Reality Sensing, Mining and Augmentation
for Mobile Citizen–Government Dialogue

FP7-288815

D4.3

Report on the Phase 2 Integrated toolset and AIV Test Report

Dissemination level:	Public
Contractual date of delivery:	Month 33, 31-10-2014
Actual date of delivery:	Month 33, 31-10-2014
Work package:	WP4 System integration
Task:	T4.3 – Technical verification with test cases environments
Type:	Report
Approval Status:	~~Draft~~ | PMB Final Draft | ~~Approved~~
Version:	1.6
Number of pages:	96
Filename:	20141029-D4.3-Report on the Phase 2 Integrated toolset and AIV Test Report-delta.docx

Abstract

This deliverable describes the Live+Gov toolset and test results of the integrated toolkit as well as the test results of the three eGovernment solutions (Mobility, Urban Maintenance, Urban Planning) developed with this toolkit during the 2nd phase of the project. Developers can use this integrated toolkit examples and test plans to develop or customize their own Mobile eGovernment solutions.

This work was supported by the EU 7th Framework Programme under grant number IST-FP7-288815 in project Live+Gov (www.liveandgov.eu)

Copyright

History

Version	Date	Reason	Revised by
0.1	01-10-2014	Initial alpha version to set outline	Frank Thiele
0.2	15-10-2014	Introduction, purpose, audience and lay-out described.	Frank Thiele
0.4	13-10-2014	Update of Sensor Data Capturing Service and Reality Mining Service. Test plans for C8, C9.1, C9.2, C14 and C15.	Heinrich Hartman
0.5	14-10-2014	Added SaaS Service Center Chapter including test plans and results.	Frank Thiele
0.6	14-10-2014	Added Traffic Service, including test plan and results.	Pekka Kaarela Laura Niittlyä
0.7	15-10-2014	Update of eGovernment Dialogue and Visualisation Service. Update of Issue Reporting Service	Frank Thiele Pieter Minnigh
0.8	15-10-2014	Live+Gov Toolkit overview, approach & architecture.	Frank Thiele Pieter Minnigh
0.9	16-10-2014	Summary of test strategy and overview of performed test reports and results.	Frank Thiele
0.10	17-10-2014	Added Personalized Content Delivery Service. Update of Augmented Reality Service, including test plans and results.	Dimitrios Ververdis Spiros Nikopoulos George Liaros
0.11	17-10-2014	Management summary	Frank Thiele
1.0	22-10-2014	Beta review: Spelling correct and multiple comments for further improvements (to the parts after page 21). Major updates in the AR section (3.4)	Dimitios Verderdis Spiros Nikopoulos
1.0	22-10-2014	Beta review performed: Spelling correction and multiple comments for further improvement.	Pekka Kaarela
1.1	23-10-2014	Processed overall beta review comments, final editing on Issue Reporting Service,	Frank Thiele
1.2	27-10-2014	Final editing on of Augmented Reality Service and Personalized Content Delivery Service	Dimitios Verderdis
1.3	27-10-2014	Final editing on Traffic Service	Pekka Kaarela Laura Niittlyä
1.4	29-10-2014	Final editing on Sensor Data Capturing Service and Reality Mining Service	Heinrich Hartmann
1.5	30-10-2014	Overall editing, Summary and Conclusions	Frank Thiele Pieter Minnigh
1.6	31-10-2014	Final editing	Matthias Thimm

Author list

Organization	Name	Contact Information
YCT	Frank Thiele	f.thiele@yucat.com
YCT	Pieter Minnigh	p.a.minnigh@yucat.com
YCT	Feike Kramer	f.kramer@yucat.com
CERTH	Spiros Nikopoulos	nikolopo@iti.gr
CERTH	Dimitios Verderdis	ververid@iti.gr
CERTH	George Liaros	geoliaros@iti.gr
UKob	Heinrich Hartmann	hartmann@uni-koblenz.de
UKob	Christoph Schäfer	chrisschaefer@uni-koblenz.de
UKob	Matthias Thimm	thimm@uni-koblenz.de
MTS	Laura Niittlyä	laura.Niittyla@mattersoft.fi
MTS	Pekka Kaarela	pekka.kaarela@mattersoft.fi

Executive Summary

This deliverable describes one of the end products of the project: the Live+Gov Toolkit, including the final test results confirming its readiness for deployment.

The toolkit consists of several applications and software libraries for creating mobile governance solutions, supported by SaaS back-end services and web applications. It offers developers reusable services and applications for sensor data collection, reality mining, augmented reality, object recognition, personalization, issue reporting, eGovernment dialogue and visualization. Furthermore advanced SaaS functionality like account management, access control, billing and diagnostics / monitoring are offered centrally to facilitate integration of the individual toolkit services

Two user groups are identified for exploitation of this toolkit: Developers and Service Providers. Developers can create their own Mobile Government Solution using the toolkit services as reusable building blocks. Service providers can offer standard customizable solutions to government officials, who would like to launch one of these solutions in their own city.

During the project three example solutions have been developed, tested, deployed and trialled in real life environments: A Mobility solution in Helsinki (Finland), an Urban Maintenance solution in Utrecht (The Netherlands) and an Urban Planning solution in Gordexola (Spain). These solutions show that the toolkit is generic, cross platform and can be applied in different scenarios and environment. These examples can be customized and deployed for other municipalities or used as an example by developers when creating or customizing their own solution.

During the Live+Gov project two Assembly-, Integration- and Verification cycles have been performed in a highly complex distributed environment with multiple companies developing and deploying reusable and integrated SaaS-based toolkit services. The followed strategy proved to provide the required quality assurance for the development and deployment of three different mobile eGovernance solutions on different trial sites. Future toolkit developers and users can adopt this strategy when developing their own toolkit extensions or SaaS solutions.

This deliverable describes the final Live+Gov Toolkit and an overview of the technical verification of the toolkit and developed solutions. More detailed information regarding the integration strategies and guidelines or the technical verification strategy can be found in *'D4.2 - System integration concept and guidelines'* [11] and *'D4.4 – Technical verification and testing strategies'* [6].

'D5.4 – Prototype demonstrator for second trials' [17] describes the three example solutions that have been developed with the toolkit. *'D5.5 – End results of trials and Live+Gov Methodology'* [18] bundles the best practices and lessons learned for adaption in future/other eGovernment initiatives.

Abbreviations and Acronyms

3D	Three Dimensional
AIV	Assembly, Integration and Validation
API	Application Programming Interface
AR	Augmented Reality
AREL	Augmented Reality Experience Language
ASP	Active Server Pages
CPMT	Citizen Participation with Mobile Technology
GPS	Global Position System
GSM	Global System for Mobile communications
GUI	Graphical User Interface
HSL	Helsinki Region Transport
HTML	HyperText Mark-up Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IBS	Image BaSed augmented reality
JSON	JavaScript Object Notation
LBS	Location BaSed augmented reality
M	Mobility
MAM	Municipal Administration Manager
MVC	Model View Controller
OS	Operation System
POI	Point of Interest
PHP	PHP: Hypertext Pre-processor
REST	Representational State Transfer
SaaS	Software as a Service
SDK	Service Development Kit
SLA	Service Level Agreement
SQL	Structured Query Language
SSF	Sensor Stream File format
StUF	Standaard Uitwissel Formaat
UM	Urban Maintenance
UP	Urban Planning
WiFi	Wireless Fidelity
WP	Work package
XML	eXtensible Mark-up Language

Table of Contents

List of Figures

List of Tables

1 Introduction

This chapter describes the purpose, intended audience and layout of this deliverable.

1.1 Purpose

This deliverable describes the final version of the Live+Gov Toolkit and the test results confirming its readiness for deployment. Both topics have been addressed in previous internal deliverables at earlier stages of the project. In order to sketch the full updated picture for the public, parts from the previous deliverables are included in summarized form. Table 1 gives an overview of the WP4 deliverables and their purpose.

Table 1: Deliverable overview

Deliverable	Purpose	Delivery date
D4.1 Report on Live+Gov toolkit requirements and architecture	Description of the first version of the Live+Gov Toolkit architecture, functional requirements, technical background, toolkit components and API's. D4.1 covers mainly task 'T4.1 – System architecture design'.	M18 Jul'13
D4.2 System integration concepts and guidelines	Description of the integration concepts and guidelines. The design of the central SaaS Service Center is present, including instructions and examples that explain how to integrate toolkit services with the central service. D4.2 covers mainly task 'T4.2 – Integration'.	M22 Nov'13
D4.4 Technical verification and test strategy	Outline of the test strategy, procedures and list of tests to be performed during the different phases in the project. A list of tests that have to be performed in each phase is presented, accompanied with the planning and responsibilities. Finally the report includes the preliminary test results until so far. D4.4 covers mainly task 'T4.3 – Technical verification within test cases environment'.	M22 Nov'13
D4.3 Report on Phase 2 Integrated toolset and AIV Test Report	Describes the Live+Gov toolset and test results of the integrated toolkit as well as the test results of the three eGovernance solutions (Mobility, Urban Maintenance, and Urban Planning) developed with this toolkit during the 2nd phase of the project. Developers can use this integrated toolkit and examples to develop their own Mobile eGovernance solutions.	M33 Oct'14

1.2 Audience

This document is meant for ICT professionals of the municipalities, service providers and developers who are interested in applying the Live+Gov Toolkit. It gives detailed information about the architecture and functionalities included the toolkit, including the available API's for development.

1.3 Layout

The layout of this document is structured as follows.

Chapter 2 introduces the Live+Gov toolkit, explaining the concept behind the Live+Gov Toolkit approach, linked to exploitation of the toolkit and its architecture.

Chapter 3 describes the different services offered in the toolkit into detail. The Services for SaaS Support (SaaS Service Center), Sensor Data Capturing, Reality Mining, Augmented Reality, Personalized Content Delivery, eGovernment Dialogue and Visualization, Issue Reporting and Traffic are presented, including detailed API descriptions for developers.

Chapter 4 presents the testing strategy followed during the development of the Live+Gov toolkit and the development and customization of the three SaaS-based mobile governance solutions. Future toolkit developers and users can reuse this strategy when developing their own toolkit extensions or SaaS solutions.

Chapter 5 gives a summary of the test results of the service level-, integration- and system test performed in preparation of the 2nd trials for Mobility, Urban Planning and Urban Maintenance.

Chapter 6 summarizes this document and discusses the conclusions.

The Appendix includes the concrete test scenarios (manuscripts) and results for the tests that are performed in preparation of the 2nd trial round. Appendix A contains a test report template partners used to report the tests. Appendix B contains all system test plans and results. Appendix C contains all service level and integration test plans and results.

2 Live+Gov Toolkit overview

This section gives a summarised overview of the Live+Gov Toolkit. First an introductory description of the Toolkit approach is given, explaining the goals and aims. After the context of the Toolkit is set, the architecture is described, with specific focus on the involved services and components. The section will be concluded with an overview of the concrete Live+Gov Use Case field examples. A full description of the architecture and requirements can be found in *'D4.1 Report on Live+Gov toolkit requirements and architecture'* [5]

2.1 Approach

The Live+Gov adopts a Toolkit approach which allows efficient development, validation, deployment and maintenance of mobile governance solutions, following the Software-as-a-Service (SaaS) deployment model. The Live+Gov Toolkit includes a number of reusable software services. These services can be pieces of software used in mobile applications, web applications or supporting services. Within the project the toolkit is applied to customise SaaS-based mobile governance solutions for each of the use-cases (Urban Planning, Urban Maintenance and Mobility). However, the toolkit can be used to develop SaaS solutions for other eParticipaton scenarios. The different services are composed of different toolkit components. The different services or components can be recombined to cover more application scenarios. The different building blocks of the Toolkit will save the municipality time and costs during development, deployment, and maintenance of new governance solutions.

In order to offer the different developing consortium partners the maximum freedom to make use of their specific expertise, a distributed toolkit architecture is chosen. Different consortium partners are responsible for different components and services. Each partner is free to adopt their specific techniques or choice in order to be able to offer Live+Gov their maximum potential expertise. This strategy has impact on the architecture. The architecture should be of supportive nature instead of an all-encompassing framework. In order that the different services and components can actually be combined, WP4 defined generic integration concepts and guidelines and offers a central SaaS Center offering generic SaaS services required for functional integration. These concepts and guidelines have been reported in D4.2 – Integration concepts and guidelines [11].

Live+Gov offers conceptual guidance in assessing the exact combinations of components for tailored eParticipaton scenarios. The guidance is built upon the Live+Gov Citizen Participation with Mobile Technology (CPMT) approach as developed in WP2. The approach is giving guidance on a conceptual level how to assemble software solutions aiming for Open Government. The conceptual framework guides in deciding the needed components for a specific pillar or combination of pillars and their variants of Open Government: Transparency, Participation, and Collaboration. More on the approach and the practical usage of the approach can be found in D2.1 [2], D2.3 [19], and D2.4 [20]. The conceptual model has been applied in the SaaS Service Center as well, assisting the service providers and customers in the configuration of (component) permissions. An example is shown in Figure 1.

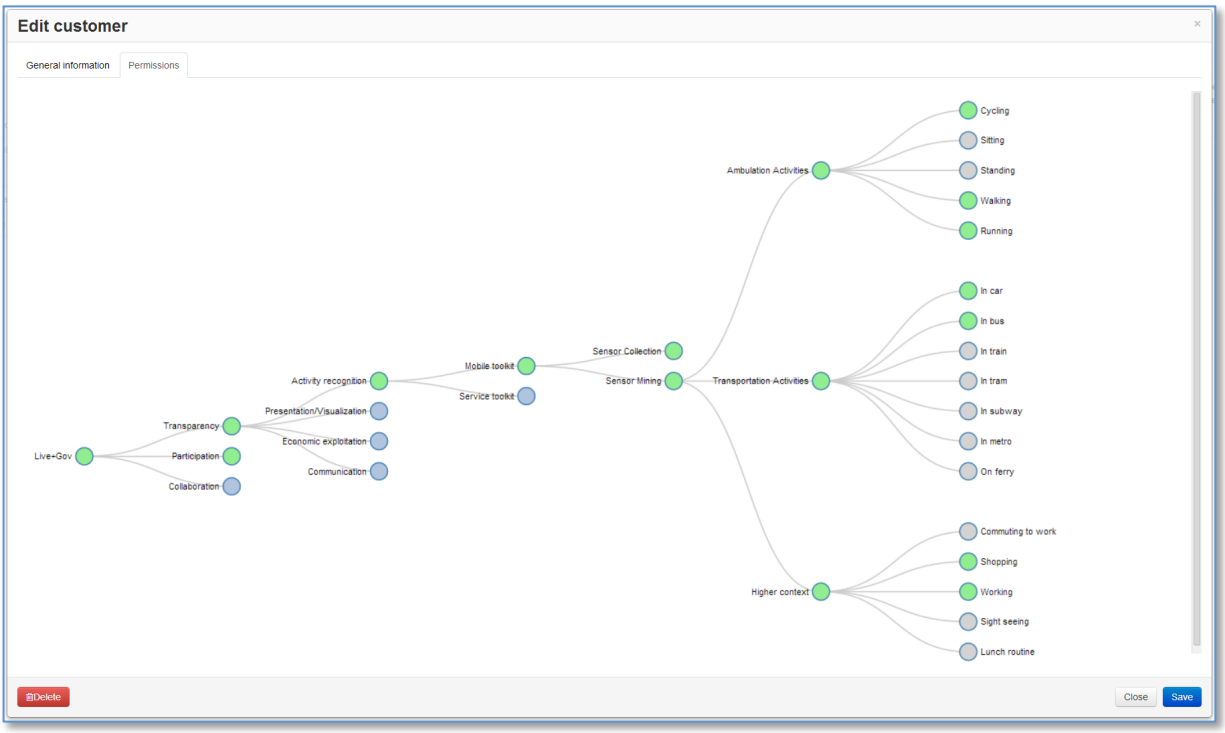

Figure 1: Conceptual model in SaaS Service Center

The overview of the architecture of the Live+Gov Toolkit will be described in more details in section 2.2, in which specific attention will be given to the involved services (section 2.3), the components (section 2.4). In section 2.5, we conclude the overview of the Live+Gov Toolkit architecture with the concrete Live+Gov Use Case field examples developed with the Toolkit in section. Here, we will show the involvement of the different components of the Live+Gov toolkit in the software prototypes that have been used in the field trials.

