simpli·city

The Road User Information System Of The Future

WP6 – Personal Mobility Assistant

D6.1: Dialogue Interface Prototype

Deliverable Lead: TALK

Contributing Partners: ASC, CRF, TIE

Delivery Date 03/2014

Dissemination Level: Public

Version 1.00

This deliverable describes the work carried out during the development of the prototype of the Dialogue Interface component of the SIMPLI-CITY platform. It specifies the scope of the delivered software 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	Fredrik Kronlid, TALK
Internal Reviewer 1	Stefan Schulte, TUV
Internal Reviewer 2	Vadim Petrenko, TIE
Type	Deliverable
Work Package	WP6: Personal Mobility Assistant
ID	D6.1: Dialogue Interface Prototype
Due Date	31.03.2014
Delivery Date	04.04.2014
Status	Approved

Document History	
Contributions	V0.1, Stefan Schulte, Philipp Hoenisch, TUV, 04.12.2012, Added document structure. V0.2 Fredrik Kronlid, TALK, 21.03.2014, First Version for D6.1 V0.3 Fredrik Kronlid, TALK, 24.03.2014, Added 2.2.X entries, Exec summary, etc. V0.4 Pontus Lindström, TALK, 24.03.2014, Added sections 4 and 5.. V0.5 Fredrik Kronlid, TALK, 24.03.2014, Ready for internal TALK Review V0.6 Fredrik Kronlid, TALK, 24.03.2014, Ready for Review V0.7 Fredrik Kronlid, TALK, 27.03.2014, Reference to video V0.8 Fredrik Kronlid, TALK 28.03.2014, Ready for 2nd review V0.9 Pontus Lindström, TALK 31.03.2014, Update of installation instructions V0.10 Fredrik Kronlid, TALK 04.04.2014, Integrated Review Comments V1.00 Fredrik Kronlid, TALK 04.04.2014, Final revision
Final Version	April 4th, 2014

Disclaimer

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

The document describes the Dialogue Interface prototype, which is the user interface component of SIMPLI-CITY. The Dialogue Interface has successfully been adapted to the selected platforms (Android and Linux), and is possible to run on an Android handset connected to a server running the Dialogue Interface's backend components.

Of the three listed "Must Have Requirements", this prototype delivers a 100% fulfilment of two of them, while the third is fulfilled by 20%. Other requirements have not so far been addressed.

The document describes in detail how to install and run the backend components on a Linux server and how to install and run the frontend components on an Android handset.

Finally, the limitations of the prototype are discussed, most notably that the server can only handle one connected client at a time, and how these limitations will be addressed.

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 Dialogue Interface Prototype 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 modeled, 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 describe how to use the prototype of the Dialogue Interface and exploit its functionalities. For this, the scope and requirements of this prototype, the requirements and preparations for users and developers, an installation and usage guide as well as a brief overview of the envisioned enhancements for the second prototype of the Multimodal Dialogue Interface, are provided.

The first Dialogue Interface prototype is the outcome of the discussions and implementation work done in project months 9 to 18. It provides a preliminary implementation of the functionalities of the Dialogue Interface as provided with SIMPLI-CITY deliverables D3.2.1 (Functional Specification), and D3.2.2 (Technical Specification). This deliverable D6.1 will be superseded by deliverable D6.2.2, which will be the final prototype of the Multimodal Dialogue Interface, which includes the Dialogue Interface.

1.3 Document Status and Target Audience

This document is a public delivery and is aiming to describe the Dialogue Interface development work to anyone who is interested. However, since the content is preliminary and subject to changes, depending on the upcoming development work within SIMPLI-CITY task T6.2, the information in this document will be superseded by D6.2.2, which will be the complete description of the Multimodal Dialogue Interface of SIMPLI-CITY, of which the Dialogue Interface is an integral part.

1.4 Abbreviations and Glossary

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

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

1.5 Document Structure

This deliverable is broken down into the following sections:

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

Section 2 provides an overview of the scope and relationship of the prototype, showing how the Dialogue Interface fits into the overall SIMPLI-CITY software framework and the outcome of the first 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 if they want to make use of the Dialogue Interface.

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 current limitations of the Dialogue Interface prototype and gives an outlook on the final prototype.

Finally, Section 7 provides a summary of the document.

2 Prototype Scope and Requirements Coverage

2.1 Dialogue Interface – General Information

This component is the user interface layer of SIMPLI-CITY, taking user input in the form of utterances, managing the need for further user input, and eventually transforming a sequence of utterances into SIMPLI-CITY app calls. The result of the app call is then fed back to the user, using speech. User input is collected using an Automatic Speech Recognition (ASR) subcomponent. The speech input is interpreted as dialogue "moves" (dialogue acts, such as *ask*, *answer*, and *request*) with some semantic content. The dialogue component can also be triggered by app activity, and can fetch input data from the app instead of asking the user, if suitable.

