WP4 – Mobility-related Data as a Service

D4.5.2: Media Data Streams and Data Prefetching Prototype II

Deliverable Lead: ASC

Contributing Partners: IBM, TUV

Delivery Date: 21.04.2015

Dissemination Level: Public

Version 1.0

This deliverable describes the work carried out during the development of the final prototype of the Media Data Streams and Data Prefetching 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	Kristof Kipp, ASC
Internal Reviewer 1	Daniel Burgstahler, TUDA
Internal Reviewer 2	Vadim Petrenko, TIE
Type	Deliverable
Work Package	WP4: Mobility-related Data as a Service
ID	D4.5.2: Media Data Streams and Data Prefetching Prototype II
Due Date	31.03.2015
Delivery Date	21.04.2015
Status	For Approval

Document History	
Contributions	V1.0, Kristof Kipp, ASC, 31.03.2015, Incorporated internal reviews and added the prototype zip archive V0.7, Stefan Schulte, TUV, 08.04.2015, Incorporated internal reviews V0.6, Kristof Kipp, ASC, 31.03.2015, Incorporated internal reviews V0.5, Kristof Kipp, ASC, 30.03.2015, Formatting and final changes V0.4, Stefan Schulte, TUV, 30.03.2015, Added paragraph to the summary V0.3, Xavier Cases Camats, WORLD, 27.03.2015, Added paragraph to the summary. V0.3, Xavier Cases Camats, WORLD, 24.03.2015, Added information about the Media Data Streaming API V0.2, Stefan Schulte, TUV, 16.03.2015, Added Prefetching Algorithms Sections V0.1, Kristof Kipp, ASC, 22.02.2015, Added document structure.
Final Version	April 21st, 2015

Note

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

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

This deliverable describes the work which was carried out during the development of the final prototype of the SIMPLI-CITY Media Data Streams and Data Prefetching component. For this, this document starts with introducing the Media Data Streams and Data Prefetching functionalities and describing the scope of the final prototype.

Afterwards, the degree of fulfilment of each requirement to be covered by the component and specified in the Requirements Analysis Report (D2.3) is described.

The current version of the Media Data Streams and Data Prefetching component contains the methods to transcode media files in different file sizes (which will provide different levels of audio quality to the customer) and offer them to consumers via a RESTful interface and native Java APIs with the Media Playback API and the Data Prefetching API. The Prefetching Algorithms provide the means to decide which data services are to be invoked at which time of a journey (e.g., before a trip or while being on the road). The data will be stored while making use of the Prefetch Proxy, which stores requests and their respective responses.

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 Media Data Streams and Data Prefetching component will be presented. The document accompanies the corresponding software prototype, which is the main content of the deliverable.

1.1 SIMPLI-CITY Project Overview

Analogously to the "App Revolution", SIMPLI-CITY adds a "software layer" to the hardware-driven "product" mobility. SIMPLI-CITY will take advantage of the great success of mobile apps that are currently being provided for systems such as Android, iOS, or Windows Phone. These apps have created new opportunities and even business models by making it possible for developers to produce new apps on top of the mobile device infrastructure. Many of the most advanced and innovative apps have been developed by players formerly not involved in the mobile software market. Hence, SIMPLI-CITY will support third party developers to efficiently realise and sell their mobility-related service and app ideas by a range of methods and tools, including the Mobility Services and App Marketplaces.

In order to foster the wide usage of those services, a holistic framework is needed which structures and bundles potential services that could deliver data from various sources to road user information systems. SIMPLI-CITY will provide such a framework by facilitating the following main project results:

- Mobility Services Framework: A next-generation European Wide Service Platform (EWSP) allowing the creation of mobility-related services as well as the creation of corresponding apps. This will enable third party providers to produce a wide range of interoperable, value-added services, and apps for drivers and other road users.
- Mobility-related Data as a Service: The integration of various, heterogeneous data sources like sensors, cooperative systems, telematics, open data repositories, people-centric sensing, and media data streams, which can be 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 provide the means to use the final prototype of the Media Data Streams and Data Prefetching component and exploit its functionalities. For this, the scope and requirements of the component, the requirements and preparations for users, and an installation and usage guide are provided.

The final Media Data Streams and Data Prefetching prototype is the outcome of the discussions and implementation work done in project months 19 to 30. It provides the final implementation of the functionalities as discussed within SIMPLI-CITY deliverables D3.2.1 (Functional Specification), and D3.2.2 (Technical Specification).

1.3 Document Status and Target Audience

This document is listed in the Description of Work (DoW) as “Public”. It provides the means to exploit the functionalities of the SIMPLI-CITY Media Data Streams and Data Prefetching component as defined in deliverable D3.2.2 (Technical Specification).