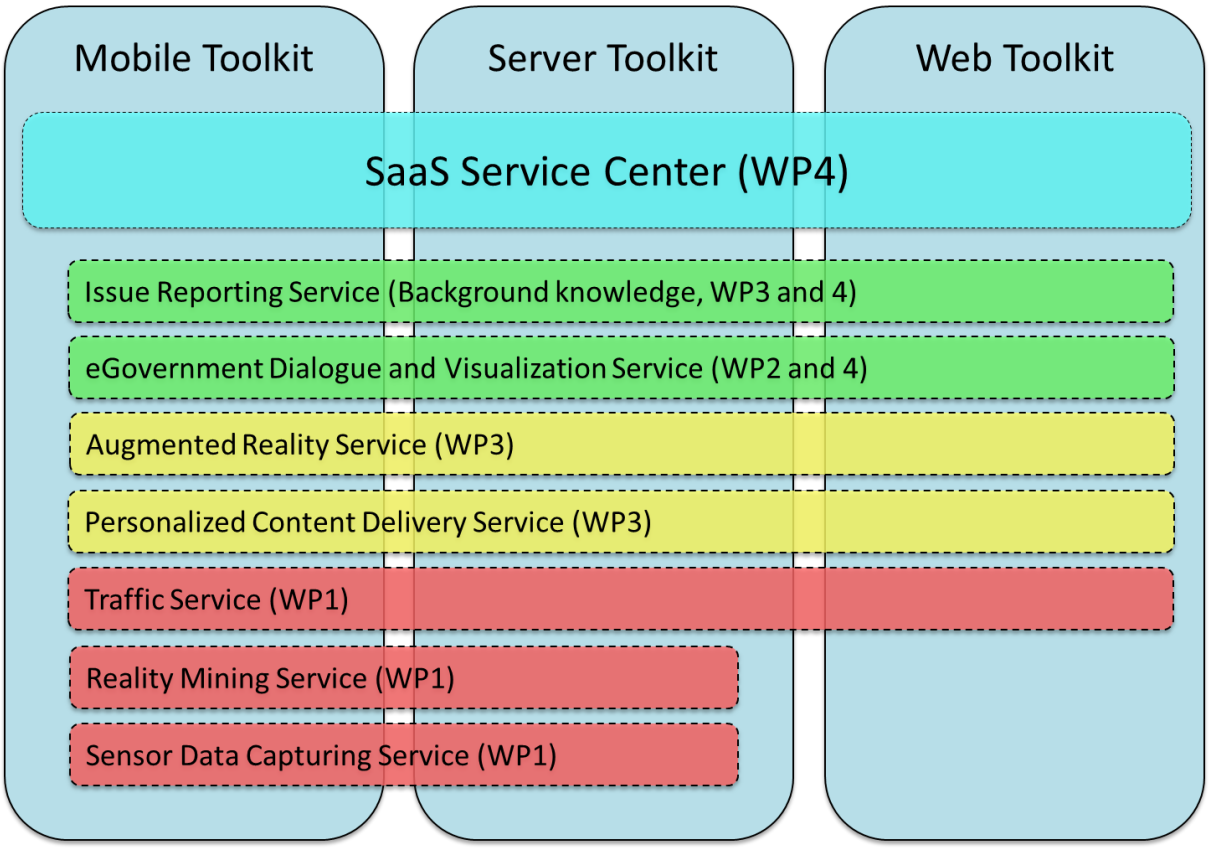

Figure 2: Live+Gov service overview

2.2 Architecture

The Live+Gov Toolkit is composed of different services and components. The very nature of the Toolkit is that it has a distributed architecture. That said, the toolkit is integrated the interaction between services and the central SaaS Service Center service. This service offers and channels the shared functionalities in a SaaS way in order to be able to exploit the generic Toolkit services for specific, tailored end-solutions. SaaS specific functionalities include account management, access control, billing and multi-tenancy. The other toolkit components utilize these services instead of including their own fragmented mechanisms. Also a public API and plug-in option are included to ensure interoperability with external systems like the public administration and transport system. The public API includes plugin for open standards for exchanging issues, like StUF [9].

Overall, the toolkit is divided in a Mobile-, Server- and Web toolkit and offers a comprehensive set of tools that offer functionality for sensor data capturing and mining, Augmented Reality presentation, visual recognition, personalized content delivery, eGovernment dialogue and visualisation and Issue Reporting. The functionality is offered by a set of distributed components that follow the principles of a Service-Oriented Architecture [5], which allows us to offer a generic applicable toolkit, in which components have a clear function and interface, allowing generic services to be applied in multiple scenarios and solutions.

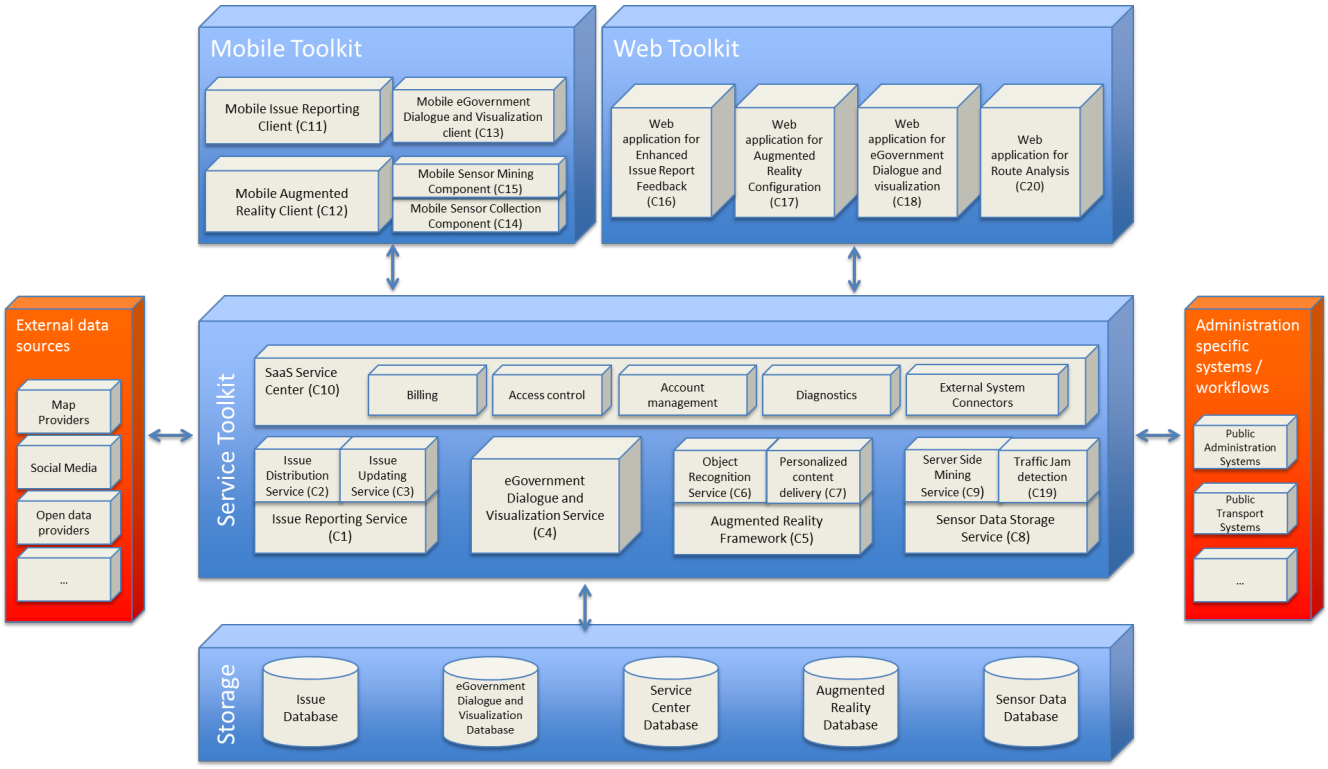

Figure 3: Live+Gov architecture

In following sections the different Services (section 2.3) and components (section 2.4) are briefly described as an introduction for section 3, in which the services are described in more detail. Finally, in section 2.5, the concrete Live+Gov field examples are described, which demonstrates the variety of possibilities of the Live+Gov toolkit.

2.3 Services

The Live+Gov Toolkit includes a number of reusable software services. These services can be pieces of software used in mobile applications, web applications or supporting services. Table 2 gives an overview of the services in the toolkit. Detailed descriptions of the services are given in Chapter 3.

Table 2: Toolkit services

No.	Service	Description	Section
S1	SaaS Service Center	Service that supports SaaS specific functionalities like account management, access control, diagnostics and monitoring, billing and providing a HUB to connect with external systems.	3.1
S2	Sensor Data Capturing Service	Service responsible for battery aware collection of sensor data from mobile devices and central storage of the relevant sensor data.	3.2
S3	Reality Mining Service	Service responsible for performing: • Energy aware smartphone based reality mining • Complex data analysis on the server, which cannot be run on individual smartphones. • Aggregated views and comprehensive summaries	3.3

S4	Augmented Reality Service (including object recognition)	Service responsible for mobile augmentation, object recognition, storage of point of interest, meta tags, 3D models and proximity triggers.	3.4
S5	Personalized Content Delivery	Extension on the AR Presentation Service responsible for filtering the content presented to the user based on his explicit, or implicit preferences	3.5
S6	eGovernment Dialogue and Visualisation Service	Responsible for offering citizen government dialogue and visualizing aggregated data in a user-friendly way, e.g.: • Present initiatives (e.g. issue reports, proposals, co-maintenance spots) • Add own initiatives (both citizens and government) • Provide opinions and feedback on initiatives • Facilitate polls and questionnaires • Moderator functions for government to prevent abuse	3.6
S7	Issue Reporting Service	Service supporting submitting and processing issue reports in the urban space. • Post issue reports including location, photo, category, and contact information. • Forward issues to external systems (e.g. Public Administration or Public Transport Systems) • Give enhanced feedback on issues to citizens	3.7
S8	Traffic Service	Service providing traffic related services like: • Traffic Jam Detection • Route Analysis • Connection with external Traffic systems	3.8

2.4 Components

Services consists of one or multiple components. Table 3 offers an overview of the components within the offered toolkit services. Detailed descriptions of the components are given in Chapter 3.

Table 3: Components overview

No.	Service	No.	Component	Mobility	Urban Maintenance	Urban Planning
S1	SaaS Service Center	C10	SaaS Service Centre	✓	✓	✓
S2	Sensor Data Capturing Service	C08	Sensor Data Storage Service	✓		
		C14	Mobile Sensor Collection Component	✓		
S3	Reality Mining Service	C09	Server Side Mining Service	✓		
		C15	Mobile Sensor Mining Component	✓		
S4	Augmented Reality Service	C05	Augmented Reality Framework	✓		✓
		C06	Object Recognition Service			✓

		C17	Web application for Augmented Reality Configuration	✓		✓
		C12	Mobile Augmented Reality Client	✓		✓
S5	Personalized Content Delivery Service	C07	Personalized Content Delivery Service	✓		✓
S6	eGovernment Dialogue and Visualisation Service	C04	eGovernment Dialogue and Visualisation Service		✓	✓
		C13	Mobile eGovernment Dialogue and Visualisation Client			✓
		C18	Web application for eGovernment Dialogue and Visualisation		✓	✓
S7	Issue Reporting Service	C01	Issue Reporting Service	✓	✓	
		C02	Issue Distribution Service	✓	✓	
		C03	Issue Update Service	✓	✓	
		C11	Mobile Issue Reporting Client	✓	✓	
		C16	Web application for Enhanced Issue Report Feedback	✓	✓	
		C21	External connector for Public Administration Systems		✓	
S8	Traffic Service	C19	Traffic Jam Detection	✓		
		C20	Web application for route analysis	✓		
		C22	External connector for Public Transport System	✓		

2.5 Field examples

Table 3 offers an overview of the usage of the difference Live+Gov Toolkit services and components during the three Live+Gov use case trials. All components have been used during the trials and most components have been applied in multiple use-cases, which demonstrates the general and reusable character of the Toolkit.

More information about these field examples can be *'D5.4 – Prototype-demonstrator for second trials'* [17].

2.6 Exploitation

The Live+Gov Toolkit is one of the exploitable end-products of the Live+Gov project. Currently three exploitation pathways that involve the Toolkit, are foreseen. This paragraph gives a preview to a selection of possible exploitation pathways, the final exploiting pathways will be described in *'D6.5 – Final Exploitation Plan'*. Table 4 describes this selection of possible exploitation pathways, relevant within the Live+Gov Toolkit and/or architecture context.

Table 4: Selection of possible exploitation pathways

Pathway	Description
Live+Gov Toolkit Libraries	The project delivers a number of libraries that can be used (under license) by the consortium members to build tailor maid solution for new customers. These libraries will help developing mobile and web applications for citizens and municipalities in a faster and more efficient way. The libraries will be ready for exploitation at the end of the project.
Live+Gov Toolkit SDK	A compiled software development kit (SDK) and related guides and documentation allowing easy development of specific solutions by 3rd parties. Compiling this SDK and guides requires considerable additional effort and it is estimated that it can be ready for exploitation 12 months after the project.
Live+Gov SaaS Solutions	The project has used the Toolkit to develop prototype SaaS solutions for three Use Cases: Urban Planning, Urban Maintenance and Mobility. These solutions are SaaS prototypes, implemented using the software toolkit and guided by a conceptual model describing the use-case and the assessment of components. The solutions are currently being trialled for evaluation and testing within Live+Gov. The final prototypes will become available for exploitation as a SaaS solution at the end of the project.

The service based architecture of the Live+Gov Toolkit allows flexible combining of services and their underlying components. For exploitation this offers the opportunity to create different commercial packages, existing of any of the valid combination of components mentioned in Table 3. The conceptual model [2] can assist a service provider in creating valid combinations of components.

The central SaaS Service Center is the component where these combinations can be configured by the service provider. The contracted services per customer and permissions will be configured in this central SaaS Service Center. Furthermore the SaaS service provider benefits from (can make use of?) several other functionalities of the Service Center:

- Add new customers who have subscribed for a SaaS solution.
- Add new modules and permissions when the toolkit is extended with new functions or services.

- Configure roles, defining access profiles for different types of users.
- Check the health status and log files of the different services for support and maintenance purposes.
- Configure billing counters for pay-as-you-go exploitation models.

This service provider first creates a custom profile for each customer, which defines the available functionality of the SaaS solution is available for this specific customer, depending on his type of subscription. The customer administrator then uses the portal to add the users and configure the roles within its SaaS solution, based on these boundaries set by the service provider. This means that a service provider can offer many combinations of services, however from commercial perspective, it might be interesting to distinguish standard service level, e.g. basis, pro and premium. An example impression for this model is depicted in Figure 4 . This approach will be further elaborated in the final exploitation plan D6.5.

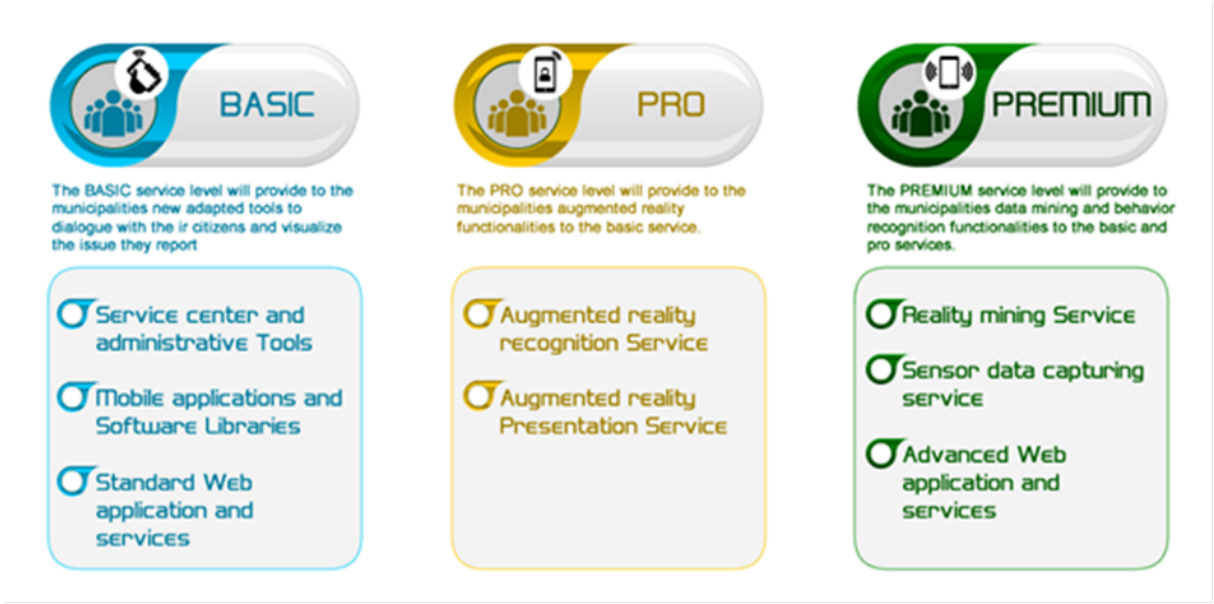

Figure 4: Example SaaS Service Levels for exploitation

3 Live+Gov Toolkit Services

The previous chapter described the toolkit approach and global structure, whereas this chapter gives an update of the Live+Gov toolkit services focussed on their Software API's, Web API's and integration with the Service Center. An overview of all toolkit components and services is given in the beginning of the deliverable by Table 2 in Chapter 2.2. The source code of the open source toolkit services can be found in the Live+Gov Source Code Repository (https://github.com/LiveGov/)

3.1 SaaS Service Center (S1)

The Service Center offers SaaS specific functionalities like account management, access control, billing and diagnostics. These services are required in every typical SaaS solution and affect all other toolkit components. Instead of each component implementing its own fragmented mechanisms, a central service is offered that takes care of this functionality.

