simpli·city

The Road User Information System Of The Future

WP5 – Mobility Services Framework

D5.1.2: Service Development API Prototype II

Deliverable Lead: TIE

Contributing Partners: TIE, WORLD, TUV

Delivery Date: 10/2014

Dissemination Level: Public

Version 1.00

This deliverable describes the work carried out during the development of the final prototype of the Service Development API component of the SIMPLI-CITY platform. It specifies the scope of this final version and the degree of fulfilment of the requirements to be covered by the component. It specifies how to install and execute the different subcomponents implemented.

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

Document Status	
Deliverable Lead	Arturo Brotons, TIE
Internal Reviewer 1	Philipp Hoenisch, TUV
Internal Reviewer 2	Fredrik Kronlid, TALK
Type	Deliverable
Work Package	WP5: Mobility Services Framework
ID	D5.1.2: Service Development API Prototype II
Due Date	30.09.2014
Delivery Date	08.10.2014
Status	Approved

Document History	
Contributions	V0.01, Arturo Brotons, TIE, 25.08.2014 – Initial document V0.02, Alfonso Marín, WORLD, 27.08.2014 – Server side description V0.03, Arturo Brotons, TIE, 27.08.2014 – Corrected introduction and marked deprecated sections V0.04, Arturo Brotons, TIE, 28.08.2014 – Corrected installation process of plug-in V0.05, Arturo Brotons, TIE, 25.09.2014 – WORLD sections incorporated, corrected screenshots V0.06, Arturo Brotons, TIE, 29.09.2014 – Sections 5.1, 5.2 completed V0.07, Vadim Petrenko, TIE, 01.10.2014 – First version for internal reviewers V0.08, Arturo Brotons, TIE, 02.10.2014 – Second version for internal reviewers V0.09, Arturo Brotons, TIE, 03.10.2014 – Formatting correction and updated screenshot V1.00, Arturo Brotons, TIE, 06.10.2014 – Final version after second review
Final Version	October 8th, 2014

Disclaimer

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

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

Project Partners

Vienna University of Technology (Coordinator), Austria

Ascora GmbH, Germany

TIE Nederland B.V., The Netherlands

Technische Universität Darmstadt, Germany

IBM Research – Ireland
Smarter Cities Technology Centre

Forschungsgesellschaft Mobilität, Austria

Talkamatic AB, Sweden

Atos Worldline, Spain

Centro Ricerche FIAT, Italy

SRM – Reti e Mobilità, Italy

Executive Summary

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

The document starts by introducing the Service Development API components and describing the scope of the final prototype. Within the final prototype the RESTful interface (Service Management Client/Server) for managing services in the Service Runtime Environment (SRE) by service developers has been fully implemented. The complete version of the Service Development IDE (Integrated Development Environment) plugin used for service development is delivered and the demonstration of integration of the plugin within the chosen IntelliJ IDEA IDE is included.

Next, the degree of fulfilment of each requirement to be covered by the components of the deliverable as specified in the Requirements Analysis Report (D2.3) is described.

The following sections of the document describe how potential users (i.e. service developers in the case of the plugin, but also SIMPLI-CITY administrators in the case of the server side subcomponents) can prepare, install and execute different parts of the Service Development API, detailing the different tools and development frameworks needed, and providing a step-by-step description of the process to install and use the prototype. The server side part of the Service Development API component is aimed at administrators of the SIMPLI-CITY Mobility Services Framework, whereas the plugin will be used by service developers.

This deliverable D5.1.2 supersedes the deliverable D5.1.1, which was the first prototype of the Service Development API and was internally released in project month 15 with the dissemination level “Restricted to Other Programme Participants (including the Commission Services)”.

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 Service Development API 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 explain how to use the final prototype of the Service Development API and exploit its functionalities. For this, the scope and requirements of the Service Development API and this final prototype, the requirements and preparations for users and developers, an installation and usage guide are provided.

The final Service Development API prototype is the outcome of the discussions and implementation work done in project months 16 to 24, being the continuation of the work done in the first prototype, during the months 9 to 15. It provides the final implementation of the functionalities of the Service Development API as defined within SIMPLI-CITY deliverables D3.2.1 (Functional Specification), and D3.2.2 (Technical Specification). This deliverable D5.1.2 supersedes the deliverable D5.1.1, which was a preliminary prototype of the Service Development API.

1.3 Document Status and Target Audience

This document is listed in the Description of Work (DoW) as “public”, since it provides the final prototype of SIMPLI-CITY task T5.1 and is not planned to be changed.

While the document primarily aims to be used by the project partners, this public deliverable can also be helpful 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 were the Service Development API fits into the overall SIMPLI-CITY software framework and the outcome of the final prototype. 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 to start making use of the Service Development APIs prototype.

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

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

Section 6 discusses the limitations of the final prototype of the Service Development API.

Finally, Section 7 provides a summary of the document.

2 Prototype Scope and Requirements Coverage

2.1 Service Development API – General Information

The Service Development API is a component aiming to be used by third party developers and allowing them to create and configure their own services for the SIMPLI-CITY Mobility Services Framework. These services will run in the SIMPLI-CITY Service Runtime Environment (SRE) and will be used by apps running in the SIMPLI-CITY Personal Mobility Assistant (PMA).

The Service Development API covers the full lifecycle of creation and management of services. It provides the means to easily define, design, develop and deploy services, and assists developers during the whole process, supplying templates and documentation. The templates (i.e., sample classes and the initial project structure) allow for easy starting with the service development. The Service Development API additionally supports developers in building connectors to SIMPLI-CITY external, backend and data services by providing insight into services and their methods available in the SRE. The possibility to bundle the developed services, upload, verify and register them with the Service Registry for future invocations is a part of the functionality of the Service Development API.

Figure 1 shows the location of the Service Development API in the SIMPLI-CITY Global Architecture. For the full Global Architecture, refer to deliverable D3.1.

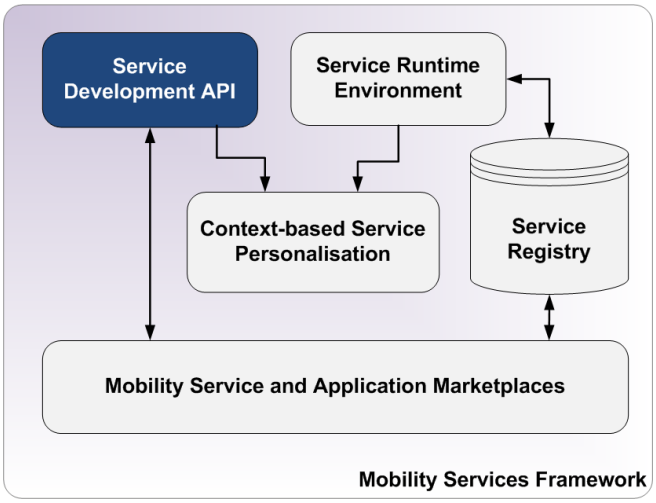

Figure 1: Location of the Service Development API in the SIMPLI-CITY Global Architecture