While the document primarily is aimed at the project partners, this public deliverable can also be useful for the wider scientific and industrial community. This includes other publicly funded projects, which may be interested in collaboration activities.

1.4 Abbreviations and Glossary

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

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

1.5 Document Structure

This deliverable is broken down into the following sections:

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

Section 2 provides an overview of the scope and relationship of the prototype, showing where the Media Data Streams and Data Prefetching component 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 2 additionally provides a description of technical changes and decisions that have been made after the first prototype.

Section 3 presents the requirements and preparations to be done by software developers if they want to make use of the final prototype of the Media Data Streams and Data Prefetching component.

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 provides a summary of the document and the final comments for Task 4.5.

2 Prototype Scope and Requirements Coverage

In this section the scope of the final prototype and its coverage of the requirements defined in deliverable D2.3 are discussed.

2.1 Media Data Streams and Data Prefetching – General Information

The Media Data Streams and Data Prefetching realizes the means for media handling in terms of media data streaming and media prefetching (e.g., for music streaming). Secondly, this component provides service prefetching functionalities (for pre-invocation of data services and backend services).

Media streaming and media data prefetching may be used to create apps that support the playback of music or other media information. By nature, the consumption of media is sensitive to interruptions: Even a connectivity loss of a few seconds in the middle of a song the user is listening to will give the user a negative experience. For this purpose, the Media Data Streams and Data Prefetching component will integrate a media buffering solution by prefetching relevant data in a local buffer.

Data prefetching takes into account which data will be most likely needed by the user in the future and might be prone to connectivity losses. Based on this information, data services are pre-invoked and the results are stored in a cache. Consequently, future service invocations can be “re-directed” to the cache – instead of invoking the actual service, data from the cache is used. Of course, this is only feasible in particular situations, as will be explained in Section 2.3.2.

Figure 1 shows the location of the Media Data Streams and Data Prefetching in the SIMPLI-CITY Global Architecture. The Media Streams API (as shown in Figure 2) subcomponent of T4.5 is not separately depicted in the Figure, because it is a special case of prediction fetching the SIMPLI-CITY Personal Mobility Assistant (PMA) makes use of. For the full Global Architecture, refer to deliverable D3.1.

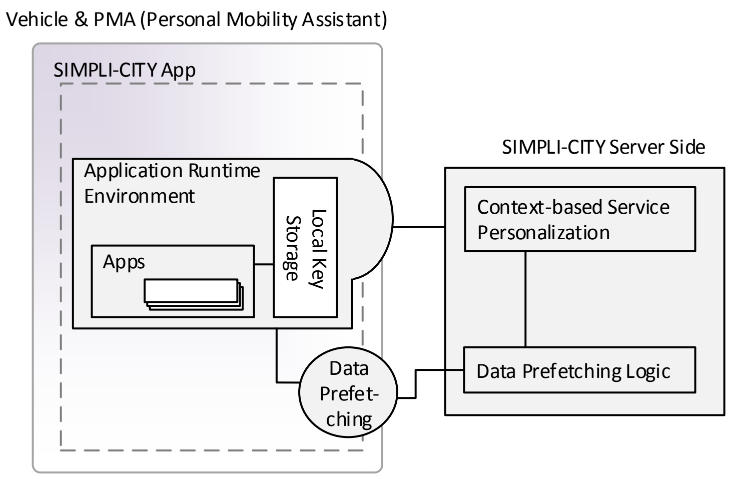

Vehicle & PMA (Personal Mobility Assistant)
SIMPLI-CITY App
Application Runtime Environment
Apps
Local Key Storage
Data Prefet-ching
SIMPLI-CITY Server Side
Context-based Service Personalization
Data Prefetching Logic

Figure 1: Location of the Media Data Streams and Data Prefetching Component in the SIMPLI-CITY Global Architecture