The dialogue moves are forwarded to the central subcomponent Dialogue Move Engine (DME). The DME contains a description of the dialogue context, including information about recent utterances, recent questions, active apps etc., and makes decisions about the system's next move based on this information. The next move can be to ask a question, answer a question, to carry out an app call, or to do nothing. Any resulting dialogue move output from the DME is sent to the Generate subcomponent, which will generate an utterance. The final step in the iteration is to speak the utterance, which is done using the Text-To-Speech (TTS) subcomponent. A Turn Manager subcomponent is used to manage the right and obligation of a dialogue partner to speak at a certain point in time during the dialogue (to make an utterance in a dialogue is often referred to as a "turn"). The Turn Manager distributes the turn between the user and the system.

Figure 1 shows the location of the Dialogue Interface (a part of the Multimodal Dialogue Interface) in the SIMPLI-CITY Global Architecture. For the full Global Architecture, refer to deliverable D3.1.

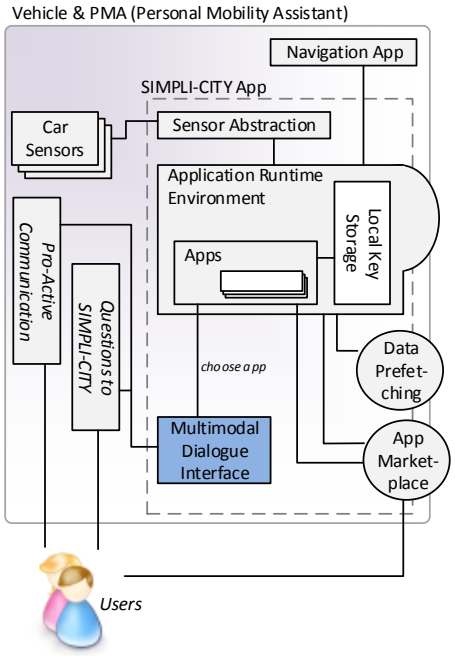

Figure 1: Location of Dialogue Interface (as Part of the Multimodal Dialogue Interface) in the SIMPLI-CITY Global Architecture

