simpli·city

The Road User Information System Of The Future

WP6 – Personal Mobility Assistant

D6.2.2: Voice-based Multimodal User Interface Prototype II

Deliverable Lead: TALK

Contributing Partners: ASC, CRF, TIE

Delivery Date 04/2015

Dissemination Level: Public

Version 1.0

This deliverable describes the work carried out during the development of the second prototype of the Multimodal Dialogue Interface component of the SIMPLI-CITY platform. It specifies the scope of this final version and the degree of fulfillment 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	Vadim Petrenko, TIE
Internal Reviewer 2	Michaela Kargl, FGM
Type	Deliverable
Work Package	WP6: Personal Mobility Assistant
ID	D6.2.2: Voice-based Multimodal User Interface Interface Prototype II
Due Date	31.03.2015
Delivery Date	21.04.2015
Status	For Approval

Document History	
Contributions	V0.1, Stefan Schulte, Philipp Hoenisch, TUV, 04.12.2012, Added document structure. V0.2 Fredrik Kronlid, TALK, 21.03.2015, First Version for D6.2.2 V0.3, Fredrik Kronlid, TALK, 24.03.2015, Added comments about changes V0.4, Fredrik Kronlid, TALK, 24.03.2015, Most changes done. V0.5, Pontus Lindström, TALK, 24.03.2015, Installation and preparation sections. V0.6, Fredrik Kronlid, TALK, 25.03.2015, FMC Graphics V0.7, Fredrik Kronlid, TALK, 25.03.2015, Proofread. V0.8, Pontus Lindström, TALK, 07.04.15. Comments from reviewers V0.9, Fredrik Kronlid, TALK, 07.04.15. Comments from reviewers V1.0, Fredrik Kronlid, TALK, 07.04.15. Final version.
Final Version	April 21st, 2015.

Note

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

Disclaimer

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

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

Project Partners

Vienna University of Technology (Coordinator), Austria

Ascora GmbH, Germany

TIE Nederland B.V., The Netherlands

Technische Universität Darmstadt, Germany

IBM Research – Ireland
Smarter Cities Technology Centre

Forschungsgesellschaft Mobilität, Austria

Talkamatic AB, Sweden

Atos Worldline, Spain

Centro Ricerche FIAT, Italy

SRM – Reti e Mobilità, Italy

Executive Summary

The document describes the final Multimodal Dialogue Interface prototype, which is the user interface component of SIMPLI-CITY to be used in the car. The Multimodal Dialogue Interface can run on an Android handset connected to a server running the Multimodal Dialogue Interface's backend components.

This prototype delivers a 100% fulfilment of the three "Must Have Requirements" listed in the D2.3 Requirements Analysis. Of the four "Should have Requirements" there is a 44 % fulfilment, compared to a 56 % fulfilment of the "Could have Requirements".

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, and how these limitations will be addressed during the remaining months of the project.

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 Multimodal Dialogue Interface Prototype is 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 describe how to use the final prototype of the Multimodal 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 are provided. There is also a brief discussion of the expected changes of the prototype during the final months of the project.

The final Multimodal Dialogue Interface prototype is the outcome of the discussions and implementation work done in project months 19 to 30. It provides an implementation of the functionalities of the Multimodal Dialogue Interface as provided with SIMPLI-CITY deliverables D3.2.1 (Functional Specification), and D3.2.2 (Technical Specification). This deliverable D6.2.2 is the final official prototype of the Multimodal Dialogue Interface. There will however be some changes to the software in the final months of the project, and we will deliver the updated software on request.

1.3 Document Status and Target Audience

This document is listed in the Description of Work (DoW) as “Public”, since the content is functional complete (beta prototype) but subject to changes, depending on the upcoming demands in other work packages and/or issues with the current implementation.

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 Multimodal Dialogue Interface 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 as they want to make use of the Multimodal 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 second prototype of the Multimodal Dialogue Interface.

Finally, Section 7 provides a summary of the document.

2 Prototype Scope and Requirements Coverage

2.1 Multimodal 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 and graphics. User input is collected using an Automatic Speech Recognition (ASR) subcomponent, as well as input from interaction with the screen. The speech input is interpreted to 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 do nothing, to ask a question, answer a question or to carry out an app call. Any resulting dialogue move output from the DME is sent to a Generate subcomponent, which will generate GUI contents and an utterance, respectively. The final step in the iteration is to do 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 Multimodal Dialogue Interface in the SIMPLI-CITY Global Architecture. For the full Global Architecture, refer to deliverable D3.1.