3.1.1 Description

The Service Center consists of a Web Portal and a software API. The Web portal offers a web interface for performing administrative tasks by service providers and customers. Administrators can use this interface for e.g. adding customers, adding users, defining roles and permissions, viewing logs and the current systems health status. The software API is used by other toolkit components to integrate with the Service Center programmatically.

A more detailed description of the service center and the integration guidelines is given in D4.2 [11].

3.1.2 Requirements

The following requirements are recapitulated from D4.1 [5] for the SaaS Service Center:

Table 5: Toolkit SaaS requirements

Requirement	Description
Account management	The solution needs to be able to identify single users and user groups (e.g. organizations). For example to separate data from different users and organization, to enable access control or to personalize application content.
Access control	Specific functionalities and services offered by the SaaS solution are only accessible to subscribers or certain roles, e.g. an administrator. It can also depend on the type of subscription which functionalities are available.

Billing	Depending on the exploitation model, billing can be handled in various ways: • Pay-per-user: Periodic (e.g. monthly) fee depends on the number of users and their role. • Pay-as-you-go: Pay for usage, e.g. number of issue reports or number of log-ins. • Unlimited use: Flat fee for an unlimited number of users. Especially Pay-as-you-Go requires metering services of these variables in order to be able to specify the bill.
Scalability / multi-tenancy	Scalability is an important aspect for SaaS solutions since the exploitation model often relies on economies of scale. The amount of effort required to add new subscribers should be reduced. It is up to the solution provider how to handle this. *Note: this requirement also has to be covered by the individual services*
High availability	Customers have to rely on the SaaS provider to the keep services running. This is still one of the prime barriers for SaaS adoption, even though SaaS providers often have better up-time track records than most enterprises. Measure can be taken to increase availability, e.g. adding redundancy, load balancing or specifying Service Level Agreement or define disaster recovery plans. *Note: this requirement also has to be covered by the individual services*
Diagnostics and monitoring	Service Level Agreements (SLA) are often part of SaaS contracts. Such an agreement includes measurable performance indicators (like up-time percentage) and quality levels, agreed between the customer and solution provider. The service provider needs to pro-actively monitor and report on the appropriate metrics to ensure and to prove that these agreements are met.
Security and privacy	Security and privacy is still the prime barrier for the adoption of SaaS solutions. The solution provider is responsible to take the appropriate security measures for securing data transmission, data storage, identity management and offering audit trails.

External system connectors	In order to provide interoperability SaaS solution should offer web service API's to allow interaction with external systems, e.g. public administration systems.

3.1.3 Architecture

The Service Center functionality is organised into separate modules. These separate module are recognisable in the Web Application, as well in the API structure. Table 6 describes the service center modules.

Table 6: Service Center Module

Module	Purpose
Organisation Management	In the organisation management section the customers are defined by the Service Provider Administrator. A customer is an organisation that has signed up for one of the SaaS solutions offered by the Service Provider.
Configuration Management	In the configuration management section the modules and permissions are defined. A module corresponds to a toolkit component.
User Management	In the user management section users, devices and roles are defined by the customer's administrator.
Billing	Billing counters are used to enable a pay-as-you go license model. Solution Providers can add a billing counter for specific customers. Counters are used to keep track of number of times a specific piece of functionality is used, e.g. the number of issue.
Diagnostics	In the diagnostics section module log files and modules health status can be consulted by the Service Provider or the Customer Administrator. The log files can be used for troubleshooting. The health status reports can be used to generate report with information regarding up-time and availability of the SaaS solutions.

The SaaS Service Center is installed and hosted by the SaaS Service Provider.

The main components are 1) the Live+Gov Webserver, which serves the SaaS Service Center Web Portal (ASP .NET), that offers the graphical user interface for the administrators, and 2) the Web API (RESTful Web API), that enables other Toolkit components to make use of the Service Center Services.

Depending of the type of data, the data is either stored in the database (PostgreSQL) or on a file share. Especially larger files (e.g. log files) are stored on the file share, with a reference to its location in the database.

Figure 5 shows the overview of the Service Centre Architecture. A detailed installation manual [13] is available in which the installation is explained step by step.

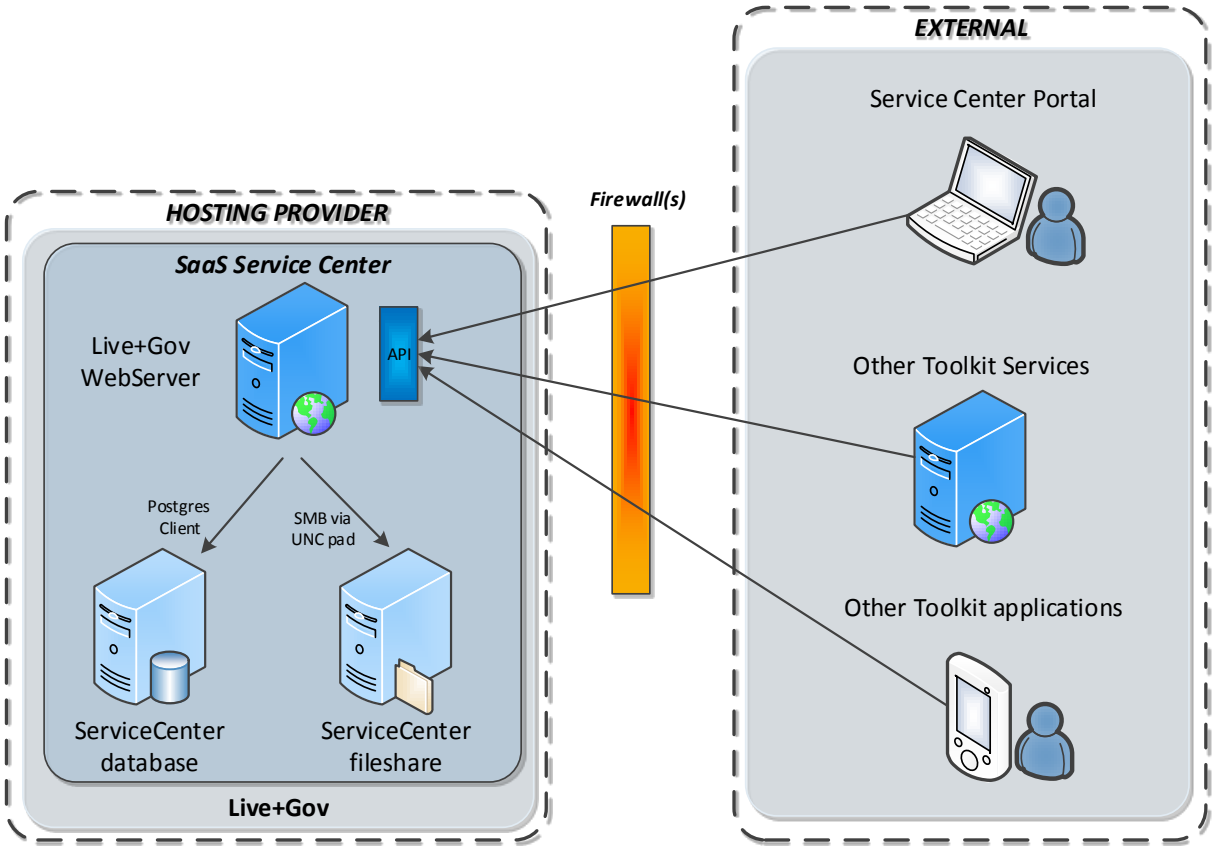

Figure 5: Service Center – Technical architecture

3.1.4 Web API

This chapter gives a summary of the SaaS Service Center API description. The API is used by other toolkit components to utilize the services offered by the Service Center. A detailed description of the API, including message examples and code examples is available in a separate document [12].

The API's in the Live+Gov toolkit are implemented using a RESTful web API. More details on the guidelines for offering and consuming this type of APIs can be found in D4.2 [11]

3.1.4.1 Organisation management API

Table 7 describes the available REST API calls for the organisation management module.

Table 7: Organisation management API

Function name	Purpose	Request Parameters	Response parameters
Get Customer	Get the details of a specific or all customer	Customer id	Customer details
Get Customers	Get the details of multiple	(Customer code)	List with

	customers		details of multiple customers

3.1.4.2 User management API

Table 8 describes the available REST API calls for the user management module.

Table 8: User management API

Function name	Purpose	Request Parameters	Response parameters
Create Registered User	Add the information of a new user to a specific customer in the service center.	Customer id Account info (Personal info) (Contact info)	User id User type
Create Anonymous User	Add the unique identifier of an anonymous user to the service centre. Optionally coupled to customers.	Unique hardware identifier (List of customer id's)	Anonymous user id
Update Anonymous User	Update the customers an anonymous user is coupled to.	Anonymous user id (List of customer id's)	None
Get Registered Users	Get a list of registered users registered in service center for a specific customer	Customer id (User type)	List with details of users
Get Anonymous Users	Get a list of anonymous users registered in service center for a specific customer	Customer id	List with details of anonymous users
Get User Info	Get the details the authenticated user	None (based on authentication)	User details
Get Registered User	Get the details of a specific registered user	User id	User details
Update Registered User	Update the details of a specific registered user	User id (Personal info) (Contact info)	None
Set Password	Set the password for a specific registered user	User id Password	None
Delete Registered User	Delete a registered user	User id	None

3.1.4.3 Authentication and authorization API

Table 9 describes the available REST API calls for the authentication and authorization module.

Table 9: Authentication and authorization API

Function name	Purpose	Request Parameters	Response parameters
Authenticate Registered User	Check provided credentials of a user and get user details	Account info	User details
Has Permission	Check if the authenticated user has been granted a specific permission.	Permission name	None
Has Permission for User	Check if a specific user has been granted a specific permission	User id Permission name	None
Get Permissions	Get list of granted permissions for the authentication user	None	List of granted permissions
Get Permissions for User	Get list of granted permissions for a specific user	User id	List of granted permissions

3.1.4.4 Diagnostics API

Table 10 describes the available REST API calls for the diagnostics module.

Table 10: Diagnostics API

Function name	Goal	Request Parameters	Response parameters
Post Log File	Periodically submit log files to central repository	Log file info Log file	None
Get Health Checks for Module	Get health status information for a specific module.	Module id	List with registered health check signals
Set Health Check	Periodically give module health status information to service center	Status Date / Time	None

3.1.4.5 Billing API

Table 11 describes the available REST API calls for the billing module.

Table 11: Billing API

Function name	Purpose	Request Parameters	Response parameters
Get Billing Counter(s)	Get the details of a specific or all billing counters	Billing counter name	Billing counter details
Decrease Billing Counter	Register a transaction on a billing counter	Billing counter id Amount	None

		Module id (User id)	

3.1.4.6 System

Table 12 describes the available REST API calls for the system module.

Table 12: System API

Function name	Purpose	Request Parameters	Response parameters
Get Module(s)	Get the details of a specific or all system modules	Module name	List of module details

3.1.5 Integration

The service center offers functionalities and API's for other services in the Live+Gov Toolkit, therefore other service integrate with the Service Center. Comprehensive instructions for integrating your service with the Service Center are found in the integration concepts and guidelines D4.2 [11].

3.1.6 Phase 2 updates

Table 13 describes the updates performed in preparation of the 2nd trials rounds.

Table 13: Service Center Phase 2 updates

SaaS Service Center Web application		
Update	**Update description**	**Rationale**
View on billing counters and transactions	Added a web presentation (dashboard) of the current status of the billing counters.	Required for 2nd use case in Gordexola to keep track of the number of submitted questionnaires during the trial.
View on health status	Added a web presentation (dashboard) of the current health status of the different modules in the solution.	Added for the 2nd trials to easily monitor the system components during 2nd trial round execution.
Conceptual model to support configuration	Added graphical presentation of the system modules according the conceptual view presented in WP2.	Added for exploitation to demonstrate a user friendly way to configure the system.
Fixed error in sort of list view	Fixed incorrect sorting of list views in the web application	Bug fix to improve user experience
Fixed error in redirect after logout	Fixed an error in the redirect after a user logs out. Previously a page not found was show, now the login screen in show.	Bug fix to improve user experience

SaaS Service Center API		
Update	**Update description**	**Rationale**

API extension	Added customer identifier in registered user controller	Required for integration with the issue reporting portal for Urban Maintenance, enabling customer specific views in issue reporting web application in a single instance.

3.2 Sensor Data Capturing Service (S2)

This chapter describes the Sensor Data Capturing Service, which is responsible for collecting and storage of sensor data from mobile devices.

3.2.1 Description

The Sensor Data Capturing Service consists of two components, the Sensor Data Storage Service (C8) and the Mobile Sensor Collection Component (C15). The Mobile Sensor Collection Component is responsible for collecting data from a large variety of sensors in a battery aware way. The Data Storage Service archives the collected sensor data and makes it available for data inspection and mining applications.

3.2.2 Requirements

The functional requirements for the Sensor Data Capturing Service have been translated into Sensor Capturing Requirements in the Deliverables D1.1 [1] and D4.1 [5]. In the following table we refer to these Sensor Capturing Requirements as presented in D1.1.

Table 14: Sensor Data Capturing requirements

No.	Sensor Capturing Requirement	Outcome	Part of 1st trials?	Part of 2nd trials?
RA.1	GPS – On spot location	The Mobile Sensor Collection Component (C15) is able to record samples with the appropriate frequencies.	Y	Y
RA.2	GPS – Location tracking		Y	Y
R-SC.1	GPS – Every 10 sec.		Y	Y
R-SC.2	Accelerometer (20Hz)		Y	Y
R-SC.3	Gyroscope (20Hz)		Y	N[1]
R-SC.4	Magnetometer (5Hz)		Y	Y
R-SC.5	WiFi (1/30 sec)		Y	N[1]
R-SC.6	GSM (1/30 sec)		Y	N[1]
R-SC.7	Microphone (1Hz)		N	N[2]

Please see D1.1 [1], p.13, p.27 for further details about these requirements and their relation to the data mining end products.

[1] For Power consumption and Privacy Protection reasons we have decided to limit the data collection in the second field trial to the minimal sensors necessary for the Activity Recognition and Service Line Detection. The corresponding requirements are all met.

[2] The microphone sensor was not used in the project and therefore the requirement R-SC.7 was effectively dropped and we did not implement audio recording functionality into the Sensor Collection Component.

3.2.3 Architecture

The Architecture of the Sensor Data Capturing Service is described in D1.1 [1], p.24. We will only provide a short summary here.

The Mobile Sensor Collection Component records sensor samples from the mobile device. The collected samples are transferred in bulk to the Sensor Data Storage Service. The data is persisted on the mobile device and transferred to the Sensor Data Storage Service over a wireless network connection (GSM, 3G or WiFi). The sensor data is stored on the server in two different ways: Firstly, on the file system for backup reasons, and secondly in a database to facilitate further processing and inspection.

The Sensor Data Storage Service includes a data inspection web front-end which is used for testing purposes.

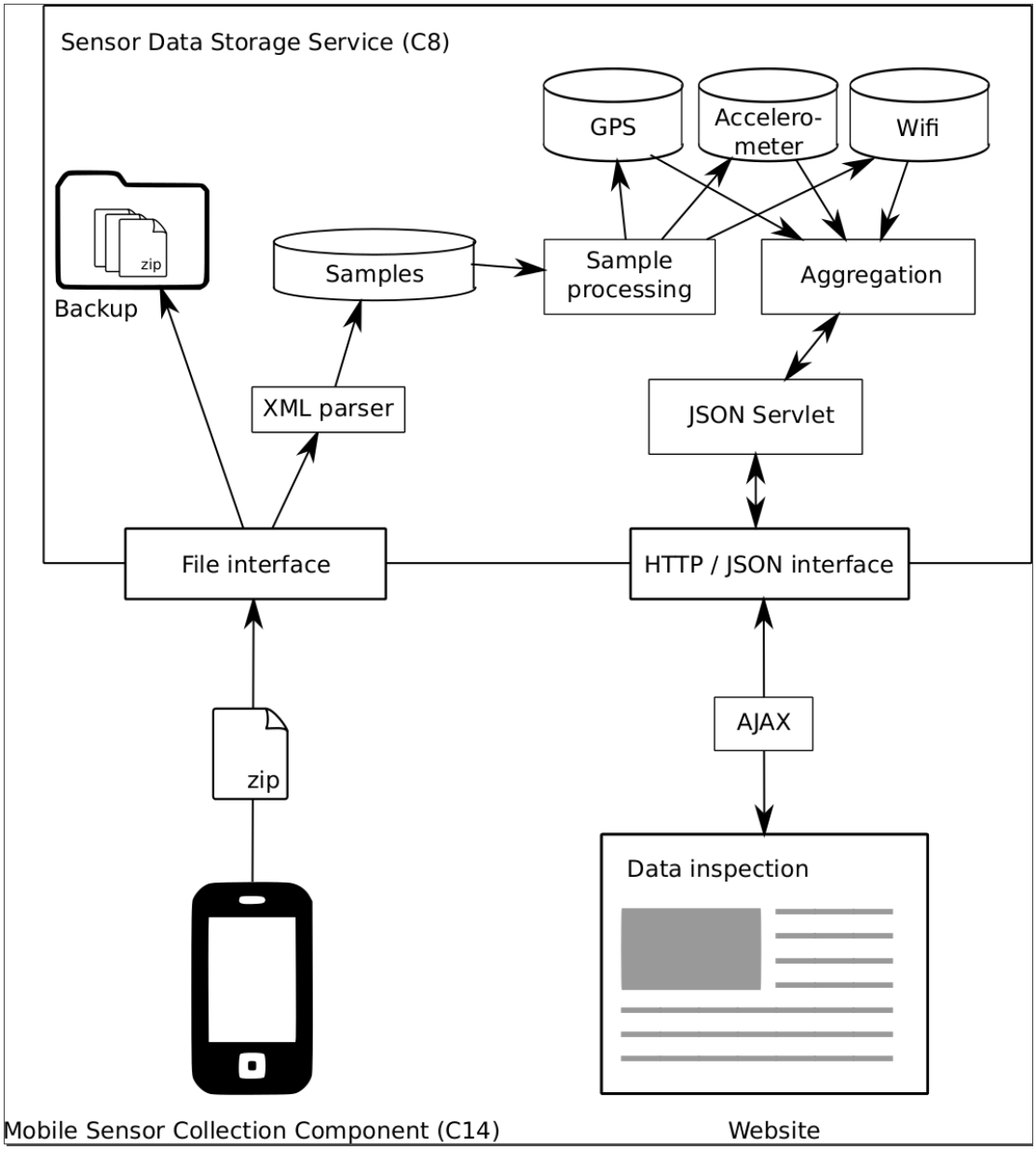

Figure 6: Sensor Collection Architecture

3.2.4 Web API

The Sensor Data Storage Service exposes the following REST-full API to the mobile client. This API is intended for internal use within the Sensor Data Capturing Service (S2).