2.2 Scope of the Final Prototype

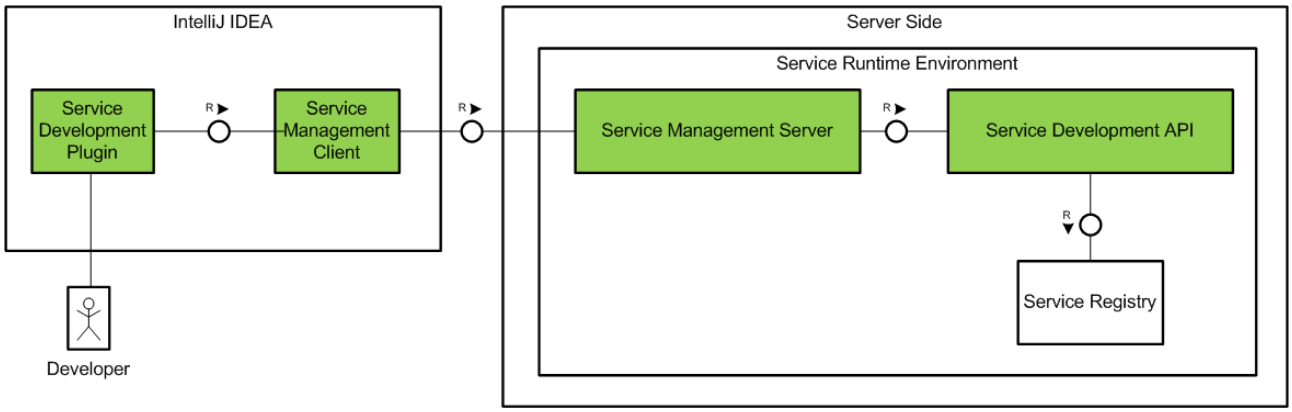

Figure 2: Scope of the Final Prototype of the Service Development API

Figure 2 depicts the status of development of the final prototype of the Service Development API, showing the subcomponents that are covered within this prototype. Note that this figure shows the Service Registry which is not developed within the Service Development API and is depicted only to show interconnection between components.

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

- Green: Fully implemented.
- Orange: Partially implemented.
- White: No implementation so far, 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 Service Management Server

The Service Management Server provides a RESTful interface to be used by the Service Development Plugin for service bundle persistence and management. Within the final prototype, a complete set of functionalities have been developed, including CRUD (Create, Read, Update and Delete) operations on services in the Service Registry.

2.2.2 Service Management Client

The Service Management Client is a subcomponent of the Service Development Plugin that connects to the RESTful interface of the Service Management Server. This subcomponent is provided as a jar library which is already integrated within the Service Development Plugin and which permits the communication with the Service Management Server component. Similarly to its server counterpart, it implements client side CRUD operations for services and allows the retrieval of information about other services registered in the Service Registry so that developers can make use of them.

2.2.3 Service Development API

This subcomponent exposes functionalities and auxiliary classes to be used during service development and deployment time.

The presented integration of the Service Development API with the Service Registry allows for bundle validation, i.e. verifying that the service being deployed fulfils the set of integrity and format constraints in order to become eligible for SRE deployment.

2.2.4 Service Development API Plugin

The Service Development API Plugin provides service developers with a set of functionalities needed for comfortable development and deployment of SIMPLI-CITY services. It seamlessly integrates with the IntelliJ IDEA IDE and presents a form of interaction with the developer via custom designed screens. The plugin allows creating a new service project in the IDE with the recommended structure already containing files necessary to start developing. Bundling and deploying of services (including the service manifest editor) is another part of the plugin together with a set of tutorials, guides, best practices and how-tos and the ability to explore remote SRE services for consumption forming a complete ecosystem for the SIMPLI-CITY service developers.

2.3 Covered Requirements

This section describes the degree of fulfilment of the requirements covered by the Service Development API and specified in the Requirements Analysis Deliverable (D2.3).

Table 1: Requirements Related to the Service Development API and their Degree of Fulfilment

Requirement	Degree of Fulfilment	Comment
Must Have Requirements		
U106: Access to cloud services (P1)	100%	Access to Cloud Services is done by standard calls to the RESTful API of the Cloud Services component.
U148: Service Registration (P1)	100%	The Service Development API Plugin is able to upload and register service bundles with the Service Registry. An additional validation of the bundle integrity is performed by the Service Development API on the server side.
U149: Service extension and modification (P1) U150: Service management (P1)	100%	The IDE Plugin allows for the full development cycle of services, editing and modification. Service management functionality (deployment, replacement, and deletion of services in the SRE) is fully implemented.
U152: SLA support (P1)	100%	The manifest editor of the IDE Plugin fully supports editing of the Service Level Agreement (SLA) part of the manifest.
U161: Support for data services (P1) U162: Support for backend services (P2)	100%	The IDE Plugin fully supports exploration of backend services located at the SRE, including service method peek (and also data services, represented by backend service proxies).

Requirement	Degree of Fulfilment	Comment
U168: Provision of source code examples (P1) U169: Provision of UI templates (P1) U170: Provision of best practices (P1) U171: Provision of tutorials (P1) U172: Provision of guidelines (P1) U173: Provision of examples (P1)	100%	The IDE Plugin contains the required set of documentation, ranging from how to start developing a service tutorial to the best practices and guidelines. Upon a new project creation, the Plugin adds example service classes to the project. UI templates are a part of the T6.4 Application Design Studio as the services have no user interface.
U179: Service Metadata (P1)	100%	The IDE Plugin contains a service manifest editor, allowing service metadata editing.
Should Have Requirements		
U59: User centric data services (P1)	100%	Access to User Centric data is performed by standard calls to the RESTful API of the User Centric Data component.
U114: Configuration of the frequency of update of the data from data sources (P1)	100%	Configuration is done by the proxy used to connect to a backend service, which is generated using the IDE Plugin.
U116: Unified data model (P1)	100%	Functionality covered by this prototype as a unified service model has been developed which is used across the SIMPLI-CITY platform.
U151: Service versioning (P1)	100%	Service versioning is supported within the service manifest model, and also the uploading of new versions of a service using the RESTful interface.
U163: Easy to use environment (P2) U164: Low investment required (P3)	100%	The IDE Plugin demonstrates the convenience and elegance, while providing perfect integration with the IDE allowing the developer to work in a familiar and comfortable environment.
U166: Identification of the developer / signature (P1)	0%	Functionality not covered by this prototype as it was decided it adds no significant value to the project.

Requirement	Degree of Fulfilment	Comment
Could Have Requirements		
U165: User-friendly developer studio interface (P2)	100%	User friendliness is covered by selecting IntelliJ IDEA IDE for the development of the Plugin.
U180: Easy debugging of services (P3) U183: Provision of bug reports (P4)	0%	Functionality not covered by this prototype.

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 server side part of the Service Development API component will be executed by administrators of the SIMPLI-CITY Mobility Services Framework, whereas the plugin will be used by service developers.

3.1 Server Side (System Administrators)

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

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

The first prototype was tested on a Windows machine running Windows 7 64 bits, Java(TM) SE Runtime Environment (build 1.7.0_51-b13) and Apache Maven 3.0.5.

In addition, it is required that the environment variable JAVA_HOME is defined as per Oracle guidelines[1], as well as the variable M2_HOME pointing to the Maven 3.0.5 installation folder.

