simpli·city

The Road User Information System Of The Future

WP5 – Mobility Services Framework

D5.2.2: Context-based Service Personalization Prototype II

Deliverable Lead: TUV

Contributing Partners: TIE

Delivery Date: 03/2015

Dissemination Level: Public

Version 1.0.0

This deliverable describes the work carried out during the development of the Context-based Service Personalization prototype of the SIMPLI-CITY Mobility Services Framework. It specifies how to install and execute the different implemented subcomponents.

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	Daniel Burgstahler, TUDA
Internal Reviewer 2	Kristof Kipp, ASC
Type	Deliverable
Work Package	WP5: Mobility Services Framework
ID	D5.2.2: Context-based Service Personalization Prototype II
Due Date	31.03.2015
Delivery Date	30.03.2015
Status	For Approval

Document History	
Contributions	V0.0.1, Philipp Hoenisch, TUV, 29.01.2015, Added document structure. V0.0.2, Philipp Hoenisch, TUV, 11.02.2015, Added TODOs for the partners V0.0.3, Philipp Hoenisch, TUV, 19.02.2015, Extended usage description V0.0.4, Philipp Hoenisch, TUV, 20.02.2015, Interface description added and limitations extended V0.0.5, Xavier Cases Camats, WORLD, 25.02.2015, Added information needed about Context Manager, Context Sensor and Context Database. V0.0.6, Philipp Hoenisch, TUV, 26.02.2015, provided feedback to Worldline V0.0.7, Xavier Cases Camats, WORLD, 26.02.2015, Reviewing and fixing some sections V0.0.8, Vadim Petrenko, TIE, 03.03.2015, Merging contribution into main document V0.0.9, Rostyslav Zabolotnyi, TUV, Added information on Proactive User Notification (5.1.3) V0.1.0, Philipp Hoenisch, TUV, 08.03.2015 Updates, review and clean up V0.1.1 Xavier Cases Camats, WORLD, 11.03.2015 Adding some figures to exemplify. V0.1.2, Philipp Hoenisch, TUV, 12.03.2015 Updates, review and clean up V0.1.4, Xavier Cases Camats, WORLD, 18.03.2015 Rephrase and structure V0.1.5, Philipp Hoenisch, TUV, 23.03.2015 Updates after review V1.0.0, Philipp Hoenisch, TUV, 28.03.2015 Finishing up for approval
Final Version	March 30th, 2015

Note

This deliverable is subject to final acceptance by the European Commission.

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 – Reti e Mobilità, Italy

Executive Summary

This deliverable describes the work carried out during the development of the final prototype of the Context-based Service Personalization component of the SIMPLI-CITY Mobility Services Framework.

The document starts by introducing the Context-based Service Personalization component and describing the scope of this final prototype.

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 provides Java interfaces for the Context Manager, Context-based Services and for the Publish-Subscribe Middleware. The latter one also provides a RESTful interface, thus this component can be used from outside of the Service Runtime Environment (SRE), e.g., by Apps running on the Application Runtime Environment (ARE) or third-party services.

In order to allow service developers to create personalized (backend) services, Java interfaces are provided and a description is given in this document. Further, it is possible to subscribe to context sensors via the Publish-Subscribe Middleware. For that, a client application plus three example sensors are provided (an alarm sensor, a temperature sensor and a weather sensor).

Furthermore, this document describes how potential users (i.e. service developers and administrators) can prepare, install and execute the different parts of the Context-based Service Personalization component. For this, a step-by-step process to install and make use of the prototype is provided.

This deliverable D5.2.2 is the final prototype of the Context-based Service Personalization.

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 final prototype of the SIMPLI-CITY Context-based Service Personalization component 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 Context-based Service Personalization component and exploit its functionalities. For this, the scope and requirements of the final Context-based Service Personalization prototype, the requirements and preparations for users and developers, and an installation and usage guide are provided.

The final Context-based Service Personalization prototype is the outcome of the discussions and implementation work done in project months 18 to 30. It provides the final implementation of the functionalities of the Context-based Service Personalization component as provided in the SIMPLI-CITY deliverables D3.2.1 (Functional Specification), and D3.2.2 (Technical Specification).

This deliverable is the final prototype of the Context-based Service Personalization component.

1.3 Document Status and Target Audience

This document is listed as “Public” in the Description of Work (DoW). 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 is primarily 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 how the Context-based Service Personalization component fits into the overall SIMPLI-CITY software 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 Context-based Service Personalization component.

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

Section 5 describes how the provided functionalities can be used.

Section 6 provides a summary of the document.

2 Prototype Scope and Requirements Coverage

The following subsections describe the scope of this final prototype of the Context-based Service Personalization and how the requirements have been covered.

2.1 Context-based Service Personalization – General Information

The Context-based Service Personalization component offers base functionalities for personalizing a service's output based on the context of a particular user. For this, the Context-based Service Personalization component provides four different functionalities.

- The location-based data service selection, which allows the adaptation of a service outcome based on the location of a user, i.e., the outcome of the service will be location-aware.
- Proactive user notification, which provides the means of notifying a user in advance about a certain piece of information based on the current situation, e.g., indicating Points of Interest (POIs) in the near environment.
- Support of prefetching relevant context, which allows controlling the data prefetching functionality, i.e., it tells apps or services when to start prefetching data.
- Context-based service execution, which enables the automated usage of user context data in service executions.

Figure 1 shows the location of the Context-based Service Personalization component in the SIMPLI-CITY Global Architecture. For the full Global Architecture, please refer to deliverable D3.1.

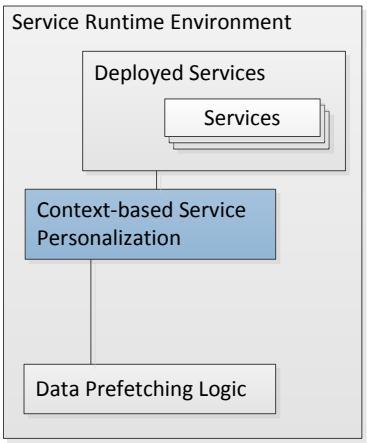

Figure 1: Location of the Context-based Service Personalization in the SIMPLI-CITY Global Architecture