Table 15: Sensor data capturing Web API

Function name ***Description***	**Request**	**HTTP method**	**Response**	**Remark**
Upload Samples Upload Samples to server.	*UploadServlet/* *The sensor data is attached as multipart/form-data in a field called "upfile".* The data is expected in the sensor stream format described below.	POST	On success: Status: 202 ACCEPTED Moreover, human readable status information is attached in the body of the response. On error: Status 400 BAD REQUEST	The upload file format has been revised since D1.1

Example Request:

```
POST / HTTP/1.1
ID: 61c206d1a77d509e
Content-Length: 2019
Content-Type: multipart/form-data; boundary=g3lJT9mnoGKMD2s1cQhkfzSCUqqR3TB
Host: 141.26.71.84:6000
Connection: Keep-Alive
User-Agent: Apache-HttpClient/UNAVAILABLE (java 1.4)

--g3lJT9mnoGKMD2s1cQhkfzSCUqqR3TB
Content-Disposition: form-data; name="upfile"; filename="sensor.stage.ssf"
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary

ACC,1381584735464,61c206d1a77d509e,-0.0435791 0.031417847 9.720917
GPS,1381584735607,61c206d1a77d509e,50.3551875 7.586807 0.0
ACC,1381584735738,61c206d1a77d509e,-0.0519104 0.030227661 9.716156

<---------------- MORE SAMPLES ---------------->

ACC,1381584735879,61c206d1a77d509e,-0.048339844 0.008804321 9.750671
LAC,1381584735880,61c206d1a77d509e,0.005099442 -0.0175697 -0.07921791
GRA,1381584735880,61c206d1a77d509e,-0.05331197 0.026311187 9.80647

--g3lJT9mnoGKMD2s1cQhkfzSCUqqR3TB--
```

Sensor Stream Format

The sensor stream file format (SSF) is inspired by the Common Log Format used by many webservers. We view incoming sensor values as events and record them as a simple stream of CSV-rows. Each sensor has an individual prefix but writes into the same file.

The SSF file follows the following Schema

```
SENSOR PREFIX, TIME STAMP, USER/DEVICE-ID, SENSOR VALUES
```

The filed `SENSOR PREFIX` takes one of the following values:

- GPS GPS sensor
- ACC Accelerometer
- GYR Gyroscope
- MAG Magnetometer

- WIFI WiFi networks
- BLT Bluetooth
- GSM GSM cells
- TAG User defined tags
- ACT Google Play Services Activity
- LAC Linear Acceleration
- GRA Gravity

The field `TIME STAMP` is the UNIX time the sample was recorded in milliseconds.

The field `USER/DEVICE-ID` contains a unique ID to identify the user. Usually this is the user ID provided by the Live+Gov Service Center. However, also custom IDs will work.

The field `SENSOR VALUES` contains the actual values of the recorded sensor sample. This field is customized for each sensor type. It is required to avoid the character "," in order to simplify parsing. Valid values of this filed are as follows:

- ACC/GYR/MAG/LAC/GRA X,Y,Z-values separated by space (" ") characters
- WIFI ESSID / mac address as escaped string
- GPS lat,lon,alt-values separated by space (" ") characters
- ACT Activity name and confidence score separated by space (" ") character
- TAG tag value as escaped text, with "," characters removed.

```
ACC,1377605748123,9HAD-FEJ3-GE3A-GRKA,0.9813749 0.0021324 0.0142523
GPS,1377605748156,9HAD-FEJ3-GE3A-GRKA,50.32124 25.2453 136.5335
WIFI,1377605748426,9HAD-FEJ3-GE3A-GGEA,"Uni-Koblenz WIFI"
TAG,1378114981049,ab85d157c5260ebe,"test tag"
```

Example Rows:

3.2.5 Software Library API

The Mobile Sensor Collection Component offers the following API on the mobile device. The current implementation uses Android Intent messages but can be adapted to messaging systems of iOS or Windows Phone. The Intent API is defined in a java class *IntentAPI.java* that can be consulted for the technical details.

Table 16: Sensor data capturing Software Library API

Function name ***Description***	**Request** **as Intent Action Strings. All listed Strings are prefixed with: "eu.liveandgov.sensorcollectorapi"**	**Response**	**Remark**
Status Request	.action.GET_STATUS	.return.STATUS The returned intent carries the following data fields: • "sampling" – boolean value that indicates if the service is collecting data • "transferring" – boolean value that indicates if the service is currently transferring data to a server. • "samples stored" – boolean value that indicates if there are samples stored on the device, that are not transferred to the server. • "id" – String value that contains the current user ID.	

Start Sample Collection	.action.RECORDING_ENABLE	.return.STATUS A full status intent is returned in response to this request.	
Stop Sample Collection	.action.RECORDING_DISABLE	.return.STATUS	
Transfer Samples Triggers transfer of samples manually.	.action.TRANSFER_SAMPLES	.return.STATUS	This action only works if an internet connection is available.
Send Annotation	.action.ANNOTATE This intent carries a String field “tag”.		
Set User ID	.intent.action.SET_USER_ID This intent carries a String field “id”	.return.STATUS	For use by the integrated application.

3.2.6 Integration

The communication of the Sensor Data Capturing Service (S2) and the Service Center (S1) running on one of the Live+Gov servers is summarized in Figure 20.

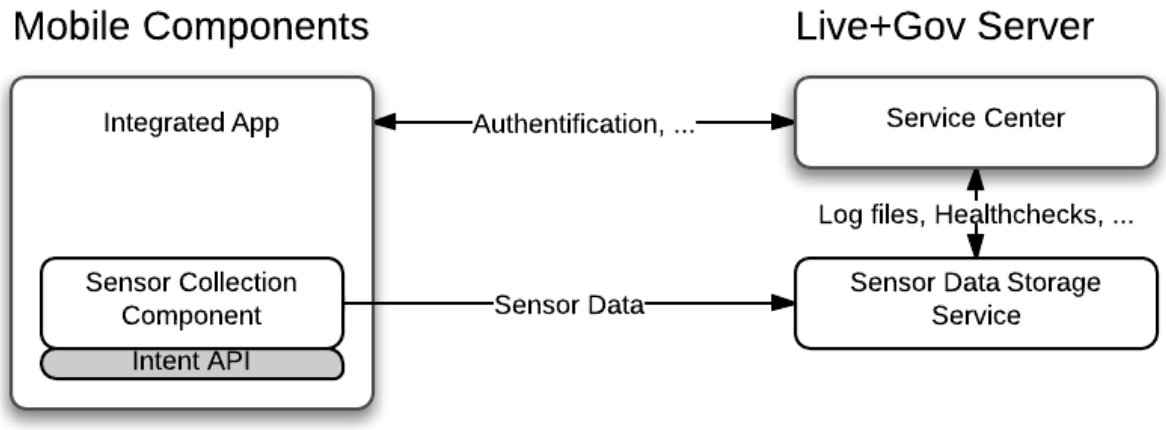

Figure 7: Sensor Collection Integration

The Sensor Collection Component is used in an integrated application which takes over the communication with the Service Center. Only the information about the User ID is passed to the Sensor Collection Component, via the Intent API. This allows to link the recorded sensor data to a registered user.

The Sensor Data Storage Component sends regular Health Checks to the Service Center and transfers log files every day to monitor the usage of the storage service.

3.3 Reality mining Service (S3)

This chapter describes the reality mining service which is responsible for extracting information from the data collected on the mobile device. Reality mining is performed on the server as well as on the mobile device itself.

3.3.1 Description

The Reality Mining Service (S3) contains the components Server Side Mining Service (C9) and the Mobile Sensor Mining Component (C15). In its current version the Server Side Mining Service implements Human Activity Recognition, the detection of service lines, means of transportation and the detection of traffic jams; the Mobile Sensor Mining Component implements activity recognition. The implementation of Human Activities Recognition as a service on the server, simplifies the integration of this service on multiple mobile platforms.

3.3.2 Requirements

The Functional Requirements listed in D5.1 [7] have been translated into Context Mining Requirements in D4.1 [5] and D1.1 [1]. In the following table we refer to the numbering used in D1.1.

Table 17: Reality Mining Service Requirements

No.	Live+Gov system requirements	Outcome	Part of 1st trials?	Part of 2nd trials?
RB.1-6	Detect users activity. The following attributes should be recognized: Sitting, Standing, Walking, Running, Cycling, Driving Car.	These requirements are covered by the Mobile Sensor Mining Component (C15).	Y	Y
RB.7-13	Detect transportation means. The following means of transport should be recognized: In bus, in train, in tram, in subway, in metro, in ferry	These requirements are covered by the Server Side Mining Component. This component is moreover able to detect the precise service line the user is traveling in.	Y	Y
RC.1-5	Detect higher context. The following attributes should be recognized: Commuting to work, Working, Lunch routine, Shopping, Sight Seeing	Due to limited resources and disappointing initial experiments, the development of his feature was not further pursued.	N	N
RD.1	Geographical Analysis	These requirements are addressed by the Server Side Mining Component. Their development is independent from the field trials.	N	Y
RD.2	Text Mining		N	-
RD.6	Clean-up existing datasets		-	-
RD.4	Route Analysis	These features depend on	N	Y

RD.5	Preferred Routes	the outcomes of the Activity Recognition and Service Line Detection Components. The features have been added for the second field trial.	N	Y

3.3.3 Architecture

The Architecture of the Reality Mining Service (S3) can be summarized as follows.

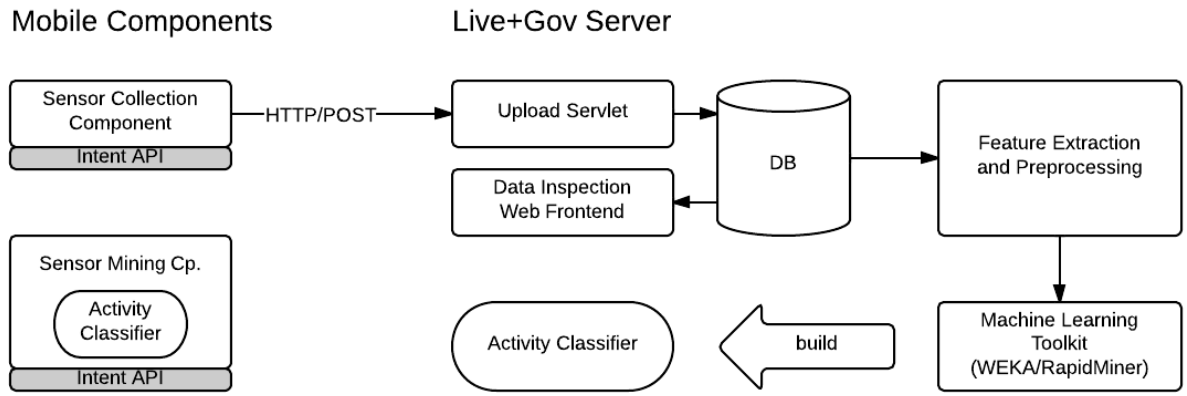

Figure 8: Mobile Sensor Mining Architecture

Using components of the Sensor Collection Service, a set of annotated samples is collected in a database. These samples undergo further quality control and editing using the Data Inspection Web Frontend. On the basis of the prepared samples a set of feature vectors is computed. These feature vectors serve as input for a machine learning toolkit which is used to train an activity classifier.

The Mobile Sensor Mining component retrieves sensor samples directly from the operating system of the mobile device, computes the feature vectors and uses the trained classifier to detect the user's activity.

The HAR Service is a web service that recognizes human activities. It receives data from a file sent within a POST request. The file must contain acceleration data in the SSF format. The service is able to handle arbitrary amounts of data, however 5 seconds of recording are Recommended. The recording frequency has to be around 50 Hz. The service returns the activity as a simple string in the body.

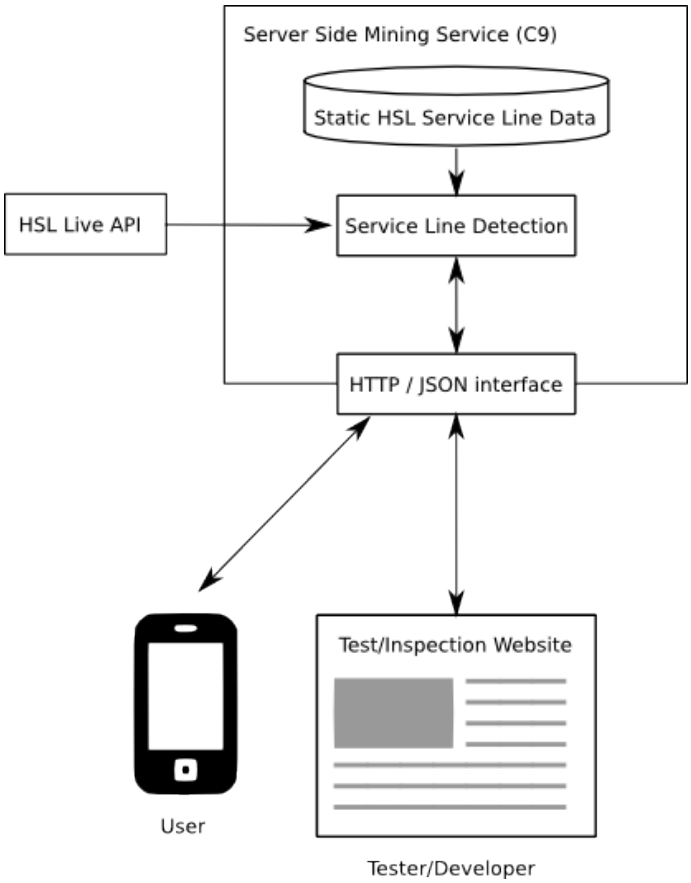

Figure 9: Service Line Detection Architecture

The Service Line Detection as part of the Server Side Mining Service (C9) is implemented as RESTful service cf. Figure 8. Its input is a GPS track and its output a list of ranked service lines matching the given track. It makes use of the static HSL Service Line Data (routes, timetables) as well as the HSL Live API which provides real-time position of some vehicles. User access the service via HTTP POST requests. The same HTTP interface can be used by testers and developers.

3.3.4 Web API

The Service Line Detection REST API receives GPS tracks (minimum one GPS coordinate) and replies a ranked list of Service Lines matching a given track in JSON format.

Table 18: Reality Mining Web API – Service Line Detection

Function name ***Description***	**Request**	**HTTP method**	**Response**	**Remark**
Service Line Detection *Get the HSL Service Line matching a given GPS track*	*/ServiceLineDetection* POST data: list<GPS_samples>: GPS_samples { - longitude - latitude - timestamp }	POST	list<ServiceLine>: ServiceLine { -route_id -shape_id -trip_id -score }	

Input: A list of recorded GPS-samples together with time-stamps:

```
Lat0,Lon0,Timestamp0,DayOfWeek0
Lat1,Lon1,Timestamp1,DayOfWeek1
Latn,Lonn,Timestampn,DayOfWeekn
```

- Lat and Lon coordinates in WGS84 format.
- Timestamp is in format "YYYY-MM-DD hh:mm:ss". It is in the local time where the sample is collected.
- DayOfWeek is one of [Mon, Tue, Wed, Thu, Fri, Sat, Son]

Example Service Line Detection API request:

```
POST /ServiceLineDetection HTTP/1.1
ID: 61c206d1a77d509e
Content-Length: 2019
Content-Type: multipart/form-data; boundary=g3lJT9mnoGKMD2s1cQhkfzSCUqqR3TB
Host: 141.26.71.84:8080
Connection: Keep-Alive
User-Agent: Apache-HttpClient/UNAVAILABLE (java 1.4)

--g3lJT9mnoGKMD2s1cQhkfzSCUqqR3TB
Content-Disposition: form-data; name="upfile"; filename="gps-track.csv"
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary

60.1652805,24.95296666,2013-09-23 20:06:36,Mon
60.1652115,24.95291232,2013-09-23 20:06:59,Mon

--g3lJT9mnoGKMD2s1cQhkfzSCUqqR3TB--
```

Output: The Service Line Detection Service response consists of the following parts:

- route_id refers to a HSL route in the HSL dataset provided in gtfs format.
- shape_id refers to the geographical shape of the route, also provided in the gtfs dataset.
- trip_id refers to the time depending trip the traveler is on. For example on route X every five minutes starts a new trip.
- score the higher the score the more likely is that the traveller really is on this trip.