3.2 SIMPLI-CITY Service Development Plugin (Service Developers)

In order to deploy the Service Development plugin for service developers, it is necessary to have IntelliJ IDEA Community Edition 13.0.1 or newer installed on the user's computer. It can be downloaded from:

- http://www.jetbrains.com/idea/download

Detailed instructions on downloading and installing IntelliJ IDEA can be found here:

- http://confluence.jetbrains.com/display/IntelliJIDEA/Basics+and+Installation

The SIMPLI-CITY Service Development IDE Plugin was developed and tested against versions 13.0.1 and above of the IDE.

The plugin was developed to be used with the IntelliJ IDEA IDE only. It will not work with other IDEs like Eclipse or NetBeans (the choice of the target IDE has been made in the corresponding section of the Technical Specification D3.2.2).

The plugin is written in the Java language and is cross-platform. It is a part of the IntelliJ IDEA and is subject to the same platform requirements as the host IDE. Please refer to the following page for a detailed list of system requirements:

- http://www.jetbrains.com/idea/download

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

Further, in this document it is presumed that IntelliJ IDEA 13.0.1 Community edition is successfully installed and running on the developer's computer.

4 Installation

This section provides guidelines on how to install and deploy the final prototype of the Service Development API.

4.1 Server Side (System Administrators)

To use the Service Development API, the SRE platform is required and is supplied together with the necessary preinstalled services as a part of this deliverable.

This section provides guidelines on how to install and run the Service Runtime Environment with the services required.

Although a batch script is supplied for Windows (as well as its counterpart for Linux), a manual installation is described in the following section:

1. Unpack the provided archive file (service-runtime-environment.zip) and navigate to the resulting folder (referred to as SRE_ROOT in the document), where the following directory structure should appear: bin/, data/, deploy/, etc/, instances/, lib/, lock/, log/ and system/.
2. Navigate into the “bin” folder.
3. Type into the terminal: “karaf”. This will start the Service Runtime Environment. If the output is similar to Figure 3, the Service Runtime Environment should be up and running.

Figure 3: Server Side Installation Using Karaf

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

4. The following bundles (provided under the bundle folder) have to be copied to the Service Runtime Environment deployment folder ([SRE_ROOT]/deploy) for enabling the API:

- eu.simpli-city.api.interfaces-0.0.1-SNAPSHOT.jar
- eu.simpli-city.api.implementation-0.0.1-SNAPSHOT.jar
- eu.simpli-city.api.rest-0.0.1-SNAPSHOT.jar

5. Finally, to check if the bundles were successfully installed in the OSGi container, the “list” command needs to be executed in the console. It needs to be checked that the state of all these bundles is active (see Figure 4). If not, it is necessary to wait for a while and check the state again with the “list” command.

```
karaf@root>list
START LEVEL 100 , List Threshold: 50
 ID | State  | Lvl | Version        | Name
-------------------------------------------------------------------------------------------------------------
 71 | Active |  50 | 1.1.1          | geronimo-annotation_1.0_spec
 72 | Active |  50 | 1.1.1          | geronimo-jta_1.1_spec
 73 | Active |  50 | 1.1.1          | geronimo-jms_1.1_spec
 74 | Active |  50 | 1.0.1          | geronimo-j2ee-management_1.1_spec
 75 | Active |  50 | 1.6.0          | Commons Pool
 76 | Active |  50 | 3.2.1          | Commons Collections
 77 | Active |  50 | 2.6            | Commons Lang
 78 | Active |  50 | 1.4            | Commons Codec
 79 | Active |  50 | 1.7.0.4        | Apache ServiceMix :: Bundles :: velocity
 80 | Active |  50 | 2.0.8.3        | Apache ServiceMix Bundles: oro-2.0.8
 81 | Active |  50 | 1.9.0.1        | Apache ServiceMix :: Bundles :: jasypt
 82 | Active |  50 | 1.9.0          | Apache ServiceMix :: Specs :: Stax API 1.0
 83 | Active |  50 | 1.1.0.4c_5     | Apache ServiceMix :: Bundles :: xpp3
 84 | Active |  50 | 1.6.2          | Joda-Time
 85 | Active |  50 | 1.1.0.4        | Apache ServiceMix :: Bundles :: jdom
 86 | Active |  50 | 1.6.1.2        | Apache ServiceMix Bundles: dom4j-1.6.1
 87 | Active |  50 | 1.3.0.3        | Apache ServiceMix Bundles: xstream-1.3
 88 | Active |  50 | 0.3.0          | Apache Aries Transaction Manager
 89 | Active |  50 | 5.7.0          | activemq-core
 90 | Active |  50 | 5.7.0          | kahadb
 91 | Active |  50 | 5.7.0          | activemq-console
 92 | Active |  50 | 5.7.0          | activemq-ra
 93 | Active |  50 | 5.7.0          | activemq-pool
 94 | Active |  50 | 5.7.0          | activemq-karaf
122 | Active |  80 | 1.0.0.SNAPSHOT | SIMPLI-CITY SRE :: Example Backend Services :: Backend Service Activator
204 | Active |  50 | 2.7.6          | Apache CXF Compatibility Bundle Jar
205 | Active |  80 | 1.0.0.SNAPSHOT | SIMPLI-CITY SRE :: Example Backend Services :: POI-Service
206 | Active |  80 | 0.0.1.SNAPSHOT | SIMPLI-CITY API :: Interfaces
207 | Active |  80 | 0.0.1.SNAPSHOT | SIMPLI-CITY API :: Implementation
208 | Active |  80 | 0.0.1.SNAPSHOT | SIMPLI-CITY API :: Service Management
karaf@root>
```

Figure 4: List of Bundles in Karaf

4.2 SIMPLI-CITY Service Development IDE Plugin (Service Developers)

The SIMPLI-CITY Service Development IDE Plugin is supplied in a form of a zip archive: *simpli-city-services-plugin.zip*. Before deploying the plugin, IntelliJ IDEA needs to be shut down, in case it is already running. Then the deliverable zip file needs to be unzipped directly into the *plugins* folder of the IntelliJ IDEA installation folder.

After unzipping, the resulting folder structure should look like (see Figure 5):

- [IntelliJ IDEA]/plugins/simplicity-service-development-plugin/
- [IntelliJ IDEA]/plugins/eu.simplicity.plugin.base.jar
- [IntelliJ IDEA]/plugins/simplicity-help.jar

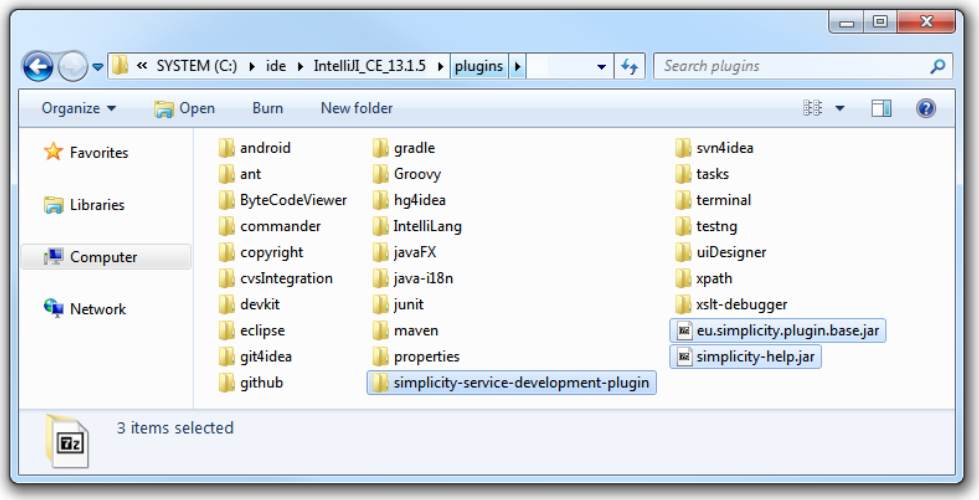

Figure 5: Extracted Files in the Plugins Folder

In the *simpli-city-services-plugin* folder, there should be exactly one folder called *lib*. It contains the Service Development API Plugin jar file, the Service Management Client jar file and auxiliary libraries needed.