2.2 Scope of this Prototype

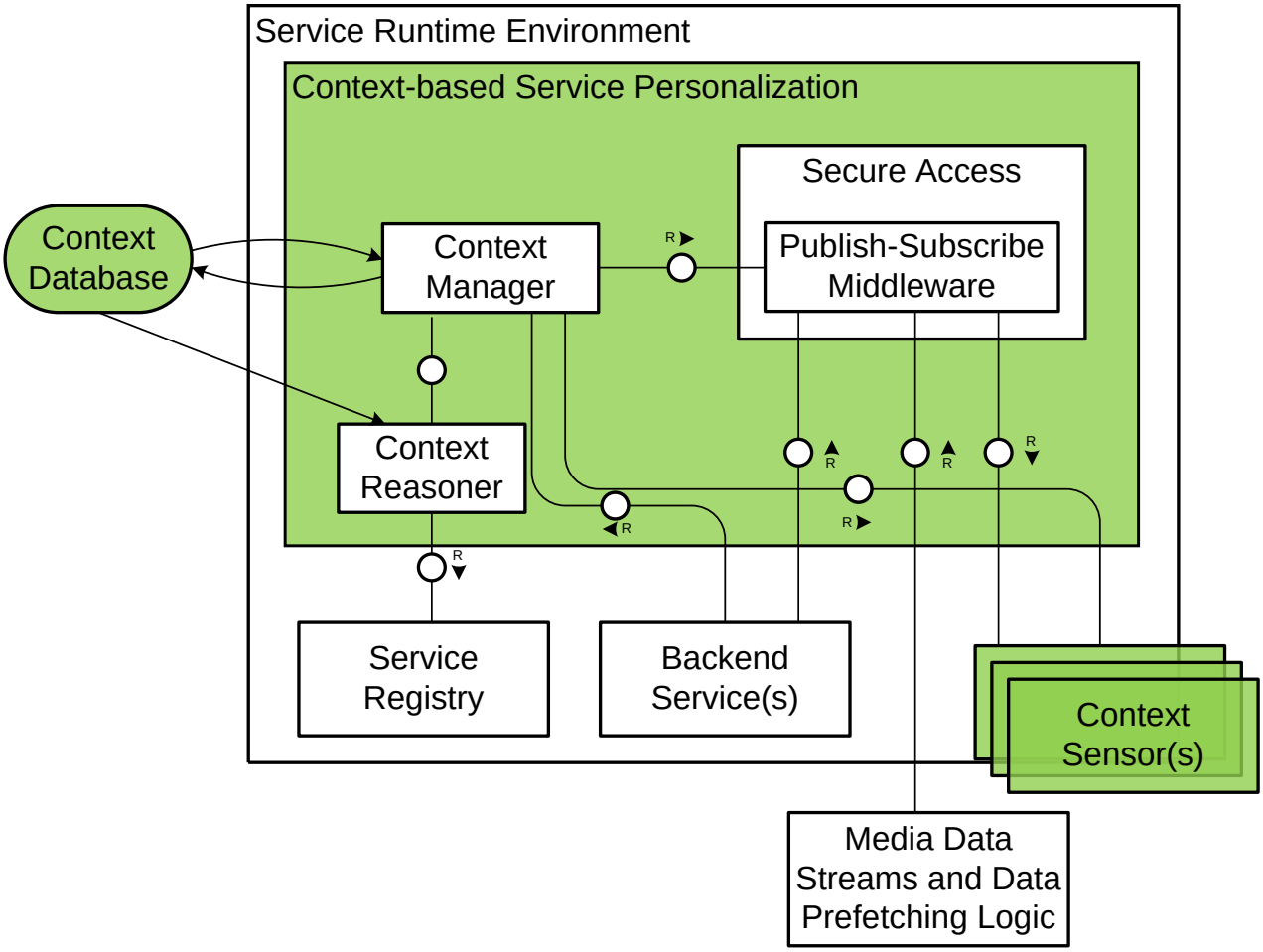

Figure 2: Scope of the Final Prototype of the Context-based Service Personalization Component

Figure 2 depicts the status of development of the Context-based Service Personalization component, showing the subcomponents that are covered. Note, that this figure shows the Context-based Service Personalization component within the Service Runtime Environment; therefore, also other parts of the SIMPLI-CITY Global Architecture, which are not developed within this component, are depicted, e.g., the Service Registry, Backend Service(s), the Service Runtime Environment and the Media Data Streams and Data Prefetching Logic component. Beside of that, for the sake of convenience and readability, the subcomponents Context Manager, Context Reasoner, and the Publish-Subscribe Middleware are not coloured, however, they are also fully implemented.

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

- Green: Fully implemented.
- Orange: Partially implemented.
- White: No implementation, or not part of this deliverable.

In the following subsections, the scope and status of the single subcomponents (as depicted in Figure 2) will be discussed in more detail. 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 Context Reasoner

The Context Reasoner subcomponent provides the core functionality for the four different context-based service personalization approaches. Within this deliverable, two context-based services are provided, i.e., a Location-based Service Selection service and a User Preference service. In addition, a backend service for notifying users proactively is provided. These subcomponents provide Java interfaces for other services and can be used by software developers to integrate the functionalities in their own services to provide user-personalized services. The Context Reasoner subcomponent is automatically deployed together with the other components that are part of this deliverable.

2.2.2 Context Manager

The Context Manager is a central subcomponent of the Context-based Service Personalization component and is responsible for communicating internally with the Context Reasoner and the Context Sensors via the Publish-Subscribe Middleware. This component enables service developers to retrieve a subscription endpoint for a known context sensor for retrieving up to date context data, or to retrieve historical data from the database. For this purpose, the Context Manager provides a Java and corresponding RESTful interfaces.

2.2.3 Context Sensors

Context Sensors are wrapping services that are connected to one particular data source. These data source could be a physical sensor such as a proximity sensor or GPS sensor, or external services, like Open Data services. Beside of this, a Context Sensor can also be connected to the Data Processing component (D4.4.1) or the Cloud-based Information Infrastructure (D4.2) where user-specific data is stored. Within this prototype, three example sensors are provided: The alarm sensor, the temperature sensor and the weather sensor.

2.2.4 Context Database

This component stores the data sent from the Context Sensors, so it is available for future requests from other services (e.g., the Context Reasoner). The Context Database is directly connected to the Publish-Subscribe Middleware component through the Context Manager. Hence, the latter stores retrieved context data in this database. For retrieving a certain set of context data, e.g., in between two dates, Java interfaces are provided.

2.2.5 Publish-Subscribe Middleware

This component is in charge of managing subscriptions and acts as an intermediary component between Context Sensors, the Context Database and the Context Manager. It allows publishing data and allows subscribing to the data by providing a context filter (e.g., “temperature lower than 8 degrees”).

The final prototype implements publishing methods in a polling (the internal publishing component requires periodically updated data) or in a pushing way (the sensor sends his

data by harnessing a provided web service interface). The Polling Publisher component provides two methods to register new publishers: a web service interface to register new polling publishers and a properties configuration file to suggest the registered publishers. All collected data is stored in the Context Database component. The final prototype provides a web service interface to register new subscriptions.

The Publish-Subscribe Middleware component works within an Apache Camel stand-alone installation or an Apache Camel instance embedded in an Enterprise Service Bus (ESB) server (JBoss, Apache Karaf …) installation. Apache Camel has a dynamic routing functionality to choose the transport service to receive and send data. This prototype uses TIE's commercial product TSB (TIE SmartBridge) as the transport service. TSB allows topic or queue based subscriptions and has been adapted to allow entity criteria subscriptions (e.g., "all data from sensor X with value lower than Y").

2.2.6 Secure Access

This component is in charge of providing secure access to the data. It wraps the Publish-Subscribe Middleware and is called automatically when data is published and the subscription filter criterion is processed. For this prototype, a security layer is provided which is based on roles, i.e., users need to have a certain role in order to be allowed to subscribe to context sensors. For each piece of data and for each subscription the Publish-Subscribe Middleware verifies if the subscriber has the necessary privileges to access that sensor data by checking the context type of the data (alarm, GPS location, …) and the roles assigned to the subscriber. If the user does not have enough privileges to access the data, the published sensor data will not result in sending any notifications to that subscription.

