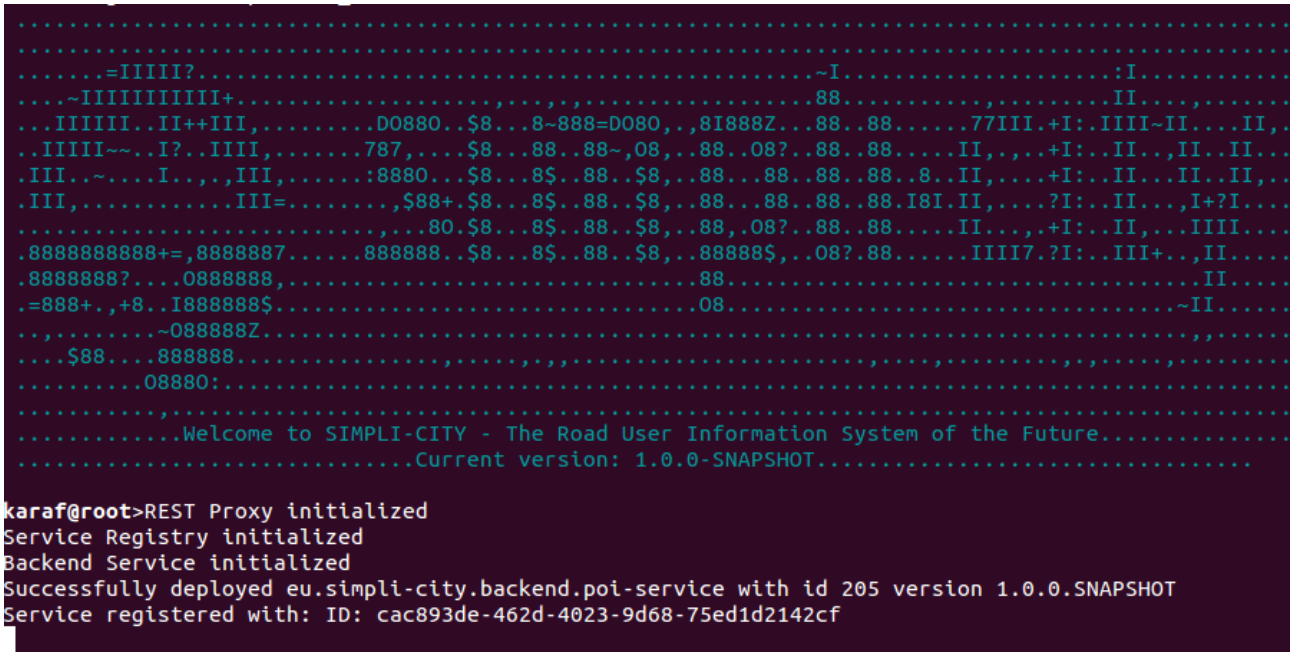

Figure 4: Service Runtime Environment Start Up Output

If no error was printed, the Service Runtime Environment has been started successfully. In addition, the second line, “Service Registry initialized” indicates that the Service Registry was successfully started.

4.1 Install Data Services

In order to test the Service Watch components it is necessary to install some additional data services. These are provided in an additional file coming with this deliverable: dataservices-0.0.1-SNAPSHOT.kar

In order to install them

1. First start the Service Runtime Environment as described above
2. Type the following command into the Service Runtime Environment console. kar:install file:/path/to/dataservices-0.0.1-SNAPSHOT.kar (Notably: /path/to has to be replaced with the correct absolute path). If the installation worked correctly, the console output should not show any error message.

http://www.simpli-city.eu/	Copyright © SIMPLI-CITY Project Consortium. All Rights Reserved. Grant Agreement No.: 318201

5 Execution and Usage of the Software

This section describes how to use the Service Registry and the Service Watch.

5.1 Server Side (System Administrators)

Within this section it is shown how the Service Registry can be tested and/or used. More in particular, a test client is explained, which demonstrates the functionality of the Service Registry including how to use the Service Watch via the command line tool CURL[3].

5.1.1 Service Registry Test Client (Server Side)

Since the Service Registry is part of the Service Runtime Environment, it is already setup and deployed by default on startup (see Section 4).