After the plugin is unzipped to the right location, the developer can start using it straight away (see Section 5.2 on executing and usage).

5 Execution and Usage of the Software

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

5.1 Server Side (System Administrators)

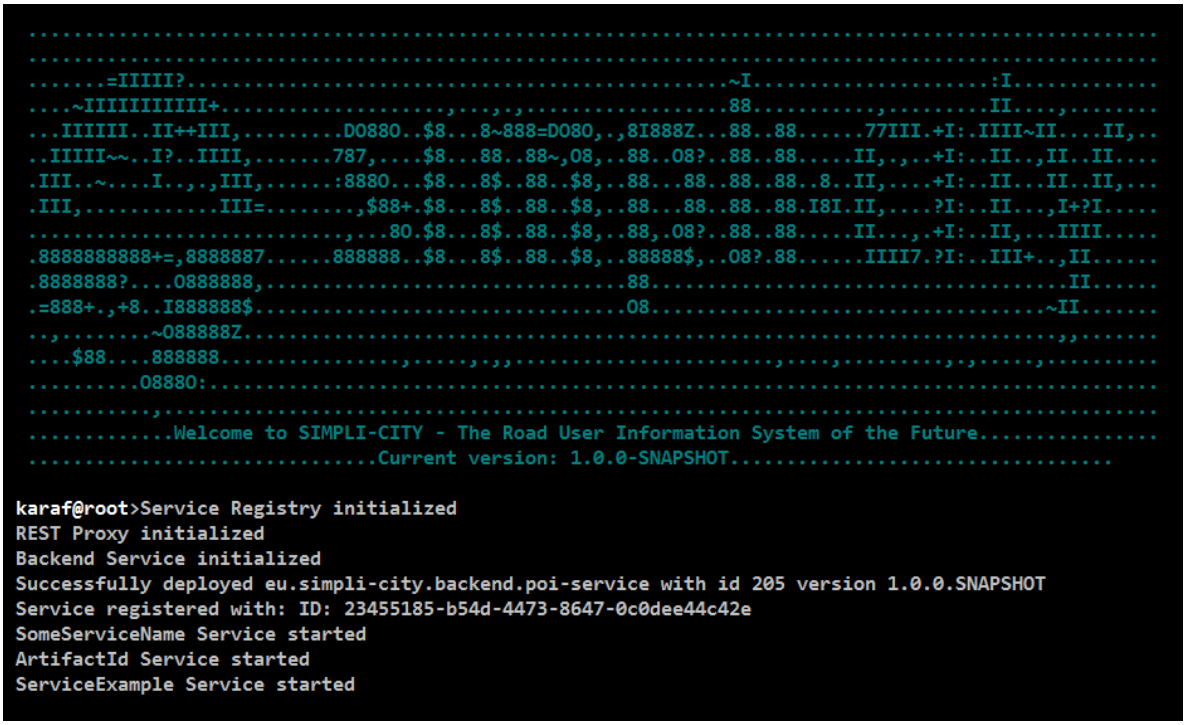

Figure 6: Service Runtime Environment with Startup Logs

In order to execute the Server Side of the Service Development API, the Service Runtime Environment must be started, by executing the following file:

- [SRE_ROOT]/bin/karaf.bat

Once the Service Runtime Environment is running, a new window appears containing all the generated start up logs. To check if it is correctly running, this URL can be opened on any web browser, where an example of JSON message should be displayed.

- http://localhost:8080/cxf/serviceManagement/test

Figure 7: Example Message to Check if Server Side of Service Development API is Running

5.2 SIMPLI-CITY Service Development IDE Plugin (Service Developers)

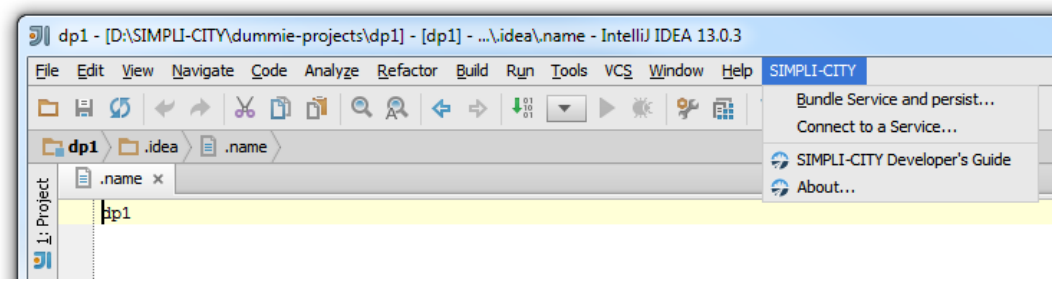

Figure 8: Checking if the Service Development Plugin is Correctly Installed

It is assumed that the user of the plugin is already familiar with the IntelliJ IDEA IDE. To access the plugin, IntelliJ IDEA has to be started and running. Once all the elements are loaded, the plugin menu entry “SIMPLI-CITY” can be found at the top right of the main menu (see Figure 8).

In order to start developing a new SIMPLI-CITY Service, the Developer needs to make use of the New Project Wizard provided by the Service Development IDE Plugin. If there is any project opened already, it can be found under File > New Project… on the main menu (see Figure 9).

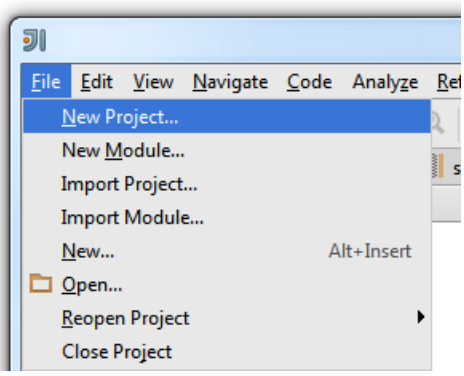

Figure 9: First Step to Create a New SIMPLI-CITY Service

The Developer can select the type of service to be developed (see Figure 10), being the available ones: Backend Service, External Service and Data Service. More details on the differences between the various types of services can be found at SIMPLI-CITY deliverables D3.2.1 (Functional Specification), and D3.2.2 (Technical Specification).

