simpli·city

The Road User Information System Of The Future

WP4 – Mobility-Related Data as a Service

D4.4.2: User-Centric Data and Open Data Management Prototype II

Deliverable Lead: IBM

Contributing Partners: IBM, TUDA, TUV

Delivery Date: 03/2015

Dissemination Level: Public

Version 1.0

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	Freddy Lecue, IBM
Internal Reviewer 1	Stefan Schulte, TUV
Internal Reviewer 2	Toni Paradell Bondia, Worldline
Type	Deliverable
Work Package	WP4: Mobility-Related Data as a Service
ID	D4.4.2: User-Centric Data and Open Data Management Prototype II
Due Date	31.03.2015
Delivery Date	31.03.2015
Status	For Approval

Document History	
Contributions	V0.1, Freddy Lecue, IBM 06.02.2015, Added document structure. V0.2, Freddy Lecue, IBM 12.02.2015, completed most sections V0.3, Martin Stephenson, IBM 15.02.2015, added content on architecture V0.4, Simone Tallevi-Diotallevi, IBM 18.02.2015, added more content on architecture V0.5, Freddy Lecue, IBM 09.03.2015, integrated all contributions in main document V0.6, Stefan (TUV) and Toni (World) Comments integrated V1.0, Martin Stephenson, Simone Tallevi-Diotallev, Freddy Lecue – Final version
Final Version	March 31st, 2015 (M30)

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 second and final prototype of the User-Centric and Open Data Management component from the SIMPLI-CITY Work Package WP4 – Mobility-related Data as a Service.

In this second prototype report for the User-Centric and Open Data Management, the scope of the prototype, the requirements coverage as well as the required preparations and deployment instructions to execute the prototype are explained.

In particular it has been shown how this prototype has been tightly integrated with all scenarios from WP7. This prototype strongly relies on the foundation of WP4 for accessing open data, car sensor data and user related data. Most of the APIs of the award winning system STAR-CITY (also part of WP4 as the data processing and reasoning component – described in D4.4.1) have been used to diagnose and predict traffic conditions in cities. It has also been shown how SIMPLI-CITY leverages existing technologies form various open APIs, services and systems such as Google Maps, Google Calendar, Apache Tomcat, and Apache CouchDB.

This document begins by giving a brief overview of SIMPLI-CITY, defining the purpose and scope of the deliverable and specifying the document structure. In Section 2 we report some related works. In Section 3 the scope of the prototype and the requirements, which have been covered, are described. For Open Data Management the collection of several of the Open Data datasets related to the SIMPLI-CITY Use-Case I topics for Dublin and Bologna are covered. Sections 4 details the architecture and the major components involved. Section 5 describes the major APIs while Sections 6 and 7 describe the necessary tasks to install the prototype. In Section 8 we provide details concerning the execution of the main deliverable.

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 second prototype of the User Centric Data and Open Data framework (tightly integrated with scenario of WP7) 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 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 (PMA) 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 second prototype of the User-Centric and Open Data Management component and exploit its functionalities. For this the scope and requirements of this component and this prototype, the requirements and preparations for users and developers and an installation and usage guide are provided. Further a brief overview of the implemented enhancements for the second prototype is given.

The User-Centric and Open Data Management Prototype II is the outcome of the discussions and implementation work done in project months 19 to 30. It provides a complete implementation of the functionalities for the User-Centric and Open Data Management as stated in SIMPLI-CITY deliverables D3.2.1 (Functional Specification), and D3.2.2 (Technical Specification). This deliverable D4.4.2 is the final prototype of the User-Centric and Open Data Management component.

1.3 Document Status and Target Audience

This document is listed in the Description of Work (DoW) as “Public”. This is the final prototype, which has been integrated with scenarios of WP7 in SIMPLI-CITY.

It provides the means to exploit the functionalities of the SIMPLI-CITY User Centric Data and Open Data as defined in deliverable D3.2.2 (Technical Specification).

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 including a general overview of the project, and outlines the purpose, scope, context, status, and target audience of this deliverable.

Section 2 overviews some related works in data integration (which is a core focus of the prototype) and positions SIMPLI-CITY with respect to relevant EU projects in the field.

Section 3 provides an overview of the scope and relationship of the prototype, showing how the User-Centric and Open Data component fits into the overall SIMPLI-CITY software framework as well as the outcome of the second prototype. Furthermore, an assessment of the requirements covered by this prototype is given.

Section 4 details the architecture and the major components involved.

Section 5 describes the major APIs.

Sections 6 and 7 describe the necessary tasks for a system administrator to prepare and install the prototype.

In Section 8 we provide details concerning the execution of the main deliverable.

Section 9 concludes this document.

2 Related Works and Past EU Projects in Data Integration

The following two sections overview some related works and projects in the context of data integration.

2.1 Related Works in Data Integration

Often, urban data is sourced from legacy non-relational systems or spreadsheets designed to be consumed by humans. The data is potentially very large, highly heterogeneous, spanning different domains, and with unknown structure (from static data to spatial-temporal data obtained from physical sensors). Moreover, the users who want to consume the data are not data integration experts and are not necessarily able to query data using structured query languages. We look at existent semantic approaches for processing, integrating and analysing such data.

Semantic technologies have been proposed to enrich the unstructured information space with ontologically annotated data, to introduce the necessary coherence, organization, data integration and dynamism to reach a highly-effective data model that can be used not only for exploration, but also to perform complex and unambiguous queries. However, converting raw government data to high quality Linked Data is costly [DLE+11] and approaches to do that at scale are limited. State of the art semantic approaches for urban data assume the existence of reference ontology(ies) to guide the conversion to RDF [ADS+07], or assume the input is a relational database, where the first row is used to suggest properties and each row refer to entities. The latest approach is used in the Datalift project [SAT+12] to automate the conversion from the source format (e.g. CSV, XLS, SHP) to the raw RDF, before transforming it to well-formed RDF by mapping to selected vocabularies through the use of SPARQL construct queries. The process used in Datalift is similar to ours. Datalift also integrates with the SILK framework for facilitating mappings between datasets. The LOD2[1] stack is also offering a variety of tools, supporting a series of information management and integration tasks.

The approach in [MCP12] is based on Google Refine for data cleaning and a reconciliation service extended with Linked Data capabilities to enable exporting tabular data into RDF. However, in our experience, this tool has limited fitness-for-use for the non-expert users.

Open content portals for cities such as London, Chicago and Dublin, to name a few, allow users to explore the relevant datasets by searching through keywords or by navigating through the different categories (e.g. weather stations, airports, arts, demographic, environment, housing, health, recreation). Datasets are searched by names (titles) and semi-structured metadata catalogues, but not by content (column names or values). Users can select a dataset and visualize the tabular data, plot it in a map or chart, and filter by column values, but they cannot aggregate data or refine exploration/queries across sources.

Content platforms for urban data require novel search and exploration models based on a hybrid space of data structure in any format (mostly tabular) and in any domain, and unstructured information, often in the form of short textual descriptions. Combining open data with visualizations can surface hidden insights and trends. However, the extreme

[1] http://lod2.eu/

heterogeneity and diversity of the relevant data makes it hard for users to discover and consume it. Furthermore, flexible data integration mechanisms are required to support information aggregation across datasets.

Traditionally, data integration architectures, based on creating a common virtual schema for a particular domain, and mapping the data to the schema, cannot cope with the scale and heterogeneity of urban data. Machine learning techniques, although capable of providing high-quality results, typically depend on the availability of training data, which is very difficult to acquire in such an open domain.

IBM City Forward[2] is a Web-based platform that allows users to create explorations across data by selecting cities, topics, and visualization types. The explorations are restricted to a set of a-priori predefined features and categories present in data from selected cities. It does not allow users to dynamically upload data or to create views by refining, selecting and combining data across diverse datasets and attributes. A similar approach to IBM City Forward is taken in the WatchDogs video game[3] where an appealing user interface is designed based on open data for a number of cities.

The approach in [CHM11] extracts structured data from tables on the Web, and allows clustering schemas that are semantically closed (based on the probability of seeing two certain attributes appearing together in a table), to suggest schema auto completion to users, and to propose semantic mappings between schemas, for example by identifying tables containing uniform data types.

Google fusion tables[4] enables users to upload (tabular) data and to visualize it in several ways (maps, timelines, and other charts), along with the ability to aggregate data across sources. It does not require the user to declare a schema upfront, but the burden to explore the relevant sources is shifted from the system to the user. There is no semantic meaning or description associated to the datasets, column names or values (no legends), and no mechanisms to help with the discovery and ranking of related datasets, or for exploring and redefining views within datasets according to user needs (e.g., spatial or content based queries).

2.2 Relevant EU Projects in Data Integration and SIMPLI-CITY

SIMPLI-CITY built on top of the European Project LarKC ("The Large Knowledge Collider"[5]), which addresses the collection and access of various types of information as well as real-time and archived sensor data. LarKC recently developed a platform for massive distributed incomplete reasoning that will remove the scalability barriers of currently existing reasoning systems for the Semantic Web. This was in particular supportive for the integration and management of Big Data (large amounts of heterogeneous data) and was therefore also useful for SIMPLI-CITY.

SENSEI – Integrating the Physical with the Digital World of the Network of the Future[6] – was an Integrated Project in the EU FP7. Its core contribution was the development of architecture and corresponding technological building blocks to bridge the physical and the

[2] http://cityforward.org/
[3] http://wearedata.watchdogs.com
[4] http://tables.googlelabs.com
[5] http://www.larkc.eu
[6] www.sensei-project.eu

digital world and thereby enable future smart environments. In this context, the focus of SENSEI was primarily on wireless sensor and actuator networks. Such devices have been considered in SIMPLI-CITY as well and thus the solutions provided by SENSEI. Nevertheless, to provide the means for the road information system of the future, SIMPLI-CITY incorporated a significantly wider variety of data sources.

SOA4ALL ("Service-oriented Architectures for All"[7]) was an FP7 project. Its aim was to develop a global, domain-independent service delivery platform where billions of parties can easily create and consume different services and data. Whereas SOA4ALL focuses on the development of a generic services framework that integrates advanced Web technology (i.e. Service-oriented Architectures, Web principles, Web 2.0, semantic Web technologies, Web of data, and context management), SIMPLI-CITY addresses an environment-aware mobility services framework that allows to define richer, more personalised services, which integrate data from various, heterogeneous sources that can be accessed via a voice-based user interface.

[7] http://www.soa4all.eu

3 Prototype Scope and Requirements Coverage

3.1 User-Centric Data and Open Data Management – General Information

The User-Centric and Open Data Management component of SIMPLI-CITY is responsible for handling the Open Data sources, which are available to SIMPLI-CITY such as traffic and weather information as well as data, provided by the end-user such as calendar information, or profile information, which indicates preferences. Both the Open Data and User-Centric data fed into the SIMPLI-CITY Unified Data Model are used in the Data Processing component where all available data (including sensor data in the final prototype) are transformed into the Unified Data Model that is described in detail in deliverable D4.1.1. This allows contextualization and reasoning over heterogeneous datasets. While the Open Data is fetched directly within the User-Centric and Open Data Management component, the User-Centric Data is accessed via the Sensor Abstraction and Interoperability Interfaces. The Sensor Abstraction and Interoperability Interfaces provide a Java interface on the Personal Mobility Assistant (PMA) side as well on the server side to directly access the user data. In addition, to provide external SIMPLI-CITY server components with access to the user data, a RESTful interface is provided on the server side.