2.2 Scope of the Final Prototype

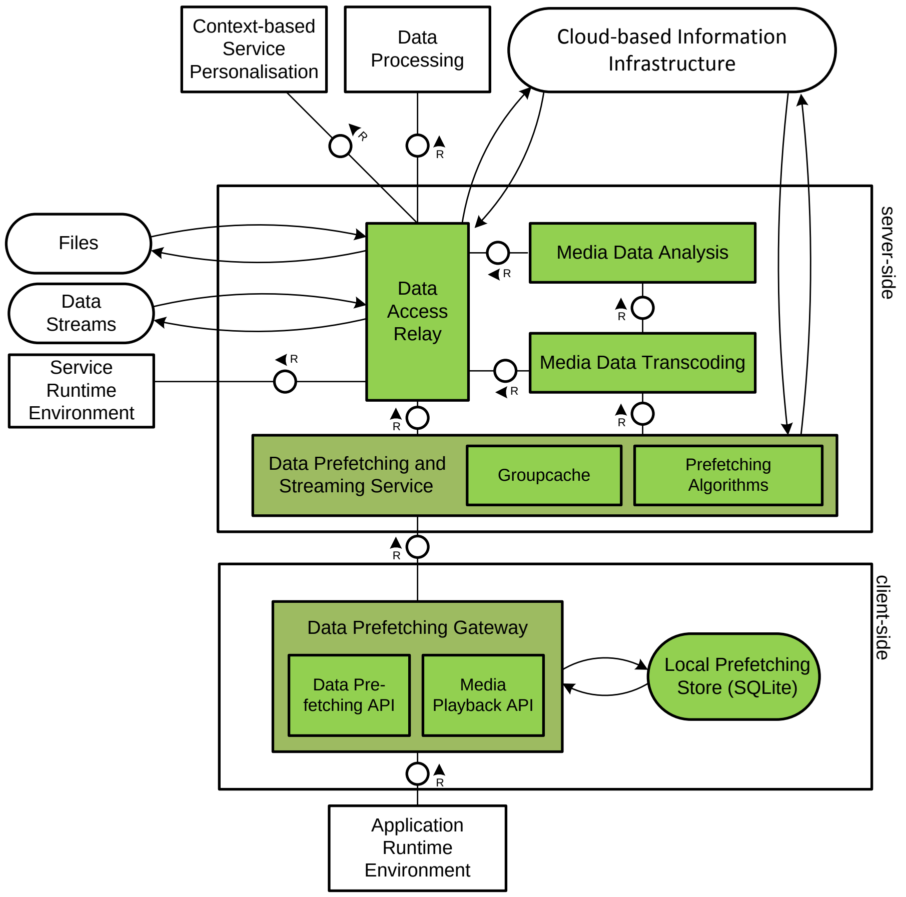


Figure 2: Scope of the Final Prototype of the Media Data Streams and Data Prefetching Component

Figure 2 depicts the status of development of the final prototype of the Media Data Streams and Data Prefetching component, showing the subcomponents that are covered within this final prototype.

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

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

In the following subsections, the scope and status of the single subcomponents 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 Media Data Analysis

The Media Data Analysis is part of the media streaming functionality. A media data stream invoked by the Media Data Transcoding can use this component to analyse the media data. The Media Data Analysis can identify music and speech from an audio data stream and handle them in different ways. The adaptive normalizer can detect the base quality of a media stream and create copies of the stream with different encoding options (e.g., lower bitrate to create smaller files). With this information, the component is able to make the best choice of encoding for the current bandwidth of the PMA. As decided in Section 5.4.3 in deliverable D3.2.1, this subcomponent was implemented in the programming language Go and makes use of the ffmpeg[1]/libav[2] toolkit to analyse the multimedia resource.

While developing the final prototype, the standard library of the Ubuntu Linux distribution has changed and ffmpeg was replaced by libav, therefore a change of commands were made according to the recommendations of the Ubuntu development team. This has only a small influence on the installation and deployment of the T4.5 component.

2.2.2 Media Data Transcoding

In general, media streams require a very high data bandwidth to encode the media data, therefore the Media Data Transcoding component has to cope with a huge data throughput. The Media Data Transcoding transforms the media stream into another encoding (probably with a lower bitrate) if monitoring, continuously performed by the Media Data Analysis, suggests that it will help to guarantee a continuous playback on the device (here: the PMA). To improve performance, this subcomponent calls the Data Access Relay before it starts transcoding and provides the transcoded multimedia data to the Data Access Relay. The Media Data Transcoding subcomponent is configured to cache its data in the Groupcache subcomponent of the Data Access Relay. Thus, if more than one user consumes the same media resource in a similar network environment, the media resource is transcoded by the Media Data Transcoding only once. As has been decided in Section 5.4.3 in deliverable D3.2.1, this subcomponent was developed in the Go programming language and the Go Media Framework is used for the transcoding.

2.2.3 Data Access Relay

The Data Access Relay provides caching functionalities to the media-streaming infrastructure of the server-side Media Data Streams and Data Prefetching component. If consuming multimedia data, data throughput peaks on some multimedia resources are common. It is anticipated that several users use the same multimedia resources – this fact is exploited in order to reduce redundant work for the Data Access Relay. The Data Access Relay is applied to minimize the traffic between the Media Data Streams and Data Prefetching Logic component and external multimedia resources and reduces the load from the Media Data Transcoding subcomponent. Moreover, already transcoded media data is cached in the Data Access Relay to reduce the load on the Media Data Transcoding subcomponent.