An empty list means, that the user is not using public transportation.

Example Service Line Detection API response:


```
{"routes":[{"route_id":"1065A","shape_id":"1065A_20120813_2","trip_id":"106
5A_20130930_Ma_2_2000","score":12},

{"route_id":"1065A","shape_id":"1065A_20120813_1","trip_id":"1065A_20130930
_Ma_1_1930","score":12},

{"route_id":"1051","shape_id":"1065A_20131001_1","trip_id":"1065A_20131001_
Ma_1_1930","score":11},

{"route_id":"1070V","shape_id":"1065A_20131001_2","trip_id":"1065A_20131001
_Ma_2_2000","score":6}]}
```

Table 19: Reality Mining Web API – Human Activity Recognition

Function name ***Description***	**Request**	**HTTP method**	**Response**	**Remark**
HAR Service Post a file with sensor data (SSF format), return the current activity	/HAR/api POST data: curl -i --form "upfile=@test.csv" http://liveandgov.uni-koblenz.de/HAR/api HTTP/1.1 100 Continue	POST	HTTP/1.1 202 Accepted Server: nginx/1.1.19 Date: Wed, 21 May 2014 14:42:48 GMT Content-Length: 8 Connection: keep-alive on_table	

Input: File with acceleration data in SSF format (https://github.com/Institute-Web-Science-and-Technologies/LiveGovWP1/wiki/Sensor-Stream-Format). The file must contain acceleration data in the SSF format [1]. The service is able to handle arbitrary amounts of data, however 5 seconds of recording are recommended. The recording frequency has to be around 50 Hz. An example file can be found in (https://github.com/Institute-Web-Science-and-Technologies/LiveGovWP1/blob/master/scripts/test.csv)

Example Human Activity API request:

```
curl -i --form "upfile=@test.csv" http://liveandgov.uni-koblenz.de/HAR/api
HTTP/1.1 100 Continue
```

Output: The service returns the activity as a simple string in the body.

Example Human Activity API response:

```
HTTP/1.1 202 Accepted
Server: nginx/1.1.19
Date: Wed, 21 May 2014 14:42:48 GMT
Content-Length: 8
Connection: keep-alive

on_table
```

3.3.5 Software Library API

The following API is offered by the Mobile Sensor Mining Component.

Table 20: Reality Mining Software Library API

Function name ***Description***	**Request as Intent Action Strings. All listed Strings are prefixed with: "eu.liveandgov.sensorminingapi.intent."**	**Response**	**Remark**
Start Activity Recognition	.action.START_HAR	.return.STATUS A full status report is sent as an immediate response. .return.ACTIVITY In regular time this intent sent back to the system. It contains a field "activity" which carries the name of the recognized activity. Valid activities are: "standing", "sitting", "walking", "running", "cycling", "in car".	Starts the automatic classification of activities.
Stop Activity Recognition	.action.STOP_HAR	.return.STATUS	Stops the classification of activities.

3.3.6 Integration

Figure 10 shows the interaction of the Service Line Detection with the Service Center. All actions performed on the service API are checked against the Service Center (has permission). Periodically log files and Health Check updates are provided to the service center. Also a future scenario is shown where a billing counter is increased every time this service is called.

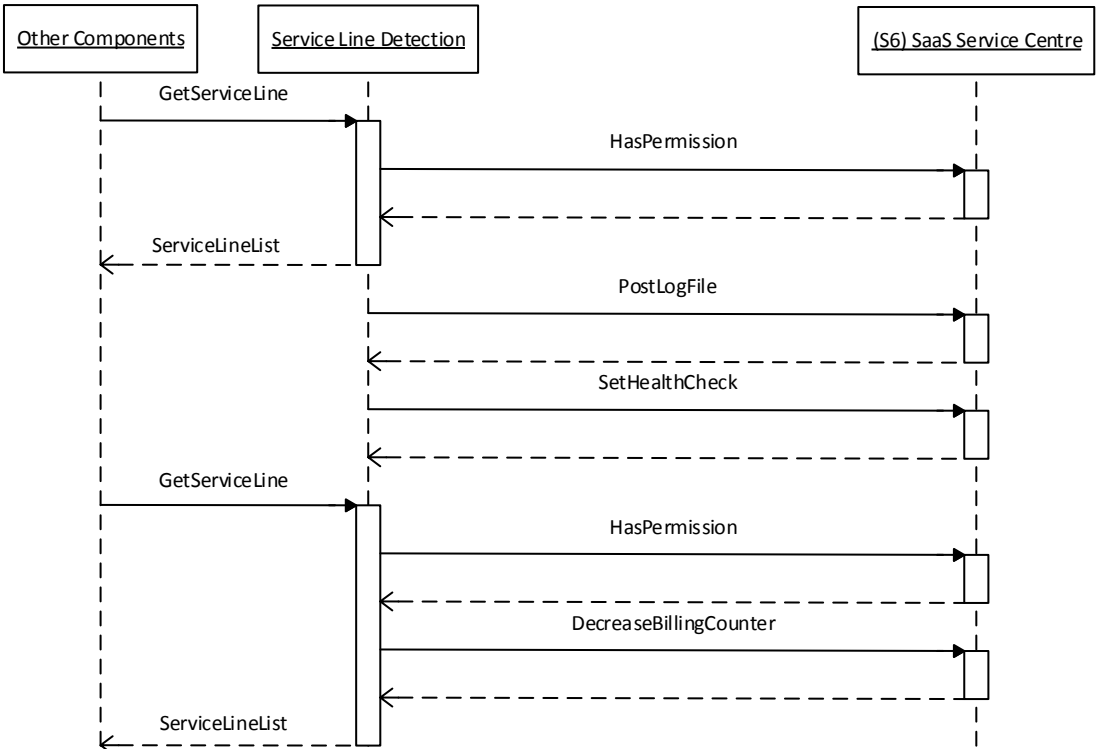

Figure 10: Service Line Detection Integration

3.3.7 Phase 2 updates

Table 21 describes the updates performed in preparation of the 2nd trials rounds.

Table 21: Reality Mining Service Phase 2 updates

Service / Update	Update description	Rationale
Service Line Detection *Integration of SLD in Storage Service, Service Center and Reporting tool.*	• *Revised our DB schema and access policies.* • *Attached user-id's to SLD queries* • *Inferred trip-id's from timestamps of the recordings*	In order to allow sophisticated queries (e.g. how are citizens traveling from one district to another) we need to link the recorded GPS tracks to the results of the SLD detection. This linkage requires communication and alignment between the sensor data storage service (storage of GPS samples), the Live+Gov Service Center (central authority for user-ids) and a reporting tool that is responsible for analytics and visualization of the query results.
Human Activity Recognition *Added Server Based Component*	The server side Activity Recognition algorithm is functionally identical to the one deployed in the mobile application. For update on this component see below.	In order to bring HAR services to other mobile platforms, we have developed a server-side HAR classification service. This service allows the mobile component to be very thin and thereby reducing the development efforts for porting the HAR service drastically.
Sensor Storage Component	*We have continued to work on the user interface of the inspection*	We have to be able to judge the quantity and quality in an easy way during the trial.

UI improvements of Inspection Tool.	*tool. It now allows to review the result of the HAR directly inside the tool. (cf. Screenshots) We are working on a similar feature for the SLD. Deletion and comments on of individual recordings from the DB is possible form the overview table.*	The inspection tool (Figure 11) has led to the discovery of several bugs during the trial.
Mobile Sensor Mining *Improved Decision Tree*	We have continued working on the recognition methods. The released component comes with a decision tree with improved performance for basic activities. This improvements have been achieved by adding further frequency domain features (FFT bin distributions), collection of further training data and improved pre-processing and cleaning of the recorded training samples.	We have continued our work on the activity classification.
Mobile Sensor Collection *Feedback of HAR results to sensor collection*	We have been working on a feature that turns off all sensor collection for a specified amount of time (e.g. 30 sec.) when the activity "ON_TABLE" is recognized. Initial experiments showed promising results. However in the following we encountered several problems such as crashes of the application and sometimes the recording did not start again after the 30 seconds had past, leading to truncated recordings.	Since the quality of data during the trial should not be compromised and the battery awareness of the component is already acceptable, we decided not to include this feature in the trial.

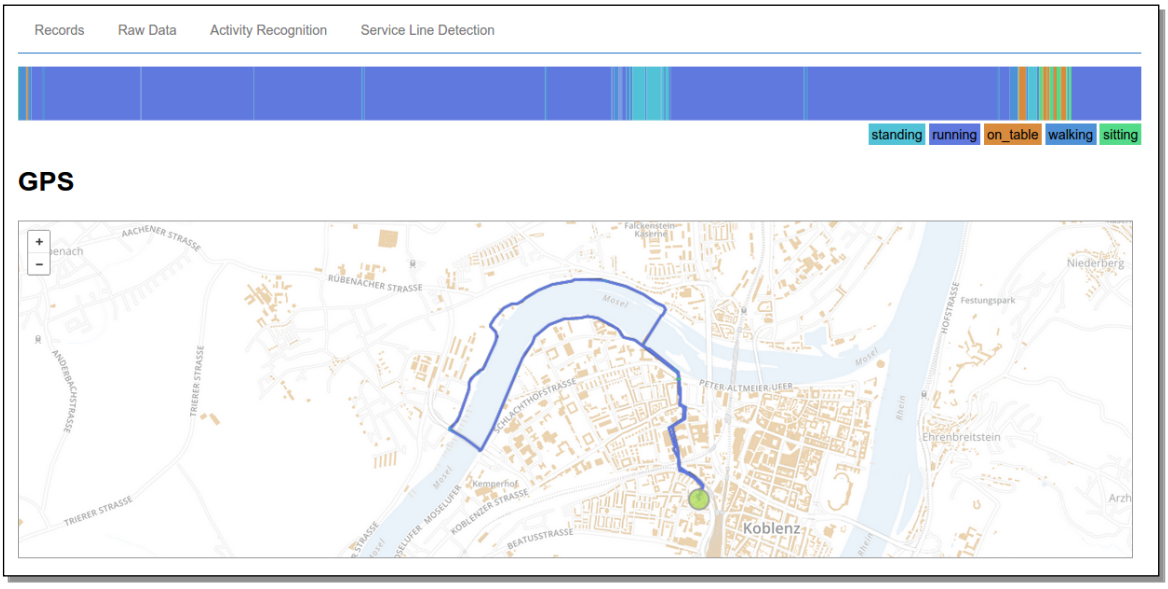

Figure 11: Revised inspection Tool

3.4 Augmented Reality Presentation Service (S4)

This chapter describes the Augmented Reality (AR) Presentation Service.

AR is based on the idea that location-based data can be overlaid on the view of the real life. In the context of Live+Gov, the goal of mobile AR is to help citizens get informed about local administration issues. For example, the local administration decides to renovate a pedestrian trail. The plan has two options: option one is an urban style renovation whereas the second option is a more natural style renovation. AR presents the 3D models for each case on the camera view of the mobile and lets the user decide which one is best.

3.4.1 Description

The Augmented Reality (AR) Service is split into several components:

1. AR Framework (C5) consisting of
    - Object Recognition Service (C6), and
    - Personalized content delivery (C7);
2. Web application for AR configuration (C17), and
3. Mobile AR Client (C12).

Object recognition is the recognition of an object from its image captured by the mobile device. The Personalized content delivery is the filtering of data according to the user preferences. The Web application is the portal of the server that allows administrators to insert, edit or delete 3d objects. Finally the Mobile AR Client is the component to perform location and image based AR that should be encompassed in the base mobile application.

3.4.2 Requirements

The AR Framework (C5) and the web application for AR (C17) are implemented on server side (AR-Server). In AR-Server, a database for storing data is found to which AR Framework and the web application for AR are directly connected. The AR information is available through two *APIs (Application Programming Interfaces)*, one for interconnecting with the mobile AR client (C12) and one for interconnecting with other Servers. The former is named as AR API, and the second is named as Authorization API. Mobile AR Client (C12) is implemented for two mobile platforms namely Android and iOS, that both retrieve data from the AR Server through AR API. Authorization API is used for the communication between AR-Server and SaaS Service Center for the verification of a power user that wishes to enter data in the AR-Server database. In total, AR service provides the features outlined in Table 22 [5] as to fulfil the project requirements.

Table 22: Requirements of Augmented Reality

No	Live+Gov system requirements	Feature status	Part of 1st Trials	Part of 2nd Trials
L+G-FR.01	Receive and store in a privacy-aware manner user information such as position age group, gender, residence, nationality, etc.	This information is send from the mobile app to the SaaS Service Center without the interference of AR framework	Y	Y
L+G-	Be able to show information in a context aware manner (location,	Information is send according to location by AR-Server and shown in	Partially	Y

FR.03	person, time-specific, situational) to create greater awareness and create mutual citizen-government understanding	Mobile AR Client. Personalized information will be retrieved from SaaS Service Center		
L+G-FR.07	Be able to capture images and recognize their content	Image based AR (IBS) is performed by the MetaioSDK locally on mobile AR client. Experimental Image Based AR (IBS*) by CERTH is performed remotely in the AR-Server by transferring the image to be recognized. Also the Junaio AR Browser by selecting the "Gordexola posters" channel can recognize the images of the IBS channel and present 3D content.	Y	Y
L+G-FR.11	Provide (mobile) presentation possibilities in an Augmented manner	AR Location and Image based view are implemented by Mobile AR Component and they are ready to use	Y	Y
L+G-FR.18	It was unrealistic to have norms from the local government. Instead focus is given on how to install AR framework to already existed content management systems in municipalities	A component to give AR functionality to Joomla/Sobipro based site was written	N	Y
L+G-FR.21	Provide possibilities to communicate urban planning plans	Urban planning plans can be communicated through AR-Server API and Mobile AR Client.	Y	Y
L+G-FR.26	Instead of gamification, we investigated ways to increase dissemination by supporting multiple free AR browsers by several companies.	Data is exported to Junaio, Layar, and Wikitude free AR browsers both for Android and iOS.	N	Y

The main requirements are split into AR visualization, Personalized Content Delivery and Object recognition. The AR Service provides receiving/storing/transferring of personal data related to AR such user position through GPS or IP, authorized access to certain entities, secure transfer of personal credentials with service centre; the image recognition capability, i.e. to capture an image with mobile phone and recognize the object in the image; and the functionality to visualize information in the mobile client such as text, image, and 3D models. The aforementioned requirements are implemented with components and APIs that are described next.

The last years' changes regard two axes. The first axis is on how to support popular content management systems (CMSs) that are already installed in several municipality servers, such as the Joomla CMS. The second axis is about supporting multiple APIs for exporting AR content. The AR API is now a set of Server to Mobile API's targeting to increase the dissemination of information of e-Government to several free AR browsers that are already

installed to several millions of devices. Junaio, Layar, and Wikitude are the supported AR browsers. Also, usage analytics monitoring is added in the mobile applications in order to see the behaviour of the users in the AR environment by exploiting several features of Google Analytics.

3.4.3 Architecture

The architecture of AR components is shown in Figure 12. It consists of two kinds of components, namely the server components, and the mobile components. Each component communicates to the other and with the rest of L+G components through APIs. These components and APIs are outlined in the following lines.

Server components are located in AR-Server. The server is an Apache server and the AR database is a MySQL database. The basic information element is called as *Entity* which represents an issue or an urban construction to be built. The Entity has text details, images, 3D models, and Object Recognition models associated with it. The user, let us say a governmental user, can insert, modify or delete Entities using the *Web Application for AR configuration (C17).* The latter is connected with the Service Center Server with the Authorization of API. The *Web Application for AR configuration (C17)* is able to modify Entities through the *AR Framework (C5)*. The latter provides the means to generate Image or Location based AR, namely IBS and LBS, respectively.

The AR content is available to the custom made applications and to the external AR browsers (Junaio, Layar, and Wikitude). The AR content is available to the custom made mobile applications for Android or iPhone through Metaio API. Both Android and iPhone AR clients exploit the free Metaio SDK for presenting AR content. It is obligatory by Metaio to be registered to the MetaioSDK application server in order to be able to use the MetaioSDK. Additionally, the Metaio, Layar, and Wikitude APIs can serve the AR content to the Junaio, Layar, and Wikitude AR browsers extending, thus, the dissemination of information. In order for these AR browsers to find these channels of information, one has to register the channel in the Junaio, Layar, and Wikitude Servers. Programming details are collected in the documentation website of AR-Server prototype and they are available online at [15].