2.3 Covered Requirements

This section describes the degree of fulfilment of the requirements to be covered by the Context-based Service Personalization component and as specified by the Requirements Analysis Deliverable (D2.3) and the Functional Specification (D3.2.1).

Table 1: Requirements Related to Context-based Service Personalization and their Degree of Fulfilment

Requirement	Degree of Fulfilment	Comment
Must Have Requirements		
U27: Proactive behaviour U195: Proactive user notifications	100%	This functionality is fully implemented. Interfaces are provided which enable service developers to notify users proactively.
U59: User centric data services	100%	Within the final prototype, a location-based service and user-preference service is provided. These are described below.

Requirement	Degree of Fulfilment	Comment
U90: Availability U91: Integrity U92: Secure access to system U93: Third party access to the system U103: Fault tolerance U104: Stability	100%	The availability and integrity is covered within the current prototype, the secure access, especially for third parties is also part of this prototype. By abiding coding guidelines and writing extensive tests, a basic fault tolerance and stability is achieved.
U106: Access to cloud services	100%	This functionality has been fully covered. Context Data is stored in the Cloud-based Information Infrastructure and various data, such as user preferences are directly read from this storage.
U50: Prefetching of media data & Offline access	100%	This functionality was covered by the final prototype in T4.5.
U201: Provision of personalized info, e.g. travel, costs	100%	This functionality is provided as part of a backend service. It takes into account user centric data for transforming units like distance or currencies.
U209: Provision of real-time information about the current route	100%	This functionality is implemented based on a request base, i.e., users can request information during their trip by sending a location to the service which will return points of interested related to the location. Further, it is possible to select services according to the user's location, e.g., find a suitable parking lot service in the current city.
Should Have Requirements		
U125: Open interfaces U126: Openness of the system U127: Extensibility	100%	These requirements are covered by well-defined interfaces and well documented code.
U196: Prioritization of notifications	0%	This functionality has not been covered within T5.2 as prioritization of notifications should be done on a user's Personal Mobility Assistant (PMA).

Requirement	Degree of Fulfilment	Comment
Could Have Requirements		
U124: Scalability of the service platform	50%	This functionality as such was not covered within the final prototype, however the overall Service Runtime Environment has been designed in a way that allows provisioning of scalability features in the future. REST Proxy being the single entry point for all method calls can be retrofitted to serve as a load balancer and the built-in monitoring and SLA watching modules can spawn new Service Runtime Environment instances should load increase and response times degrade.

3 Preparations

This section provides information about what potential users (both administrators and software developers) need to prepare in order to use the functionalities of the delivered prototype.

The main part of the Context-based Service Personalization component will be executed by administrators of the SIMPLI-CITY Mobility Services Framework, whereas the provided service's interface can be used by service developers.

3.1 Server Side (System Administrators)

In order to make use of the final prototype of 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. In addition, it is required that the environment variable JAVA_HOME is defined as per Oracle guidelines.[1]

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

The final prototype was tested on a Linux and Windows:

- The Linux machine was running Ubuntu 12.04, Java(TM) SE Runtime Environment (build 1.7.0_51-b13) and Apache Maven 3.0.5.
- The Windows machine was running Windows 8.1, Java(TM) SE Runtime Environment (build 1.7.0_55-b13) and Apache Maven 3.0.5.

An installation script is provided for Windows and Ubuntu.

1. Unpack the provided archive file and change into the resulting folder, where the following directory structure should appear:
    - services/
    - eu.simpli-city.sre.assembly-1.0.0-SNAPSHOT.tar.gz
    - eu.simpli-city.sre.assembly-1.0.0-SNAPSHOT.zip
    - start.cmd
    - start.sh
2. On Windows, double-click "start.cmd" and wait for the Service Runtime Environment to start. On other operating systems, follow these instructions.
3. If not already done, start a terminal and navigate into the resulting folder.
4. If not already happened: you can make the file "start.sh" executable by typing
    a. chmod a+x start.sh
5. Type into the terminal: "./start.sh". This will start the Service Runtime Environment and automatically deploy some example services. If the output is similar to Figure 3, the Service Runtime Environment should be up and running.

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

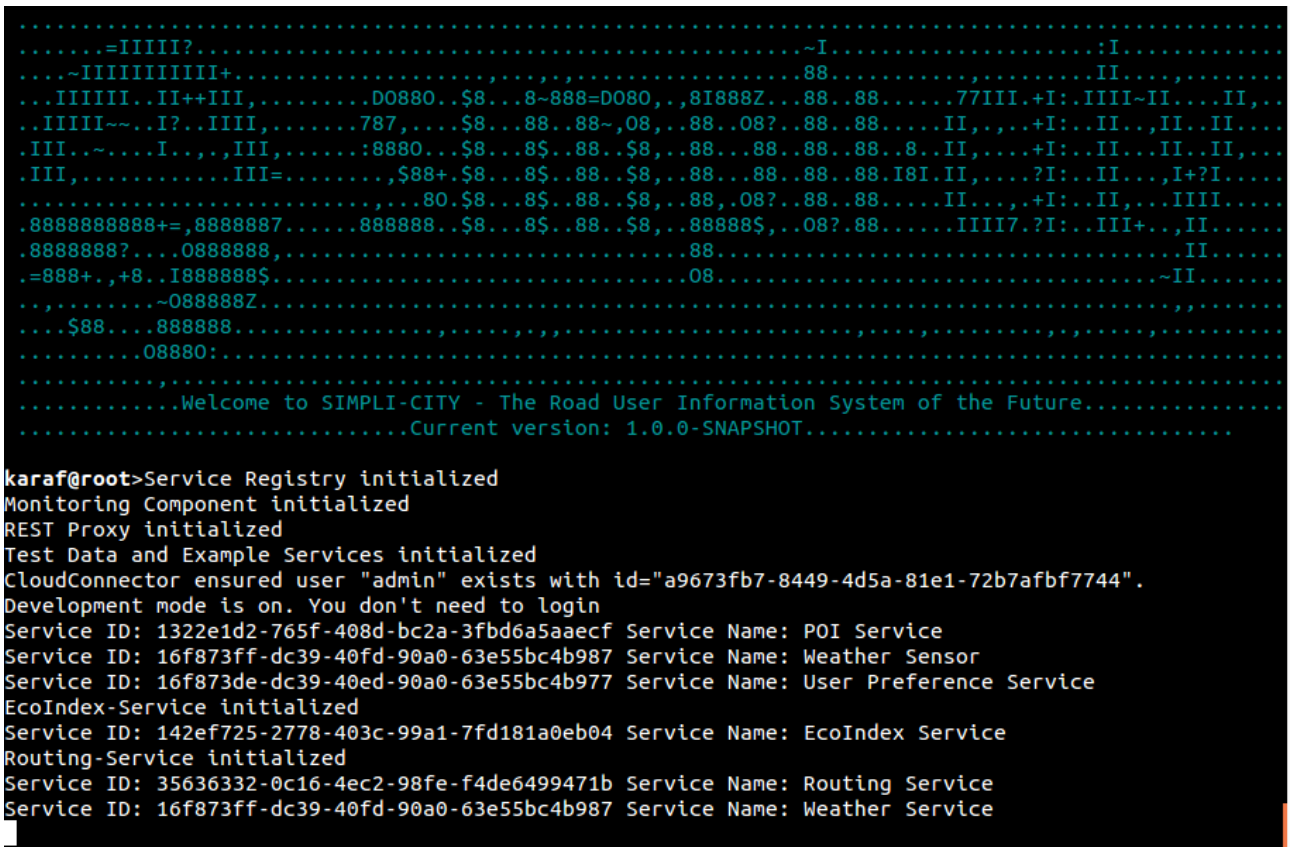

Figure 3: Service Runtime Environment Start-Up Output

If no error was printed, the Service Runtime Environment has been started successfully. In addition, the log shows that the Service Registry was successfully started and 4 example services are pre-deployed:

- A POI Service
- A User Preference Service
- An EcoIndex Service
- A Routing Service
- And a Weather Sensor

3.2 Publish-Subscribe Middleware

As mentioned before, the Publish-Subscribe Middleware can work in a stand-alone manner or be embedded in an ESB server installation (JBoss, Apache Karaf …). For this prototype, the embedded installation inside the Context-based Service Personalization component is used, so a proper Oracle's Java SE Development Kit 7 installation is necessary. The Publish-Subscribe Middleware uses the security access component which is automatically deployed and set up together with the other components of this deliverable, hence, no further actions are required to set up these components.

The prototype uses TSB (TIE Smart Bridge, a commercial ESB implementation from partner TIE) as a transport service to route messages between the Context Sensors, backend services and the Context Manager, for this reason access to a server with a TSB installation is required. For this prototype, a TSB server hosted by partner TIE is available

at the IP address: 95.211.177.222. The server is fully managed by TIE and does not require any configuring or administering by SIMPLI-CITY system administrators or partners.

On the client side, if a backend service wants to subscribe to messages that match a provided filter criteria, it has to provide a network endpoint accessible remotely.

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 final prototype of the Context-based Service Personalization component within the Service Runtime Environment.

4.1 Server Side (System Administrators)

The following describes what set up steps are required for system administrators.

4.1.1 Installation of Context-based Service Personalization Component

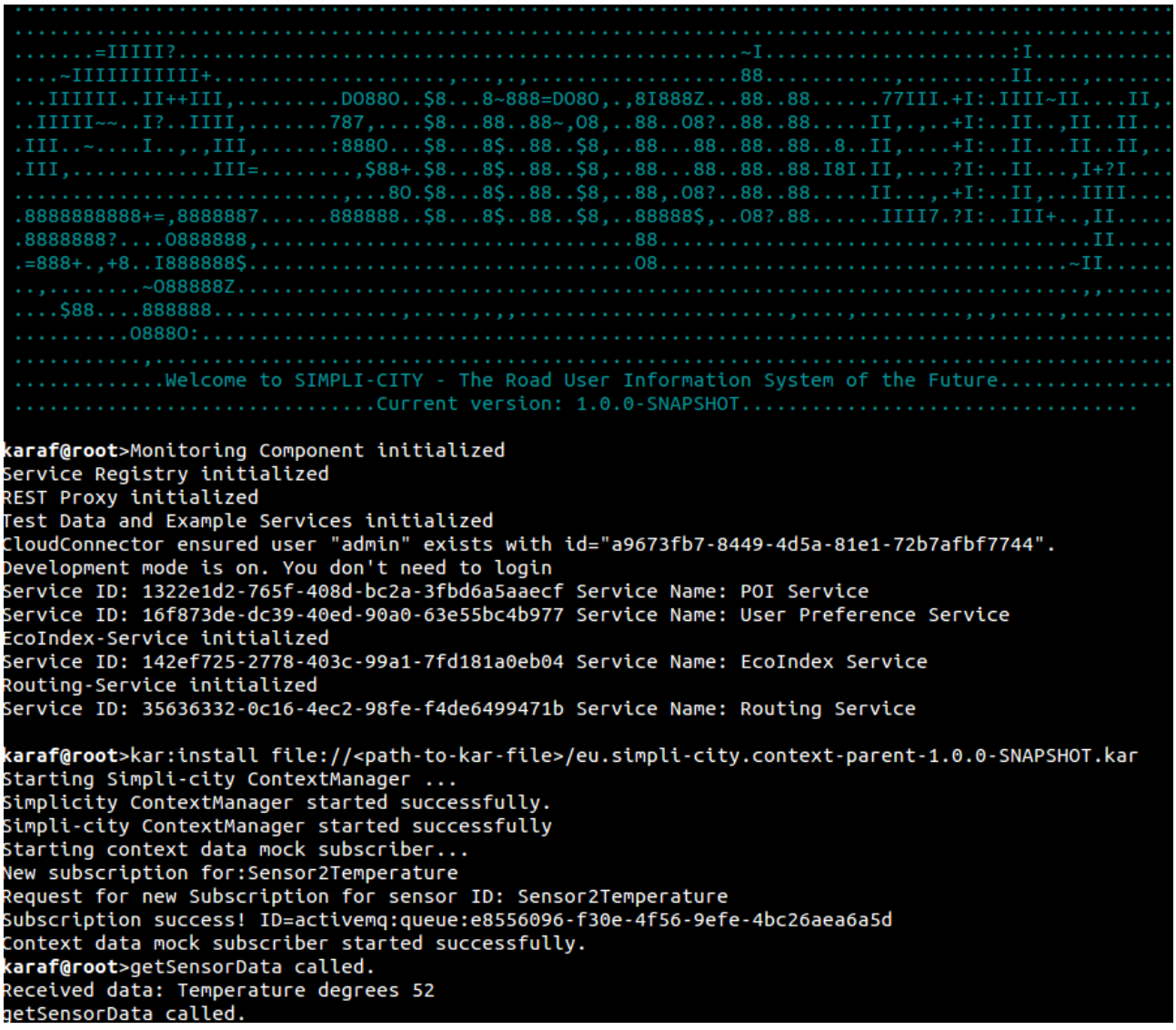

Figure 4: Example Console Output of Deploying the Context-based Service Personalization Component

In order to make use of the Context-based Service Personalization Component, the following steps have to be performed:

1. The Service Runtime Environment has to be started in a terminal as described in Section 3.1.
2. As soon as the Service Runtime Environment is up and running, the command from Listing 1 needs to be typed into the terminal:
    - Note: <path-to-context-parent.kar> is only a placeholder and has to be replaced with the absolute path to the eu.simpli-city.context-parent-1.0.0-SNAPSHOT.kar – file
3. Subsequently, the services are deployed. If no error occurred, the output should be similar to Figure 4.

Listing 1: Command to Install Context-based Service Personalization

```
kar:install file://<path-to-context-parent.kar>
```

4.1.2 Publish-Subscribe Middleware

Since the Publish-Subscribe Middleware is included and packaged in the Context-based Service Personalization component, no more actions are necessary to run it, once the Context-based Service Personalization is deployed.

4.1.3 Context Manager and Context Sensor Setup

The Context Manager is part of the full prototype and will be installed automatically with all the other components.

5 Execution and Usage of the Software

This section describes how to use the different subcomponents of the prototype.

5.1 Context-based Services

In the following, three services are described, which make use of context data in order to personalize their output. First, a backend service is described which is able to select other services by a given location and range. Second, a preference service is described, which makes use of user-centric data, such as preferences in order to personalize its output. Third, a service is explained which enables proactive user notifications.