2.2.4 Data Prefetching and Streaming Service

While developing the Data Prefetching and Streaming Service several major technical design decisions (which were not addressed within D3.2.2) had to be made. The access to the media files needed to be handled in a fast manner due to a predicted access rate of more than a thousand requests per second. This relatively high number is given by the fact that media files are stored in chunks, which are accessed and streamed on demand.

The Data Prefetching subcomponent of SIMPLI-CITY aims at mitigating the problem that mobile implementations, such as the PMA, are typically subjected to network fluctuations

[1] https://www.ffmpeg.org/
[2] https://libav.org/

or intermittent downtimes. This can be illustrated using an example of a service consumer (here: the PMA) driving through a mountain tunnel or simply in an area with bad cellular coverage. Data Prefetching itself is a type of precaching, i.e., data is put into a cache before it is actually needed. To achieve this within SIMPLI-CITY, a Prefetching Algorithm needs to take into account the future context of the PMA and also the data needs of the user. Based on this information, the algorithm is able to decide which data to prefetch at which point of time in order to enhance the user experience. Notably, prefetching should be possible for all kinds of data types, but is naturally especially critical for streaming data for providing uninterrupted playback.

2.2.5 Data Prefetching API and Media Playback API

These components (Data Prefetching API and Media Playback API) are not concerned in this deliverable, because they highly depend to the App implementations of the SIMPLI-CITY Application Runtime Environment, which is not yet finalized at the time of this deliverable. The work on these subcomponents will be intensified while implementing the real world use cases for WP7 and WP8.

2.3 Technical Decisions

This section discusses important technical decisions that had to be made after the first prototype. These decisions include the storage of the media files that will be streamed to the customers based on different parameters that will be explained in the next sections.

2.3.1 Media Streaming Technology

The technology to stream the audio data to the client device (the PMA) is the most important factor for the Media Data Streams component. It needs to support multi-channel streams, which are not limited to a single song, but offer the means to dynamically open new channels for listeners on the fly. This requirement creates a 1:n-relationship between the service and the possible large number of clients, which is an important requirement for successful realization of this component.

A technology for media streaming needed to be chosen – as there are many different methods of streaming data to a mobile device. Almost all of the common technologies offer static media streams that use a single channel for receiving and transmitting of data. As stated before, media streaming needs to support multi-channel streams, which is not foreseen in existing components. Therefore the only possible solution that is currently available – apart from pure connections (e.g., TCP or UDP Sockets) – is an RTMP (Real Time Messaging Protocol) server. RTMP supports dynamic channels that are opened as soon as a transmitter writes data into it.

As the RTMP protocol makes use of a centralized server and is just the backend for streams, the Media Data Streams subcomponent needs to dynamically open a channel and write media data into the created channel. This channel creation and writing is done with a content provider, which is usually an ffmpeg/libav process. The content provider process is controlled by the Media Data Streams service.

2.3.1.1 Streaming of Media Data

Upon opening a channel there are two possible ways of writing the chunk data to the opened channel, given the length of a chunk is 10 seconds:

a) Write every chunk on its own, meaning having to open a new RTMP connection every new chunk.
b) Have a FIFO (First In, First Out) pipe which is used as the source of the stream and constantly write the chunk data into the pipe.

A FIFO pipe is an organization method in operating systems and programming languages, where a dynamic file is created that has data providers (methods and processes that write data into the pipe) and consumers (which read the data) – therefore possibly creating endless file streams.

The first method constantly opens new streams with the same name, having the data stream consumer to reopen the connection every chunk length (here: 10 seconds) – which creates a massive overhead. This overhead may not be readable in time as the connection on the PMA may be too low to keep up with the needed bandwidth.

Using a FIFO pipe, the content provider (the Media Data Streams component) can open a dynamic channel using this pipe as an input and the RTMP server channel as the output and therefore dynamically adjust the quality of the song by writing the data directly into the pipe without having to re-establish the connection on every quality change or new chunk, which is done by having multiple bitrate-versions of a song ready for writing. The PMA keeps track of the currently available bandwidth, which is send to the Data Prefetching and Streaming Service via the Media Playback API. The Data Prefetching and Streaming Service then chooses the best suiting bitrate for the available bandwidth and uses the already opened connection to send the data to the client.

2.3.2 Prefetching Algorithms

To compensate for connection losses or bad connection quality, it is imperative to prefetch data and buffer it locally. Prefetching is the technique of querying or gathering any kind of data or service functionality before the moment it is actually needed or used. It is related to caching, not only since caching and prefetching are often combined, but also since the prefetched data have to be stored in caches or similar modules. In SIMPLI-CITY, data is consumed using the notion of services. Hence, in the following it is assumed that different data services are available, which could be prefetched, i.e., a service could be invoked before the actual time the service response is needed. The result of the service response is stored in a cache and the result is used once the actual service invocation should take place.