Figure 1 below shows the location of the User Centric and Open Data, Data Processing (both centre) and Personal Data Source (top right) components highlighted in blue in an extract from the SIMPLI-CITY Global Architecture. For the full Global Architecture, please refer to deliverable D3.1. The highlighted components contribute to this second prototype of User-Centric and Open Data Management.

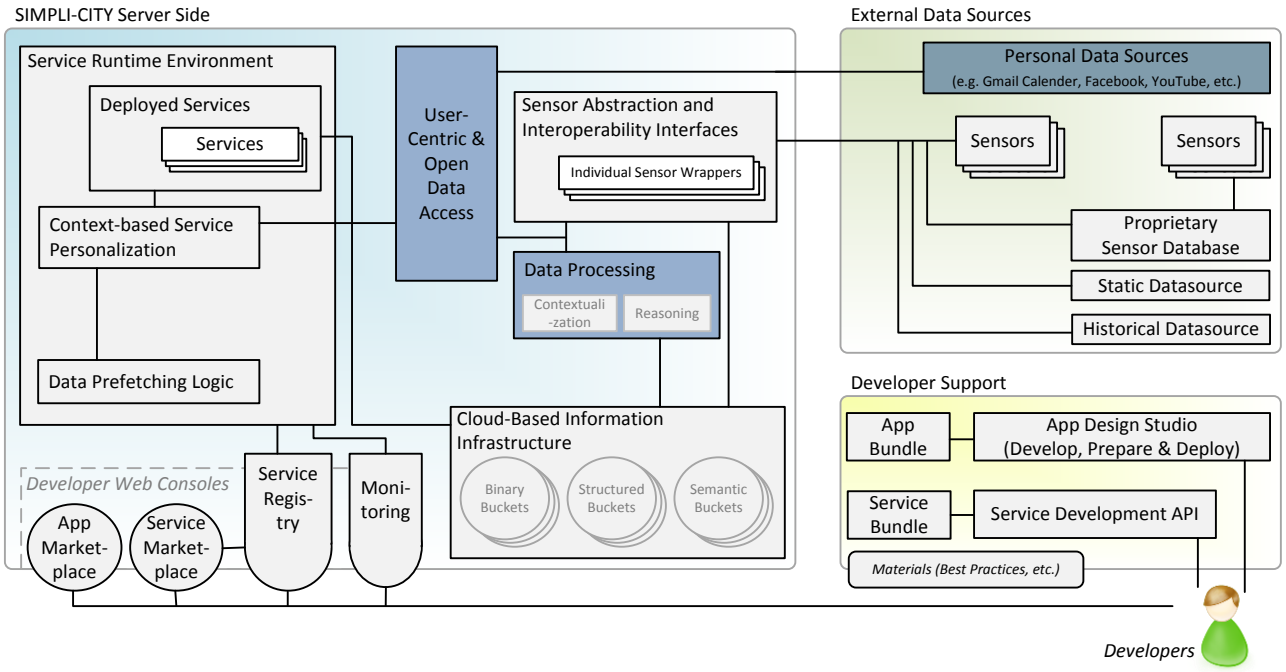

Figure 1: Location of User Centric and Open Data in the SIMPLI-CITY Global Architecture

3.2 Scope of the Second Prototype

The second prototype extends the functionalities of the first version. It focuses on the seamless integration of technologies provided by WP4, i.e., data access, storage, processing and reasoning. In the following subsections, the scope and status of all subcomponents of D4.4.2 will be discussed. For the Functional Specification and Technical Specification of these subcomponents, refer to SIMPLI-CITY deliverables D3.2.1 and D3.2.2, respectively.

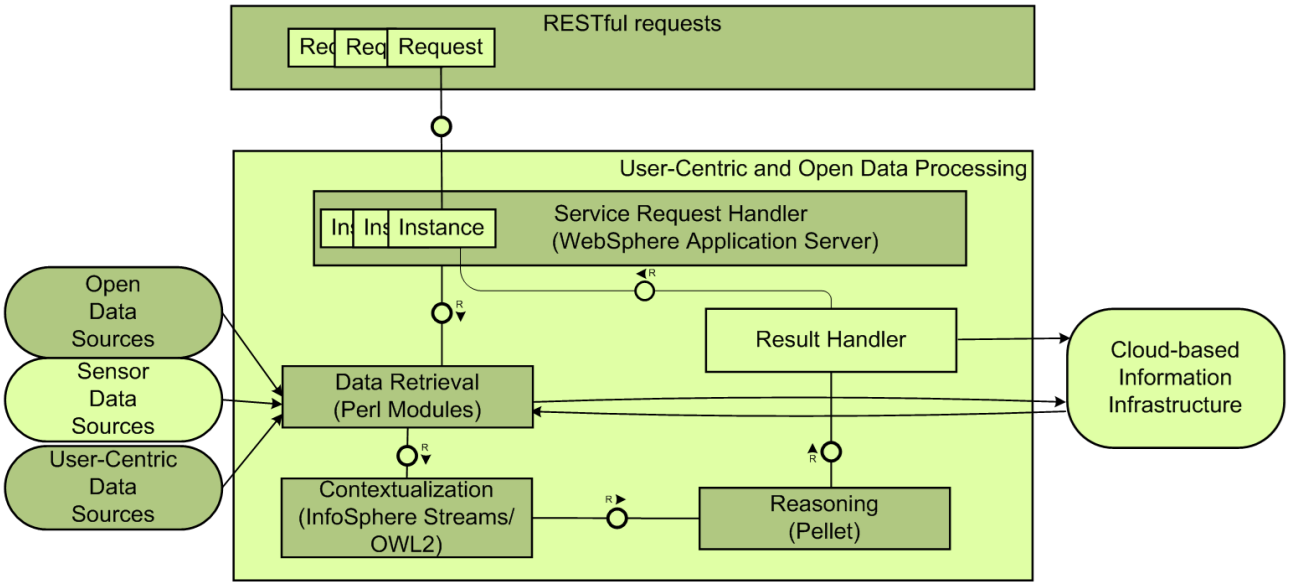

Figure 2: Scope of the Final Prototype of User-Centric Data and Open Data Management

Figure 2 depicts the scope and status of development of the final prototype of User-Centric Data and Open Data Management, showing the subcomponents that are covered within this final prototype.

The status of the implementation in Figure 2 shown using the following colour codes:

- Green: Fully implemented.
- Orange: Partially implemented.
- White: No implementation so far.

3.3 Open Data Management

Following the achievement of the first prototype, the collection of relevant Open Data datasets related to the SIMPLI-CITY Use-Case I topics for Dublin and Bologna has been done. Table 1 details all data sets, which have been collected and used for the WP7 scenario.

Table 1: List of Open Data Sets Used in this Prototype

Open Data Sets Utilized		
Dublin		
Provider	**Description**	**Base URL**
Dublin City Council	Dublin City Traffic Trip times	http://www.dublinked.ie
Dublin City Council	Dublin City Roadworks	http://www.dublincity.ie
Wunderground	Dublin Weather Info	http://www.wunderground.com
AA RoadWatch	Twitter Feed	https://twitter.com
LiveDrive information	Twitter Feed	https://twitter.com
Garda updates	Twitter Feed	https://twitter.com
Eventful	Dublin Event Info	http://api.eventful.com
Eventful	Dublin Event Info	http://api.evdb.com
Bologna		
Cisium / Municipality of Bologna	Bologna Traffic flow information	http://82.187.83.50
Cisium / Municipality of Bologna	Bologna Road links information	http://82.187.83.50
Cisium / Municipality of Bologna	Bologna Road Work information	http://82.187.83.50
Wunderground	Bologna Weather Info	http://www.wunderground.com
Eventful	Bologna Event Info	http://api.eventful.com

To demonstrate the Open Data collection and transformation mechanisms, REST APIs have been developed which return current values as raw (CSV format) or transformed (RDF format) data for the requested dataset and frequency, e.g., 1 minute interval, 10 minute interval, etc. Detailed information concerning the data access and transformation (in the SIMPLI-CITY data model) are provided in D4.1.2.

3.4 User Centric Data Provision – Personal Data Sources

In this second and final prototype of User-Centric and Open Data Management the scope of User-Centric Data is based on data extracted from the users' PMA. The different user accounts, e.g., Google accounts are merged on the PMA side and data is extracted into one common data set for contact information and one for calendar information. Contact information represents address book of the user. Calendar information contains particular event information and map information, e.g., the user's country information as well as their home and work locations.

The provision of this data in the SIMPLI-CITY data model is conducted via REST API calls to the Sensor Abstraction and Interoperability Interfaces. These calls return the relevant User-Centric Data, e.g., calendar events to the caller in the JSON schema format corresponding to sensor data.

As mentioned before, the data source of User-Centric Data is the users' PMA. On the PMA side, i.e., an Android based smartphone, the user can add several online accounts, e.g., Google or Facebook accounts. All these accounts may contain some kind of address book or calendar. All these data sources are merged within the mobile operating system, i.e., Android, which is also responsible for data synchronisation. This basic functionality is provided by all major mobile operating systems, i.e., Android, Windows Phone and iOS. The PMA runs a background service of the Sensor Abstraction and Interoperability Interfaces responsible for accessing this data. This background service provides a Java interface to provide data access to the SIMPLI-CITY Apps running in the Application Runtime Environment (ARE). Furthermore this background service communicates with the server side of the Sensor Abstraction and Interoperability Interfaces who provide an interface to other SIMPLI-CITY services to access this user data. The server side interfaces are implemented as Java interfaces for services executed on the same physical machine as well as RESTful interfaces to provide access to SIMPLI-CITY services executed on other machines.

3.5 User Centric and Open Data Processing

The User-Centric and Open Data is transformed into the SIMPLI-CITY Unified Data Model in the Data Processing component. This allows for contextualizing over the data, i.e., finding relationships between user locations, weather and traffic conditions for the relevant region (Dublin or Bologna) and to perform reasoning, predictive and diagnosis tasks. The scope of this second and final prototype for the Data Processing component is to demonstrate contextual driving experience by:

- Capturing user contexts and car sensor contexts for personalized navigation
- Diagnosing road traffic conditions related to User-Centric and Open Data
- Predicting road traffic condition using Open Data and User-Centric Data sources

3.6 Covered Requirements

This section describes the degree of fulfilment of the requirements to be covered by the User-Centric Data and Open Data Management component as specified in the Requirements Analysis Deliverable (D2.3) and the Functional Specification (D3.2.1).

Table 2: Requirements Related to User Centric and Open Data Management and their Degree of Fulfilment

Requirement	Degree of Fulfilment	Comment
Must Have Requirements		
U31: Reaction to time of the day	100%	Supports full integration of time-based data, fulfilling the SIMPLI-CITY spatio-temporal contextualisation requirements. Real-time data from user-data, car sensor data and open data have been integrated. The prototype is fully correlated with events occurring during the day.
U34: Reaction to history of usage (User-centric)	100%	Functionality fully covered by the second prototype. Previous destinations are stored and could be eventually re-played by the end-user.
U35: Reaction to traffic information, like traffic jams, train schedules, road works, accidents, and strikes	100%	Functionality fully covered by the second prototype. Traffic diagnosis and prediction are both analytics frameworks, which require the use and interpretation of traffic, and contextual conditions in real-time.
U59: User centric data services	100%	Automatic fetch of User Personal Data Sources and profile data sources from Sensor Abstraction and Cloud-based Information Infrastructure are implemented and used for contextualization. In particular users are able to navigate through events rather than destination. Events are coming from personal data from the end-user.
U80: Profile in the cloud	100%	User profile data is retrieved from the Cloud-based Information Infrastructure. Privacy and security mechanisms are ensured.