5.1.1 How to Test the Location-based Service Selection

In order to discover services for a particular location, a command line test client has been created. Table 2 shows a short description of this test client and four example inputs. A possible output can be found in Figure 5.

Table 2: Location-based Service Selection Test Client

Command	Parameters	Description
test:location	<latitude> <longitude> <radius>	This command has to be executed within the terminal of the Service Runtime Environment. Note: <latitude>, <longitude> and <radius> are placeholders and have to be replaced. Valid examples can be found below.
test:location	53.34419 -6.26751 40.0	For finding a service for Dublin
test:location	41.389 2.159 40	For finding a service for Barcelona
test:location	44.494 11.339 100	For finding a service for Bologna
test:location	48.208 16.372 100	For finding a service for Vienna

```
karaf@root>test:location 53.34419 -6.26751 40.0
Looking for a service with Longitude -6.26751, Latitude 53.34419 and Radius: 40.0
Found cities [Dublin]
Found services for cities:
Nearest city: Dublin found services 1
karaf@root>
```

Figure 5: Example Service Selection Input and Output

5.1.2 How to Use the User Preference Service

In many cases, PMA users may have different preferences and settings on their phone (PMA). For example, some user may prefer the metric system to the imperial system for measuring distances, or want to have prices directly converted into their favorite currency. For that purpose, a user preference service is provided in this deliverable, which considers user-centric data for transforming currencies and distances. The corresponding Java interfaces are described below: Listing 2 shows the usage of a method to transform from currency to another. This method accepts a userID as parameter, which references a unique user. For this user, the preferences are loaded from the Cloud-based Information Infrastructre and the given value is converted. Listing 3 shows how to transform a certain distance from one system to another. Again, a userID is required to load the user's preferences from the database and convert a given value. These services can be directly used by any backend service. For the usage within an App running on the Application Runtime Environment, please refer to the REST Proxy of the Service Runtime Environment in D5.3.3. This service does not require any specific license or user rights.

Listing 2: Convert Currency

```
/**
* This method converts a provided value from one currency to another. The target
* currency is defined by the user's preferences. If the user has no preferences
* defined, the default unit is taken, i.e., EUR
*
* @param userID: defines a unique user ID
* @param value: defines the value to be converted
* @param sourceUnit: defines original currency type, e.g., EUR, USD, GBD, …
* @return the converted currency value, e.g., 0.84
* @throws Exception is thrown if an unexpected error occurred or an invalid currency
was given
*/
public double convertCurrency(UUID userID, double value, String sourceUnit)

//get the currency rate from USD to Euro
double converted = api.convertCurrency("1322e1d2-765f-408d-bc2a-3fbd6a5aaecf", 120,
                                        "USD");
```

Listing 3: Convert Distances

```
/**
* This method converts distances between the IMPERIAL and the METRIC system (or vice
* versa). The target unit type is defined by the user's preferences.

* @param userID: defines a unique user ID for which preferences are loaded from the
* DB
* @param value: defines the value to be converted
* @param sourceUnitType: defines original distance type, e.g., IMPERIAL or METRIC
* @return the converted value, e.g., 0.621371
* @throws Exception if an unexpected error occurred
*/
public double convertDistance(UUID userID, double value,
                              DistanceUnitType sourceUnitType)

//get the currency rate from EURO to USD
double value = api.convertDistance("1322e1d2-765f-408d-bc2a-3fbd6a5aaecf", 1000
                                   UnitType.IMPERIAL);
```

5.1.3 How to Proactively Notify a User

In some cases, it is necessary to send a notification to the user's PMA even if there is no application to expect such notifications. Usually this happens due to some long-running or periodic tasks in the Service Runtime Environment or due to some external event. For example, when the calendar service running in Service Runtime Environment detects that a meeting was rescheduled, the user needs to be notified even if the calendar app is currently not running on the user's PMA. Similarly, if an unforeseen street work may influence upcoming travelling plans, the user has to be informed as soon as possible. To provide this functionality, a user notification service is provided which is able to push messages to a specified user's PMA. Listing 4 illustrates the usage of this service to notify the user that a calendar event has been rescheduling. The presented method accepts a receiver ID and a message that has to be delivered to the defined user. When the service fails to send a message, an exception is thrown.

Listing 4: Proactive User Notification

```
/**
* This method sends proactive user notifications to the defined recipient.
*
* @param recipientID: defines a unique recipient of the message.
* @param message: defines the message that needs to be sent to the recipient.
* @throws IOException is thrown if message sending failed due to networking error.
*/
public void notify(String recipientID, String message) throws IOException;

//send the message on meeting rescheduling.
api.notify("1322e1d2-765f-408d-bc2a-3fbd6a5aaecf", "The meeting with Stefanie was
rescheduled from 13:00 to 13:30.");
```

5.2 How to Add a New Sensor

To add a new ContextSensor to the list of current publishers, it is necessary to specify its configuration in a file named "*config.properties*" that is located in the project eu.simpli-city.ctx-personalize.pub-sub.ContextManager resources folder.

Listing 5: Example of Publish-Subscribe Middleware Configuration

```
PubSubMode = standalone
PublishingRouteUri = standalone://Environment1
NotificationRouteUri = standalone://Environment1
SubscriptionServerUri = netty-http:http://localhost:5668/Subscriptions
PublishersUri = netty-http:http://localhost:8283/alarmSensorService,netty-
http:http://localhost:8285/temperatureSensorService
PublishersName = Sensor1Alarm,Sensor2Temperature
PublishersAskPeriod = 5000
```

The PublishersUri property needs to contain the ContextSensor address being added and its name has to be specified in the PublishersName property. These are comma-separated properties.

The current implementation of the Publish-Subscribe Middleware requires that these URIs do not contain any dynamic parameters and if requested via HTTP would return a JSON serialized ContextData instance. For that purpose, an additional sensor is provided, i.e., a weather sensor which accepts a specific location as parameter (see Section 5.2.1). This deliverable bundles two different examples of the ContextSensor implementations as described in the following sections.

5.2.1 Weather Sensor

In this section, a short description of the Weather Sensor is provided. The Weather Sensor, in this case, is a specific backend service wrapping sensor data. It accepts a location in form of longitude and latitude and a unit type as the parameter. A detailed description is given in Listing 6 and Listing 7. Using this example, every data source can be wrapped and provided to other backend services.

Listing 6: Weather Sensor Interface Description: Full Weather Information

```
/**
* This method returns weather information for a specific location.
*
* @param longitude: The longitude of the location
* @param latitude: The latitude of the location.
* @param unit: The units (METRIC or IMPERIAL) that should be used.
* @return The object that defines the weather condition at the specified location.
* @throws IOException is thrown if message sending failed due to networking error.
*/
public WeatherInfo getWeatherInfo(double longitude, double latitude,
                                  DistanceUnitType unit) throws IOException;


//get weather info for a specific location.
WeatherInfo weatherInfo = api.getWeatherInfo(16.3667, 48.2,
                                             DistanceUnitType.METRIC);
```

Listing 7: Weather Sensor Interface Description: Temperature Only

```
/**
* This method returns temperature information for a specific location.
*
* @param longitude: The longitude of the location
* @param latitude: The latitude of the location.
* @param unit: The units (METRIC or IMPERIAL) that should be used.
* @return The value that defines the current temperature at the specified location.
* @throws IOException is thrown if message sending failed due to networking error.
*/
public double getTemperature(double longitude, double latitude,
                             DistanceUnitType unit) throws IOException;


//get temperatur info for a specific location.
Double temperature = api.getTemperature(16.3667, 48.2,
                                        DistanceUnitType.METRIC);
```

5.3 How to Run the Publish-Subscribe Middleware

The Publish-Subscribe Middleware is included in the Context-based Service Personalization component distribution, so that when the latter is installed, the Publish-Subscribe Middleware is automatically installed, too.