It should be noted that the modular structure of the SIMPLI-CITY Media Data Streams and Data Streaming subcomponent allows the replacement of the underlying Prefetching Algorithms. During the course of SIMPLI-CITY, a number of algorithms have been implemented. For more details of the Prefetching Algorithms, please refer to [HSH+14]. In general, all data services, which are consumed through the OSGi-based SIMPLI-CITY Service Runtime Environment, may be prefetched.

2.3.2.1 Data Types

One prerequisite for data prefetching is the definition of a taxonomy of data types, since it does not make sense to prefetch any kind of data at any point of time. Hence, for the purposes of prefetching, data is categorized using the following categories:

- Importance (high/medium/low): Signifies the importance of the particular data service. Since different users may attribute different levels of importance to the same data service, this category can be customized to cater for the individual user needs. For instance, some users may want to download traffic information updates in any context using the full available bandwidth for this, while other users do not necessarily appoint such a high importance to this kind of information.

- Time-Criticality (high/medium/low): Describes the time criticality of the particular data service. For instance, a traffic update data service is highly time-critical, since the data is out-dated very soon. In contrast, a media streaming data service may only feature a medium level of time-criticality.
- Access Pattern (e.g., streaming, push /poll, on demand, recurrent, polling): Represents the access patterns for a particular data service. As mentioned above, streaming data services are the focus of this SIMPLI-CITY component, but per se, other access patterns are also supported.
- (Pre-)Fetching Strategy (timely updates, pre-compute, pre-load for route, pre-load & cache, postpone if required): Describes different prefetching strategies that could be chosen based on the individual needs of a user.

As it can be seen, the different categories may lead to conflicting prefetching decisions. For example, a traffic update data service may be categorized as highly important (indicating a need for prefetching), but is also time-critical (indicating that there is no improvement in user experience if the service is prefetched). To find an optimal solution taking into account the different categories is therefore crucial.

2.3.2.2 Prefetching Scheduling Strategies

It is assumed that a data service is available in a given context: (i) Either if the network quality is sufficient to make a service invocation at the time it is required or (ii) the service request had been prefetched before. The goal of this Prefetching Algorithm is to prefetch all data services whenever requested. This implies that all requests that are scheduled for a time where the network is unavailable (or the network quality is insufficient) need to be prefetched. As a further, soft constraint for optimizing the prefetching scheduling, more time-critical data services are requested at the latest possible time. Also, precedence is given to more important data services. Two particular prefetching strategies, described below, have been implemented.

The basic type of prefetching is to periodically invoke the target data service in fixed or predefined time intervals (*Periodic Prefetching Strategy*). Periodic prefetching is not well-suited for services with high time-criticality, because prefetching may be scheduled too early (i.e., could be scheduled closer to the time when the result is actually needed). Moreover, this strategy is sub-optimal with respect to network usage, since it may perform unnecessary prefetch invocations (i.e., results which are never used). Yet, the strategy can be applied when the context evolution is not known in advance, i.e., there is no information about how the context of the user or the device are going to evolve in the future. This is the case if, e.g., the planned route of the vehicle is not available.

On the other hand, taking the context of the user or device into account will usually lead to more accurate results (*Context-Aware Prefetching Strategy*). With context-aware prefetching, it is necessary to know in advance the evolution of the user context (or at least the relevant portion of it). The future context information is used to reveal upcoming problems in connectivity and allows prefetching in a timely manner. Consequently, this leads to less requests and a higher degree of freshness of the prefetched data. Moreover, this strategy allows creating context-specific prefetching requests.

2.4 Covered Requirements

This section describes the degree of fulfilment of the requirements to be covered by the Media Data Streams and Data Prefetching and specified in the Requirements Analysis deliverable (D2.3) and the Functional Specification (D3.2.1).

Table 6: Requirements Related the Media Data Streams and Data Prefetching Component and their Degree of Fulfilment

Requirement	Degree of Fulfilment	Comment
Must Have Requirements		
U50: Prefetching of media data + Offline access	100%	The basis for prefetching is done by having several bitrates of a single media file available, the division of a media file in chunks completely done and these can be accessed at any time.
U113: Handling of multimedia data	100%	Meta information analysis is feature complete, handling and analysing all kinds of media files.
U90: Availability	100%	The availability of this component is provided by the Data Prefetching Proxy, which is running seamlessly in the background and the Media Streaming subcomponent within the scope of this prototype.
U91: Integrity	100%	Media Streams are natively available for Android based systems. The API is going to be developed within the scope of the Use Cases for better integration into the eco-system of the Personal Mobility Assistant.
U92: Secure access to system	100%	Prefetched data is stored within the mobile device and securely available to the Prefetching component. Media Data is not explicitly encrypted due to the fact that It needs to be highly available.
U93: Third party access to the system	100%	The Media Streams REST service is publicly available, while the Prefetching Proxy is running on the PMA itself without any open interfaces to the public.