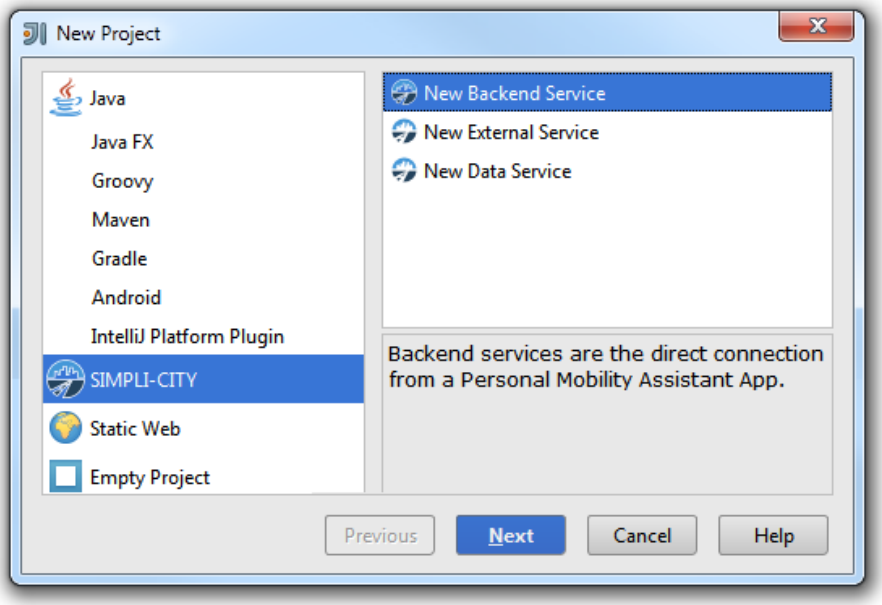

Figure 10: Types of SIMPLI-CITY Services

In order to keep the development process simple and facilitate the integration between components, SIMPLI-CITY Services projects are based on the build tool Maven. Hence, to continue, the Developer needs to choose a Group ID, an Artefact ID and the version of the project. More information about Maven naming conventions can be found in:

- http://maven.apache.org/guides/mini/guide-naming-conventions.html

The following pieces of information can be used as an example Service (as depicted in Figure 11):

- GroupId: eu.simplicity.examples
- ArtifactId: some-simplicity-service
- Version: 0.0.1

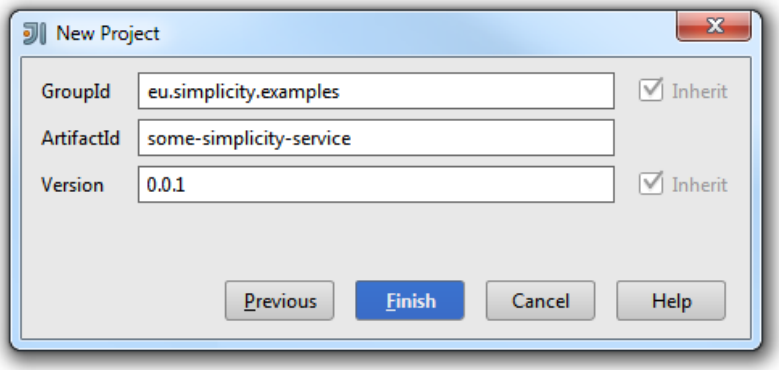

Figure 11: Example Information for a Maven Artefact

In order to finish the process of creating a new SIMPLI-CITY Service, the Developer needs to choose a location in the local storage where the files that comprise the new project will be stored (see Figure 12).

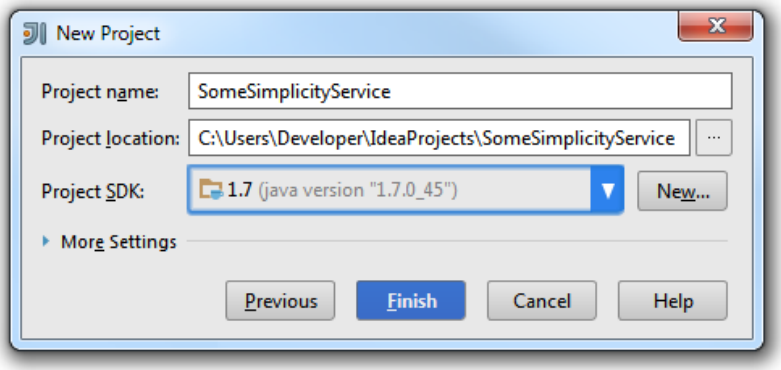

Figure 12: Project Name, Location and SDK Selection

The result of this process is the minimal structure of files, including Java code and configuration files (which can also be used as code examples), needed to generate a SIMPLI-CITY Service that can be deployed on the Service Runtime Environment (see Figure 13).

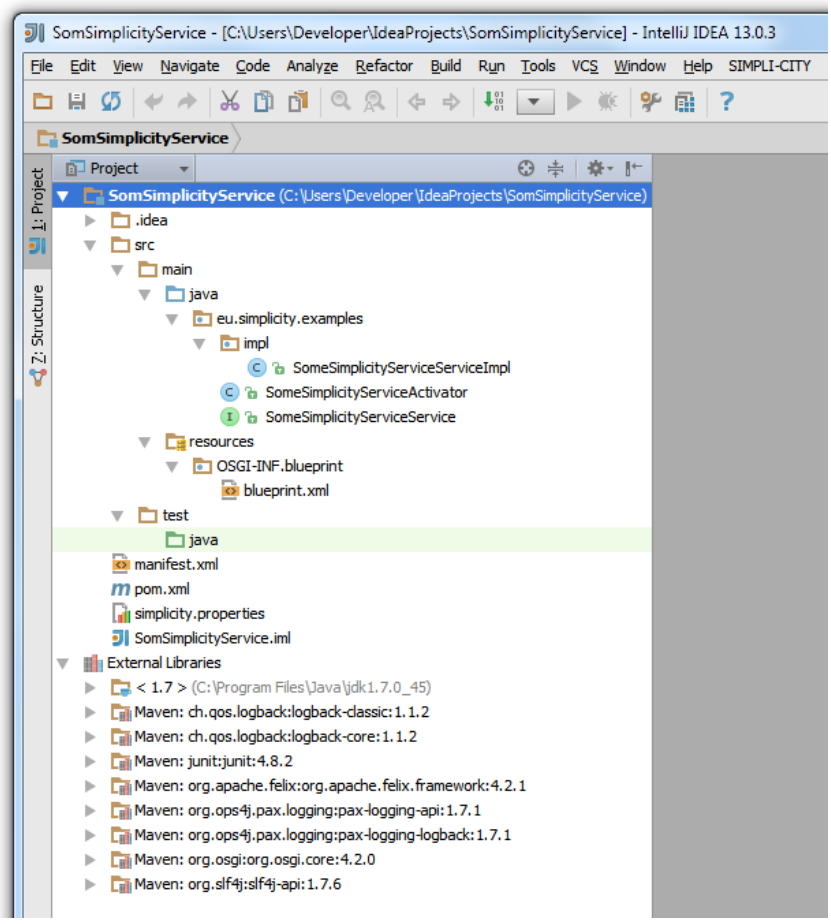

Figure 13: Project Structure of a New SIMPLI-CITY Service Project

At this point, the developer is able to generate a jar bundle that will contain the binaries of the Service, making use of the lateral panel of Maven and running the option [Project name] > Lifecycle > package (see Figure 14 and Figure 15).