Once the service is installed and the sensors and the subscriber start sending information, the user can use the following commands to manage this process:

Table 3: Commands to Manage Sensors and Subscribers

Command	Parameters	Description
Stop	exampleSensor1 exampleSensor2 exampleSubscriber	This command has to be executed within the terminal of the Service Runtime Environment. It stops the requested sensor or subscriber.
Start	exampleSensor1 exampleSensor2 exampleSubscriber	This command has to be executed within the terminal of the Service Runtime Environment. It starts the requested sensor or subscriber.

5.4 How to Subscribe to a Sensor

A complete example of a subscriber is provided in the eu.simpli-city.ctx-personalize.pub-sub.mockSubscriber project module. The example in Listing 8 obtains implementations of the ContextManagerService service via the ServiceTracker in the Activator of the bundle.

Listing 8: Subscriber Example

```
 try {

   // Create a tracker and track the service
   serviceTracker = new ServiceTracker(context,ContextManagerService.class.getName(),
null);
   serviceTracker.open();
   // Have a service listener to implement the whiteboard pattern
   fContext.addServiceListener(this, "(objectclass=" +
ContextManagerService.class.getName() + ")");
   // Grab the service
   contextManagerService = (ContextManagerService) serviceTracker.getService();

   if(contextManagerService == null)
        System.out.println("ContextManager is not registered!");
   else {
        createSubscription();
        System.out.println("Context data example subscriber started successfully.");
   }
} catch(Exception ex) {
    ex.printStackTrace();
}
```

The Service Tracker allows the new service to track the life cycle of the Publish-Subscribe Middleware component, i.e., when a new Context Manager implementation starts or stops.

The example in Listing 9 shows how a new subscription can be obtained via the Context Manager Service instance.

Listing 9: Subscription Example

```
private void createSubscription() throws Exception {
  if(contextManagerService == null) return;
  int upperthreshold = 36;
  int lowerthreshold = 26;
  // Define a new ContextFilter
  ContextFilter filter = new ContextFilter(upperthreshold, lowerthreshold, interval,
start, end);
  // New credentials of the subscriber
  Credentials credentials = new Credentials("user1","my secret password");
  // Get a new subscription from the ContextManagerService to a sensor, providing a
ContextFilter and the required credentials
  subscription = contextManagerService.getSubscriptionEndpoint("Sensor2Temperature",
filter, this, credentials);
  if(subscription == null) {
       System.out.println("Subscription is null!");
  } else {
       System.out.println("Subscription success! ID=" + subscription.getURI());
  }
```

5.5 How to Request Data from the Context Manager

Many data sources can be added to the platform and because of that a Context Manager has been implemented. The Context Manager is able to subscribe to sensors and sources, close the subscriptions and to retrieve information about these different sources. The according Java interfaces are described below:

Listing 10 shows the usage of a method to subscribe to a sensor. This method accepts a datasourceID as a first parameter, which defines a unique sensor. The second parameter is the contextFilter that configures the subscription for this sensor.

Listing 10: Get Subscription Endpoint

```
/**
 * @param datasourceID, unique ID for the Data Source for which the requester will
subscribe
 * @param contextFilter, is an optional parameter, which defines a filter which
      should be applied on the context data; examples are the
      frequency of wished updates or a threshold which has to be
      exceeded before new data is pushed.
 * @return on success a Subscription object holding the URI and its type
 * @throws DataSourceNotFoundException in case the datasourceID is invalid
 */
Subscription getSubscriptionEndpoint(String datasourceID, ContextFilter
contextFilter) throws DataSourceNotFoundException;

int upperthreshold = 36;
int lowerthreshold = 26;
int interval = 0;
long start = 0;
long end = 1000;
ContextFilter filter = new ContextFilter(upperthreshold, lowerthreshold, interval,
start, end);

//get the endpoint of the subscription to the Sensor2Temperature sensor.
subscription = contextManagerService.getSubscriptionEndpoint("Sensor2Temperature",
filter);
```

Listing 11 shows how to close a subscription. The according functions accepts a subscriptionUri as parameter.

Listing 11: Close a Subscription Endpoint

```
/**
 * Close the subscription
 * @param subscriptionUri to close.
 * @return true if success
 */
boolean closeSubscriptionEndpoint(String subscriptionUri);

//If there is a Subscription with an endpoint, the endpoint can be closed.
contextManagerService.closeSubscriptionEndpoint(subscription.getURI());
```

Listing 12 shows how to retrieve information with the UUID of a data source.

Listing 12: Get Context Data

```
/**
 * @param datasourceID, defines the Data Source for which the context data shall be
returned
 * @return a list of context data or a single context data entry
 * @throws DataSourceNotFoundException in case the data source ID is invalid
 */
ContextData getContextData(String datasourceID) throws DataSourceNotFoundException;

//Get context data by UUID.
contextManagerService.getContextData("1234-5768-9012-3456-7890");
```

Listing 13 shows how to retrieve historical information from a data source providing the UUID of a specific data source.

Listing 13: Get Historical Context Data

```
/**
 * @param datasourceID, defines the Data Source for which the context data shall be
returned
 * @return a list of context data or a single context data entry
 * @throws DataSourceNotFoundException in case the data source ID is invalid
 */
ContextData[] getHistoricalContextData(String datasourceID) throws
DataSourceNotFoundException;

//Get historical context data by UUID.
contextManagerService.getHistoricalContextData("1234-5768-9012-3456-7890");
```

RESTful interfaces are also provided and can be seen in Table 4. This service does not require any specific license or user rights.

Table 4: RESTful Interfaces for Context Manager

Functionality	Type	Command	Parameters	Description
Get a Subscription	HTTP-POST	<base_url>/contextmanger/getSubscriptionEndpoint	JSON object representing a SubscriptionRequest:{}	Will return a JSON representing a Subscription.
Close a Subscription	HTTP-GET	<base_url>/contextmanger/closeSubscriptionEndpoint/{subscriptionUri}	{subscriptionUri} : String of the Uri provided in the "/getSubscriptionEndpoint "	Will close the subscription with the provided URI.
Get ContextData	HTTP-GET	<base_url>/contextmanger/contextData/{datasourceID}	{datasourceID}	Will return a JSON representing a ContextData.
Get Historical ContextData	HTTP-GET	<base_url>/contextmanger/historicalContextData/{datasourceID}	{datasourceID}	Will return a JSON representing a ContextData Array

While Figure 6 shows how to request a subscription endpoint, Figure 7 shows the expected response. Furthermore, Figure 8 shows how to request context data from a context sensor and Figure 9 shows the expected response.