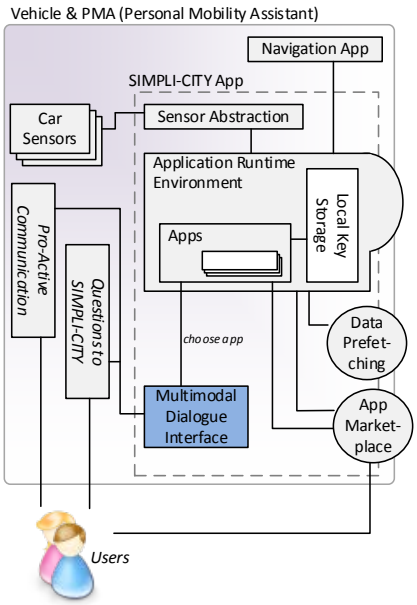

Figure 1: Location of Multimodal Dialogue Interface
in the SIMPLI-CITY Global Architecture

2.2 Scope of the Second Prototype

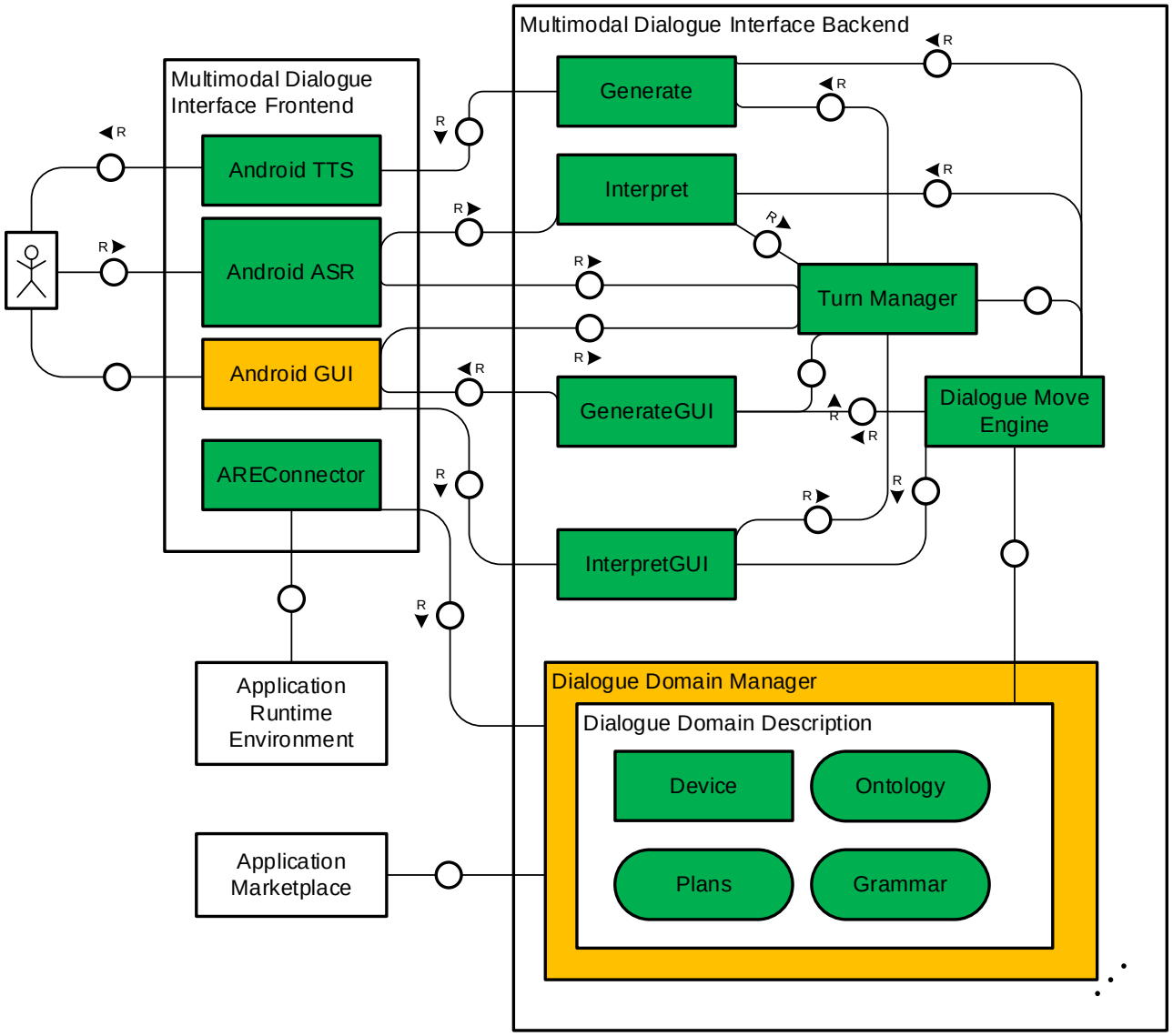

Figure 2: Scope of the Second Prototype of Multimodal Dialogue Interface

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

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

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

Two of the relevant subcomponents are not shown in this figure: 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 the Backend.