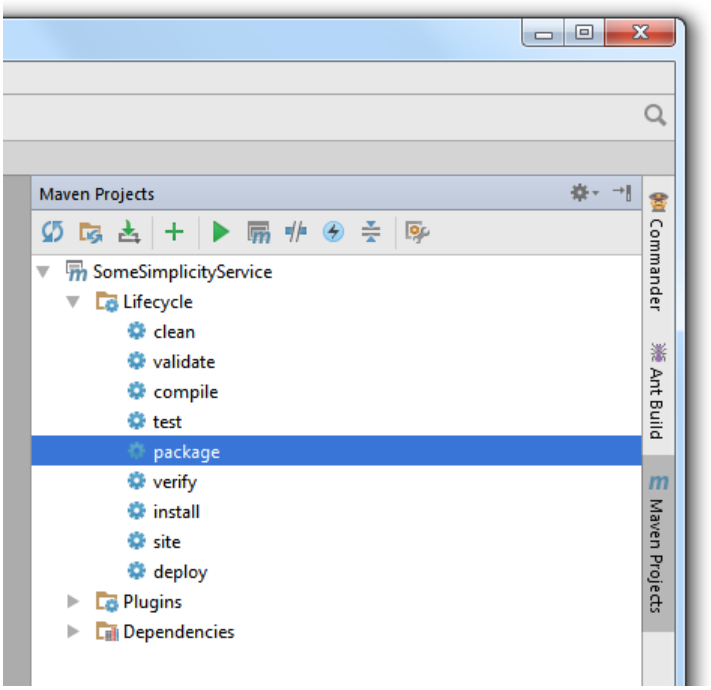

Figure 14: Generation of jar Service Bundle

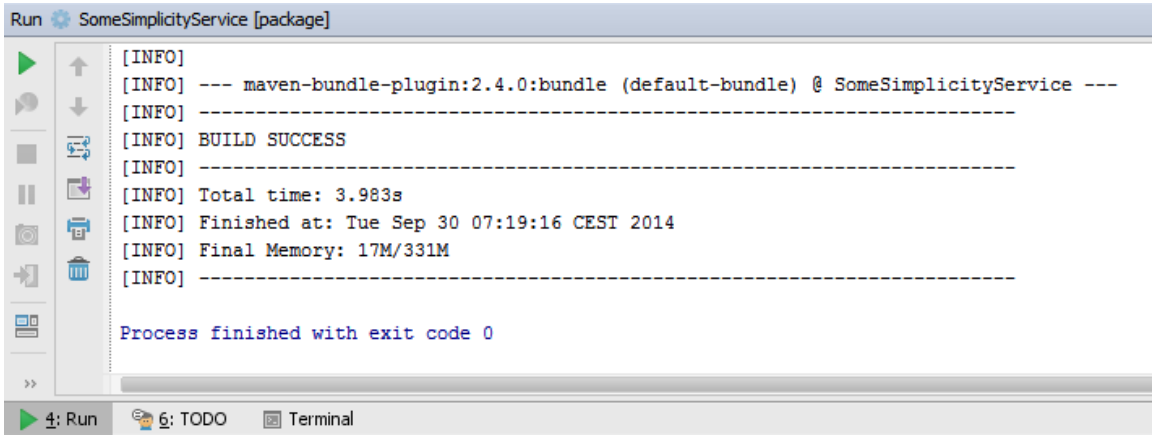

Figure 15: Result of Generating jar Service Bundle Using Maven

Once the bundle jar is generated, the developer can install it in the Service Runtime Environment using the option SIMPLI-CITY > Bundle service and persist… (see Figure 8). In the persist dialog, the jar and manifest files will be selected automatically, but in case the developer needs to use another version, other versions can be selected too (see Figure 16). When the bundle is successfully installed in the Service Runtime Environment, a service ID is assigned in UUID format and shown to the developer. This is used to identify the service during its invocations. In addition, a bundle ID is provided, which is the internal number of service used by Karaf (see Figure 17 and Figure 18).