2.2 Scope of the First Prototype

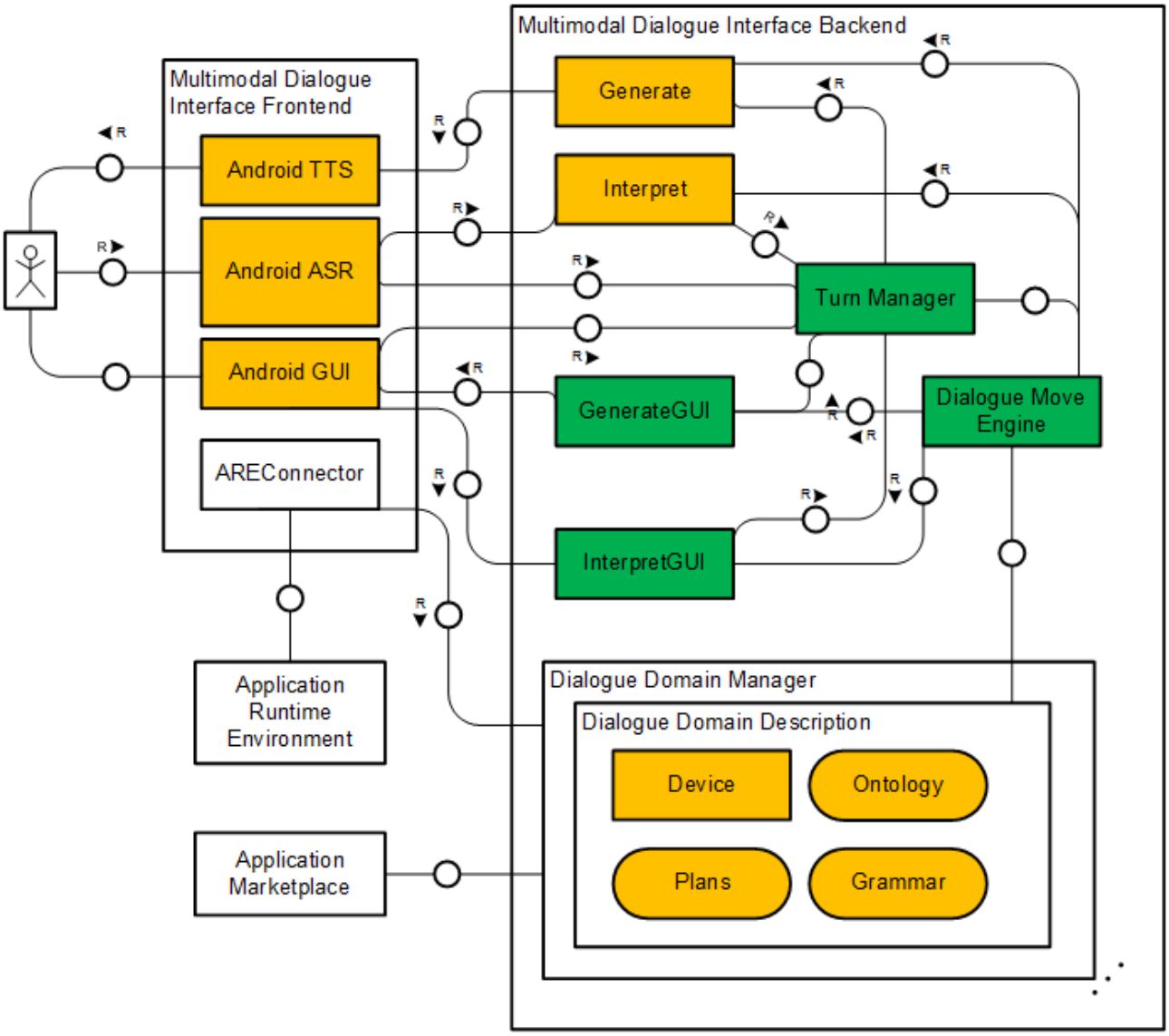

Figure 2: Scope of the First Prototype of Dialogue Interface

Figure 2 depicts the status of development of the first prototype of the Dialogue Interface, showing the subcomponents that are covered within this first prototype.

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

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

Several of the relevant subcomponents are marked as partially implemented. These implementations are considered final for this deliverable, but at the same time they form an integral part of the work in task T6.2 (Voice-based Multimodal User Interface), and will be developed further within that task.

Two of the relevant subcomponents are not showed in Figure 2: The Connector subcomponents (Connector To Backend and Connector To Frontend – see Sections 2.2.8 and 2.2.9), which are working as the connections between the Multimodal Dialogue Interface Frontend and Backend. The GenerateGUI and InterpretGUI subcomponents are not relevant for the Dialogue Interface.

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 Generate

The Generate subcomponent is basically a standard component that has been used in the Talkamatic Dialogue Manager (TDM, which is the Talkamatic product that the Dialogue Interface is based on) in its standard form. The Generate subcomponent generates text strings to be uttered by the Android TTS subcomponent. The strings that are generated are based on a grammar defined by the app developer.

2.2.2 Interpret

The Interpret subcomponent is basically a standard component that has been used in TDM in its standard form. The Interpret subcomponent interprets text strings into TDM's semantic language of dialogue moves. Since SIMPLI-CITY uses the dictation based Android ASR (as opposed to an ASR with a grammar based language model), some initial adaptations of the Interpret subcomponent for dealing with this kind of free input have been made. For instance, input is spell-checked before parsing to catch possibly existing recognition errors, and parsing of partial utterances is done in case the entire utterance cannot be parsed.

2.2.3 Dialogue Move Engine

The DME is unchanged (compared to the default Talkamatic implementation), and will likely remain that way during the rest of the project.

2.2.4 Turn Manager

The Turn Manager is unchanged (compared to the default Talkamatic implementation), and will likely remain that way during the rest of the project.

2.2.5 Android Text-To-Speech

The Android TTS (Text-to-Speech) subcomponent wraps the TTS service available via the Android API. The TTS module takes as input a string of characters to speak, sent from the Generate module via the connector modules (see below), but also information from the GUI about what element is currently in focus in order to read out screen alternatives to the user using the VoiceCursor concept. It also listens to notifications signalling if the Push-To-Talk button is pressed, or if an item is selected in the GUI, in which cases the TTS is silenced. Finally, it also signals when the current utterance was spoken or aborted.

2.2.6 Android GUI

In the Dialogue Interface, the GUI only consists of the Push-To-Talk button, but as the Multimodal Dialogue Interface (MMDI) and Dialogue Interface share the GUI subcomponent, also the MMDI functionality is handled by the GUI. The MMDI GUI offers menu choices and information to the user, and receives haptic/graphic input from the user. The user input is then sent to the backend modules via the Connectors.