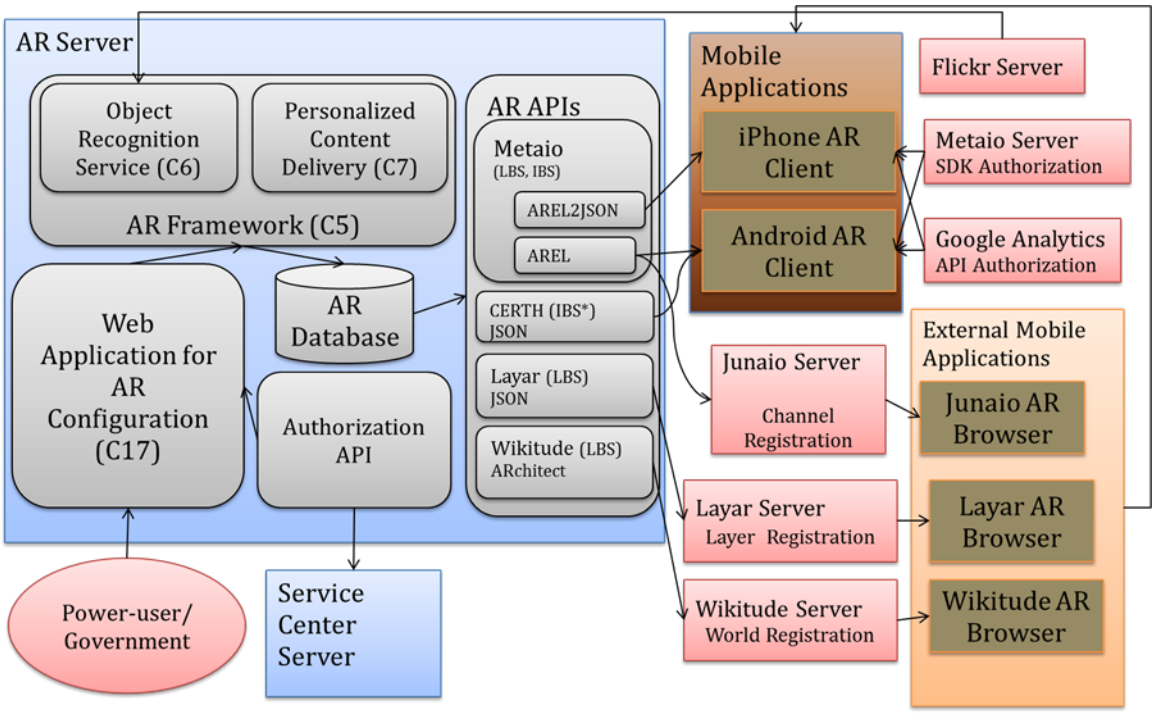

Figure 12: AR Server, Mobile AR Clients, and external mobile applications.

Web application for AR configuration (C17): It is the front-end portal of the server where power users can login and manage the information stored in the AR Database through the AR Framework (C5). C17 is written in HTML5, PHP, and JavaScript. It is comprised of three central files, the "index.php", the "AR_Main.php" and the "AR_Edit.php". The "index.php" provides the means to login using the credentials of the Server Center Server. Upon successful authorization, the user is led to "AR_Main.php" where he/she can inspect all Entities on a map and in a list. When the user selects an Entity, then the "AR_Edit.php" page appears where an Entity can be previewed and modified.

AR Framework (C5): consists of the functionalities of the AR technology that should be stored in the database. These functionalities depend on the AR modes supported. Three independent AR modes are supported, namely Location based (LBS), Image Based (IBS), and the one experimentally developed called as Image Based classification into concepts (IBS*). CERTH provides an extended image recognition service which recognizes general concepts such as tree, car, road etc. For example, IBS is able to recognize a certain car by its marker photo from which may depict the car from a certain angle. IBS* recognizes that the object in the image is a car independently the angle of the captured image and the car type. Several fields per Entity are required. All modes require a title and a description of the AR Entity. Up to four translations are allowed for the title and the description. The other fields that depend on the AR modes are described in the following.

Object Recognition Service (C6): It consists of IBS and IBS* that were described in C5. In details, IBS requires:

1. an image to be uploaded which functions as a trigger for presenting an AR Entity;
2. one, two or three 3D models that serve as alternative options for an urban plan that will be placed over the triggering image

3. the distance in meters from where the image was taken with respect to the spot that the AR Entity should be placed. This serves as a scaling parameter for the 3D model.
4. the shooting angle in degrees between the camera and the AR Entity spot with respect to the north to south line. This angle serves as rotation parameter for the 3D model so as to fit to the image perfectly in the case where orientation of the 3D model matters.

IBS* requires a sequence of images depicting a concept (instead of a single image) in order to train a model that represents that concept. This functionality is offered automatically. By typing the concept, several related images from Flickr[3] are downloaded and a model that can be used for image classification is generated.

Personalized Content Delivery (C7): Personalized information that it is exploited is the language of the mobile device, the location of the user, and the nearby items around. The language of the device controls the language of the downloaded AR Entities. The location based channel requires the GPS coordinates of each Entity. These coordinates are compared to the position of the user and only the ones that are closer to 100km are visualized.

AR APIs: There are several APIs developed mainly in the last year of the project that allow for a greater dissemination of information.

a) **METAIO AREL**: supports LBS and IBS for the custom Android AR client and the Junaio AR browser. The difference between the custom client vs. Junaio browser is that the custom app allows for the visualization of more than one 3D model per AR Entity. AREL API consists of PHP queries with configuration parameters from the Mobile MetaioSDK and sends back information to mobile in AREL format. AREL format is an XML format developed from Metaio for the needs of the mobile MetaioSDK components. AREL uses also JavaScript and CSS for defining the graphic user interface actions and the appearance in the Junaio AR browser, respectively.

b) **METAIO AREL2JSON:** a custom modification of AREL for exporting LBS and IBS content to iPhone devices. JSON is preferred since it is easier to parse JSON than AREL in iPhone.

c) **CERTH IBS* JSON:** a custom API for the support of the IBS* in the Android application.

d) **Layar (LBS) JSON:** the standard Layar API for exporting content to the Layer browser only for LBS mode. Automatic generation of an IBS channel is not supported in Layer at all.

e) **Wikitude (LBS) Architect:** the standard Wikitude HTML-JavaScript-CSS API to support LBS in Wikitude browser. Automatic generation of an IBS channel is not supported by Wikitude also.

Custom Mobile Applications: The Mobile AR Clients are based on the MetaioSDK component. MetaioSDK offers Location Based (LBS) AR and Image Based (IBS) AR. LBS shows points of interest (markers or 3D models) on the mobile screen over real-time video depending on the orientation and location of the user and the points of interest. IBS is a

[3] http://flickr.com

continuous visual search for images that match certain predefined images in the AR Server. In order to use MetaioSDK, the app name and its package name should be registered to Metaio developer portal in order to obtain a key that should be included in the app. Custom mobile AR clients also require the registration to the Google Analytics service in order to obtain usage statistics for the user of the app including the use of AR.

External Mobile Applications: The AR content can be available in Junaio, Layar and Wikitude browsers as a channel of information. First the channel has to be registered to the respective server of each AR vendor by the administrator. Then the end-user can type a keyword and the channel will be found. By exporting content to Free AR browsers, we can achieve a higher level of dissemination of information. Also, in the free AR browsers, the link to download the custom mobile application is advertised. So, if somebody wants to vote he/she can download the app and vote.

3.4.4 Software library API

The AR API allows the AR Server to communicate with the mobile devices, and the Authorization API allows the AR Server to authenticate a power user by communicating with the Service Center. These are the APIs that should be used in order to integrate the AR functionality with the rest of the components. The list of the functions for these APIs is shown in Table 23.

Table 23: AR and Authentication APIs

Function name ***Description***	**Request**	**HTTP method**	**Response**
AR API			
Metaio LBS AREL *Get Entities related to Location Based AR channel* *Used in:* • *AR Client Android* • *Junaio AR browser*	***/api_Metaio_v2/LBS/index.php?lang=en&idapp=0100&m=30&l=40.5678,22.999&distthres=100000&device=android*** *lang: language ISO-639-1, e.g. en (English), es (Spanish), eu (Basque)* *idapp: select application framework from set 1,2,3,4 by binary switching, i.e. 1000 = (1), 1100 = (1,2), 1101 = (1,2,4); 0100 is for UrbanPlanning, the others are not used* *m:max number of Entities to download, e.g. 30* *l: user location (latitude, longitude), e.g. l=40.567,22.999* *distthres: distance from user threshold (meters): Only objects closer to user below this threshold will be send to mobile app* *device: String "android" or "iphone". It is used from Junaio browser.*	GET	An XML response with the AR Entities details

Metaio IBS AREL: *Get Entities related to Image Based AR channel* • *Android AR Client* • *Junaio AR Browser*	***/api_Metaio_v2/IBS/index.php?lang=en&idapp=0100&device=android*** lang: language ISO-639-1, en (English), es (Spanish), eu (Basque) idapp: select application framework from set 1,2,3,4 by binary switching, i.e. 1000 = (1), 1100 = (1,2), 1101 = (1,2,4); *0100 is for UrbanPlanning* device: String "android" or "iphone"	GET	An XML response with the AR Entities details
Metaio IBS AREL2JSON: • iPhone AR Client	***/api_Metaio_v2/IBS/indexjson.php?lang=en*** lang: language ISO-639-1, en (English), es (Spanish), eu (Basque)	GET	A JSON response with the AR Entities details
Metaio LBS AREL2JSON: • iPhone AR Client	***/api_v3/ar_get.php?id_app=2*** idapp: select application framework from set 1,2,3,4 by binary switching, i.e. 1000 = (1), 1100 = (1,2), 1101 = (1,2,4); *0100 is for UrbanPlanning*	POST	A JSON response with the AR Entities details
CERTH IBS* JSON: Object visual recognition • Android AR Client	*/api_v3/recognizer.php* *$_FILES['upload']['tmp_name'] : the image to be recognized* *Id_app: 1,2,3,4 the application context. 2 is for Urban Planning. The others were not used.*	POST	String = "label;score; entity id" e.g. "tree; 0.1234; 15"
Layar LBS JSON: • Layar AR Browser	***/api_Layar_v1/index.php?lang=es&idapp=0100&lat=40.5678&lon=22.999&radius=10000000&layerName=Urban Planning*** lang: language ISO-639-1, en (English), es (Spanish), eu (Basque) *lat: latitude* *lon: longitude* *radius: range in meters* *layerName: a string to put as a title in the layer*	GET	A JSON with the Entities details

Wikitude LBS ARchitect: • Wikitude AR Browser	***/api_wikitude/LBS/index.php*** *Wikitude has a graphic user interface for testing where one should enter the longitude and latitude. It works with javascript on-the-fly.*	GET	An html with the Entities details
AR-Server internal API			
Web server connection to Service Center: Set credentials for a successful initialization of the authorization module in AR-Server so that power users can login using the GUI of the web portal (augreal.mklab.iti.gr).	***/auth/include/fg_memebersite.php*** *Open the php and change parameters:* *$arserver_username = 'ar_api_user';* *$arserver_password = '***';* *$admin_email = 'theadminmail@company.com';* *$site_name = 'augreal.mklab.iti.gr';* *$SC_path = "https://urbanplanning.yucat.com/servicecenter/api/";*	POST	It is a method inside the Web-portal. Upon successful login the user goes to "AR_Main.php" to see the list and the map with the AR Entities

3.4.5 Integration

The integration procedure is depicted in Figure 13. It is described how AR-Web, AR-Mobile, and AR-Server API interact with users and other components. The basic loop is the authentication or creation of a user, the request of permissions from SaaS Service Center, the retrieval of the permissions list. The base app requests the AR List from Mobile AR Component. The latter checks if the user has the permissions to do so by querying the SaaS Service Center. The Mobile AR Component then asks the Mobile AR API for data. The latter asks any other APIs if updated data is available. Permissions are sent to the other API components, and upon the confirmation of permission from other components, the data is transmitted. Next, the reception of updated AR List from the base app is achieved. Other APIs include internal or external APIs, such as Mobility API. The Action 2 describes how a user (UserB) can update the 3D data of AR-Server with the AR Web Portal. The UserB is logged in to the Web portal and authenticated through SaaS Service Center. Then he can manage the 3D models through the Web portal which calls the AR Server API.

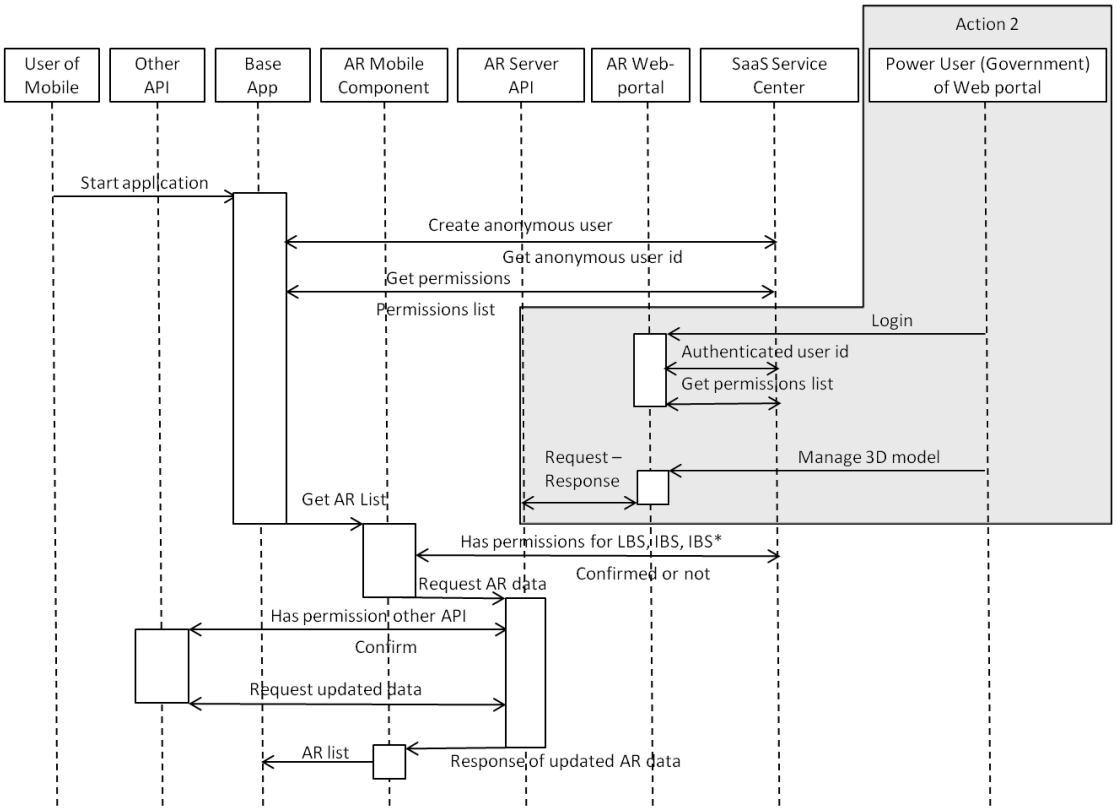

Figure 13: Augmented Reality Service Integration

3.4.5.1 Android AR Client

The AR functionality in the Urban Planning and the Mobility mobile applications is given by exploiting the MetaioSDK 5.5.2 which is provided for free by Metaio[4]. Urban Planning and Mobility serve as templates to start working for the mobile application of each use case. As seen in Figure 14, the templates for Urban Planning and Mobility are dependent in cascade mode on the Live+Gov Toolkit and LG_CERTH_Metaio_Lib where the latter is an intermediate library to ease the use of MetaioSDK.

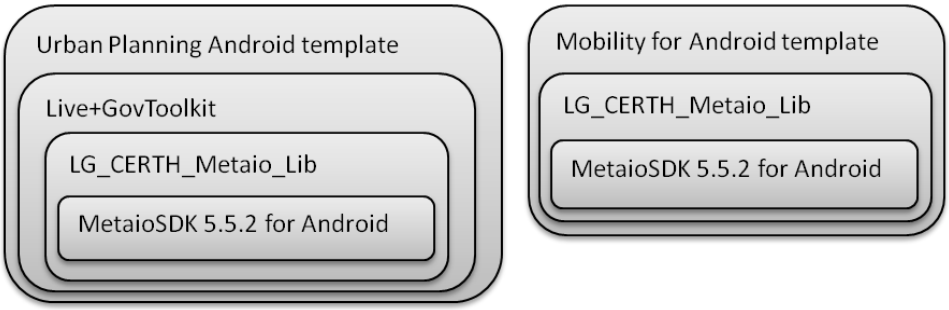

Figure 14: Urban Planning and Mobility are implemented based on MetaioSDK.