Requirement	Degree of Fulfilment	Comment
U86: Transparency U87: Confidentiality - Do not give away data to third parties U88: Data encryption U89: Certification - Only certified apps are allowed to access users data	90%	Fully implemented and supported by this prototype. These requirements have been fulfilled by following recommendations from D3.3 (Holistic Security and Privacy Concept). In particular appropriate a specific technique of anonymisation has been coupled with encryption to ensure secure access to personal data. However some extra tasks related to encryption are required to ensure complete secure connections with the cloud infrastructure (due to the level of personal data captured in the cloud infrastructure).
U90: Availability U91: Integrity U92: Secure access to system U93: Third party access to the system	100%	A high level of availability has been achieved through a stable system and well-used exception handling. In case of errors the system will remain functional and appropriate errors are sent to the server (which exposes the APIs and REST services) and the responsible person (e.g., IBM for diagnosis and predictive reasoning). Integrity and secure access has been achieved by applying authentication mechanism and secure session IDs to access the system via the REST Proxy. This ensures that services can only be accessed by authorized users and each request is logged. Further, third parties can request access to the system by requesting username and password.
U112: Support of Open Data	100%	All relevant Open Data sets for Dublin and Bologna have been retrieved, e.g., traffic data, weather, data, city events, road works, restricted area.

Requirement	Degree of Fulfilment	Comment
U114: Configuration of the frequency of update of the data from data sources	100%	Frequency manipulation is possible via REST APIs. This parameterization has an impact on scalability. It is recommended to not upgrade the frequency to keep the analytics scalable. Although that would be possible due to the REST interface cf. D4.4.1.
U115: Transformation support	100%	All relevant Open Data, user and car sensors sources can be transformed. The SIMPLI-CITY data format is used as a final representation for all data exposed in SIMPLI-CITY.
U116: Unified Data Model	100%	Data Model version II is finalized in the Unified Data Model. The model has been extended with extra representation to cover the full set of data required to accomplish the scenarios in WP7 and WP8.
U117: Data filtering U118: Data correlation	100%	Fully implemented. Both diagnosis and prediction reasoning require data filtering and correlation for scalable processing. Data has been filtered by spatial and temporal features while correlation has been achieved though association mining of stream data and ontologies.
U119: Data Summarization	100%	Fully implemented. Data is reduced to its minimal size to ensure scalable processing. In particular we strongly relied on U117 (Data filtering) and U114 (Configuration of the frequency of update of the data from data sources) to ensure compact representation.
U120: Handle data streams	100%	Stream operators implemented. They are the basis of the model. Filtering and correlation are operated at stream level, i.e., all data is processed in real-time.

Requirement	Degree of Fulfilment	Comment
U189: Unified interface for accessing sensors	100%	The Sensor Abstraction Interface handles access to the vehicle sensors as well as the smart device sensors and provides data in a format that conforms to the Unified Data Model. The interface ensures an unified access to all data from the car, therefore it is easily portable to different platforms.
U190: Unified interface for accessing user centric data	100%	Sensor Abstraction Interface provides a unified interface to access user centric data. The interface ensures unified access to all user data, therefore it is easily portable to different platforms.
U202: Diagnosis of abnormal traffic condition in real-time U203: Prediction of abnormal traffic condition U204: Support for querying diagnosis historic U205: Support for querying impact factor on traffic condition U206: Optimization of financial resources U207: Support suggestions for road/trip optimization if conditions change	100%	Functionality covered by the second prototype. Diagnosis and prediction are based on historical filtering and analysis of SIMPLI-CITY representation of data. Such reasoning provides explanation and prediction of traffic conditions. REST interfaces are provided to access traffic conditions as well as the results of diagnosis and prediction in real-time. State-of-the-art techniques are used to re-compute the route from one location to another one when conditions change.
U208: Possibility to continue a previously started trip on the same or a different device	100%	Access to APIs for user profile history data. Session IDs are provided to each REST interfaces. Accessing historical calls and obtaining relevant information from previous calls from other devices is a major reason for keeping session IDs.
U85: Interaction with car sensors	100%	The Sensor Abstraction Interface handles access to the vehicle sensors and the PMA sensors. SIMPLI-CITY captures in real-time data from car sensors, which are used for providing personalized information to end-users.

Requirement	Degree of Fulfilment	Comment
U108: Access to smart device sensors	100%	This functionality is available in the second prototype. GPS location is used from the phone to track the progress of the end-user and provide contextualized information.
U109: Access to remote sensors	100%	Open Data for traffic and weather sensors is accessed. Traffic data from different cities is captured. Examples are weather stations and loop inductions.
U50: Prefetching of media data & Offline access	100%	Related to T4.5. The Data Prefetching subcomponent of SIMPLI-CITY aims at mitigating the problem that mobile environments, including these the PMA will operate in, are typically subjected to network fluctuations and intermittent downtimes. Prefetching is possible for all kinds of data types, but is naturally especially critical for streaming data, in order to enable uninterrupted playback.
Should Have Requirements		
U52: Offline access of data used within apps U113: Handling of multimedia data	100%	Streaming of User-Centric Data is implemented in T4.4. Offline access to data is realized through prefetching as part of the work done in T4.5.
U107: Access to sensors of the vehicle	100%	T4.1 provides a unified model for data description in the JSON data format. T4.3 provides access to a set of car sensors.
Will Not Have for Now Requirements		
U110: Remote control of car components, e.g., air conditioning, heating, battery charge timing	0%	Functionality not covered by the prototype

4 Architecture

4.1 High-Level Architecture

For a detailed description of the components in this diagram see Section 5.

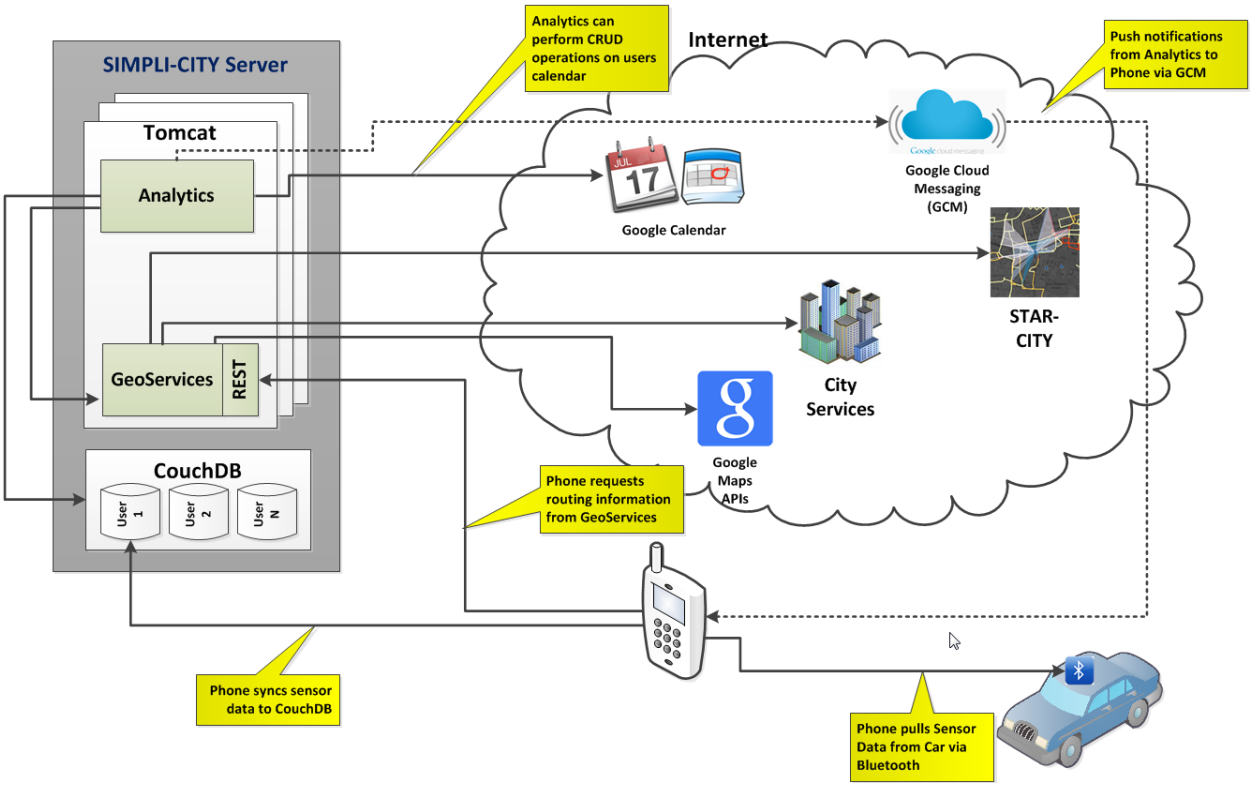

Figure 3: High Level Architecture of the User Centric Data and Open Data Component

4.2 Design Choices

4.2.1 Decision to Have SIMPLI-CITY Analytics Read the User's Calendar

The system could have been written so that the mobile app monitors the user's calendar, however, this would have been very battery intensive for the mobile device.

Additionally, due to the fact that SIMPLI-CITY Analytics are monitoring the calendar, this component can augment the event location information by checking if it is in a restricted area of the city. This changes the architectural design of the system as the analytics are now monitoring the calendar, rather than the mobile app monitoring the calendar (which would have been overly battery intensive, as mentioned previously).

4.2.2 CouchDB for Sync / Replication

CouchDB provides the possibility for multiple devices to be connected to the same account. Additionally it provides functionality to replicate data to the mobile platform, to allow user to work offline in areas with no data connection.

5 Functionalities and APIs

Considering the components described in the previous section, the functionalities and APIs are provided as follows.

5.1 STAR-CITY

STAR-CITY provides a set of RESTful APIs that can be used to find the *Anomalies* over the standard traffic flow behaviour and the *Diagnosis* for those Anomalies.

Considering the potentially large amounts of data, STAR-CITY provides pagination capabilities that facilitate a two phases interaction. A generic execution is based on two REST calls (GET or POST):

- The first call to the URL *http://[IP]:[PORT]/star-city/rest/historical* takes as input the following parameters (Listing 1):
  - *type*: the type of functionality
  - *location*: identifies the city
  - *bounding box*: defines a portion of the map
  - *temporal constraints*: multiple periods of time

The first call creates a session on the server and it returns a session id (Listing 2);

- The second call to the URL *http://[IP]:[PORT]/star-city/rest/historical/session* takes as input the following parameters (Listing 3):
  - *session id*: identifies the current session and it is needed to obtain the results of the request
  - *number of results*: number of results that will be returned by the current REST call

and the second call returns the actual results (Listing 4);

The second REST call can be iterated until no more results are found (Listing 5) and the session is destroyed on the server.

The following is an example of the two phases interaction to retrieve anomalies and diagnosis from Star-City. A first POST/GET call to the URL *http://[IP]:[PORT]/star-city/rest/historical* is used to obtain a session id (Listing 2).