2.2.7 Android Automatic Speech Recognition

The Android ASR wraps the ASR service for speech input available in the Android API. The Android ASR module takes as input a speech signal from the user (something which is handled by the Android ASR service itself). The output is a number of hypotheses of what the user said, annotated with the probability/confidence that the individual hypothesis is actually correct. The output is sent to the backend, and is broadcasted to other subcomponents, which subscribe to this kind of event, via the Connector to Backend.

The Android ASR is activated by pushing the Push-To-Talk button.

2.2.8 Connector to Backend

The Connector to Backend subcomponent, running in the frontend, is responsible for all communication with the backend. All messages to the backend from the frontend are channelled via the Connector to Backend, as well as all messages from the backend to the frontend. The frontend and backend will be physically connected over a network channel.

2.2.9 Connector to Frontend

The Connector to Frontend, running in the backend, is responsible for all communication with the frontend. All messages from the backend to the frontend are channelled via the Connector to Frontend, as well as all messages from the frontend to the backend.

2.3 Covered Requirements

This section describes the degree of fulfilment of the requirements to be covered by the Dialogue Interface specified in the Requirements Analysis Deliverable (D2.3) and the Functional Specification (D3.2.1).

Table 1: Requirements Related to the Multimodal Dialogue Interface and their Degree of Fulfilment

Requirement	Degree of Fulfilment	Comment
Must Have Requirements		
U36: Natural speech recognition	20%	A dictation speech recogniser has been connected to the system, and some first steps have been taken to ensure that its output is properly taken care of. Further work in the interpretation subcomponent is required, and also work in the individual applications.
U41: Input interactions with system via multimodal UI: on screen, voice control, and non-voice control	100%	Voice input works as expected, and is synchronised with screen input.
U42: Output interaction from system through UI	100%	System output is spoken, and is synchronised with screen output.
Should Have Requirements		
U18: Reasonable response time	20%	Response times are subjectively reasonable, more exact measurements will be done at the later stages of development.
U43: Non-distracting interaction	50%	The TDM is relatively non-distracting in itself, but research results from other projects where TALK is participating should be implemented in the Dialogue Interface to improve non-distractiveness even further.
U45: Automotive quality voice recognition (automotive acoustic models for in car use)	0%	This requirement has not been taken into account yet, but will be addressed in D6.2.2.
U46: High speech recognition rate	0%	No measurement or tuning has taken place.

Requirement	Degree of Fulfilment	Comment
Could Have Requirements		
U19: Minimum manual configuration U22: Personalization – Incremental configuration	0%	Not applicable yet, since no configuration is required.
U37: Result oriented instead of service/app recognition oriented U38: Disambiguation U39: Provision of a limited number of alternatives	0%	This work has not started yet. The system only supports one single app at the moment.
U40: System should have a friendly voice U44: Voice quality	50%	Built-in TTS voices are currently used, and no attempt to evaluate their quality or friendliness has been made.
U47: Voice interaction through the car audio system (microphone & loudspeakers) for hands free in car use	0%	Will be considered for D6.2.2.
U20: Multilingual	0%	Will be considered for D6.2.2
Will not have for now		
U21: Link voice commands to apps	0%	Will be considered for D6.2.2.

3 Preparations

This section provides information about how to enable installation of the MMDI Frontend on an Android device and the MMDI Backend on a server.

3.1 Preparing an Android Device for the MMDI Frontend

This section describes how to enable installation of the development version of the MMDI Frontend on an Android device. It should be noted that the final version of the MMDI Frontend will not require such preparations from end users.

In order to install any Android app from sources other than the Google Play Store, the Android device has to be configured to install apps from “Unknown Sources”. This setting option is found under “Settings > Security > Unknown Sources”. An easy step-by-step guide to enable it can be found below. After following these steps, the user will be able to install any Android app in the form of an Android Application Package File (APK), either downloaded from the Internet or copied to the phone’s memory (i.e., SD card or internal memory, via USB cable, Bluetooth, etc.), including the ones that are part of this deliverable.