Requirement	Degree of Fulfilment	Comment
Should Have Requirements		
U51: Avoid the download of data from 3G	100%	The Prefetch Proxy is capable of caching data that is likely going to be requested. Hence, it can be avoided to download data via 3G if this is defined in a Prefetching Algorithm.
U52: Offline access of data used within apps	100%	If feasible and meaningful, any data may be downloaded prior to its actual usage. However, as discussed in Section 2.3.2, this is not the case for all types of data.
U54: Expiration of data	100%	Using the Prefetch Proxy, the cached data is stored for a certain amount of time and will be deleted as soon as the data may be irrelevant.
Could Have Requirements		
U53: App recommendations	0%	Not considered in this prototype due to budget constraints and low importance of this requirement.

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.

3.1 Server Side

In order to deploy the Server Side part of the Media Data Streams and Data Prefetching component, it is necessary to have a working environment for the Go programming language[3] running. This prototype was developed using Ubuntu 13.10 with Go 1.2. Making use of Ubuntu built-in dpkg (Debian Package) and APT (Advanced Packing Tool) utilities makes the installation of the Go programing language straightforward.

Before installing the needed packages it is recommended to update the package index of APT. This is done in a command line window via the following command:

```
sudo apt-get update
```

To have a fully working environment available, it is required to run the following command with a user that is eligible to use “sudo” – this installs the go programming language runtime and a library to split the media files, which is used within the component:

```
sudo apt-get install golang mp3splt
```

Apart from the above mentioned packages there is one additional dependency for this prototype that needs to be installed. The commands are avprobe and avconv, which are part of the libav package. Despite choosing ffmpeg for the conversion and analysis in deliverable D3.2.2, libav replaced ffmpeg as the standard library in Ubuntu 12.04 and is therefore used in this prototype. libav is installed via:

```
sudo apt-get install libav libav-tools
```

To handle the RTMP requests and responses, the server needs to run a service that integrates into the process-line of the Media Streaming component. For convenience and the reason of being free software, the web server nginx, with a special RTMP server module, has been chosen. To install the respective service and the module the server must be prepared by installing the following packages that nginx lists as dependencies.

```
sudo apt-get install build-essential libpcre3 libpcre3-dev libssl-dev
```

Inside the home directory (`cd ~`), the nginx source code needs to be downloaded:

```
wget http://nginx.org/download/nginx-1.7.10.tar.gz
```

As of this writing, the latest stable version of nginx is 1.7.10.

The RTMP module for nginx needs to be downloaded into the same directory as the nginx archive:

```
wget https://github.com/arut/nginx-rtmp-module/archive/master.zip
```

Both archives (the nginx service and the RTMP module) need to be unpacked. The following listing explains how to unpack the archives and enter the nginx directory:

[3] http://golang.org/

```
tar -zxvf nginx-1.7.7.tar.gz

unzip master.zip

cd nginx-1.7.7
```

The source now needs to be configured for the current system architecture and built with the corresponding compiler. The following listing explains all the steps that need to be taken to configure, build and install the modified nginx server:

```
./configure --with-http_ssl_module --add-module=../nginx-rtmp-module-master

make

sudo make install
```

3.2 Client Side

This section provides the preparations that need to be done to make use of the subcomponents on the client side of the final prototype of T4.5.

3.2.1 Prefetch Proxy

To make use of the Prefetch Proxy the mobile Android Device has to be put into developer mode. Since SIMPLI-CITY partners agreed on making use of the LG Nexus 4, the following list explains how to enable the developer mode on this very device. The steps on other Android phones should generally be similar.

- The device settings menu needs to be opened. This can be done by pressing the Menu button while being on the home screen and tapping the “System settings” icon.
- The option “About phone” needs to be tapped.
- At the “About” screen, the “Build number” needs to be tapped seven times.

At this point the device is in Developer Mode and able to install the Prefetch Proxy.

4 Installation (Deployment)

This section provides guidelines on how to install and deploy the final prototype of the SIMPLI-CITY Media Data Streams and Data Prefetching component on a server as well as clients (i.e., an Android device or an Android device simulator). These different target systems are separately described in the following sections.

4.1 Server Side

The server side handles all the packages and subcomponents that are not visible to the user, but offer the functionality the clients need to make use of the Media Data Streams and Data Prefetching prototype. The following instructions are made for Debian Linux and Debian based distributions, i.e., Ubuntu or Linux Mint. Installation steps may differ when using other distributions that use other package managers.