Listing 1: http://[IP]:[PORT]/star-city/rest/historical – Example of JSON Input

```
{
   "location": "DUBLIN",
   "type": "ANOMALIES",
   "startDate": 1385942400,
   "endDate": 1387497600,
   "startTime": 0,
   "endTime": 86400,
   "boundingBox": {
      "max": {
         "x": -6.286,
         "y": 53.36
      },
      "min": {
```

```
         "x": -6.233,
         "y": 53.336
      }
   },
   "numOfDiagnosis": 1
}
```

Fields explanation (Listing 1):

- location: city identification,
- type: type of functionality,
- startDate: midnight on start date in unix timestamp,
- endTime: midnight on end date in unix timestamp,
- startTime: start time expressed in seconds of a day,
- endTime: end time expressed in seconds of a day,
- boundingBox: bounding box min (south/west) and max (north/east) coordinates,
- numOfDiagnosis: max number of diagnosis required in the results.

Listing 2: http://[IP]:[PORT]/star-city/rest/historical – Example of JSON Output

```
{
   "type": "ANOMALIES",
   "location": "DUBLIN",
   "sessionID": "56097742...b34827"
}
```

Fields explanation (Listing 2):

- type: type of functionality,
- location: city identification,
- sessionID: id of the current session.

The session id can now be used to obtain the actual results by sending it with a second call to the URL *http://[IP]:[PORT]/star-city/rest/historical/session* with the following input (Listing 3).

Listing 3: http://[IP]:[PORT]/star-city/rest/historical/session – Example of JSON Input

```
{
   "sessionID": "56097742...b34827",
   "numOfResults": 30
}
```

Fields explanation (Listing 3):

- sessionID: id of the current session,
- numOfResults: number of results for the current request.

This second request will return an array of anomalies with the most probable diagnosis: the system explanation (Listing 4).

Listing 4: http://[IP]:[PORT]/star-city/rest/historical/session – Example of JSON Output


```
{
   "startTimestamp": 1385942400,
   "endTimestamp": 1387497600,
   "startTime": 0,
   "endTime": 86400,
   "boundaryBox": {
      "max": {
         "x": -6.286,
         "y": 53.36
      },
      "min": {
         "x": -6.233,
         "y": 53.336
      }
   },
   "sessionID": "56097742...b34827",
   "results": [
      {
         "id": "A_1385943186_120_133",
         "point": {
            "x": -6.24,
            "y": 53.337
         },
         "startTimestamp": 1385943186,
         "endTimestamp": 1385943186,
         "type": "TRIPS",
         "roadName": "Clanwilliam Place",
         "severity": 4,
         "districtId": "405092",
         "locationId": "120_133",
         "diagnosis": [
            {
               "id": "E0-001-062066479-1",
               "point": {
                  "x": -6.2,
                  "y": 53.34
               },
               "startTimestamp": 1385596800,
               "endTimestamp": 1386115200,
               "eventName": "National Aca...",
               "venueName": "The Lir",
               "address": "National Academy ...",
               "type": "social",
               "subTypes": [
                  "other"
               ],
               "confidence": 0.72716814,
               "eventDescription": "descr."
            }
         ]
      }
   ]
}
```

Fields explanation (Listing 4):

- startTimestamp: midnight start on date in unix timestamp,
- endTimestamp: midnight end on date in unix timestamp,
- startTime: start time expressed in seconds of a day,
- endTime: end time expressed in seconds of a day,
- boundingBox: bounding box min (south/west) and max (north/east) coordinates,
- sessionID: id of the current session,
- results: array of Anomalies detected by the system.

Anomaly fields explanation (Listing 4):

- id: unique id of the anomaly,
- point: GPS coordinates,
- startTimestamp/endTimestamp: period of anomaly (unix timestamp),
- type: type of anomalies,
- roadName: name of the road with the anomaly,
- severity: severity of the anomaly,
- districtId: id of the city area,
- locationId: road segment id,
- diagnosis: Array of diagnosis detected by the system.

Diagnosis fields explanation (Listing 4):

- id: unique id of the diagnosis,
- point: GPS coordinates,
- startTimestamp/endTimestamp: period of diagnosis (unix timestamp),
- eventName: name of the event
- venueName: name of the venue
- address: address of the event
- type: type of the event,
- subType: list of sub-types of the event,
- confidence: confidence of the diagnosis,
- eventDescription: description of the event.

5.2 Geo-Services

The Geo-Services component is responsible for providing the binding of the routing information obtained from the Google Maps API[8] with the relevant anomalies and diagnosis detected by the STAR-CITY APIs (above section). In this way the Geo-Services component augments basic routing information (like the ones provided by Google), by providing a facility to infer any anomalies that may happen for the current traffic data and an explanation (diagnosis) for these anomalies. By exploiting a REST interface the information is retrieved by a POST request to *http://[IP]:[PORT]/geoServices* with the following input (Listing 5)

Listing 5: http://[IP]:[PORT]/geoServices – Example of JSON Input

```
{
   "start": 1418061180,
   "end": 1418061780,
   "orig": "53.36635,-6.30178",
   "dest": "53.3437471,-6.2547592",
   "sensor": false,
   "mode": "driving",
   "alternatives": true,
   "threshold": "0.025",
   "gas_station": true
}
```

As can it be seen from Listing 5, Geo-Services requires the following input:

- *Temporal constraints*: the relevant period of time
- *Spatial constraints*: the origin and the destination of the route
- *Google Maps APIs parameters*: extensively described in the next section
- *Threshold*: the anomaly distance threshold used to correlate the anomalies with the points of the route
- *Gas station*: flag to indicate if a stopover at a gas station is required

The REST services will answer with the following JSON Array (Listing 6), containing:

- Routing info: Information coming from Google API and described here https://developers.google.com/maps/documentation/directions/

All information coming from STAR-CITY and its diagnosis component are described in the paragraph following Listing 4.

Listing 6: http://[IP]:[PORT]/geoServices – Example of JSON Output

```
[
   {
      "_id": "route_0",
      "type": "route",
      "geometry": "line",
      "info": {
         "duration": {
            "text": "13 mins",
```

[8] https://developers.google.com/maps/do cumentation/webservices/

```
            "value": 796
         },
         "distance": {
            "text": "6.0 km",
            "value": 6018
         },
         "end_location": {
            "lng": -6.25,
            "lat": 53.34
         },
         "start_address": "129-179 Dublin...",
         "end_address": "1 P..., Dublin...",
         "start_location": {
            "lng": -6.3,
            "lat": 53.36
         }
      },
      "bounds": {
         "southwest": {
            "y": 53.34,
            "x": -6.3
         },
         "northeast": {
            "y": 53.36,
            "x": -6.25
         }
      },
      "anomalies_count": 12,
      "name": "Route-option 0",
      "stops_over": [
         {
            "point": {
               "y": 53.37,
               "x": -6.28
            },
            "address": "None Defined",
            "name": "Esso",
            "type": "location.gas_station",
            "geometry": "point"
         }
      ],
      "diagnosis": [
         {
            "_id": "E0...@2014",
            "event_subTypes": [
               "other"
            ],
            "startTimestamp": 1418063400,
            "endTimestamp": 1418077800,
            "event_type": "social",
            "confidence": 0.92340904,
            "type": "diagnosis",
            "geometry": "point",
            "point": {
               "y": 53.343,
               "x": -6.254
```

```
            },
            "eventDescription": "some description",
            "address": "Trinity College Dublin",
            "eventName": "Trinity College Dublin...",
            "venueName": "The Science Gallery"
         }
      ],
      "anomalies": [
         {
            "_id": "A_1...52",
            "event_subTypes": [
               "disruptions"
            ],
            "startTimestamp": 1418061607,
            "endTimestamp": 1418061607,
            "diagnosis": [
               "E0-001-...7-8"
            ],
            "severity": 5,
            "type": "anomalies",
            "roadName": "Arran Quay",
            "geometry": "point",
            "point": {
               "y": 53.34,
               "x": -6.27
            },
            "locationId": "37_152",
            "districtId": "405092",
            "anomaly_type": "TRIPS"
         }
      ],
      "points": [
         {
            "y": 53.35,
            "x": -6.28
         },
         {
            "y": 53.35,
            "x": -6.28
         },
         {
            "anomalies": [
               "A_1..._20"
            ],
            "y": 53.35,
            "x": -6.281
         }
      ]
   }
]
```

5.3 City Services

City services represent the services provided by the different cities for the use case. Services from Bologna and Dublin were used.

5.4 Bologna – SARA Services

The SARA services provide information about the restricted area for Bologna city. These services are implemented with a SOAP architecture and the restricted areas can be retrieved calling the SOAP method "*getUnauthorizedAccessAreas*" described in the WSDL specification available here https://sara.comune.bologna.it/WsZoneSirio/services/ZtlService?wsdl

The SOAP method *getUnauthorizedAccessAreas* requires the following input (Listing 7):

- Username/Password: allow the identification of the user
- PermitId: identification of the authorization
- LicensePlate: licence plate number of the car
- StartTime: the start of the time window requested
- EndTime: the end of the time window requested

Listing 7: Example of Input of the getUnauthorizedAccessAreas Method

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:types.ztl.sara.comune.bologna.it">
   <soapenv:Header />
   <soapenv:Body>
      <urn:Request><![CDATA[<?xmlversion='1.0' encoding='UTF-8'?>
            <p:GetUnauthorizedAccessAreas
                 EndTidme='2001-12-31T12:00:00'
                 LicensePlate='AD223ER' PermitId='DE/1442'
                 StartTime='2001-12-31T12:00:00' xmlns:p='urn:
                 types.ztl.sara.comune.bologna.it' xmlns:xsi='
                 http://www.w3.org/2001/XMLSchema-instance'

xsi:schemaLocation='urn:types.ztl.sara.comune.bologna.itZtlGetNotAuthorizedAccessAreasS
ervice.xsd'>
                  <p:Credentials Password='' Username=''/>
                  </p:GetUnauthorizedAccessAreas>]]></urn:Request>
   </soapenv:Body>
</soapenv:Envelope>
```

By exploiting these information (Listing 7) the system detects restricted areas for the specific user, e.g., some user may not have the same restriction because they are residents in a restricted areas or they had been already authorized.

The SOAP method *getUnauthorizedAccessAreas* will answer with the following output (Listing 8):

- Result code: execution outcome

- Unauthorized access areas: a KMZ string representing a file expressed in Base64 and containing the unauthorized access areas (https://developers.google.com/kml/documentation/kmzarchives)

Listing 8: Example of Output of the getUnauthorizedAccessAreas Method

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<GetUnathorizedAccessAreasResponse xmlns="urn:types.ztl.sara.comune.bologna.it">
    <Result Description="KML generato correttamente" ResultCode="OK"/>
    <UnauthorizedAreas>PD94bWwgdmVy.....</UnauthorizedAreas>
</GetUnathorizedAccessAreasResponse>
```

The value contained inside the tag *UnauthorizedAreas* can then be processed and transformed in KML-format like in the example below (Listing 9):