In order to test the Service Registry, a simple test client is deployed in the Service Runtime Environment and a simple example service is provided with this prototype (see additional files vienna-service.jar and vienna-service.xml). In addition, a Point of Interest (POI) service is deployed on startup. The test client provides the following functionality:

Table 2: Service Registry Test Client Functionality

Functionality	Command	Description
Create a service	registry:create /path/to/vienna-service.jar /path/to/vienna-service.xml	By executing this command, a service is deployed and started within the Service Runtime Environment. Notably: The absolute path to the JAR and XML file has to be specified. If no error occurred the console should show an output similar to Figure 5.
Update a service description	registry:update f333b510-b4fb-11e3-a5e2-0800200c9a66 /path/to/vienna-service.xml	By executing this command, a particular service description is updated within the Service Runtime Environment. Notably, the given ID should be changed and the absolute path to the XML file has to be specified. If no error occurred the console should show an output similar to Figure 6.
Get service description	registry:getService "//ServiceDescription[//keyword='Vienna']"	Retrieves the service description according to the provided filter and prints it on the screen. An example output can be found in Figure 7.
Delete a service	registry:delete f333b510-b4fb-11e3-a5e2-0800200c9a66	Deletes a particular service by the given ID. An example output can be found in Figure 8.

[3] http://curl.haxx.se/

Copyright © SIMPLI-CITY Project Consortium. All Rights Reserved. Grant Agreement No.: 318201

```
karaf@root>registry:create                          /simpli-city/t5-3/D5.3.2/vienna-service.jar
                /simpli-city/t5-3/D5.3.2/vienna-service.xml
Deploying a new service
Vienna Service started
Service ID: 2fd9be13-e2a4-404f-9f48-77e5eab3cdfc
Status Code: 201; Status Message: Successfully deployed eu.simpli-city.viennaService with id 206 ver
sion 1.0.0.SNAPSHOT
karaf@root>
```

Figure 5: Service Registry: Create Example Service

```
karaf@root>registry:update 2fd9be13-e2a4-404f-9f48-77e5eab3cdfc                    /simpli-city/
t5-3/D5.3.2/vienna-service.xml
Updating service
Status Code: 200; Status Message: Update successful
karaf@root>
```

Figure 6: Service Registry: Update Service

```
karaf@root>registry:getService "//ServiceDescription[//keyword='Vienna']"
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><ServiceDescription><name>ViennaService</name
><version>1.0</version><description>Vienna Service is just an example service</description><keywords
><keyword>Vienna</keyword></keywords><runtimeVersion>add me if ram is considered</runtimeVersion><li
cense><licenseID>51588387-d2aa-41f9-895a-8d0e47745e21</licenseID><freeService>true</freeService></li
cense><marketPlaceInfo><faviconURL>http://thecatapi.com/api/images/get?format=src&amp;type=gif</favi
conURL></marketPlaceInfo><serviceID>2fd9be13-e2a4-404f-9f48-77e5eab3cdfc</serviceID><serviceType>BAC
KEND_SERVICE</serviceType><bundleId>206</bundleId></ServiceDescription>
karaf@root>
```

Figure 7: Service Registry: Get Service Description

```
karaf@root>registry:delete 73b9f3ef-4679-409d-a8f8-adfca161eda9
StatusCode: 200 StatusMessage: Service deleted
karaf@root>
```

Figure 8: Service Registry: Delete Service

5.2 Service Runtime Environment Service Developer Usage Guide

The following subsections shows how the subcomponents covered within the first prototype can be used by service developers.

5.2.1 Service Registry RESTful Interfaces

Besides the test client (see Section 5.1.1), the Service Registry also exposes its interfaces as RESTful interfaces. These can be called from outside the Service Runtime Environment (e.g., a browser or another application). A valid WADL file for the Service Registry can be found (provided that the Service Runtime Environment is started) at http://localhost:8080/cxf/serviceRegistry?_wadl.

So far, the interfaces listed in Table 3 are implemented. For the corresponding API references, please refer to deliverable D3.2.2.