4.1.1 Media Data Streaming

The prototype is delivered alongside a demo webserver, which is part of the archive provided with this deliverable. Before the server is started, the $GOPATH environment variable needs to be set within a terminal window, this includes two steps:

```
cd <projectpath>/src

export GOPATH=`pwd`
```

Afterwards the missing Go packages need to be installed. This is automatically done by the golang toolset:

```
go get
```

After installing all the missing packages the prototype is ready to run. The compilation and initiation of the prototype is done via the command:

```
go run main.go
```

Now the webserver is running on port 3000 of the local machine, so it is accessible via a web browser. The URL to access the server is “http://localhost:3000”.

4.1.2 nginx

By default nginx is installed to `/usr/local/nginx`, so to start the server following command needs to be run:

```
sudo /usr/local/nginx/sbin/nginx
```

To make sure the nginx service is running, the following command can be executed:

```
curl http://localhost/
```

In case `curl` is not installed, the URL (`http://localhost/`) can be opened inside a web browser.

The nginx configuration file needs to be modified in order to enable the Media Data Streams and Data Prefetching component is able to stream the media. The configuration file is located by default at `/usr/local/nginx/conf/nginx.conf`. The lines shown in Listing 1 need to be added at the end of this file.

Listing 1: Lines to be Added to Configuration File of nginx

```
#user nobody;
worker_processes 4;

#error_log logs/error.log;
#error_log logs/error.log notice;
error_log logs/error.log info;

#pid logs/nginx.pid;


events {
     worker_connections 1024;
}

rtmp {
     server {
          listen 1935;
          ping 20;
          chunk_size 4096;
          publish_time_fix off;
          play_time_fix off;

          application simcity {
               live on;
          }
     }
}
```

This configuration was used in the development progress. After saving the configuration file, the nginx service needs to be restarted for the changes to take effect.

```
sudo /usr/local/nginx/sbin/nginx -s stop

sudo /usr/local/nginx/sbin/nginx
```

4.2 Client Side

The Prefetch Proxy is part of D6.3.2 (SIMPLI-CITY Mobile Application Runtime Environment Prototype II) and will be installed through following the instructions found in public deliverable D6.3.2.

5 Execution and Usage of the Software

This section describes how to use the different subcomponents of the prototype after they have been installed on their respective devices. All data examples used in this deliverable are not part of the deliverable package due to license reasons.

5.1 Analysis and Transcoding

The Analysis and Transcoding components are independently working packages, which can be used by the means of the Go programming language. Since they are not part of the execution of the prototype, these packages will not be described any further.

5.2 Streaming REST API

This component provides a REST API as the main communication method for the Apps of the Personal Mobility Assistant. This API provides simple stream control methods to get a list of available streams, to start and stop a stream and to change the bandwidth available on the device.

5.2.1 getStreamList

The getStreamList function returns the list of all songs that are instantly available for streaming. The listing below shows a sample response, which contains a list of three songs, identified by the ID field and with the artist and title names as additional fields. These fields may contain spaces, even though the example in Listing 2 does not make use of it.

Listing 2: JSON Response of the Stream List

```
[
   {
      "Id": "840b6e2193cd7fcd5a1057e928d38c95",
      "Artist": "Weezer",
      "Title": "HashPipe"
   },
   {
      "Id": "e50775d37e96bc727f2d1ea928928aa8",
      "Artist": "SilversunPickups",
      "Title": "LazyEye"
   },
   {
      "Id": "e8f94e354e8d8d292e6ae2e24277562a",
      "Artist": "ArcticMonkeys",
      "Title": "Brianstorm"
   }
]
```

5.2.2 setBandwidth/:id/:kbps

This function takes an ID as a parameter to identify the stream to change the bandwidth for and the current bandwidth. This will immediately take effect on the stream provided to the mobile device. This method returns “ACK” if the request is valid and the corresponding error message if the request is invalid.

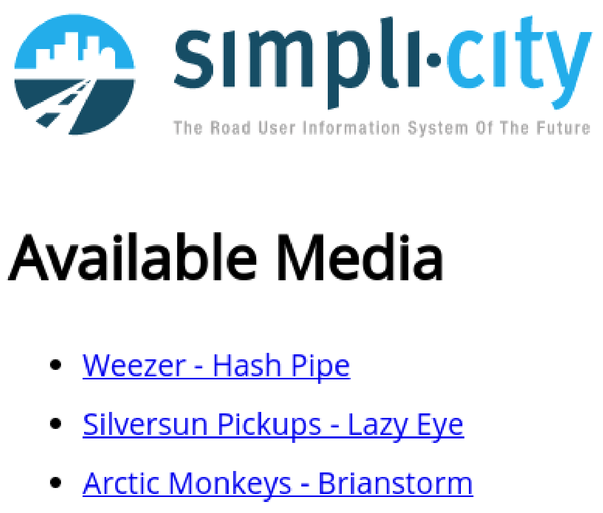
5.2.3 play/:id and stop/:id

To start a stream, the app needs to get an ID from the getStreamList method; this id is then used by the play method to generate a media-file/PMA-ID correlation and returns this combination as a unique ID, which is used to control a specific stream. The controlling of a specific stream is done by the play- and the stop-method of this API. This method returns "ACK" if the request is valid and the corresponding error message if the request is invalid.