Listing 9: Example of UnauthorizedAreas Output in KML-Format

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2" xmlns:atom="http://www.w3.org/2005/Atom"
xmlns:gx="http://www.google.com/kml/ext/2.2"
xmlns:kml="http://www.opengis.net/kml/2.2">
   <Document>
      <name>DE</name>
      <open>1</open>
      <Folder>
         <name>folder 1</name>
         <TimeSpan>
            <begin>2014-04-11T06:30:00Z</begin>
            <end>2014-04-11T07:00:00Z</end>
         </TimeSpan>
         <Placemark>
            <name>zone</name>
            <open>1</open>
            <Polygon>
               <outerBoundaryIs>
                  <LinearRing>
                     <coordinates>11.3466024,44.5042188,0
11.3455296,44.50449429999999,0 11.3444996,44.5011581,0 11.3398647,44.5019233,0
11.3358307,44.4931686,0 11.3386631,44.4916686,0 11.3400364,44.49142359999999,0
11.339736,44.4866783,0 11.3561296,44.4846882,0 11.3576746,44.485821,0
11.3558722,44.5009132,0 11.3466024,44.5042188,0</coordinates>
                  </LinearRing>
               </outerBoundaryIs>
            </Polygon>
         </Placemark>
      </Folder>
   </Document>
</kml>
```

The KML-format output above (Listing 9) describes the restricted area in GPS coordinates. A full description of the KML format can be found in the latter Listing (Listing 9).

5.5 Bologna – Purchase Ticket for Restricted Area

This service is just a mockup that has been used to demonstrate the possibility of buying the ticket for a restricted exploiting a service provided by the city.

5.6 Geo-Maps API

The Google Maps APIs are a set of APIs provided by Google, to provide geographic data for map applications. Specifically, SIMPLI-CITY exploits the Google Directions API that gives directions between locations. One can find the complete list of functionalities and API specifications at https://developers.google.com/maps/documentation/directions/.

5.7 Analytics

The Analytics are responsible for notifying the user of new updates in their calendar, or relevant new information that is available in the CouchDB database, e.g., the fuel in the car is low. This component, however, does not provide any externally callable set of APIs. It simply monitors the data and sends push notification via the Google Cloud Messaging service to the user's mobile app.

5.8 CouchDB

CouchDB is a NoSQL database that stores data with JSON documents. It allows full access to the documents and query functionality via RESTful API calls. CouchDB also provides a mechanism to distribute the data efficiently using incremental replications with master-master setups and automatic conflict detection. The instance of a CouchDB can be accessed with an HTTP request at *http:///[IP]:[PORT]:/_utils/.* All possible APIs of CouchDB can be found on the according website[9].

The system exploits the synchronization features of CouchDB, enabling the replication of data between the mobile and the SIMPLI-CITY server databases. Due to this architecture, every change on the server database will be replicated to the mobile one and vice versa. This feature allows the user to access the app functionalities either offline or online, e.g., the app can collect data from the car even offline and synchronize this information to the SIMPLI-CITY server when the app is back online.

The complete specification of the replication API can be found on the according website[10].

5.9 Google Cloud Messaging

Google Cloud Messaging (GCM) is a service that allows third parties to send push notifications to Android based apps[11]. In the context of the SIMPLI-CITY system, the push notifications are sent from the SIMPLI-CITY Analytic component whenever relevant data updates (or deletions) are on the server database. When a new event in the user's calendar is available, it is stored in the server database, causing a push notification to be sent to the mobile app to tell the user about the new event.

[9] http://docs.couchdb.org/en/1.6.1/intro/api.html
[10] http://docs.couchdb.org/en/latest/api/
[11] https://developer.android.com/google/gcm/index.html

6 Software Requirements

This section provides information about what system administrators need to prepare in order to use the functionalities of the delivered prototype.

6.1 Server Side (System Administrators)

6.1.1 Servlet Container

In order to deploy the SIMPLI-CITY server, it is necessary to have a Tomcat server installed (we recommend Tomcat 7.0). It can be downloaded from:

- http://tomcat.apache.org/download-70.cgi

Detailed instructions on installing Tomcat can be found here:

- http://tomcat.apache.org/tomcat-7.0-doc/setup.html
- http://tomcat.apache.org/tomcat-7.0-doc/RUNNING.txt

In this document, it is presumed that Tomcat 7.0 is successfully installed and running on the developer's computer.

It is also necessary that the operating system hosting the server side must have the TCP networking port 8080 free, since this is the port that will be used by the Tomcat server.

6.1.2 CouchDB

This second prototype has been tested on the Ubuntu 12.04.5 LTS Operating System using the Java SE Development Kit 6, even though it is usable under all major Java platforms running in Windows or Linux.