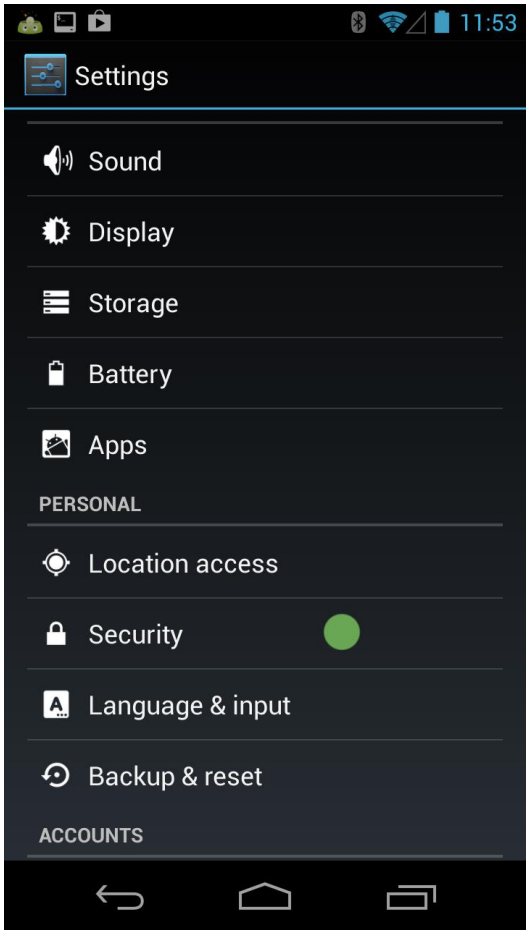

Figure 3: Selecting Security Options in Settings Menu

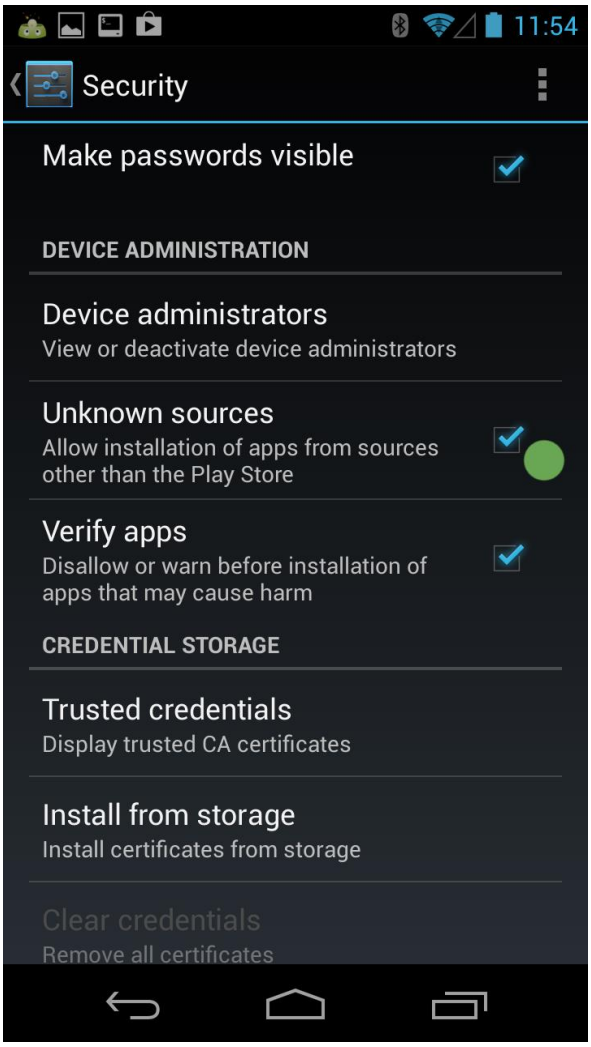

Figure 4: Enabling Installation of Apps from Unknown Sources

The user needs to set up Android Studio together with Android SDK on the computer in order to be able to upload the MMDI Frontend to the Android device. The Android SDK can be downloaded within the Android Studio. Installation instructions to install Android Studio can be found on the official web page: http://developer.android.com/sdk/installing/studio.html.

3.2 Preparing a Server for the MMDI Backend

The MMDI Backend requires Python 2.7 and supports Linux, Mac OS X and Windows. It requires the package *python-dev.* This package can be installed using the command (assuming a Linux environment): *sudo apt-get install python-dev.* The other dependencies can be found in the file *pip_dependencies* and are installed using the command depicted in the following listing:

Listing 1: Installation of MMDI Dependencies

```
talkamatic@machine:~$ sudo pip install -r pip_dependencies
Downloading/unpacking Twisted==10.2.0 (from -r pip_dependencies (line
1))
Downloading Twisted-10.2.0.tar.bz2 (2.7Mb): 2.7Mb downloaded
Running setup.py egg_info for package Twisted

Downloading/unpacking jsonpickle==0.4.0 (from -r pip_dependencies
(line 4))
Downloading jsonpickle-0.4.0.tar.gz
Running setup.py egg_info for package jsonpickle
```

4 Installation (Deployment)

This section provides guidelines on how to install and deploy the MMDI Frontend on an Android device and the Backend on a server.