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 component is basically the standard component that has been used in the Talkamatic Dialogue Manager (TDM, which is the Talkamatic product the Dialogue Interface is based on) in its standard form. The Generate subcomponent generates text strings to be uttered by the Android Text-To-Speech subcomponent. The strings that are generated are based on a grammar defined by the app developer. The standard component has been extended in order to decorate dialogue moves with background information, so that a question can give background information. For instance, when the Multimodal Dialogue Interface (MMDI) asks the user about the preferred mode of transportation, the question can be decorated with the destination, or any fact that is deemed relevant by the developer: “What mode of transportation would you like to select on your way to Paris?”

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. In the first delivery of the MMDI, some initial adaptations for dealing with input from dictation based ASRs were made (partial parsing, spell checking). This work has been extended with a more efficient handling of partial parsing, allowing for parsing of utterances, which are only partially relevant for the application.

A more important change is the implementation of dynamic ontologies. A limiting factor for the usefulness of the TDM has been the necessity of knowing all of the individuals of a domain at build time. By using a construction, where a Named Entity Recogniser (NER) decides, if a part of an utterance contains a reference to a previously unknown individual, and in that case assigns it a unique ID, we allow for a gradual discovery of a domain, increasing the flexibility of TDM immensely. New data can be added to the application as necessary.

2.2.3 Dialogue Move Engine

The Dialogue Move Engine (DME) is mostly unchanged (compared to the default Talkamatic implementation), but it has been necessary to undertake a few updates. The updates are related to the introduction of background information (see Section 2.2.1), and to the inclusion of image handling. It is now possible to use images as context information and also as visualisations of answers to questions.

2.2.4 Turn Manager

The Turn Manager has been updated to include also Push To Initiate, as a complement to the existing Push To Talk functionality. Push to Initiate means that the user initiates the dialogue by pushing the button, and that the user and the systems take turn speaking with no further haptic interaction necessary. The system signals to the user when it’s the user’s turn to talk.

2.2.5 Android Text-To-Speech

The Android TTS (Text-to-Speech) component 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 Speech Cursor concept[1]. 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. In addition to these improvements over the final prototype, there has also been an extension, making it possible to use a TTS implementation provided by Nuance Mobile instead of the default Google TTS.

2.2.6 Android GUI

Besides the Push-To-Talk button, the Multimodal Dialogue Interface (MMDI) GUI offers visual menu choices and information to the user, and receives haptic input from the user. The user input is then sent as output from the module to the backend modules via the Connectors. The GUI has been extended with the ability to show images as answers to questions, as illustrations to menu options and also as context information. Two experimental implementations of the Speech Cursor concept have been made, but the actual integration of the technology into the GUI is still remaining.

2.2.7 Android Automatic Speech Recognition

The MMDI Automatic Speech Recognition (ASR) wraps the ASR service for speech input available in the Android API. The MMDI 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 about what the user said, annotated with the probability/confidence that the respective hypothesis is actually correct. The output is sent to the backend, and is broadcast 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 / Push-To-Initiate button. In the case of Push-To-Talk, this needs to be done every time the user wants to say something, while in the case of Push-To-Initiate, the ASR is activated automatically every time - the system requests some information from the user.

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 as well as all messages from the frontend to the backend are channelled via the Connector to Backend. The frontend and backend are physically connected over a network channel. The Push-To-Initiate functionality and also some other extensions, have required to extend the protocol that the connectors implement.

[1] The SpeechCursor is a Talkamatic technology for browsing lists of information without needing to look at a screen. The technology is patent pending in Germany and in the US.

2.2.9 Connector to Frontend

The Connector to Frontend, is a component running in the backend, it is responsible for all communication with the frontend. All messages from the backend to the frontend as well as all messages from the frontend to the backend are channelled via the Connector to Frontend. The Push-To-Initiate functionality and also some other extensions, have required to extend the protocol that the connectors implement.

2.2.10 GenerateGUI

The GenerateGUI component is the GUI counterpart of the Generate module. It receives as input a semantic description of an utterance, and generates a description of a screen suitable for accompanying the utterance, which will be generated and spoken.

2.2.11 InterpretGUI

The InterpretGUI module makes semantic interpretations of user activities in the GUI. InterpretGUI will take as input a GUI event, representing a menu choice, a button being pressed or similar. It will transform this event into a semantic description, which will be passed on to subscribers – primarily the Turn Manager.

2.2.12 Session Management

The MMDI can handle multiple connected frontend devices through the Session Manager. The Session Manager is a server that can be installed in the cloud and can run independently of the Dialogue Domain Description.

Installing the MMDI Frontend on a second Android device and connecting it to the same server will lead to two separate sessions, which will be handled independently of each other. The session management is divided into three parts: Session Manager, Backend Manager and Backend.