The server side storage system is CouchDB, which is a JSON document store.

It can be downloaded from:

- http://couchdb.apache.org/#download

Detailed instructions on installing CouchDB can be found here:

- http://docs.couchdb.org/en/1.6.1/

In this document, it is presumed that CouchDB 1.6.1 (this is the version we recommend) is successfully installed and running on the developer's computer.

It's also necessary that the operating system hosting the serversSide must have the TCP networking port 5984 free, since this is the port that will be used by CouchDB.

7 Installation (Deployment)

This section provides guidelines on how to install and deploy the SIMPLI-CITY Server Web App.

7.1 Services

The Geo-Services and the Analytics components are provided as WAR (Web Archive) files and deployed on Apache Tomcat (Section 6). The information regarding how to deploy a WAR file can be found at http://tomcat.apache.org/tomcat-7.0-doc/deployer-howto.html.

Detailed instructions on how to deploy a WAR file are available here:

- https://tomcat.apache.org/tomcat-6.0-doc/deployer-howto.html

7.2 SIMPLI-CITY Mobile App

Since the SIMPLI-CITY App is not yet available in the Google Play Store, the installation requires additional steps. To be able to install third party apps on an Android device one may needs to modify the configuration by enabling the "*Unknown Sources*" option (Figure 4).

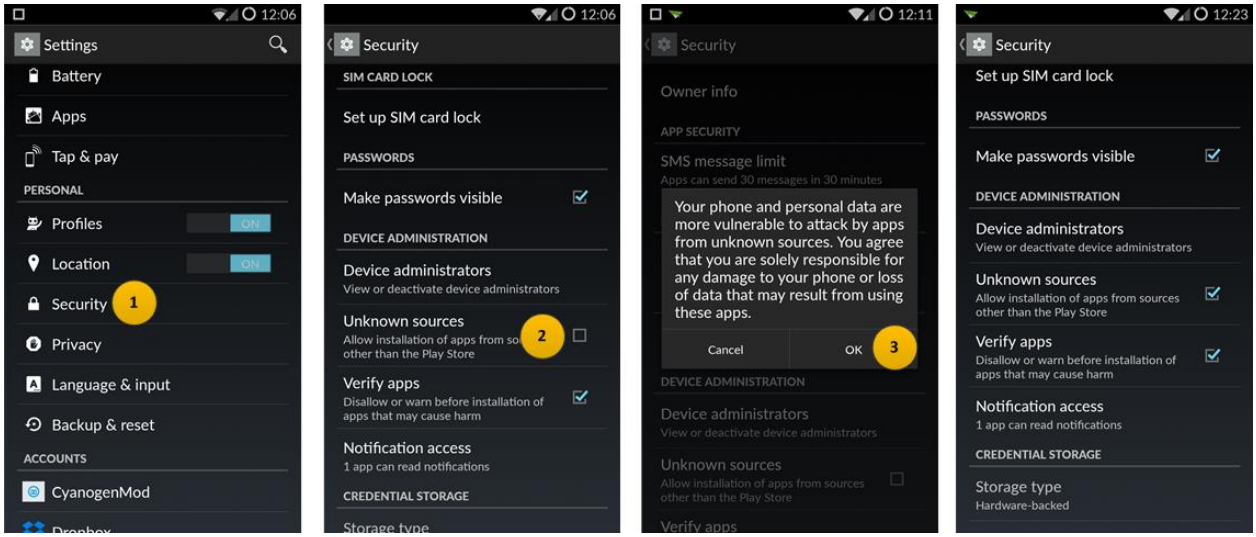

Figure 4: Enabling Unknown Sources

Once the "*Unknown Source"* options have been set, is it possible to proceed with the installation of the app by copying the APK to the Android device. Once the APK has been copied, double click on it to start the installation process. The installation steps can be seen in Figure 5.

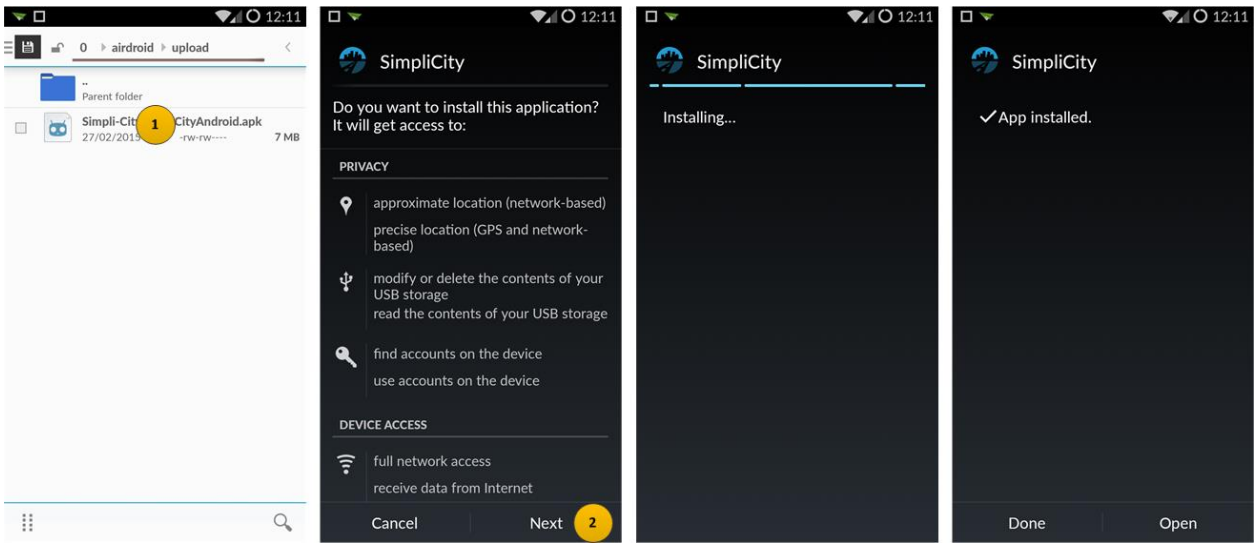

Figure 5: SIMPLI-CITY Mobile App Installation Process

8 Execution and Usage of the Software

8.1 Basic Usage of the SIMPLI-CITY Mobile App

8.1.1 How to Start the SIMPLI-CITY Mobile App

To start the SIMPLI-CITY Mobile App, tap the SIMPLI-CITY icon on the mobile phone (see (1) in the screenshot below).

The SIMPLI-CITY home screen will then be displayed:

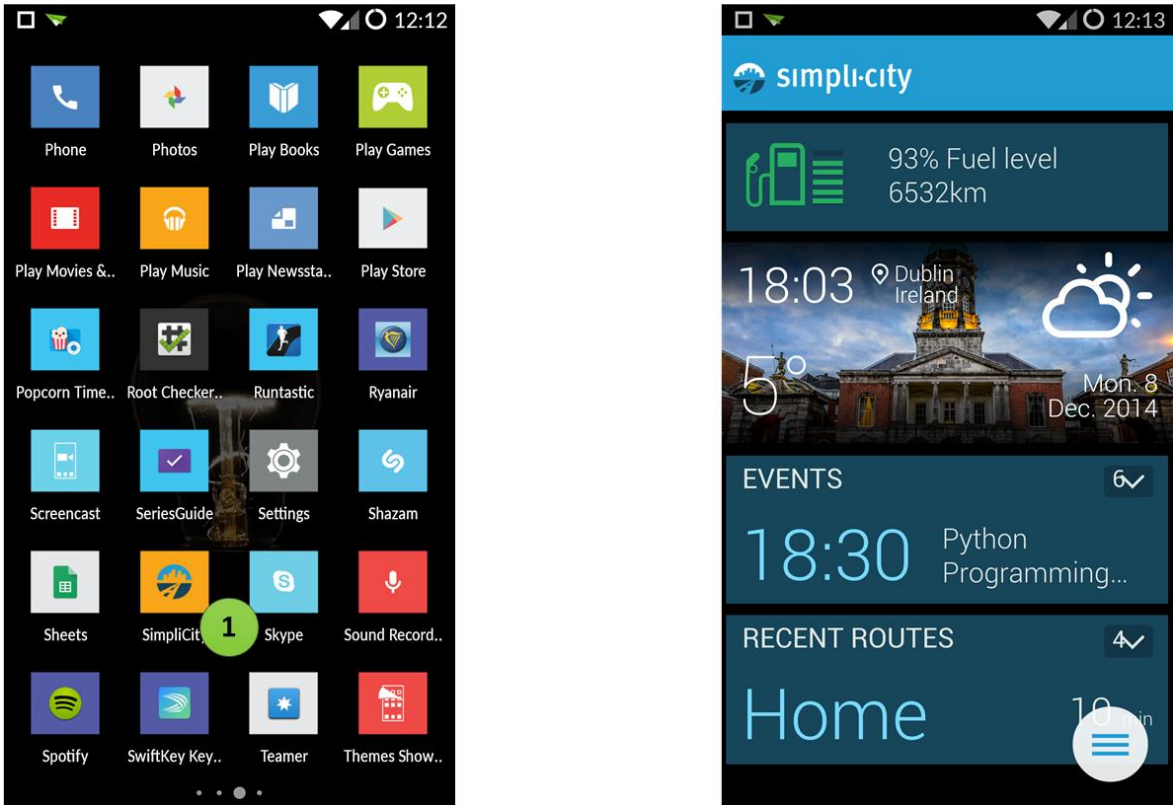

Figure 6: How to Start the SIMPLI-CITY Mobile App

As part of the initial setup of the SIMPLI-CITY app, you should first go to the SIMPLI-CITY settings screen and enter your mail credentials to allow the system to access your calendar. You can access the setting screen by clicking the menu icon (1), then clicking the settings icon (2) and insert your credentials (3):

Figure 7: SIMPLI-CITY Set Up

8.1.2 Selecting an Event to Attend

Click the “Events” area (1) and you will be presented with the list of all your events that are in your calendar (2) – you may have to scroll up and down to see the complete list:

Figure 8: Selecting an Event to Attend

Click on the event you want to attend, in this example we selected the “Python Programming” Event. Once you click this event you will be presented with the suggested route. In order to view all possible routes click the arrow icon beside the event (1) and then you can select your preferred route (2). Finally, click the “GO” button to start route guidance (3):

Figure 9: Route Selection

8.1.3 Approaching a Traffic Anomaly During Routing

If there is a traffic anomaly (e.g., an event that causes congestion) on your route, the SIMPLI-CITY app will inform you when you are close to the anomaly, along with the cause of the anomaly (in this example the music band “Kasabian” is playing a concert):

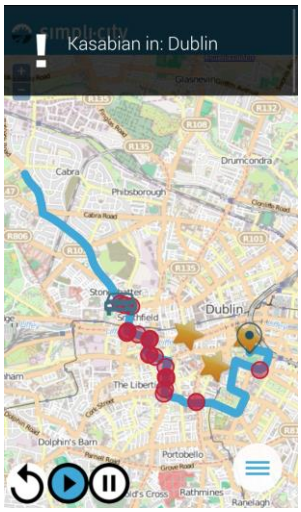

Figure 10: Traffic Anomaly

8.1.4 Viewing Car Sensor Information

At any time you can view the information being read by the SIMPLI-CITY mobile app from the car sensors. To do this, click the menu icon (1), then click the car icon (2) and you will be presented with the sensor information screen:

Figure 11: Viewing Car Sensor Information

8.1.5 Routing to an Event in a Restricted Area of the City

Occasionally, an event you wish to attend may be in a restricted area of the city (in our example this happens in the city of Bologna). If the event is in a restricted are, an icon and message informing you this will be displayed (1).

In order to go to an event in the restricted area you must pay for access to this area (2).

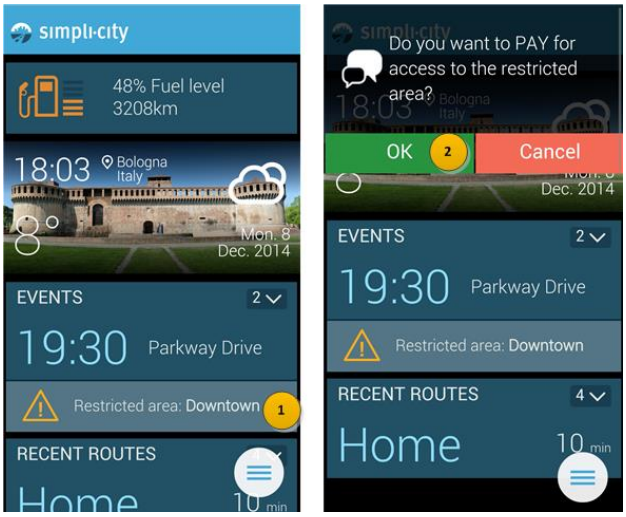

Figure 12: Routing to an Event in a Restricted Area of the City

NOTE: Your payment details are entered in the settings screen – see Section 8.1.1 on how to access the settings screen.

8.1.6 Low Fuel Scenario

If you are running on low fuel, the SIMPLI-CITY mobile app will inform you of this (1).

If this example, we route to the “Peter Pan” event (2) and during routing, the app informs us of the low fuel situation and asks if we want to re-route to a gas station (3). Once we select to re-route we then click “GO” (4) to start routing to the event via the gas station:

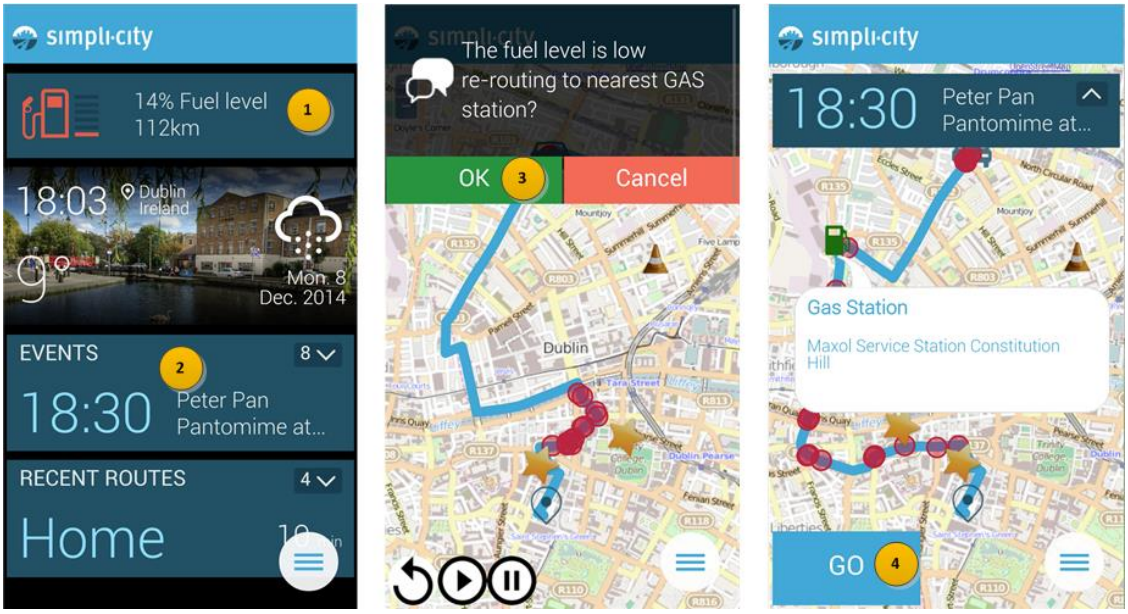

Figure 13: Low Fuel Scenario

8.1.7 Receiving an SMS while using the SIMPLI-CITY App