Table 3: Service Registry RESTful Interface

Functionality	Type	Command	Parameters	Description
Create a service	HTTP PUT	http://localhost:8080/cxf/serviceRegistry/createService/	service.jar and manifest.xml as stream in the body (see D3.2.2 for more information)	Allows deploying a service via an HTTP PUT command.
Update a service	HTTP POST	http://localhost:8080/cxf/serviceRegistry/updateService?serviceID=<….>/	service.jar as stream in the body and the service ID as parameter (see D3.2.2 for more information)	Allows updating a service file via an HTTP POST command.
Update a service	HTTP POST	http://localhost:8080/cxf/serviceRegistry/updateServiceDescription	manifest.xml as stream or String in the body (see D3.2.2 for more information)	Allows updating a service description via an HTTP POST command.
Get service description	HTTP GET	http://localhost:8080/cxf/serviceRegistry/getServices?filter=<filter>	<filter> (see D3.2.2 for more information): String	Allows retrieving a service description according the provided filter (xpath format). An example output can be found in Figure 9.
Get service binaries	HTTP GET	http://localhost:8080/cxf/serviceRegistry/getServiceBinaries?serviceID=<serviceID>	<serviceID>: UUID	Allows retrieving the service binaries defined by <serviceID>.
Delete a service	HTTP DELETE	http://localhost:8080/cxf/serviceRegistry/deleteService?serviceID=<serviceID>	<serviceID>: UUID	Allows deletion of a service defined by its ID.

http://www.simpli-city.eu/ | Copyright © SIMPLI-CITY Project Consortium. All Rights Reserved. Grant Agreement No.: 318201

Figure 9: Service Registry: Result of Get Service Description via Browser

5.2.2 Service Watch RESTful Interfaces

As the Service Registry, the Service Watch also provides RESTful interfaces, which can be executed from outside the Service Runtime Environment. Compared to D3.2.2, a few changes were made to the interfaces, i.e., parameter types have changed. Updated tables for these interfaces are provided below (Table 4, Table 5, Table 6). Valid URL examples can be found in Listing 1, Listing 2 and Listing 3.

Table 4: RESTful Interface Description – Create a Service Watch

Method	PUT	**URL**	`$API_ROOT/createServiceWatch?:serviceID&:lifetime&:interval&:retentionPeriod`				
Description	Creates a new Service Watch for a particular service						
Parameter	serviceID	**Required**	yes	**Possible Values**	UUID	**Description**	Specifies the service to be watched
Parameter	lifetime	**Required**	yes	**Possible Values**	Long	**Description**	Defines the lifetime (in milliseconds) for the service watch
Parameter	interval	**Required**	yes	**Possible Values**	Long	**Description**	Defines the interval (in milliseconds) for the service watch
Parameter	retentionPeriod	**Required**	yes	**Possible Values**	Long	**Description**	Defines the retention period (in milliseconds) for the service watch
Example URL	`$API_ROOT/createServiceWatch?serviceId=temperature-vie&lifetime=600000&interval=1000&retentionPeriod=3600000`						
Response	HTTP status code + JSON object						
HTTP Status Code		**Required**	yes	**Possible Values**	200	**Description**	Service watch created successfully
					400		Bad request
					404		No service found
					424		Unknown error occurred
JSON Object	http://SIMPLI-CITY.eu/ServiceRegistry/JSON-Schema/StatusCode						
JSON Attribute	serviceID	**Required**	yes	**Possible Values**	128 bit UUID	**Description**	ID for the service
JSON Attribute	statusCodeNum	**Required**	yes	**Possible Values**	number	**Description**	Status code as number (same value as the HTTP status code)
JSON Attribute	statusCodeText	**Required**	yes	**Possible Values**	any string	**Description**	Status code in form of text
Example Response	`HTTP/1.1 200 OK`						

http://www.simpli-city.eu/	Copyright © SIMPLI-CITY Project Consortium. All Rights Reserved. Grant Agreement No.: 318201

Table 5: RESTful Interface Description – Deleting a Service Watch

Method	DELETE	**URL**	`$API_ROOT/deleteServiceWatch?:serviceID`				
Description	Deletes a Service Watch from the Service Registry						
Parameter	serviceID	**Required**	yes	**Possible Values**	UUID	**Description**	Specifies the service watch to be deleted
Example URL	`$API_ROOT/deleteServiceWatch?serviceID=temperature-vie`						
Response	HTTP status code + JSON object						
HTTP Status Code		**Required**	yes	**Possible Values**	200	**Description**	Service Watch successfully deleted
					404		Service Watch to be deleted not found
JSON Object	http://SIMPLI-CITY.eu/ServiceRegistry/JSON-Schema/StatusCode						
JSON Attribute	serviceID	**Required**	yes	**Possible Values**	128 bit UUID	**Description**	ID of the service
JSON Attribute	statusCodeNum	**Required**	yes	**Possible Values**	number	**Description**	Status code as number (same value as the HTTP status code)
JSON Attribute	statusCodeText	**Required**	yes	**Possible Values**	any string	**Description**	Status code in form of text
Example Response	`HTTP/1.1 200 OK`						