2.2.12.1 Session Manager

The Session Manager handles all the sessions and can be installed on a server in the cloud or on the local computer. The Session Manager routes packages between MMDI frontends and Backends. It also does work load management and distributes the Backends over the connected Backend Managers. It tells Backend Managers when to start new Backends and when to terminate old ones.

2.2.12.2 Backend Manager

A Session Manager can have several Backend Managers. The Backend Manager handles the creation and termination of Backends. The Backend Manager connects to the Session Manager and waits for commands.

2.2.12.3 Backend

The backend is created by the Backend Manager. It is the actual dialogue system. All the input given by the user to the MMDI Frontend is handled by the Backend.

2.2.13 Analysis Tools

In order to gain insights on how users are communicating with the MMDI, all user utterances are logged. As we also log the utterances that cannot be interpreted by the system, we can also analyze the grammar coverage of user utterances, in order to optimize the grammars.

2.2.14 ARE Connection

The ARE Connection connects the MMDI Frontend to the Application Runtime Environment (ARE). The ARE Connection is responsible for fetching data from the Apps, sending action requests to the Apps and receiving push notifications from the Apps.

2.3 Covered Requirements

This section describes the degree of fulfilment of the requirements to be covered by the Multimodal Dialogue Interface as specified in the deliverables Requirements Analysis Report (D2.3) and 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	100%	A dictation speech recogniser has been connected to the system, and its output is properly taken care of. Experience has shown that care needs to be taken when designing the grammars, analysing carefully which user utterances are systematically problematic.
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	100%	Response times are subjectively reasonable. Measuring the response times should preferably be made during evaluation. Informal estimations show that response times are on par with or even better than Apple's Siri™.

Requirement	Degree of Fulfilment	Comment
U43: Non-distracting interaction	75%	The TDM is relatively non-distracting in itself, but research results from other projects (SIMSI[2]) show that Multi-Modal Dialogue extended with the Speech Cursor technology significantly reduces the head-down time when compared to other modality conditions in simulator use. There are two implementations of Speech Cursor for Android, but integration into the GUI remains to be done.
U45: Automotive quality voice recognition (automotive acoustic models for in car use)	0%	The ASR technology chosen (Googles ASR) does not have specific acoustic models for in-vehicle use.
U46: High speech recognition rate	0%	No measurement or tuning has taken place.
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	30%	The system supports multiple apps, but not yet in one single dialogue domain. This means that these requirements are neither met nor meaningful, yet. However, since implementation of the single-domain system is planned to be finished before the end of the project, these requirements will likely be met.
U40: System should have a friendly voice U44: Voice quality	100%	The Android built-in TTS solutions are used. This means that the rather artificailly sounding built-in voice can be replaced with voices bought from Google Play. Another option is to build the Android front-end with support for Nuance Mobile TTS.
U47: Voice interaction through the car audio system (microphone & loudspeakers) for hands free in car use	70%	This work is in progress, and delivery is estimated before end of project.
U20: Multilingual	100%	The MMDI supports running applications in both English and Italian.
Will not have for now		
U21: Link voice commands to apps	0%	Will probably not be implemented due to shortage of resources.

[2] http://www.vinnova.se/sv/Resultat/Projekt/Effekta/SIMSI-Sakra-multimodala-talgranssnitt-i-fordon/

3 Preparations

This section provides guidelines on how to install and deploy the MMDI Frontend on an Android device and the MMDI Backend on a server. 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 a live 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 in Figure 3 and Figure 4. After following these steps, the user will be able to install any Android app, including the ones that are part of this deliverable, in the form of an Android Application Package File (APK), either downloaded from the Internet or copied to the phone’s memory (e.g., SD card or internal memory, via USB cable, Bluetooth, etc.),.