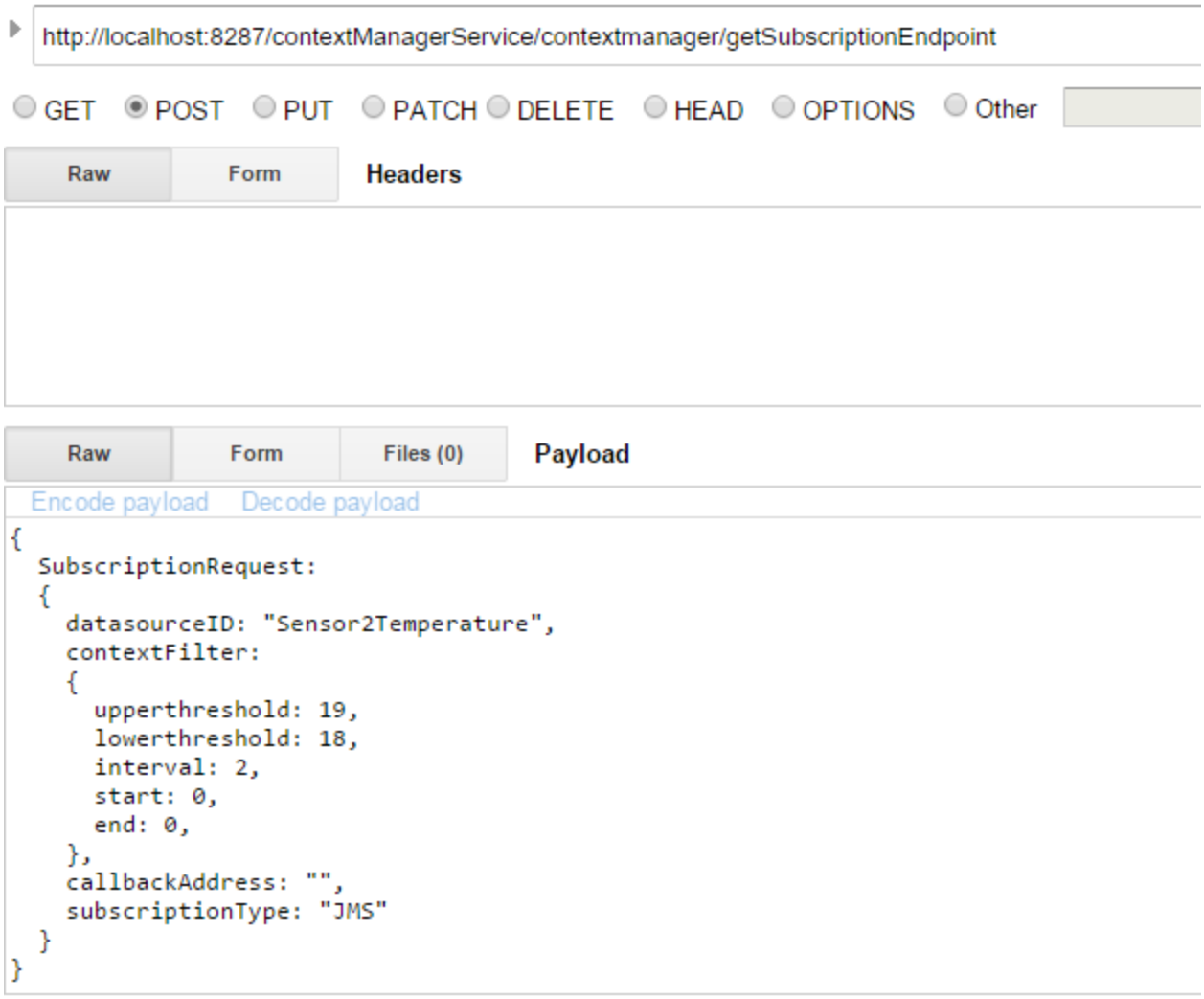

Figure 6: Request: Get Subscription Endpoint

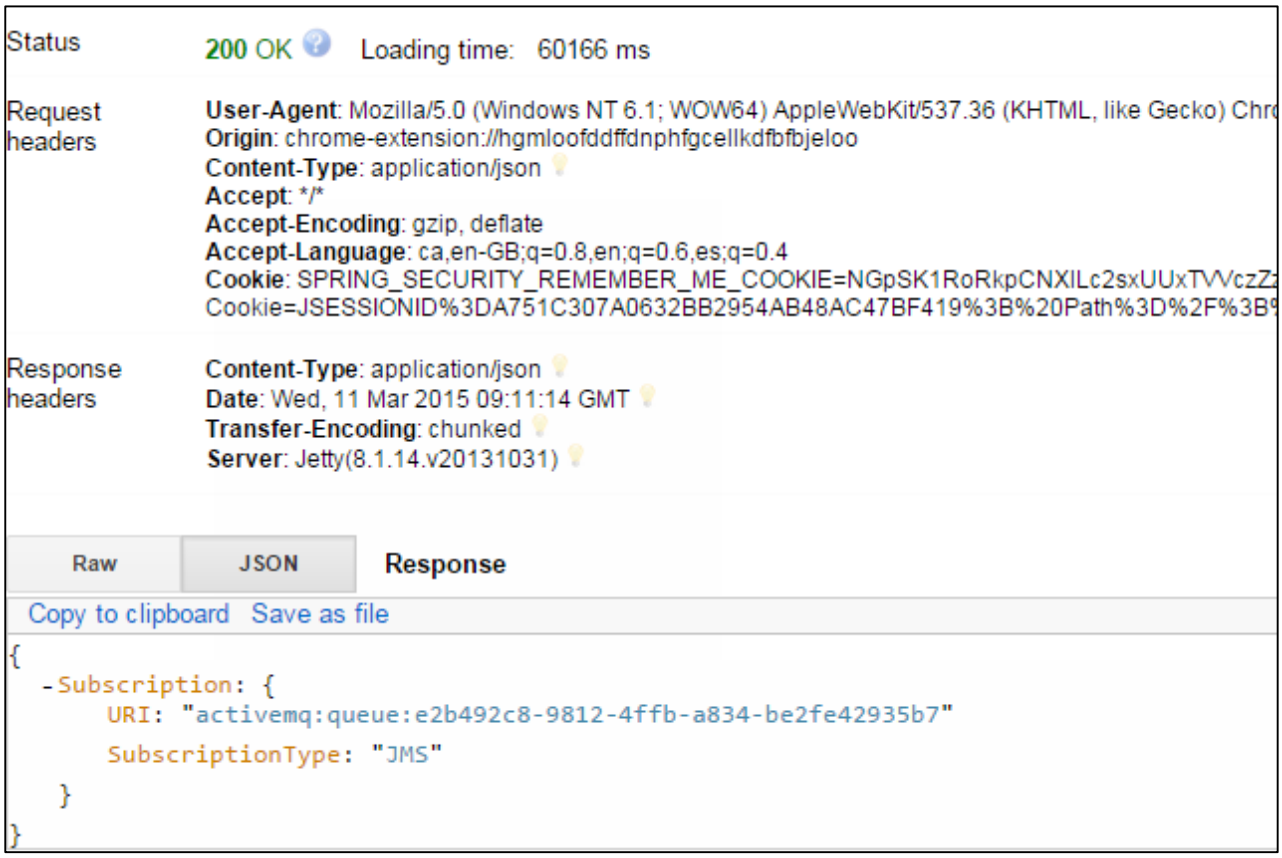

Figure 7: Response: Get Subscription Endpoint

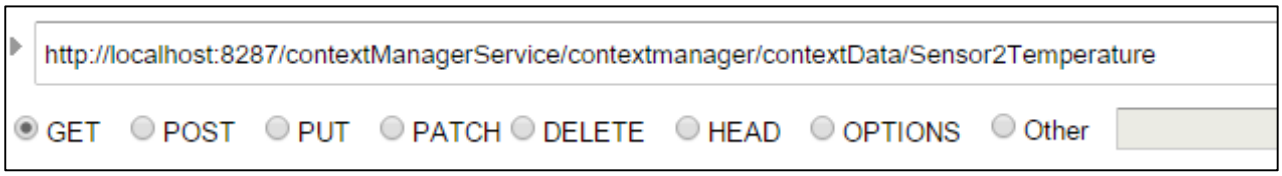

Figure 8: Request: Get Context Data from a Context Sensor

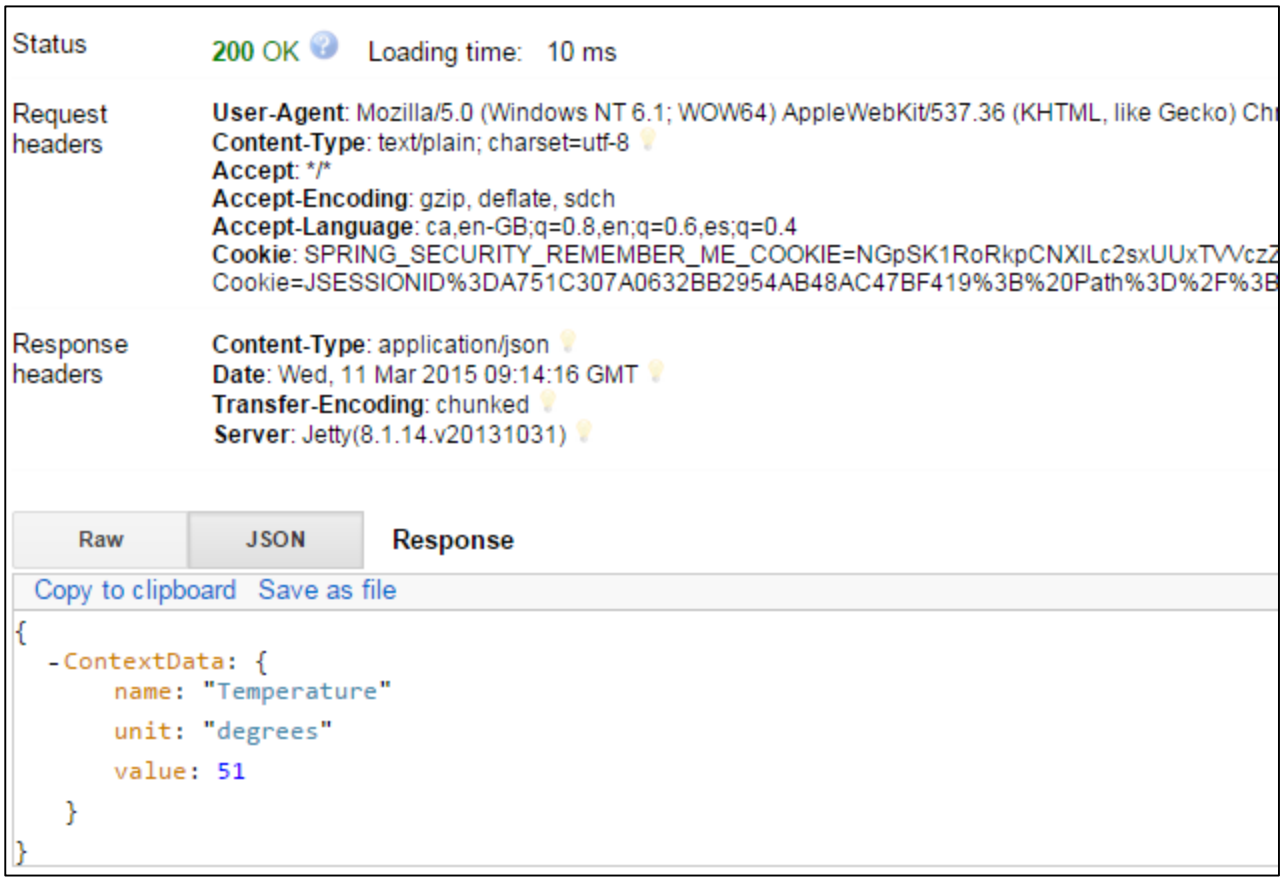

Figure 9: Response: Get Context Data from a Context Sensor

6 Summary

This section provides a brief overview of what is provided in the final prototype.

6.1 Reasoners

In the final prototype of the Context-based Service Personalization component, several different services have been developed. Different kinds of data are taken into account when reasoning about actions. For example, a service was developed which takes into account a location in order to find data services, e.g., in order to find a parking spot nearby. Another service was developed, which is able to consider user-centric preferences while generating an output, e.g., for transforming currencies or distances.

Although the provided services are able to reason about different actions and adapt to a given context, these services are rather passive than active. The different reasoners have been designed to work in a request-response manner, i.e., other backend services, or apps running on the PMA can actively send a request in order to retrieve personalized output. However, through the provision of a push notification mechanism, it is possible that another backend service pushes the personalized output directly to a user's PMA. These services provide Java interfaces and some RESTful interfaces, hence, it is possible to integrate them directly into other backend services or apps running on the PMA.

6.2 Publish-Subscribe Middleware