Table 6: RESTful Interface Description – Retrieve Service Watch Data Entries

Method	GET	**URL**	`$API_ROOT/getServiceWatchData?:serviceId&:start&:end`				
Description	Retrieves a collection of Service Watch data entries for a particular service in a defined timespan						
Parameter	serviceId	**Required**	yes	**Possible Values**	UUID	**Description**	Specifies service for which the serviceWatchData shall be returned
Parameter	start	**Required**	yes	**Possible Values**	long	**Description**	Specifies the start date (unix time) for the first entry
Parameter	end	**Required**	yes	**Possible Values**	long	**Description**	Specifies the end date(unix time)
Example URL	`$API_ROOT/getServiceWatchData?serviceID=temperature-vie&start=1404900600&end=1404915000`						
Response	HTTP status code + JSON object						
HTTP Status Code		**Required**	yes	**Possible Values**	200	**Description**	Service Watch successfully deleted
					404		Service Watch to be deleted not found
JSON Object	http://SIMPLI-CITY.eu/ServiceRegistry/JSON-Schema/ServiceWatchData						
JSON Attribute	serviceID	**Required**	yes	**Possible Values**	128 bit UUID	**Description**	ID of the service
JSON Attribute	timestamp	**Required**	yes	**Possible Values**	date	**Description**	The date when this entry was recorded
JSON Attribute	responsetime	**Required**	yes	**Possible Values**	long	**Description**	The responsetime of the watched service
JSON Attribute	not_responding	**Required**	yes	**Possible Values**	boolean	**Description**	Indicates if the service was responding
Example Response	`HTTP/1.1 200 OK`						

Beside of the tables above, Table 7 shows the available commands including a short description.

Table 7: Service Watch RESTful Interface

Functionality	Type	Command	Parameters	Description
Create/Update a Service Watch	HTTP PUT	http://localhost:8080/cxf/createServiceWatch?<serviceID>&<lifetime>&<interval>&<retentionPeriod>	<serviceID>: UUID: defines a service for which the Service Watch shall be created <lifetime>: long: defines the lifetime for the Service Watch, i.e., how long the Service Watch should exist <interval>: long: defines the interval of how often the service should be checked <retentionPeriod>: long: defines the retention period for the Service Watch Data	As there can only exist one Service Watch per service, the same method can be used for creating or updating a Service Watch, i.e., if a Service Watch already exists, the old one gets deleted.
Delete a Service Watch	HTTP DE-LETE	http://localhost:8080/cxf/deleteServiceWatch?serviceID=<serviceID>	<serviceID>: UUID	Deletes an existing service watch for a particular service
Retrieve Service Watch Data	HTTP GET	http://localhost:8080/cxf/getServiceWatchData?<serviceID>&<start>&<end>	<serviceID>: UUID <start>: long: a date represented as milliseconds <end>: long: a date represented as milliseconds	Retrieves a list of Service Watch data for a particular service in a particular period defined by start and stop.

Examples of how to create a Service Watch via command line using the tool CURL can be seen in Listing 1; Figure 10 shows a screenshot of how the result looks like. Listing 2 shows the CURL command of how to retrieve Service Watch data via the command line and Figure 11 shows the output of it. Further, Listing 3 shows the command for deleting a Service Watch and Figure 12 shows the output of that command.