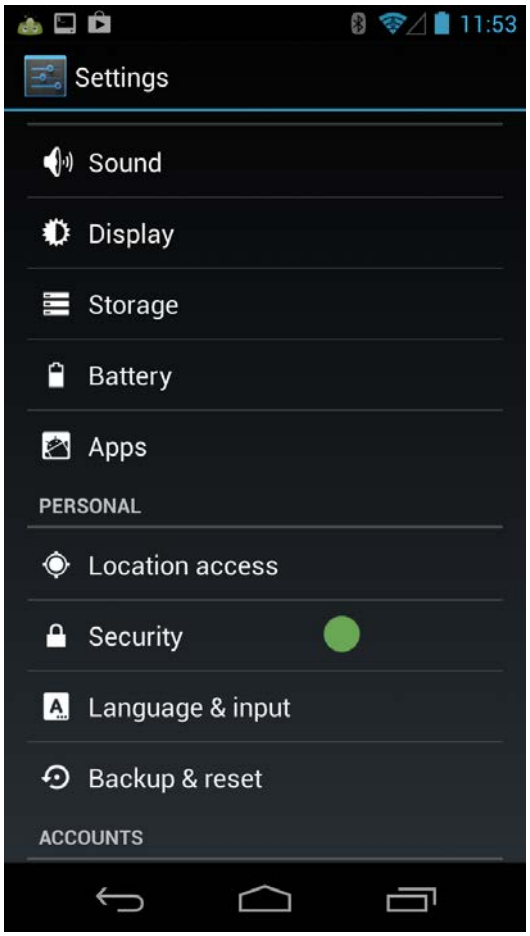

Figure 3: Selecting Security Options in Settings Menu

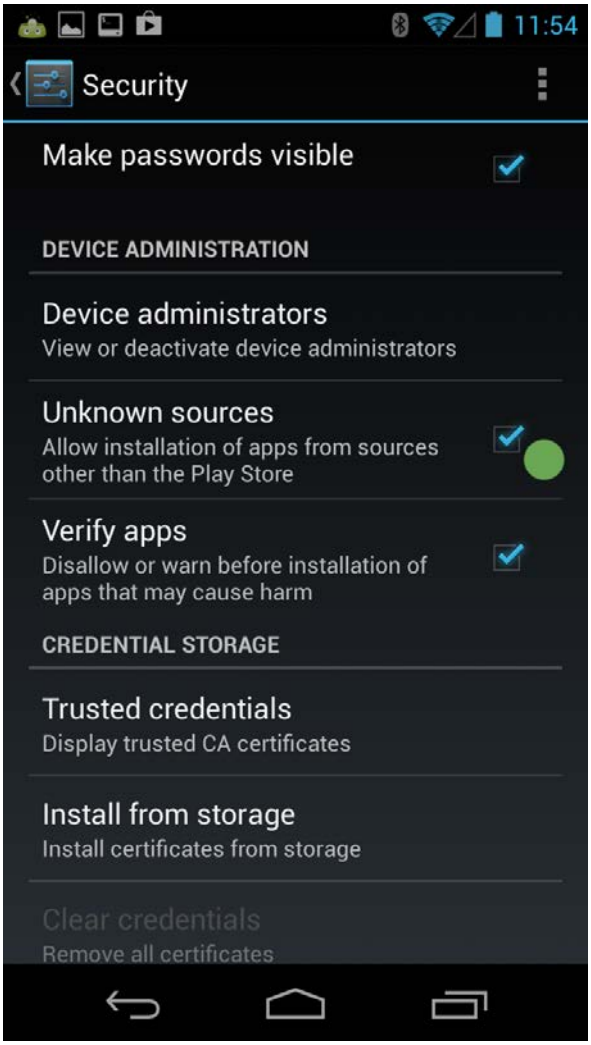

Figure 4: Enabling Installation of Apps from Unknown Sources

3.1 Preparing a Server for the MMDI Backend

The MMDI Backend requires Python 2.7 and supports Linux, Mac OS X and Windows. The python dependencies can be found in the file *pip_dependencies*, this file is located in the zip file simpli-city-final-release.zip. The dependencies 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 Whoosh==2.3.2 (from -r pip_dependencies
(line 1))
Downloading Whoosh-2.3.2.tar.gz (882kB): 882kB downloaded
Running setup.py (path:/tmp/pip_build_root/Whoosh/setup.py)
egg_info for package Whoosh

Downloading/unpacking tornado==2.4.1 (from -r pip_dependencies
(line 2))
Downloading tornado-2.4.1.tar.gz (348kB): 348kB downloaded
Running setup.py (path:/tmp/pip_build_root/tornado/setup.py)
egg_info for package tornado
Downloading/unpacking ws4py==0.3.2 (from -r pip_dependencies (line
3))
Downloading ws4py-0.3.2.tar.gz
Running setup.py (path:/tmp/pip_build_root/ws4py/setup.py)
egg_info for package ws4py
Installing collected packages: Whoosh, tornado, ws4py
Running setup.py install for Whoosh