4.1 MMDI Frontend

In the current state, the SIMPLI-CITY app has to be custom built for every server running the TDM Backend. The IP number of the server is stored statically and can only be set when building the SIMPLI-CITY app. For this reason, the user has to email info@talkamatic.se to get an instance of the app built for their setup. Of course, this step will not be necessary for the final prototype.

The SIMPLI-CITY app is installed by transferring it to the Android phone and then using an APK installer app to install the app on the phone. The two apps AirDroid and APK Installer are used to transfer the APK and installing it to the phone. Both apps can be downloaded from Google Play.

The following procedure is illustrated in Figure 5.

Transferring the APK to the phone is done using the app AirDroid. AirDroid makes the phone accessible by setting up a web server on it. The user needs to access the phone's web server with a web browser (the IP address of the phone is displayed in the AirDroid app). Furthermore, the browser needs to be allowed to the phone (Step 1 in Figure 5).

On the web page, “Files” need to be chosen (Step 2); this will show the file tree on the SD card on the Android device. The folder where the APK should be uploaded needs to be entered and the blue “Upload” button needs to be clicked (Step 3). Then, the APK file can be chosen (Step 4).

As depicted in Figure 6, the next steps are to launch the APK Installer, to choose the uploaded APK file, and to click on “Install”. If everything works, the SIMPLICITY app can be found in the apps menu where all the other apps are located.