If you receive a text message (SMS) while using the SIMPLI-CITY app it will intercept this message and inform you that it has received a message (1). You have the choice to read the message if you want to (2).

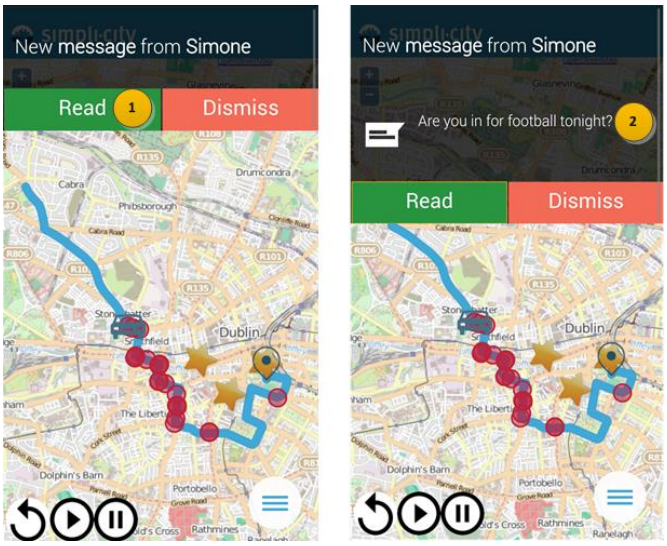

Figure 14: Receiving an SMS while Using the SIMPLI-CITY App

NOTE: It is not recommended to read messages while driving, however, in a future version of the app, it will be able to read the message out on your behalf.

8.2 Typical Usage Scenarios

There are a number of possible usage scenarios of the SIMPLI-CITY mobile app. This outlines the main usage scenarios in more detail from a technical point of view. Each Scenario is described in text and with a swim lane diagram. The test and diagram should be used in conjunction with each other for a full view of the scenario.

8.2.1 Scenario 1 – Navigation to an Event (in a Non-Restricted Area)

8.2.1.1 Description

This scenario, part of the WP7 use case, can be split into two “phases” as follows:

First Phase (new event)

1. A new event is entered into the user’s calendar.
2. The analytics detect the new event. The location of the event is sent to City Services (cf. Section 5.3) to check if it is in a restricted area of the city. In this case, it is not in a restricted area.
3. The analytics then store the details of the event, e.g., that it is not in a restricted area, to the server database and notifies the mobile app of the new event via GCM.
4. The mobile app then syncs with the server database in order to access all the details of the new event. Finally the mobile app alerts the user about the new event.

Second Phase (User decides to attend the event)

1. The user looks at the mobile app and decides to go to the location of the event. (We recommend that the user does not do this while driving)
2. Since the event is not in a restricted area of the city, the user just asks the mobile app to route him/her to the event.
3. The mobile app contacts the SIMPLI-CITY Geo-Services to ask for the routing information.
4. SIMPLI-CITY Geo-Services now gets the set of possible routes to the destination from Google Map APIs (a set of possible routes is needed in case there is an anomaly on the main route).
5. SIMPLI-CITY Geo-Services ask STAR-CITY if there are any current anomalies in the bounding-box that covers all the possible routes returned in the previous step
6. SIMPLI-CITY then picks the most relevant route for the user. If all routes have anomalies then the user must chose the preferred route.

8.2.1.2 Swim Lane Diagram

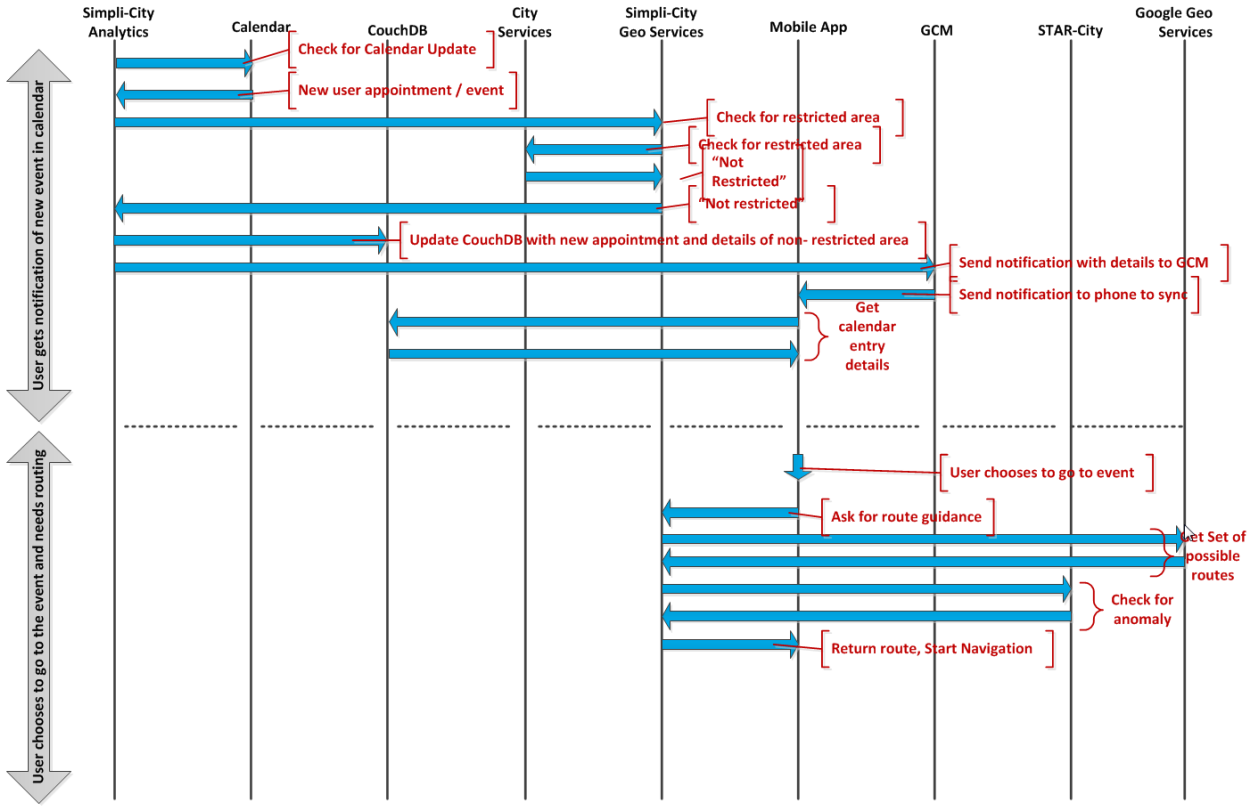

Figure 15: Swim Lane Diagram for navigation to an event in the city

8.2.2 Scenario 2 – Navigation to an Event in a Restricted Area of the City

8.2.2.1 Description

This use case can be split into two “phases” as follows:

First Phase (new event)

1. A new event is entered into the user’s calendar.
2. The analytics detect this new event. The location of this event is sent to City Services (Section 5.3) to check if it is in a restricted area of the city. In this case, it is in a restricted area.
3. The analytics then store the details of the event, e.g., that it is in a restricted area to the server database and it notifies the mobile app of the new event via GCM.
4. The mobile app then syncs with the server database in order to have all the details of the new event available. Finally the mobile app alerts the user that there is a new event.

Second Phase (User decides to attend the event)

1. The user looks at the mobile app and decides to go to the location of the event.
2. The event is in a restricted area, so the user decides to buy the relevant ticket from City Services. The fact that the user has purchased the ticket is synchronized to the server database by the mobile app.
3. The user then asks the mobile app for guidance.
4. The mobile app contacts the SIMPLI-CITY Geo-Services to request the routing information.
5. SIMPLI-CITY Geo-Services receive the set of possible routes to the destination from Google Map APIs. The set of possible routes is required in case there is an anomaly on the main route.
6. SIMPLI-CITY Geo-Services ask STAR-CITY if there are any current anomalies in the bounding-box that covers all the possible routes returned in the previous step
7. SIMPLI-CITY then picks the most relevant route for the user. If all routes have anomalies then the user must chose the preferred route.

8.2.2.2 Swim Lane Diagram

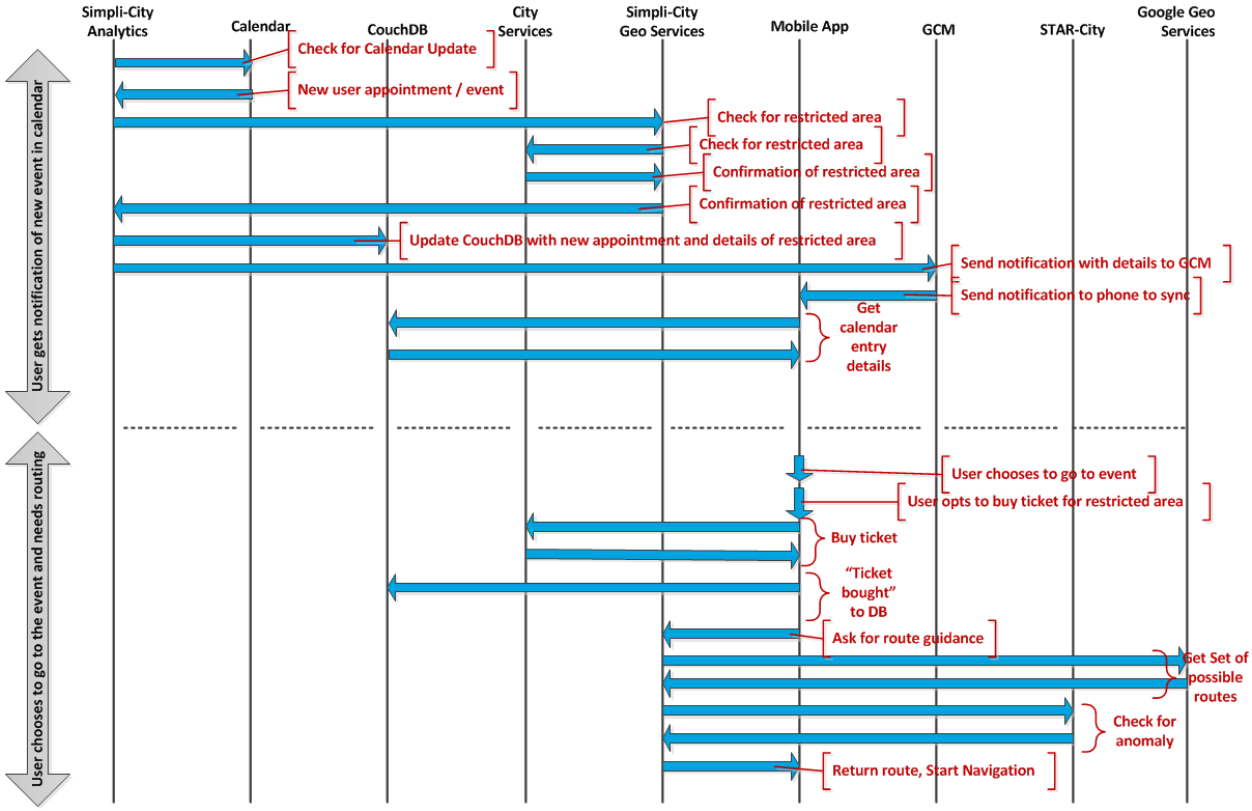

Figure 16: Swim Lane Diagram (Restricted Area)

8.2.3 Scenario 3 – User Gets SMS Message While Navigating to Event

8.2.3.1 Description

The use case is as follows:

1. The user selects an event to attend.
2. The mobile app asks the SIMPLI-CITY Geo-Services for route guidance.
3. During the navigation to the event, the user receives a text message (SMS).
4. The mobile app intercepts this message and displays a notification about the SMS to the user. It also gives the user the choice to either read the message or to dismiss the notification.
5. The user chooses to dismiss the notification and continues with the navigation to the event.

8.2.3.2 Swim Lane Diagram

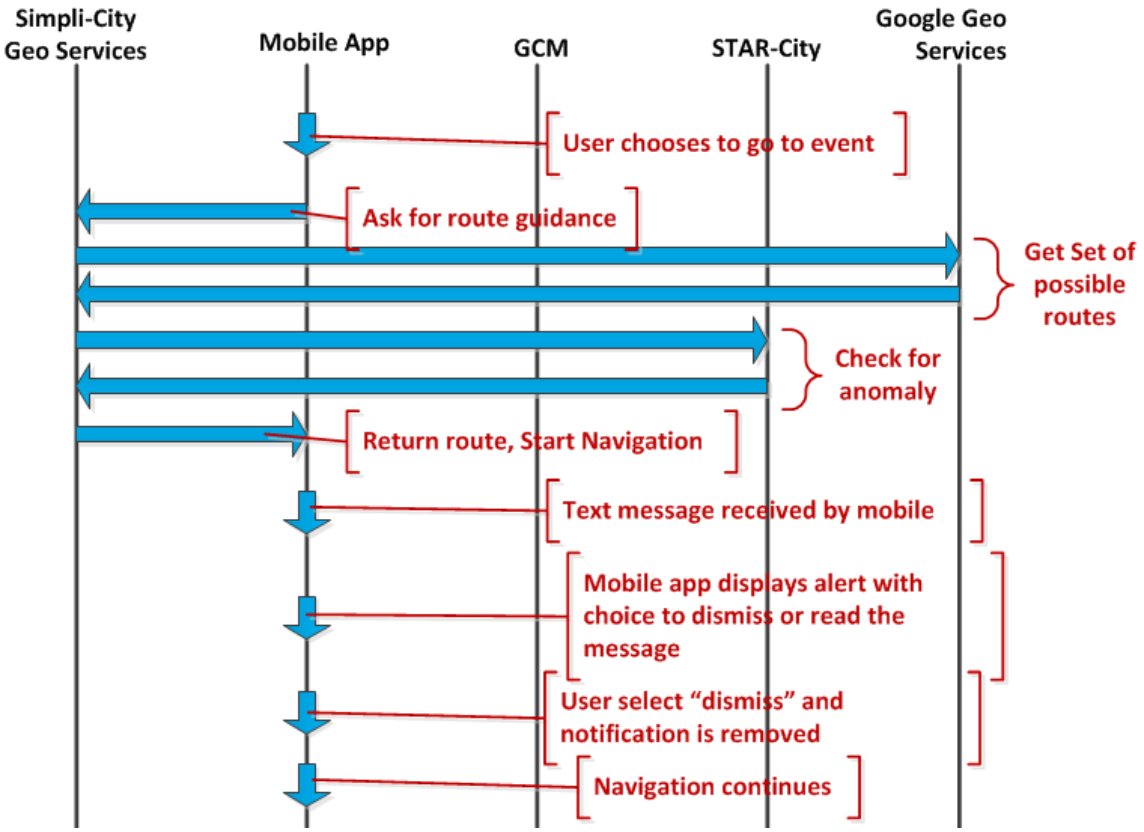

Figure 17: Swim Lane Diagram (User Gets SMS Message While Navigating to Event)