Running setup.py install for tornado
Running setup.py install for ws4py
Successfully installed Whoosh tornado ws4py
Cleaning up...
```

4 Installation (Deployment)

4.1 MMDI Frontend

The SIMPLI-CITY app is installed by transferring the APK file MMDI-Frontend-release-final.apk 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: Uploading the APK using AirDroid:

- 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 access the phone (Step 1). 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: Installing the APK Using APK Installer, 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.

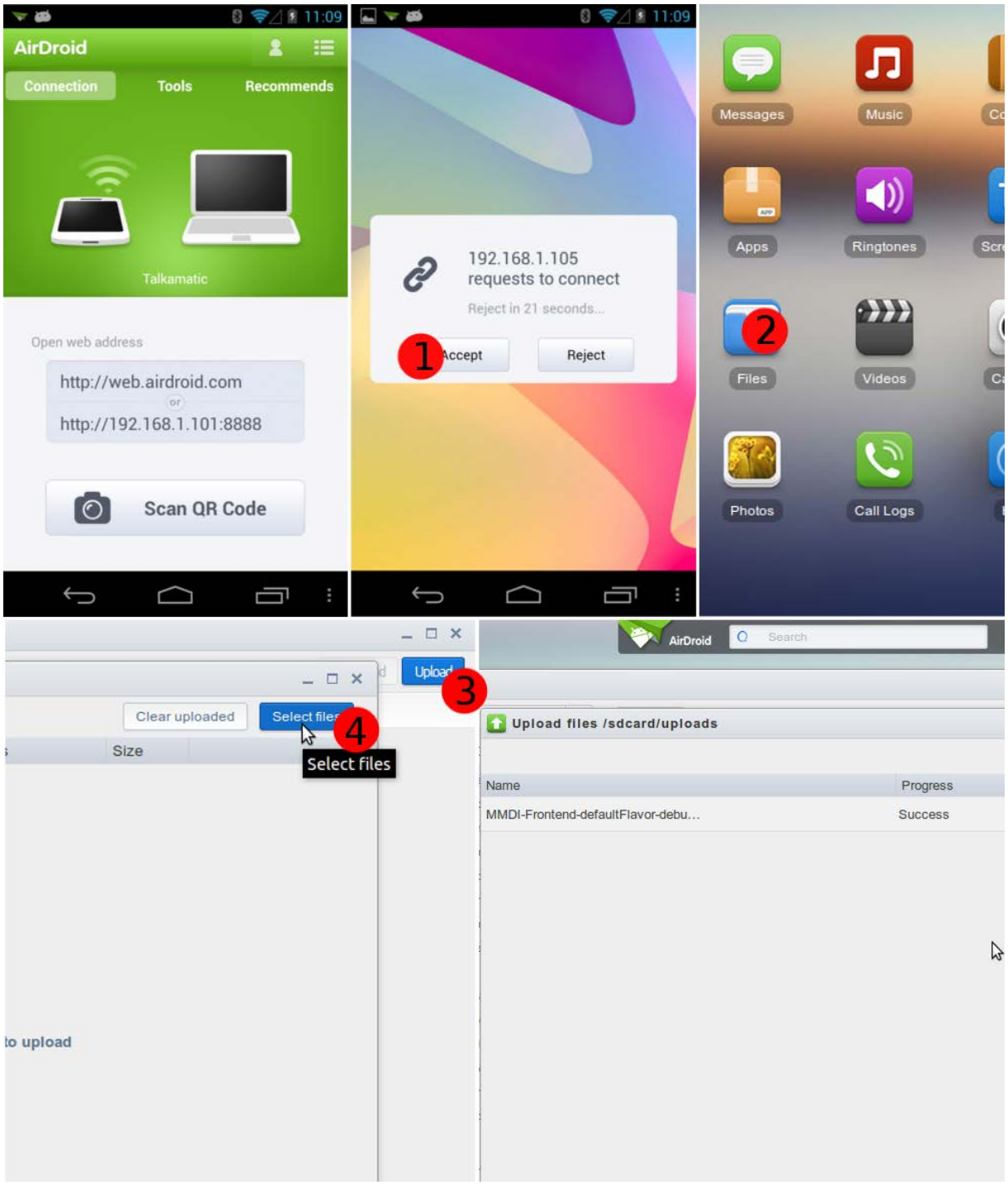

Figure 5: Uploading the APK using AirDroid

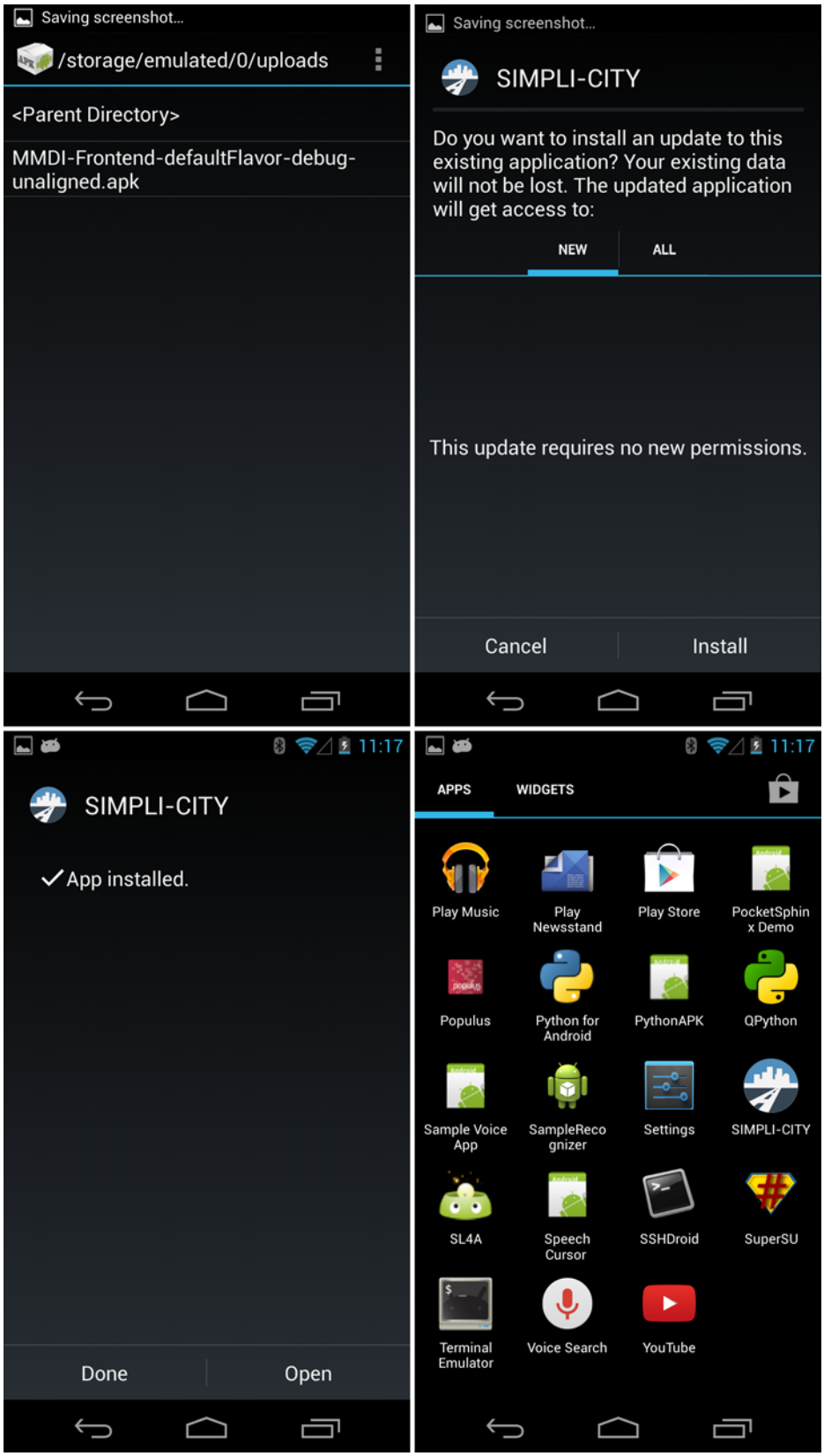

Figure 6: Installing the APK Using APK Installer

4.2 MMDI Backend

The Backend package, which can be found in the file simpli-city-release-final.zip, contains two python packages called eggs and an application folder Unzip this file, and enter the folder from a terminal. Installing the python eggs is done using the command easy_install with the egg as argument.

Listing 2: Installation of MMDI Dependencies