Figure 5: Step by Step Guide to Transferring SIMPLI-CITY App to Device

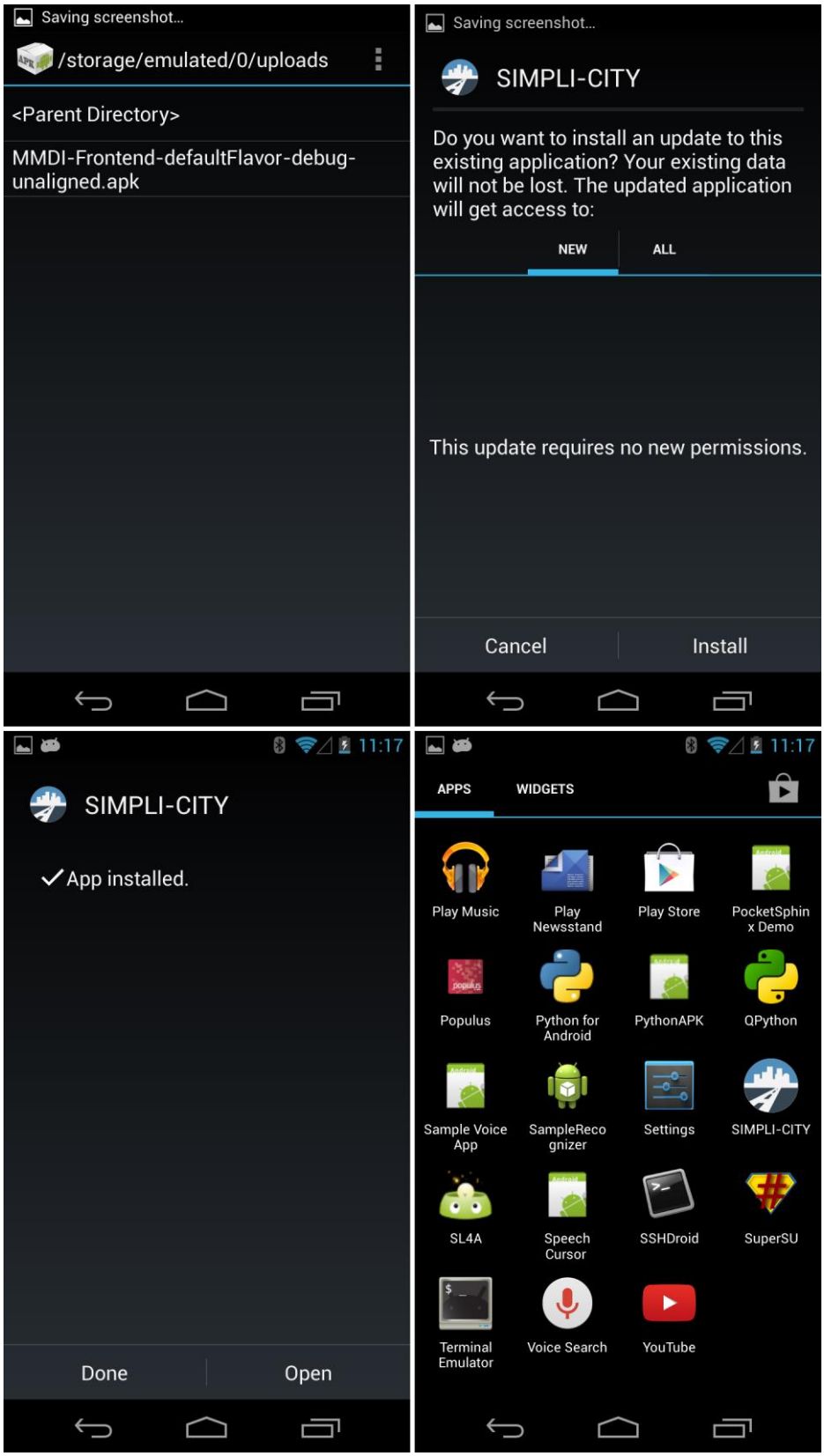

Figure 6: Installing APK Using APK Installer

4.2 MMDI Backend

The Backend is packaged as several Python packages, and each package is installed by using the command *easy_install* with the package provided as first parameter, as depicted in Listing 2.

Listing 2: Installation of MMDI Backend Packages

```
talkamatic@machine:~$ sudo easy_install tdm-1.0.egg maharani-1.0.egg
tdm_loader-1.0.egg
Processing maharani-1.0-py2.7.egg
removing '/usr/local/lib/python2.7/dist-packages/maharani-1.0-
py2.7.egg' (and everything under it)
creating /usr/local/lib/python2.7/dist-packages/maharani-1.0-py2.7.egg
Extracting maharani-1.0-py2.7.egg to /usr/local/lib/python2.7/dist-
packages
maharani 1.0 is already the active version in easy-install.pth

Installed /usr/local/lib/python2.7/dist-packages/maharani-1.0-
py2.7.egg
Processing dependencies for maharani==1.0
Finished processing dependencies for maharani==1.0
```

It is also necessary to set an environment variable TDM to “tdm” as depicted in the following listing:

Listing 3: Setting the TDM environment variable

```
talkamatic@machine:~$ export TDM=tdm
```

A prototypical SIMPLI-CITY app called “Eco Index” is provided in order to enable testing and validation of the interaction between the Backend and Frontend. The user can validate that the installation of the Backend is correct by running a suite of tests. A script is provided which executes interaction tests for the “Eco Index” app. It is executed with the command *bin/test_interactions.py* as depicted in Listing 4.

Listing 4: Running the Eco Index Test Suite

```
talkamatic@machine:~/eco_index$ python bin/test_interactions.py
Running tests from eco_index/test/interaction_tests_eng.txt
Using TDM from system directory
Using maharani from system directory
....
----------------------------------------------------------------------
Ran 1 test in 3.066s

OK
Running tests from eco_index/test/interaction_tests_gui_eng.txt
Using TDM from system directory
Using maharani from system directory
....
----------------------------------------------------------------------
Ran 1 test in 3.349s

OK
```

5 Executing MMDI Backend and MMDI Frontend

This section describes how to start the MMDI Backend and the MMDI Frontend.

5.1 Executing MMDI Backend

To run the Backend with the “Eco Index” app, there is a script called *eco_index_android.py* script in the "bin" folder of the *eco_index* package. The script starts the Backend and waits for the Frontend to connect, as depicted in Listing 5.

Listing 5: Executing the MMDI Backend

```
talkamatic@machine:~/eco_index$ python bin/eco_index_android.py
Using TDM from system directory
Using maharani from system directory
```

5.2 Executing MMDI Frontend

The MMDI Frontend app can be found in the list of Android apps under the name “SIMPLI-CITY” and is started by clicking on it. When the app is started, a main menu with two options shows up: “What is my eco-index?” and “How can I improve my eco-index?”. The bottom right corner contains the Push-To-Talk icon. When tapping the icon, a sound is played and the app starts listening to the user.

The “Eco Index” app is very simple, and can only give pre-defined answers to two questions. When the user asks “What is my eco-index?” the system answers “55”. When the user asks “How can I improve my eco-index?”, the system answers “Use a higher gear”.