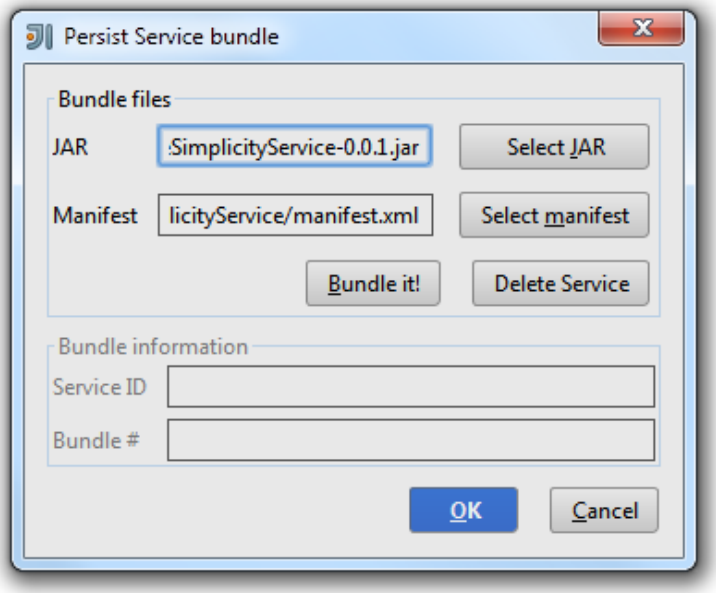

Figure 16: Before Installing a Service Bundle

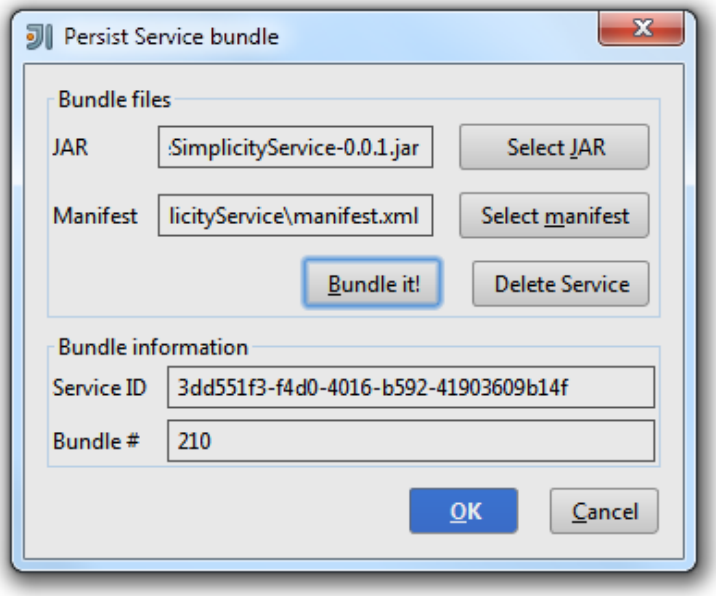

Figure 17: After Installing a Service Bundle (Client Side)

```
karaf@root>SomeSimplicityService Service started
Service ID: 3dd551f3-f4d0-4016-b592-41903609b14f
list
START LEVEL 100 , List Threshold: 50
 ID | State  | Lvl | Version        | Name
-------------------------------------------------------------------------------------------------------------
 71 | Active |  50 | 1.1.1          | geronimo-annotation_1.0_spec
 72 | Active |  50 | 1.1.1          | geronimo-jta_1.1_spec
 73 | Active |  50 | 1.1.1          | geronimo-jms_1.1_spec
 74 | Active |  50 | 1.0.1          | geronimo-j2ee-management_1.1_spec
 75 | Active |  50 | 1.6.0          | Commons Pool
 76 | Active |  50 | 3.2.1          | Commons Collections
 77 | Active |  50 | 2.6            | Commons Lang
 78 | Active |  50 | 1.4            | Commons Codec
 79 | Active |  50 | 1.7.0.4        | Apache ServiceMix :: Bundles :: velocity
 80 | Active |  50 | 2.0.8.3        | Apache ServiceMix Bundles: oro-2.0.8
 81 | Active |  50 | 1.9.0.1        | Apache ServiceMix :: Bundles :: jasypt
 82 | Active |  50 | 1.9.0          | Apache ServiceMix :: Specs :: Stax API 1.0
 83 | Active |  50 | 1.1.0.4c_5     | Apache ServiceMix :: Bundles :: xpp3
 84 | Active |  50 | 1.6.2          | Joda-Time
 85 | Active |  50 | 1.1.0.4        | Apache ServiceMix :: Bundles :: jdom
 86 | Active |  50 | 1.6.1.2        | Apache ServiceMix Bundles: dom4j-1.6.1
 87 | Active |  50 | 1.3.0.3        | Apache ServiceMix Bundles: xstream-1.3
 88 | Active |  50 | 0.3.0          | Apache Aries Transaction Manager
 89 | Active |  50 | 5.7.0          | activemq-core
 90 | Active |  50 | 5.7.0          | kahadb
 91 | Active |  50 | 5.7.0          | activemq-console
 92 | Active |  50 | 5.7.0          | activemq-ra
 93 | Active |  50 | 5.7.0          | activemq-pool
 94 | Active |  50 | 5.7.0          | activemq-karaf
122 | Active |  80 | 1.0.0.SNAPSHOT | SIMPLI-CITY SRE :: Example Backend Services :: Backend Service Activator
204 | Active |  50 | 2.7.6          | Apache CXF Compatibility Bundle Jar
205 | Active |  80 | 1.0.0.SNAPSHOT | SIMPLI-CITY SRE :: Example Backend Services :: POI-Service
206 | Active |  80 | 0.0.1.SNAPSHOT | SIMPLI-CITY API :: Implementation
207 | Active |  80 | 0.0.1.SNAPSHOT | SIMPLI-CITY API :: Service Management
208 | Active |  80 | 0.0.1.SNAPSHOT | SIMPLI-CITY API :: Interfaces
210 | Active |  80 | 0.0.1          | SomeSimplicityService
karaf@root>
```

Figure 18: After Installing a Service Bundle (Server Side - Karaf)

Another functionality implemented in this final prototype of Service Development API Plugin is the manifest editor, which is a visual editor for the service manifest XML file. It helps the developer in the task of configuring the services in an easier way than editing the actual XML file. This editor is automatically displayed when the manifest.xml file is opened through the Project Structure lateral tool (see Figure 13). It is organized by sections General, Import, SLA, License and one of Export, External or Data, depending on the type of service being developed.

As an example, the description and keywords of the example service created in Section 5.2 above can be easily changed by opening the manifest.xml file, going to the General tab and putting some description and keywords (see Figure 19). Changes are automatically done on the actual XML file (see Figure 20).

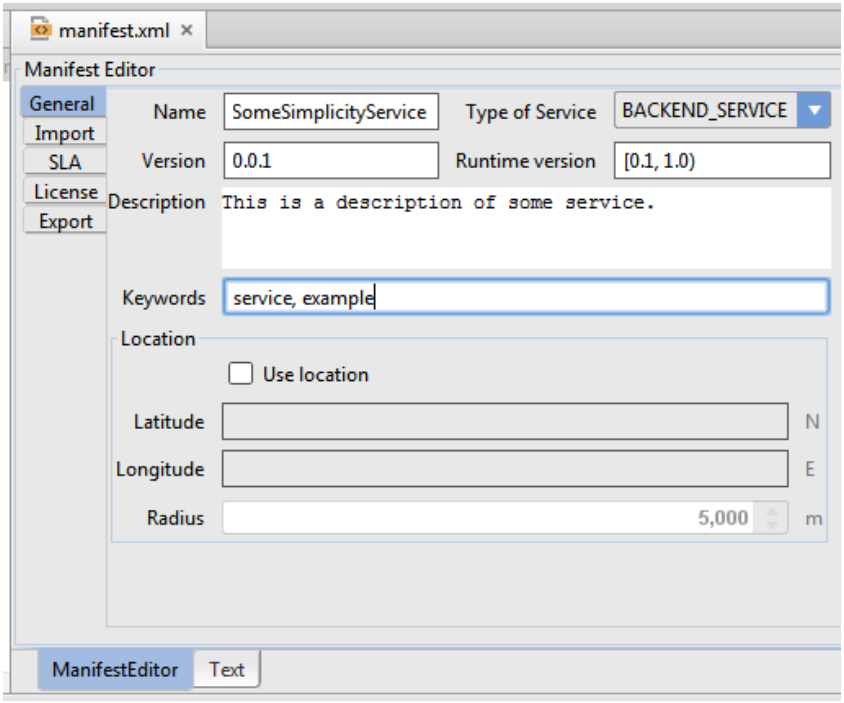

Figure 19: Service Manifest Visual Editor

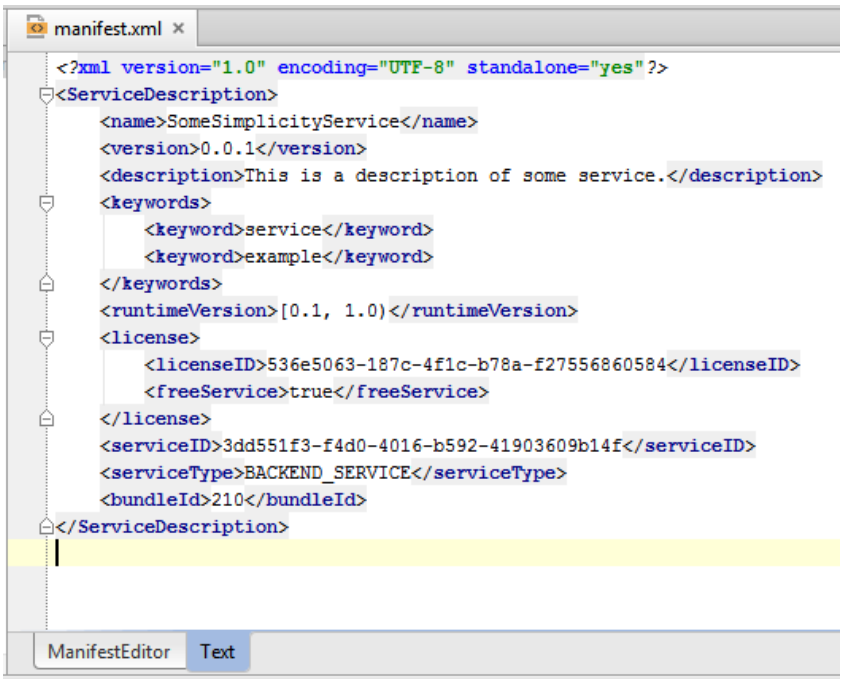

Figure 20: Service Manifest Textual Editor

When developing a SIMPLI-CITY service, some other SIMPLI-CITY services available in the SRE may be needed. In order to facilitate their usage, the Service Development API Plugin provides the means to discover them and show how to make use of their methods. This can be done by using the main menu option SIMPLI-CITY > Connect to a Service… (see Figure 8). When the form is opened, the list of services currently installed on Service Runtime Environment is displayed. By selecting any of the services, a list of its methods is shown below with their exact name and signature (see Figure 21).