The final prototype of the Context-Based Service Personalization component implements the backend functionalities to publish all kinds of context data, store it and route it to the registered subscribers. This prototype uses TSB (TIE Smart Bridge, a commercial ESB implementation from partner TIE) for sending and receiving messages, but it can easily be changed to another transport service, e.g., a JMS (Java Message Service) message broker, XMPP (Extensible Messaging and Presence Protocol) message broker. In order to meet security criteria, the Publish-Subscribe Middleware is equipped with a secure layer, i.e., the Secure Access component. It validates credentials and provides role management to control the access to the sensor data. Last but not least, the final prototype provides a FilterContext processor which resolves filters. It can be extended to manage more complex filters like logic operators (AND and OR).

6.3 Context Manager

The final version of the Context-based Service Personalization component provides a fully implemented Context Manager, which is connected to the Context Database. This enables the storage of context data, which can be requested by a later point of other service. For this purpose, Java interfaces are provided for internal communication, i.e., within the Service Runtime Environment, and RESTful interfaces for usage from outside the Service Runtime Environment, e.g., from the PMA side.

6.4 Context Sensors

This prototype provides three example sensors. If a third-party developer wants to make use of a certain sensor, a traditional backend service can be created. Furthermore, this

backend service can be provided to other service developers. Thus, it acts as a (context) sensor on its own. An example is shown within this document in form of the Weather Sensor in Section 5.2.1.