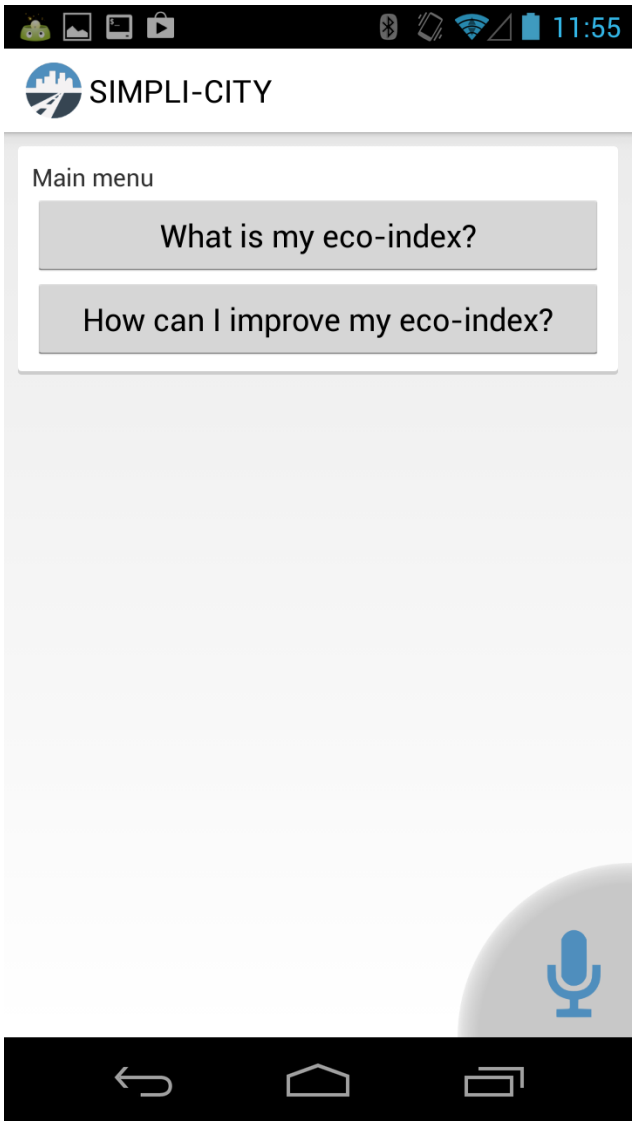

Figure 7: Prototype App Showing Two Menu Items with the Microphone Icon Showing at the Bottom Right.

6 Limitations and Further Developments

This first prototype in work package WP6 delivers the SIMPLI-CITY Dialogue Interface. This dialogue interface, based on the Talkamatic Dialogue Manager (TDM), is a fully-fledged, specific-purpose dialogue system, potentially able to understand a wide variety of user requests. The implementation covers the whole chain of interaction, from speech recognition, via dialogue modelling up to the point where the dialogue manager will provide input to services and output, respectively, to end user apps.

Although it covers the whole chain from ASR to TTS, it does have some limitations, as described below. All of these issues will be regarded in the coming iterations of the MMDI (T6.2).

6.1 Session Management

The current version of the Dialogue Interface can only handle a single connected device. In upcoming iterations, the solution will expanded to support multiple end-user devices connected to the same SIMPLI-CITY server.

6.2 Natural Speech Recognition Interpretation

The Automatic Speech Recogniser in Android is dictation based, and the Dialogue Interface does not handle the variation of potentially recognised utterances. Upcoming iterations will support an improved coverage of utterances that go beyond the grammar specified by the app developer.

6.3 ARE Connection

The current Dialogue Interface is not connected to the Application Runtime Environment, but shortcuts the interaction using predefined results in the Dialogue Domain Description. The ARE Connection component, allowing for dynamic calls to the actual app, will be implemented in the next iterations.

6.4 One Dialogue Domain

So far, the MMDI has only been tested with a single SIMPLI-CITY app. The solution needs to be expanded in order to support multiple installed apps. This will require a more dynamic handling of Dialogue Domain Descriptions.

6.5 Hybrid Ontology/Taxonomy and Hierarchical Ontologies

The Ontology/Taxonomy has not yet been adapted to the SIMPLI-CITY context, as the main focus of the implementation work so far has been on getting the system to run on the desired platform. This functionality does not mean anything in terms of functionality, but has a larger impact when it comes to defining new apps.

7 Summary

This component is the user interface layer of SIMPLI-CITY, taking user input in the form of utterances managing the need for further user input, and transforming them into app calls.

The subcomponents that have been implemented for this prototype are the following:

- Generate
- Interpret
- Dialogue Move Engine
- Turn Manager
- Android Text-To-Speech
- Android GUI
- Android Automatic Speech Recognition
- Connector to Backend
- Connector to Frontend

Of the three must have requirements defined for the task, two are completely fulfilled. The third is 20% fulfilled (natural speech recognition).

To successfully run the software, the Backend subcomponents are installed and run on a Linux server, while the Frontend subcomponents run on an Android handset.

The current version of the implementation has limitations, most notably that the Backend components only handle one handset at a time. The limitations will be addressed in the upcoming D6.2.2 deliverable.

The functionalities and execution steps discussed in this deliverable are also presented in a separate video, which is delivered along with the software package and this document.