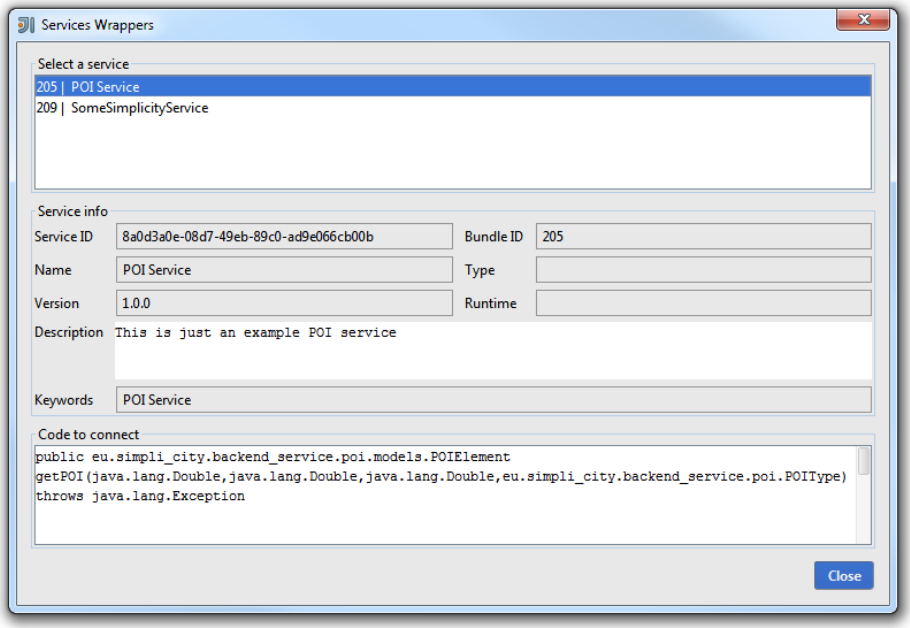

Figure 21: List of Methods of a Service

The last piece of functionality to be mentioned is the Help, found under SIMPLI-CITY > SIMPLI-CITY Developer's Guide (see Figure 8). It is presented as a JavaHelp system which contains a set of articles to aid the developer in designing, creating and developing a service. It is organized in sections, starting with an introduction of the SIMPLI-CITY project and its architecture, and continuing with the description of the first steps for creating a service, using the Service Development API Plugin and best practices (see Figure 22). As the internal structure of this JavaHelp system is a set of HTML documents and images, along with CSS styling files, it can be easily exported to any website to make it accessible from a web browser.

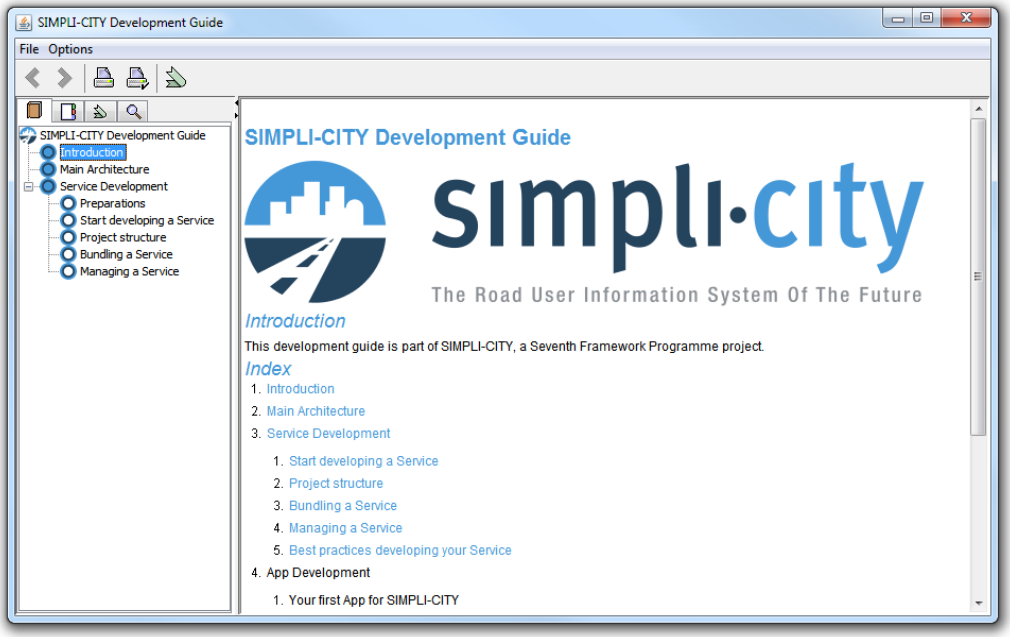

Figure 22: SIMPLI-CITY Developer's Guide

6 Limitations and Further Developments

Within this final prototype, the development of the different subcomponents of the Service Development API has been completed.

On the server side, the Service Development API Interface and Implementation and the Service Management Server has been fully implemented and integrated with the Service Management Client and the Service Runtime Environment, allowing the installation and management of services by the developer straight from the Service Development API Plugin.

On the client side, the Service Development API Plugin has been fully implemented and integrated with the Service Management Client. The main subcomponents of the Plugin are: wizard for creating new services, a visual editor for manifest files allowing easy configuration of services, functionalities for managing services in addition to assistance in connecting to other SIMPLI-CITY services. The Plugin has been tested on IntelliJ IDEA 13.0.1 and above.

Unless any issue or malfunction shows up no further developments are required.

7 Summary

This deliverable has presented and described the scope of the final prototype of the Service Development API component of the SIMPLI-CITY Mobility Services Framework.

In this prototype, the final version of the RESTful interface to be used by service developers to deliver new services in the SIMPLI-CITY Service Runtime Environment has been implemented. Secondly, the final version of the Service Development API Plugin is provided, allowing developers creating, configuring, developing and managing services that will be run in the Service Runtime Environment.

An analysis of the degree of fulfilment of the requirements to be covered by the component as specified in the Requirements Analysis Report (D2.3) has been presented.

Moreover, all the required steps to install, deploy, and execute the different subcomponents have been detailed, including the ones for the Server Side subcomponents and the ones for the Plugin.

Finally, the current state of limitations has been discussed.