8.2.4 Scenario 4 – Low Fuel Level Detected While Navigating to Event

8.2.4.1 Description

1. Users selects event to attend.
2. The user then asks the mobile app for route guidance.
3. The mobile app contacts the SIMPLI-CITY Geo-Services to ask for the routing information.
4. SIMPLI-CITY Geo-Services receive the set of possible routes to the destination from Google Map APIs. The set of possible routes is required in case there is an anomaly on the main route.
5. SIMPLI-CITY Geo-Services ask STAR-CITY if there are any current anomalies in the bounding-box that covers all the possible routes returned in the previous step.
6. SIMPLI-CITY then picks the most relevant route for the user. If all routes have anomalies then the user must chose the preferred route.
7. Navigation starts.
8. The mobile app gets latest sensor readings from the car via Bluetooth (see architecture diagram).
   Note: The mobile app is constantly getting the sensor readings and updates them.

9. Mobile app syncs the latest readings to CouchDB.
10. The Analytics read from the CouchDB and determine that there is low fuel.
11. A notification with the low fuel details is sent to Google Cloud Messaging (GCM).
12. GCM pushes the message to the mobile device.
13. The mobile app gets the low fuel message and asks the user if they want to re-route via the nearest gas station.
14. User asks to be re-routed via the nearest gas station.
15. Mobile app sends the re-route request along with the current location to SIMPLI-CITY Geo-Services.
16. SIMPLI-CITY Geo-Services calculate the nearest gas station based on the user's current location.
    Note: The gas station locations are pre-loaded into the SIMPLI-CITY Geo Services.
17. SIMPLI-CITY Geo-Services ask the Google APIs for a new route from current location to the original destination via the gas station.
18. Google APIs return the new route to SIMPLI-CITY Geo Services.
19. The new route is sent back to the mobile app and the user is re-routed.

8.2.4.2 Swim Lane Diagram

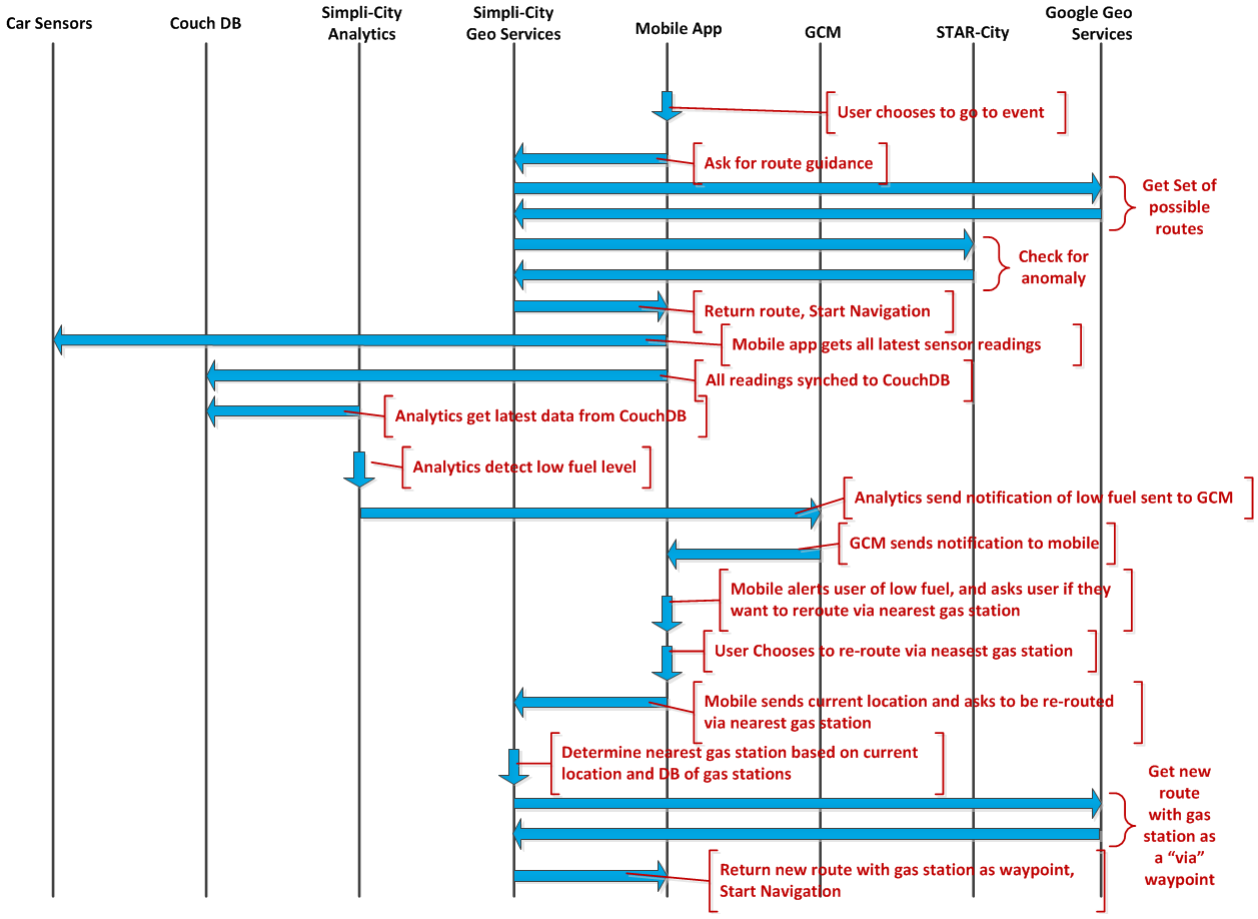

Figure 18: Swim Lane Diagram (Low Fuel Level Detected While Navigating to Event)

8.2.5 Scenario 5 – Event Cancelled While Driving

8.2.5.1 Description

1. User is already en-route to an event.
2. SIMPLI-CITY Analytics check the calendar for any updates.
3. The Calendar has as an update: the event has been cancelled.
4. SIMPLI-CITY Analytics store this new information in the CouchDB.
5. SIMPLI-CITY Analytics send a notification to GCM to tell the mobile app to sync with the Couch DB.
6. GCM pushes the notification to the phone.
7. The mobile app receives the notification and syncs with CouchDB.
8. The Mobile app detects that the current event has been cancelled. It notifies the user and cancels the navigation.

8.2.5.2 Swim Lane Diagram

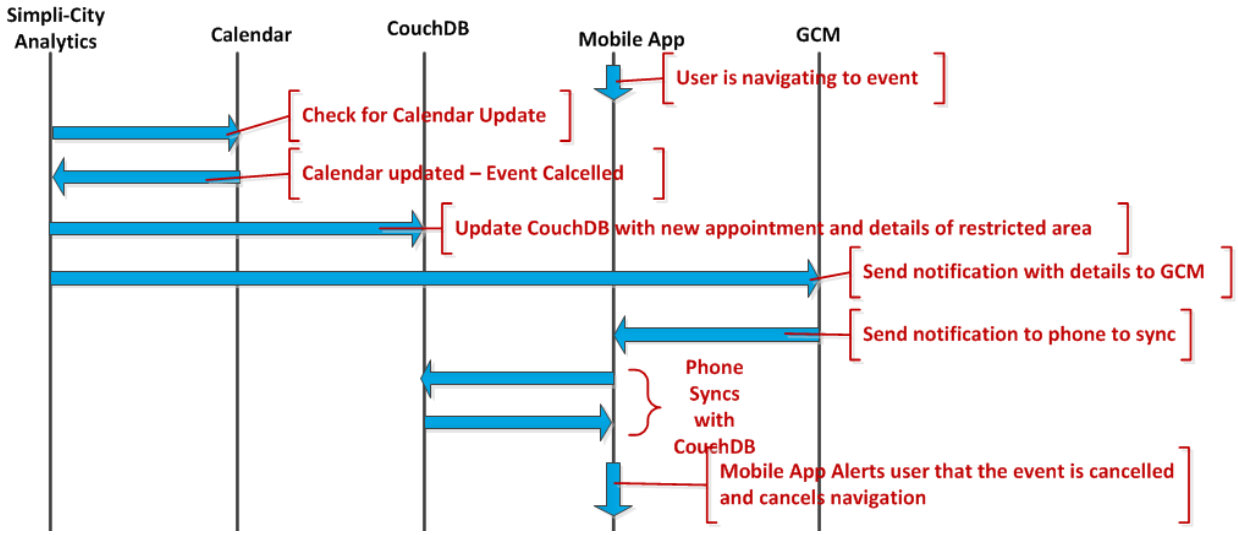

Figure 19: Swim Lane Diagram (Event Cancelled While Driving)

9 Summary

In this second prototype report for User-Centric and Open Data Management the scope of the prototype, the requirements coverage as well as the required preparations and deployment instructions to execute the prototype have been explained.

In particular it has been shown how this prototype has been tightly integrated with all relevant SIMPLI-CITY scenarios. In particular we focused on WP7 scenario as illustration in this deliverable, as it is the most representative scenario of SIMPLI-CITY covering both User-Centric and Open Data. This prototype strongly relies on foundation from WP4 for accessing Open Data, car sensor data and user related data. Most of APIs of the award winning system STAR-CITY (also part of WP4 as the data processing and reasoning component – described in D4.4.1) have been used to diagnose and predict traffic conditions in cities. Further it has been shown how SIMPLI-CITY leverages existing technologies form various open APIs, services and systems such as Google Maps, Google Calendar, Apache Tomcat, Apache CouchDB.

References

[ADS+07] H. Alani, D. Dupplaw, J. Sheridan, K. OHara, J. Darlington, N. Shadbolt, and C. Tullo, “Unlocking the potential of public sector information with semantic web technology”, in *9th Int. Semantic Web Conf.,* Busan, Korea, 2007, pp. 708-721.

[CHM11] M. J. Cafarella, A. Halevy, J. Madhavan, “Structured data on the web,” *Communications of the ACM*, vol. 54, no. 2, pp 72-79, 2011.

[DLE+11] L. Ding, T. Lebo, J. S. Erickson, D. DiFranzo, G. T. Williams, X. Li, J. Michaelis, A. Graves, J. G. Zheng, Z. Shangguan, J. Flores, D. L. McGuineness and J. A. Hendler, “TWC LOGD: A portal for linked open government data ecosystems,” *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 9, no. 3, pp. 325-333, 2011.

[MCP12] F. Maali, R. Cyganiak, V. Peristeras, “A publishing pipeline for linked government data,” in *9th Extended Semantic Web Conf.*, Heraklion, Greece, 2012, pp. 778-792.

[SAT+12] F. Scharffe, G. Atemezing, R. Troncy, F. Gandon, S. Villata, B. Bucher, F. Hamdi, L. Bihanic, G. Kepeklian, F. Cotton, J. Euzenat, Z. Fan, P-Y. Vandenbussche, and B. Vatant, “Enabling linked-data publication with the datalift platform”, in *AAAI Workshop on Semantic Cities*, Toronto, Canada, 2012.