Listing 1: Create a Service Watch via CURL – Input Example

```
curl -i -H "Accept: application/json" -X PUT
"http://localhost:8080/cxf/serviceRegistry/createServiceWatch?serviceID=dc8715dd-
86bb-42d2-85c9-2735739a6748&lifetime=100000&interval=10000&retentionPeriod=60000"
```

Listing 2: Get Service Watch Data via CURL – Input Example

```
curl -i -H "Accept: application/json" -X GET
"http://localhost:8080/cxf/serviceRegistry/getServiceWatchData?serviceID=dc8715dd-
86bb-42d2-85c9-2735739a6748&start=1410176545076&end=1410179545076"
```

Listing 3: Delete a Service Watch via CURL – Input Example

```
curl -i -H "Accept: application/json" -X DELETE
"http://localhost:8080/cxf/serviceRegistry/deleteServiceWatch?serviceID=dc8715dd-
86bb-42d2-85c9-2735739a6748"
```

```
laptop ~$curl -i -H "Accept: application/json" -X PUT http://localhost:8080/cxf/serviceRegistry/createService
Watch?"serviceID=2b775583-bb11-4bd6-a09a-0c1a5e89d31c&lifetime=100000&interval=10000&retentionPeriod=10000"
HTTP/1.1 200 OK
Content-Type: application/json
Date: Fri, 05 Sep 2014 09:23:06 GMT
Transfer-Encoding: chunked
Server: Jetty(8.1.14.v20131031)

{"statusCode":{"code":201}}
```

Figure 10: Create a Service Watch via CURL – Output Example

```
laptop ~$curl -H "Accept: application/json" -X GET http://localhost:8080/cxf/serviceRegistry/getServiceWatchD
ata?serviceID=2b775583-bb11-4bd6-a09a-0c1a5e89d31c
{"serviceWatchData":[{"serviceID":"2b775583-bb11-4bd6-a09a-0c1a5e89d31c","timestamp":"2014-09-05T11:28:57.235+02:00","re
sponsetime":856,"notResponding":false,"retentionPeriod":10000}]}
```

Figure 11: Get Service Watch Data via CURL – Output Example

```
laptop ~$curl -H "Accept: application/json" -X DELETE http://localhost:8080/cxf/serviceRegistry/deleteService
Watch?serviceID=2b775583-bb11-4bd6-a09a-0c1a5e89d31c
{"statusCode":{"code":200}}
```

Figure 12: Delete a Service Watch via CURL – Output Example

http://www.simpli-city.eu/	Copyright © SIMPLI-CITY Project Consortium. All Rights Reserved. Grant Agreement No.: 318201

6 Limitations and Further Developments

Although both components, the Service Watch and the Service Registry provide Java interfaces next to RESTful interfaces, it is not foreseen that unauthorized users make use of them. For that reason, the REST Proxy, developed in the Service Runtime Environment, provides basic authentication mechanisms. The REST Proxy is the only one which can directly communicate with the Service Registry, all other components are supposed to make use of the REST Proxy.

As mentioned before, the Service Registry as well as the Service Watch are not standalone components. Furthermore, both components are already pre-deployed in the Service Runtime Environment and do not have to be started separately. Because of this, it is required to attach a prototype of the Service Runtime Environment to this deliverable. However, it remains to mention, that this Service Runtime Environment is not a final prototype and its functionalities are partly implemented and therefore it is not foreseen for any productive use. The final prototype of the Service Runtime Environment will be provided in deliverable D5.3.3.

http://www.simpli-city.eu/	Copyright © SIMPLI-CITY Project Consortium. All Rights Reserved. Grant Agreement No.: 318201

7 Summary

The deliverable at hand presents and describes the final prototype of the Service Registry and the Service Watch. Within this final prototype, the promised functionalities were implemented.

It is possible to create and deploy new backend services, update them by providing either new binaries, a new service description, or both. Furthermore, it is possible to query for a service by using XPath queries and delete and undeploy them from the Service Runtime Environment. All of these methods provide a basic error handling and fault tolerance, i.e., the caller will be informed if an invalid input was provided or if an unexpected error occurred on the server side.

Beside of this functionality, a Service Watch can be created which monitors data services, i.e., it “contacts” them in a defined interval in order to check its state. The monitoring data can also be retrieved.

Apart from the prototype description, an analysis of the degree of fulfillment of the requirements that need to be covered by this component (as specified in the Requirements Analysis Report – D2.3) is presented.

In addition, all required steps to install, deploy, and execute the different components have been described. Finally, the limitations of the current prototype including their dependability on the Service Runtime Environment have been specified.

http://www.simpli-city.eu/	Copyright © SIMPLI-CITY Project Consortium. All Rights Reserved. Grant Agreement No.: 318201