```
talkamatic@machine:~$ sudo easy_install maharani-1.0-py2.7.egg
Processing maharani-1.0-py2.7.egg
Copying maharani-1.0-py2.7.egg to /usr/local/lib/python2.7/dist-
packages
Adding maharani 1.0 to easy-install.pth file

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

talkamatic@machine:~$ sudo easy_install tdm-1.0-py2.7.egg
Processing tdm-1.0-py2.7.egg
Copying tdm-1.0-py2.7.egg to /usr/local/lib/python2.7/dist-packages
Adding tdm1.0 to easy-install.pth file

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

To test the integrity of the installation, interaction tests can be run in the provided application. These tests will run a test suite of scripted dialogue to see that the system works as intended.

Listing 3: Running Simplicity Apps Test Suite

```
talkamatic@machine:~/simplicity-apps$ python
bin/test_interactions.py
Running: tdm_test_interactions.py -p simplicity_project -f
simplicity_common/test/interaction_tests_eng.txt -L eng
Accepting connections on ws://127.0.1.1:9197/maharani
...
----------------------------------------------------------------------
Ran 1 test in 3.492s

OK
Running: tdm_test_interactions.py -p simplicity_project -a
eco_assistant -f eco_assistant/test/interaction_tests_eng.txt -f
eco_assistant/test/interaction_tests_gui_eng.txt -L eng
Accepting connections on ws://127.0.1.1:9211/maharani
....................
----------------------------------------------------------------------
Ran 1 test in 39.154s

OK
Running: tdm_test_interactions.py -p simplicity_project -a
increased_mobility -f
increased_mobility/test/interaction_tests_eng.txt -f
increased_mobility/test/interaction_tests_gui_eng.txt -L eng
Accepting connections on ws://127.0.1.1:9202/maharani
........................
----------------------------------------------------------------------
Ran 1 test in 52.312s


OK
Running: tdm_test_interactions.py -p simplicity_project -f
simplicity_common/test/interaction_tests_it.txt -L it
Accepting connections on ws://127.0.1.1:9191/maharani
...
----------------------------------------------------------------------
Ran 1 test in 3.346s

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 4.

Listing 4: 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 simple test app, 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”. It supports all the functionality available for the developers such as showing pictures and handling push events.

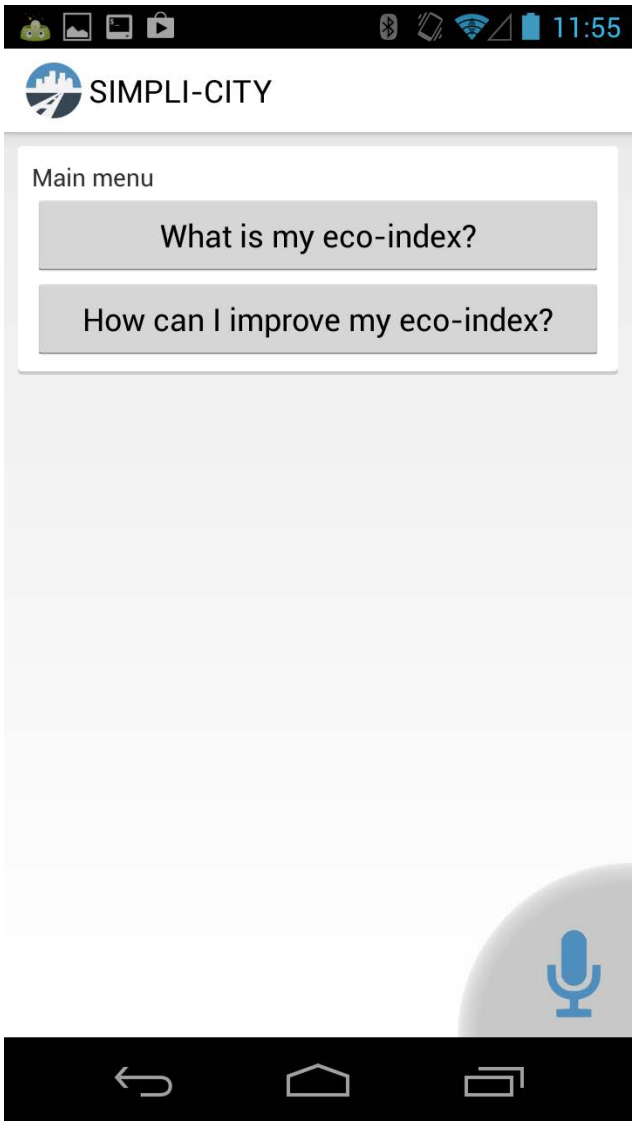

Figure 7: Prototype App Showing Two Menu Items with the Microphone Icon shown at the Bottom Right

6 Limitations and Further Developments

This prototype delivers the final version of the SIMPLI-CITY Multimodal Dialogue Interface. This Multimodal Dialogue Interface, built around 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, 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 path from ASR to TTS, it does have some limitations, described below. Most of these issues will be regarded in the bug-fixing and integration phase of the project.

6.1 One Dialogue Domain

The Dialogue Domain Description does not support dynamically loading of new domain descriptions, which is the information needed by the Backend to handle a dialogue in a certain domain. The current solution is to have all the Dialogue Domain Descriptions loaded in the backend on startup. The user can switch application at any time by naming an application, and can install and uninstall apps at their own will.

6.2 Hybrid Ontology/Taxonomy and Hierarchical Ontologies

It was planned to change the Ontology/Taxonomy into a hierarchical structure, where different applications made use of a common taxonomy. In the current implementation this change has not been done. This does not change the functionality of the system, but would have had a large impact on the definition of new apps. It is unlikely that this adaptation of the Ontology/Taxonomy will be implemented within the project, as other features have been judged to be more important.

6.3 Security

Currently the connection between the MMDI frontend and the MMDI Backend is a websocket connection. Preparatory work has been done to replace the websocket connection with a WSS connection (Web Socket Secure), but the integration remains to be done. Also other security measures (e.g. nonces) need to be implemented. The work is expected to be carried out before the project ends.

7 Summary

The MMDI 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 the user's utterances into app calls.

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

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

All three "must have requirements" defined for the task in the deliverables D2.3 "Requirements Analysis Report" and D3.2.1 "Functional Specification" are completely fulfilled. About 50% of the "should have and could have requirements" are fulfilled.

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

The current version of the implementation has some limitations, most notably the insecure connection between the frontend and the backend. Most of the limitations (including the secure connection) will be addressed in the "integration and bug-fixing" phase of the project.