5.3 Streaming Website

The Streaming Website is only for showcase purposes and was created for the second review meeting in December, 2014. It shows the use of the getStreamList method in a readable format and the possibility to start a stream with the play method. After clicking on one of the links the browser is redirected to a stream-view page which includes a slider to change the bandwidth of the current stream. This website is part of the package included within this deliverable.

The following figures were taken using Google Chrome (Version 41.0) on Ubuntu Linux 14.10 – the results may vary depending on the Browser and Operating System used.


simpli·city
The Road User Information System Of The Future

Available Media

- Weezer - Hash Pipe
- Silversun Pickups - Lazy Eye
- Arctic Monkeys - Brianstorm

Figure 3: Demo Website – List of Available Media

As seen in Figure 3 the website shows a list of available songs. When clicked on a song a new page will open similar to the one depicted in Figure 4.

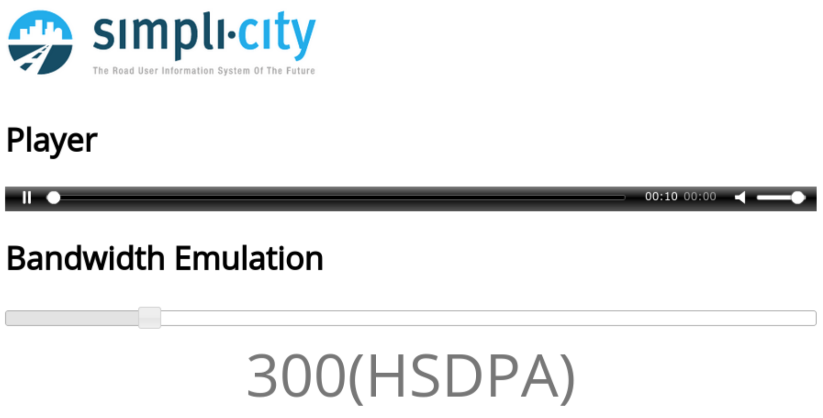


Figure 4: Demo Website – Running Stream

The Page depicted in Figure 4 offers the means to control the running stream by adjusting the virtually available bandwidth to demonstrate the bandwidth adaptation of the streaming component.

5.4 Prefetch Proxy

The Prefetch Proxy does not provide a single executable App, which is going to be used for execution. This functionality is transparently provided by a service that is running in the background of the PMA. It is automatically used by SIMPLI-CITY Apps that are running inside the PMA. Therefore, this section does not provide a particular use-case or example for usage of the Prefetch Proxy.

6 Summary

The final Media Data Streams and Data Prefetching prototype offers capabilities to stream media to the SIMPLI-CITY PMA and to store prefetched data on the PMA.

The Media Data Streams component automatically converts media files into a common output format and can start RTMP streaming upon invocation. This format is supported by any kind of clients, which includes the PMA. The PMA includes the functionality of the Media Data Streams component and wraps it around the Media Playback API. This API will be usable within native source code that is used by app developers.

The Data Prefetching algorithm provides the logic to decide at what points of time which particular data items should be prefetched. It is especially aiming at mobile users and road users, which are subject to fluctuating mobile networks. Different prefetching strategies are covered; the actual selection of a strategy is based on the importance of the data, its time-criticality, and the data access pattern. Thanks to SIMPLI-CITY's loosely coupled architecture, the prefetching algorithm may be replaced by other algorithms which may be more sufficient in different environments. Also, it is possible to extend the current algorithm.

In addition to what has been discussed in this deliverable, the Media Playback API and the Data Prefetching API will also provide a set of functionalities to be able to manage the streams (e.g. buffering, changing bitrate, controlling of streams) on the PMA. All these functionalities will be implemented in a manner that allows PMA application developers to understand and use them in a very easy way. As these APIs have a tight relationship with the app development, they will be delivered in the upcoming deliverables D7.2 and D8.2.

References

[HSH+14] W. Hummer, S. Schulte, P. Hoenisch, and S. Dustdar, “Context-Aware Data Prefetching in Mobile Service Environments,” in The 4th IEEE International Conference on Big Data and Cloud Computing (BDCloud 2014), Sydney, Australia, 2014, pp. 214-221.