In detail, LG_CERTH_Metaio_Lib implements an AR view as a fragment which is a basic visualization element that can be inserted into the main activity, e.g. in a tab of an activity is a fragment. It was not implemented in MetaioSDK where an Activity was only given. AR view as a fragment can be inserted into a tab host next to a map and a list view. An additional

[4] http://metaio.com

feature is that Location Based AR (LBS), Image Based AR (IBS), and the Image recognition by CERTH (IBS*) functionalities are in the same AR fragment. The user can switch between the three modes with a button whilst the Metaio SDK is still in memory. This lowers the loading times and makes the application to run smoothly. It also provides the downloading functionality from AR-Server. In the LG_CERTH_Metaio_Lib, a file named Constants_API.java contains three parameters that should be modified in each installation:

- ServerName = "http://augreal.mklab.iti.gr" (it is the name of the AR-Server)
- rangeSTR = "5000000" (Distance from user (range) in meters where all Items within this range should be downloaded)
- Idapp = "0100" (0100 is the code for Urban Planning and 0001 is for Mobility use cases)

Further developer details can be found in the online developer manual in [15].

Live+Gov Toolkit provides the basic visualization and e-Government functionalities. The visualization functionalities is the construction of a ListView, User Information Questionnaire, Urban Plan Questionnaire, Urban Plan Questionnaire Results, MapView, and ARFragment Wrapper. The parameters that should be configured are located in the DownloadHelper.java file, namely:

- UP_SERVICE_CENTER_USER_NAME = "up_client_account";
- UP_SERVICE_CENTER_PASSWORD = "****";
- SERVICE_CENTER = "https://urbanplanning.yucat.com/servicecenter/api/" (it is the API endpoint for registering a mobile device)
- DIALOG_AND_VISUALIZATION_SERVICE="https://urbanplanning.yucat.com/dialogandvisualization/api/" (it is the API endpoint for the e-Government layer of the app)

Urban Planning Template is a shell where the functionalities of the Live+Gov Toolkit are inserted. The structure of the app is depicted in Figure 15.

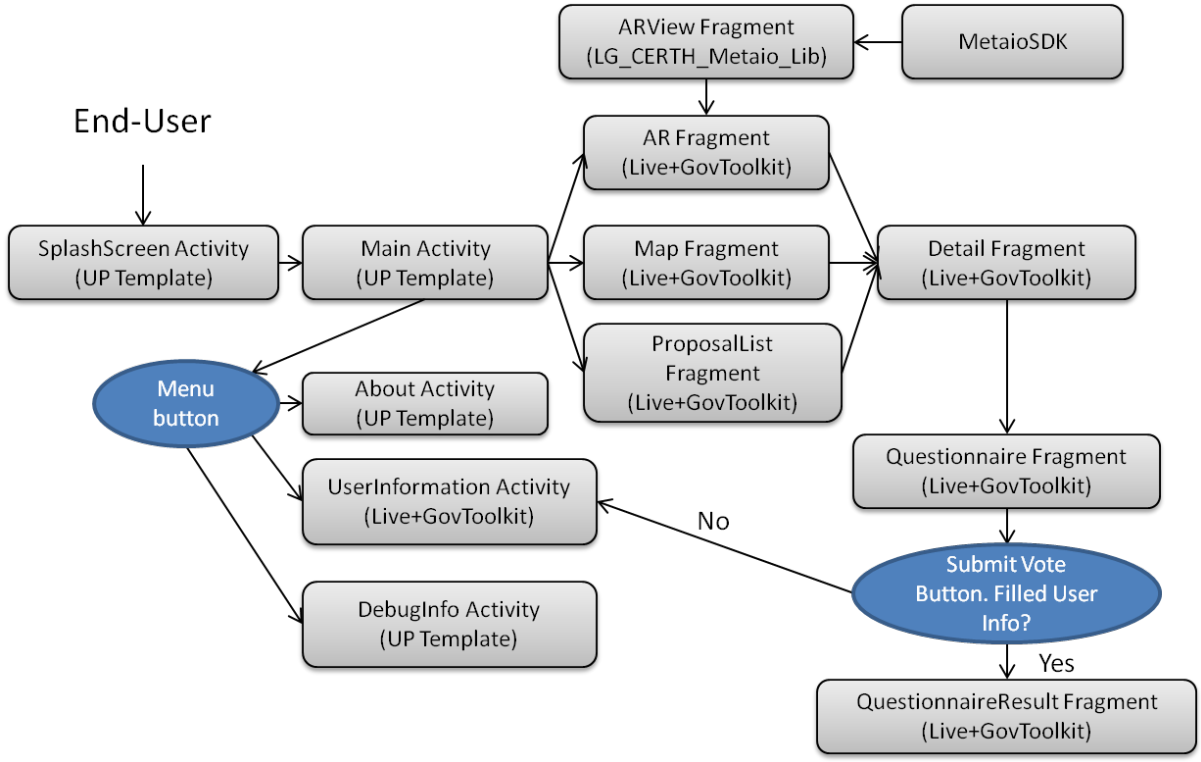

Figure 15: General structure of Urban Planning Android

The ARFragment contains several buttons that are related to its modes, namely the Location based (LBS), the Image Based (IBS), and CERTHs' Image Based Classification (IBS*) mode. A graphic representation of the functionalities is shown in Figure 16.

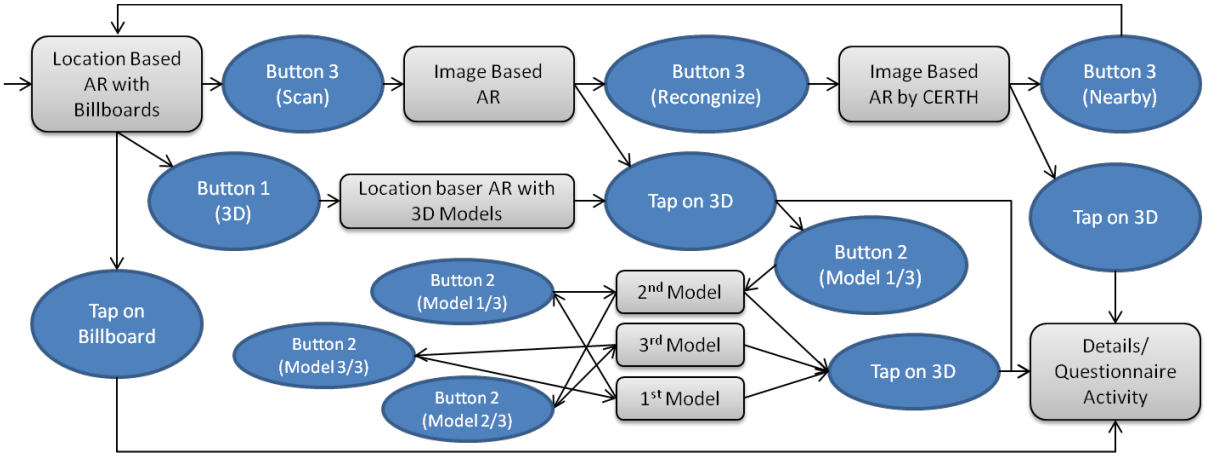

Figure 16: The diagram of the AR graphic user interface.

The first mode is the Location Based AR with Billboards (LBS Billboard) mode. In the LBS Billboard mode, only the billboards (points of interest, POI) and two buttons are shown. The Button 1 is a toggle button which toggles between Billboards and 3D models, i.e. Billboards disappear and the 3D models for each urban plan appears, and more specifically only the 1st model per urban plan is visible. The other button (Button 3) switches between LBS, IBS, and CERTH's IBS* channels. In LBS, the actions are coded in the following way:

A. By tapping on a billboard mode, the Details/Questionnaire activity is shown;

B. In LBS and IBS 3D mode, by clicking on a 3D model:

   I. If the clicked model corresponds to an Urban plan with no alternative 3D models (an urban plan can have up to 3 3D models) then Details/Questionnaire Activity appears

   II. If the clicked model has alternative 3D models then a new button appears named as 'Model 1/3' which can be used to replace the currently clicked model with the next alternative model in a cyclic fashion;

   III. If the clicked Urban plan is 're-clicked' then the Details Activity appears;

In IBS mode, a 3D model appears if its corresponding printed pattern is recognized. The recognition in IBS is done locally in the mobile device. In IBS* mode, the image is send to the remote server for recognition through HTTP. If the image is recognized with a positive recognition score, the label derived by the recognition procedure is matched against the Urban plans hosted in the AR Server. In case of a positive match, the matched Urban plan Details activity appears. The differences between IBS and IBS* are explained in detail in D3.1 [4].

The Debug Information Activity it is activated by pressing the menu button and it presents text data about the current status of the AR client and in general for the whole application. It

can be used for providing feedback to the user. Namely, the current temperature of the device, the position fixes through time, the data downloaded and uploaded are shown.

3.4.5.2 Mobility AR Client

Mobility AR Client is based on the LG_CERTH_Metaio_Lib library which provides an easy to use interface for the MetaioSDK 5.5.2 library. It should be used in a cascade manner as seen in Figure 17. The LBS mode is only used for the Mobility use case.

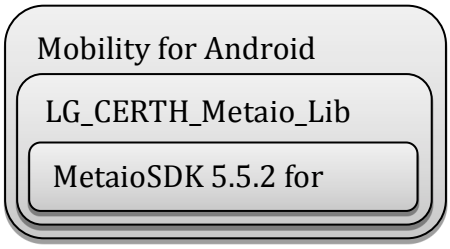

Figure 17: Architecture of Mobility AR Client

The integration steps of the Mobility AR Client are the same as described in section 3.4.5. The main difference is that a different source of data must be defined, like the HSL API. The AR view can be started as any other Android Activity via a button or some other event. A billboard view is the first thing that is loaded with the provided data such as bus stops. A detailed information screen can be shown as a popup when a user taps on a billboard. The detailed information screen can show additional information like the timetables of the bus stop. In order to use the HSL API, the user should register at the HSL developer API website and request an account[5].

3.4.5.3 iOS

An iPhone AR client for Urban Planning was written from scratch. It resembles the Android AR client for Urban Planning. The Urban Planning template for iPhone is based on the MetaioSDK 4.5 for iPhone. Around the MetaioSDK a wrapper was written by CERTH in order to provide its functionalities in the template. Additionally an Asynchronous HTTP library named as "ASIHTTPRequest.framework" was used in order to provide communication capabilities over HTTP network with several Servers. A graphic representation of the Template is shown in Figure 18.

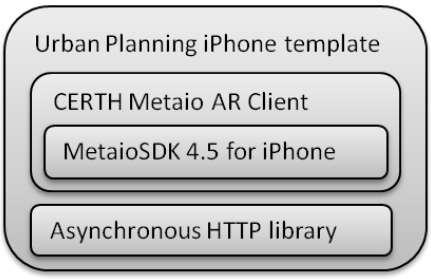

Figure 18: Organization of the Urban Planning iPhone Template

CERTH Metaio AR Client is a wrapper around MetaioSDK but not a library. The necessary classes for the wrapper are inside the template. The problem for making an intermediate

[5] http://developer.reittiopas.fi/pages/en/other-apis.php

library for iOS is that “MetaioSDK.framework” needs several configurations that will make it hard to support in any future MetaioSDK releases.

“ARELViewController.mm” is the file that creates the AR views and supports 5 AR “modes”. The first 3 modes are used in L+G project and correspond to the 3 modes of the Android version. Namely:

- **Location Based with custom AREL interpreter**: This creates an AR View without using the Metaio AREL interpreter but with an XML parser by CERTH which allows for high level of customizability e.g. custom billboards, radar, identifying the touched geometry etc. This mode requires as input the XML of the AR-Server LBS PHP script.
- **Image Based (Tracking application, IBS)**: This view uses a tracking image to render 3D objects on the user’s screen.
- **Image Based Recognition (Object Recognition, IBS*)**: This view captures frames from the camera and using the visual recognition service by CERTH recognizes an object. A recognition score is shown in the screen and if it exceeds the 0 threshold, a related 3D model is rendered in the centre of the screen.

The other 2 modes are not currently used in L+G project, but can be useful in future releases:

- **Location Based for other data available in memory**: This demonstrates how to create an AR View with your own data which can be retrieved from a different source other than AR-Server.
- **Location Based with Metaio’s AREL interpreter**: A Location Based AR view using the Metaio’s AREL interpreter. This view doesn’t allow much customization of the final output.

Detailed examples on how to instantiate this controller are included in “ViewController.mm” file. From the “LG_AR_Template” XCode project see “startARWithOptions” function. Depending on your location you may not be able to see any Billboards/3D models in the location based AR View because the distance of the entities exceeds a certain threshold. This threshold prevents Entities to be rendered on the screen when they are too far from the mobiles current location. To configure the rendering distance threshold you may need to call the following functions (please refer to the metaioSDK documentation for this - the default max distance is set to 10 km)

- “m_metaioSDK->setRendererClippingPlaneLimits”
- “m_metaioSDK->setLLAObjectRenderingLimits”

For location based applications you will have to define a View Controller class to launch when the geometry is touched. In the “touchesBegan” method in “ARELViewController.mm” is the correct place to define one.

3.4.6 Phase 2 updates

Table 24 describes the updates performed in preparation of the 2nd trials rounds.

Table 24: Augmented Reality Service Phase 2 updates

Augmented Reality Service		
Service / Update	**Update description**	**Rationale**
Content Server to Free AR Browsers (Junaio, Layar, Wikitude)	• LBS Junaio • IBS Junaio • LBS Layar • LBS Wikitude	Millions of users have these browsers. By having content to these browsers we increase the possibility of the project and our custom app to be found.
New Parameters for presenting 3D Models	• Scale • Rotation around z-axis	3D models are presented in the image based recognition in the correct scale and rotation depending on the photo shooting distance and angle.
Multilingual support for title and description per Urban Plan	Up to 4 translation are supported per plan	Multilingualism
Mobile Augmented Reality Client		
Service / Update	**Update description**	**Rationale**
Stability improvements in mobile AR client	Several improvements were made	Android AR Client had several bugs with respect to the asynchronous nature of downloading, loading and presenting AR entities. They were fixed.
iPhone support	A native AR Client was written for iPhone that presents the same content as the Android AR Client.	The AR client can be used in Android and iPhone applications
Speed improvements in downloading AR Entities from mobile AR Clients	Both the developed iPhone and Android AR Clients now use a cache system that it is used to avoid double downloading of files from AR-Server	There is no reason to download the same file twice. Files for AR are usually too big because they include 3D models. Speed and required bandwidth of the mobile applications are significantly improved.
Web application for Augmented Reality Configuration		
Service / Update	**Update description**	**Rationale**
New design for AR-Server web-portal	Information is organized into tabs to avoid overloaded first screens. The new web-portal for editing AR entities is based on a common API that underlies below the visualization part of the portal.	Better user interface and organization of the AR-Server web-portal
FastAR component with an Urban Planning template for Sobipro (UPARTS) were written for Joomla to allow a single click installation to already existed systems in	FastAR component and UPARTS provide an easy way to encompass in Joomla the functionalities of AR-. Further details are found in D3.2[21]	Several municipalities already have a content management system like Joomla. We need a sufficient way to install AR-Server to Joomla.

municipalities		

3.5 Personalized Content Delivery Service (S5)

The Personalized Content Delivery Service is responsible for filtering the content presented to the user based on his explicit, or implicit preferences

3.5.1 Description

The Personalized Content Delivery Service is provided as an Android library project and has four core functionalities

- Collecting location points at regular intervals
- Storing the data in the device's local database
- Processing the data to form the user's commuting profile
- Provide personalized information to the user according to the profile that has been formed

The general purpose of the service is to generate personalized content based on the most visited places of the user. The service has the ability to detect the times that the user is usually travelling to those places. By exploiting these data an application using this service can provide personalized information to the user such as routing options, public transportation timetable updates, service disruption information and more.

The service has been designed to run on the background without consuming additional power from the sensors or using much of the phones resources.

3.5.2 Architecture

The Personalized Content Delivery Service can be embedded in any Android application project. It has been designed to run exclusively on the mobile device side meaning that all the data collection and processing is handled locally. This decision was taken mainly due to the fact that the service handles sensitive data and communicating them to an external service will raise privacy concerns to the end users.

The component is divided in three parts:

a) The Personalization Service ensures that the data collection persists regardless of the state of the application. The service is automatically started on phone startup in order to always stay alive and collect data when the device is turned on.

b) The Database handles the data processing and storage. It also provides APIs to communicate directly with the client mobile application.

c) The Notification Scheduler is responsible for delivering personalized notifications at the appropriate time. It has the ability to communicate directly to the user via system notifications or notify the client application.

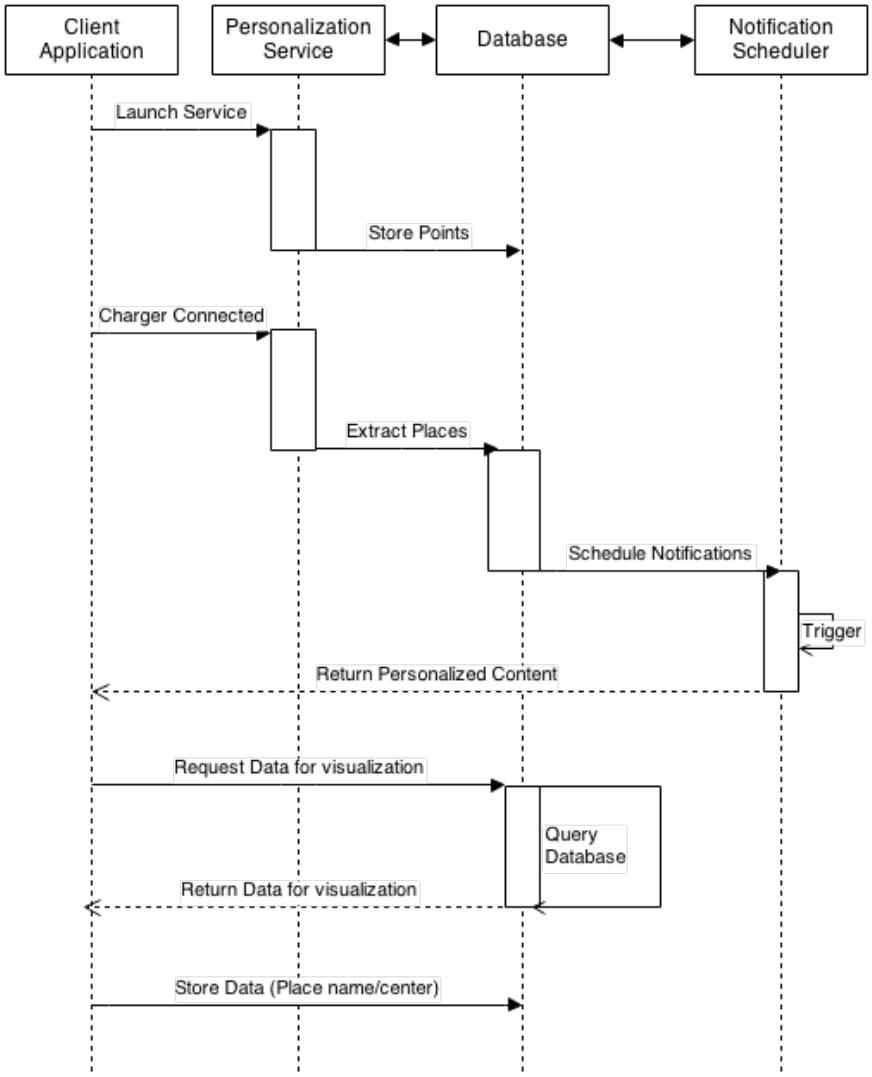

Figure 19: Integration with Client application

3.5.3 Integration

To incorporate the Personalized Content Delivery Service into an existing application, two BroadcastReceivers must be registered to launch the Personalization Service. The first one will be responsible to launch the service on the device start-up and the second one when the charger cable has been connected to the phone. This will ensure that the service will be enabled when the device is on and that the data process execution will run only when the device is charging, in order to minimize the impact on the device battery. A typical usage scenario is shown in Figure 19.

There are also some parameters that can be defined by the developer to adapt the service to his needs as shown in Table 25. Optimal values are predefined.

Table 25: Service parameters

Parameter	Description
INTERVAL_MINUTES	Interval for collecting location points in minutes

TRIP_NOTIFICATION_CANCEL_AFTER_MILLIS	The system notifications will be dismissed after this time has passed
TRIP_NOTIFICATION_SHOW_BEFORE_MINUTES	Minutes to send the notification before the arrival time to a place. The value should be between 0-59.
TRIP_NOTIFICATION_SHOW_BEFORE_HOURS	Hours to send the notification before the arrival time to a place.
MINIMUM_TIME_SPENT_IN_PLACE_SECONDS	Minimum duration that must be spent on a place by the user, in order to be considered a “Favourite Place”.
CLUSTER_RADIUS_METERS	The radius used to form a “Favourite Place”. The points that fall inside the radius are clustered together to form 1 Place.
REVISIT_TIME_PERIOD_SECONDS	A time period that is used to distinguish separate visits to a place.
DEVIATION_THRESHOLD_MINUTES	Maximum standard deviation allowed for accepting arrival and departure times in minutes. If the arrival times of a place exceeds this threshold, the arrival times will not be set, because they cannot be accurately estimated.

3.5.4 Software Library API

The client application is able to communicate with the Personalized Content Delivery Service via the API shown in Table 26. This API can be useful to visualize the data to the user and let the user correct the results of the automatically detected places.

Table 26: Software API

Function	Parameters	Returned Value	Description
addPoint	latitude, longitude, accuracy, timestamp	-	Stores a point to the database.
getPlacesFiltered	duration	list<Place>	Returns places that are not hidden, and that the user has visited for a defined duration.
getPlaces	-	list<Place>	Returns all detected places.
getPlaceByID	placeID	Place	Returns a single place by ID.
fixCenters	-	-	The places centers are re-calculated. This is automatically called at regular periods.
updatePlaceCenter	placeID, latitude, longitude	-	Manually update the center of place. This allows the user to correct the center of the place if the automatically detected one is not correct.
updatePlaceName	placeID, name	-	Sets a name for the

			detected place.
hidePlace	placeID	-	Hides a place.
showNotification	placeID	-	Shows a notification for the defined place.
addPlaces	-	-	Assigns location points to existing or new places. This is called automatically when charger is connected to the device but can be also called manually if necessary.

3.5.5 Phase 2 updates

The Personalization service is introduced in the 2nd Phase

3.6 eGovernment Dialogue and Visualisation Service (S6)

This chapter describes the eGovernment Dialogue and Visualization Service.

3.6.1 Description

The eGovernment Dialogue and Visualization service is responsible for offering citizen government dialogue and visualizing aggregated data in a user-friendly way. The following functions are offered by this service.

- Present initiatives (e.g. issue reports, proposals, co-maintenance spots)
- Add own initiatives (both citizens and government)
- Provide opinions and feedback on initiatives
- Facilitate polls and questionnaires
- Moderator functions for government to prevent abuse

3.6.2 Requirements

The eGovernment Dialogue and Visualisation Service is involved in realization of parts of the requirements described in

Table 27 [5]. For each of the requirements the current status is discussed and it is indicated if the requirement is implemented for the first or second trials. Please note that often the combination of multiple components is required to cover the whole requirement.

Table 27: Requirements eGovernment Dialogue and Visualisation

No	Live+Gov system requirements	Feature status	Part of 1st Trials	Part of 2nd Trials
L+G-FR.03	Be able to show information in a context aware manner (location, person, time-specific, situational) to create greater awareness and create mutual citizen-government understanding	The web application is able to present data. Both citizens and municipalities have access to information that provides possibilities for better understanding of contextual dependent reality.	Y	Y
L+G-FR.05	Provide communicative tools for government and citizens (including reactions, alerts, messages)	Communication possibilities are provided by the Web application. Future development will intensify the web-mobile interaction.	Y	Y
L+G-FR.06	Provide possibilities to visualize knowledge and patterns from data by data-mining techniques	In the first cycle of development, the possibilities for visualisation of knowledge patterns are provided with the web application.	Partial	Y
L+G-FR.12	Provide possibilities to retrieve personal opinions, votes or comments from users	Together with L+G-FR.05, the web application provides possibilities for free-form comments, reactions and opinions. Also the specific mobile questionnaires service is realised, which make specific surveying possible.	Y	Y
L+G-FR.20	Provide possibilities to present co-maintenance initiatives in an area	The web application is able to present co-maintenance initiatives in a map-based manner as well as a gallery-view is provided. Apart from co-maintenance initiatives, the web application is set-up in a generic fashion that makes it possible to present a large range of types of data.	Y	Y
L+G-FR.21	Provide possibilities to communicate urban planning plans	The mobile client implements the option to show items (e.g. urban planning plan) in a list and map view. Furthermore each item can be equipped with a questionnaire.	Y	Y

3.6.3 Architecture

The eGovernment Dialogue and Visualization Service consist of three components:

1) The eGovernment Dialogue and Visualization Service (C4), which offers the following functions that can be integrated using the Web API.

- Store and serve initiatives
- Store and serve feedback
- Store and serve photo's
- Store and serve poll results

- Store and serve questionnaire results

2) The Mobile eGovernment Dialogue and Visualization Client, which offers the following functions that can be integrated in mobile applications using the software API. E.g. the Urban Planning trial application.

- Show initiatives on map (e.g. issue report, proposal, co-maintenance spot)
- Show initiatives on photo gallery
- Add own initiatives as a civilian or government
- Show feedback on initiatives
- Give feedback to initiatives
- Facilitate polls and questionnaires

3) The Web application for eGovernment Dialogue and Visualization, which offers the following functions for the end-user, e.g. citizens and government officials.

- Show initiatives on map (e.g. issue report, proposal, co-maintenance spot)
- Show initiatives on photo gallery
- Add own initiatives (both civilians as government)
- Show opinion / feedback on initiatives
- Give feedback to initiatives
- Facilitate polls and questionnaires

Figure 20 shows the architecture of the eGovernment Dialogue and Visualization Service, the relation between the components in this service and the interaction with the SaaS Service Center.

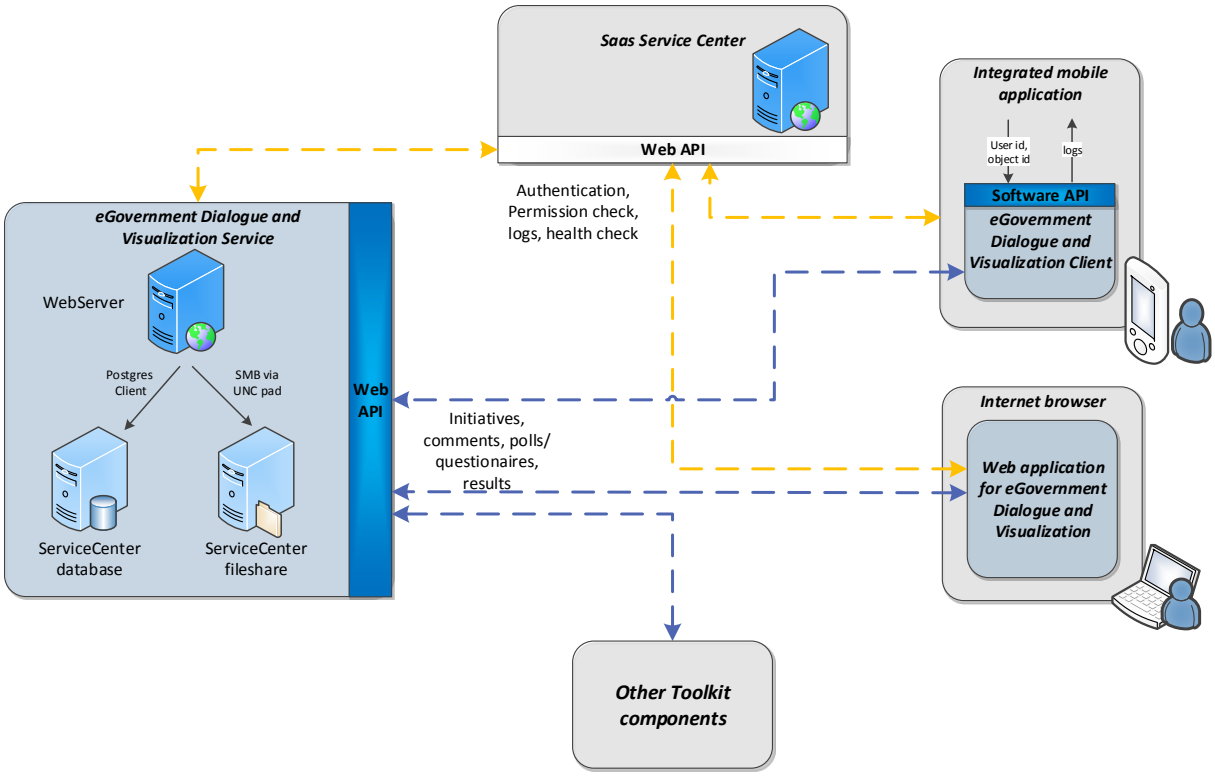

Figure 20: eGovernment Dialogue and Visualization architecture

3.6.4 Web API

Table 28 shows a summary of the eGovernment Dialogue and Visualization Web API. A full description of this API, including message example can be found in [16].

Table 28: eGovernment Dialogue and Visualization Web API

Function name	Purpose	Request parameters	Response parameters
Get layers	Gets the defined presentation layers in the visualisation. E.g. co-maintenance spots, issue reports, urban planning plans.	-	List with layers and properties
Get features in layer	Get the available features in the specified layer. E.g. the co-maintenance sports themselves.	Layer id	List with features and properties
Get feature	Get details of a specified feature.	Feature id	Details of feature
Get comments in feature	Get comments that are added to specified feature. E.g. comments of other citizens to co-maintenance sport.	Feature id	Comments
Get own features	Get the features associated with the logged-in user.	-	List with features and properties
Add feature	Add new feature to specified layer.	Layer id, Feature details	-
Add comments in feature	Add a comment to the specified feature.	Feature id, Comment details	-
Post contact	Send email message to specified email address	Email details	-
Post login	Login with username and password	-	-
Add image with feature	Add image to specified feature	Feature id, Image details	-
Update user	Update the details of the logged in user	User details	Full user details
Update feature	Update the details of a feature	Feature id, Feature details	Full feature details
Remove image from feature	Remove image from feature	Feature id, Image id	-
Remove feature	Remove feature from layer	Feature id	-
Submit questionnaire	Submit questionnaire for object	Anonymous user id	-
Get questionnaire	Get questionnaire and results for object	Anonymous user id, Questionnaire code, Object code}	Empty questionnaire, Questionnaire results

3.6.5 Software Library API

The mobile eGovernment Dialogue and Visualization Client (library) can be embedded in the trial application. The library offers screens (fragments) for presenting items and their details in a list view, map view. Furthermore a questionnaire/poll can be submitted for each item. After submitting the questionnaire/poll the intermediate results of the (statistics) of the questionnaire/poll can be presented. Figure 21 shows the interaction between the library and the trial application.

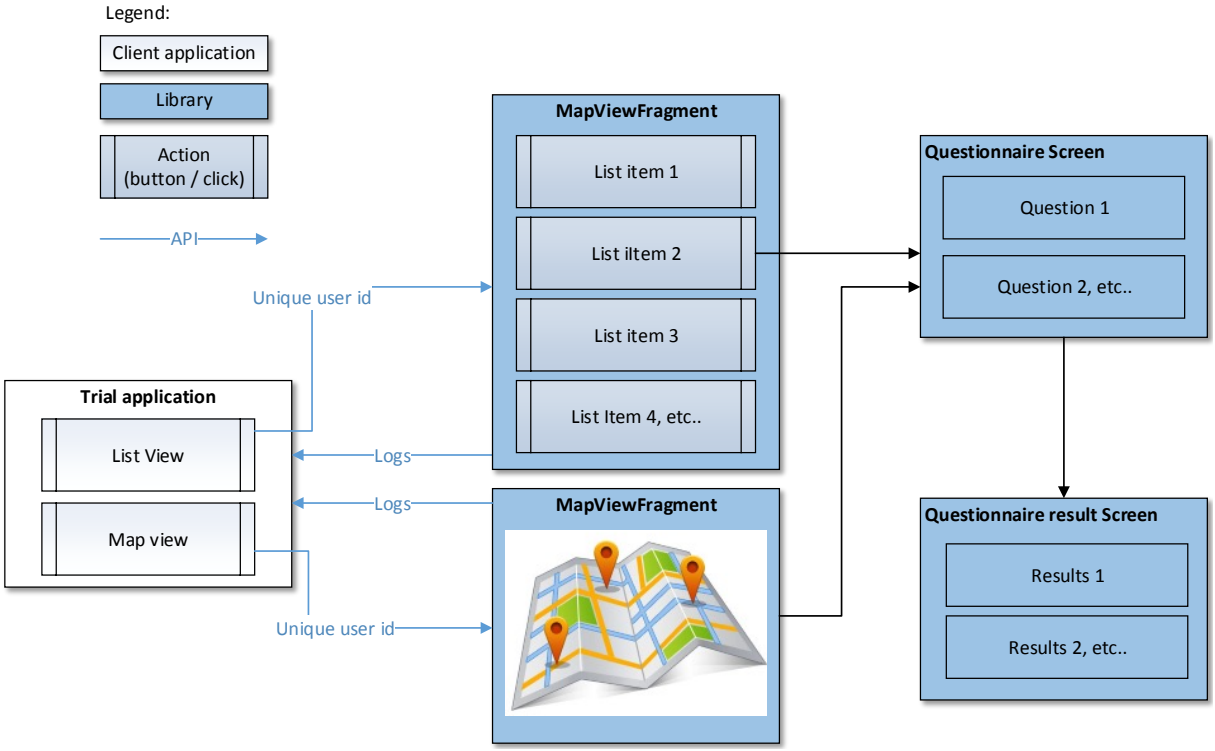

Figure 21: eGovernment Dialogue and Visualization Software API

3.6.6 Integration

Figure 22 shows the interaction between the different components in the eGovernment Dialogue and Visualization Service and the foreseen integration with the Service Center. The scenario shows a user that browses through the web application. The user logs in and is authenticated with the Service Center. A permission list is received, which determined which functionality is available for this user. Furthermore all actions performed on the service API are checked against the Service Center (has permission). Periodically log files and Health Check updates are provided to the service center.

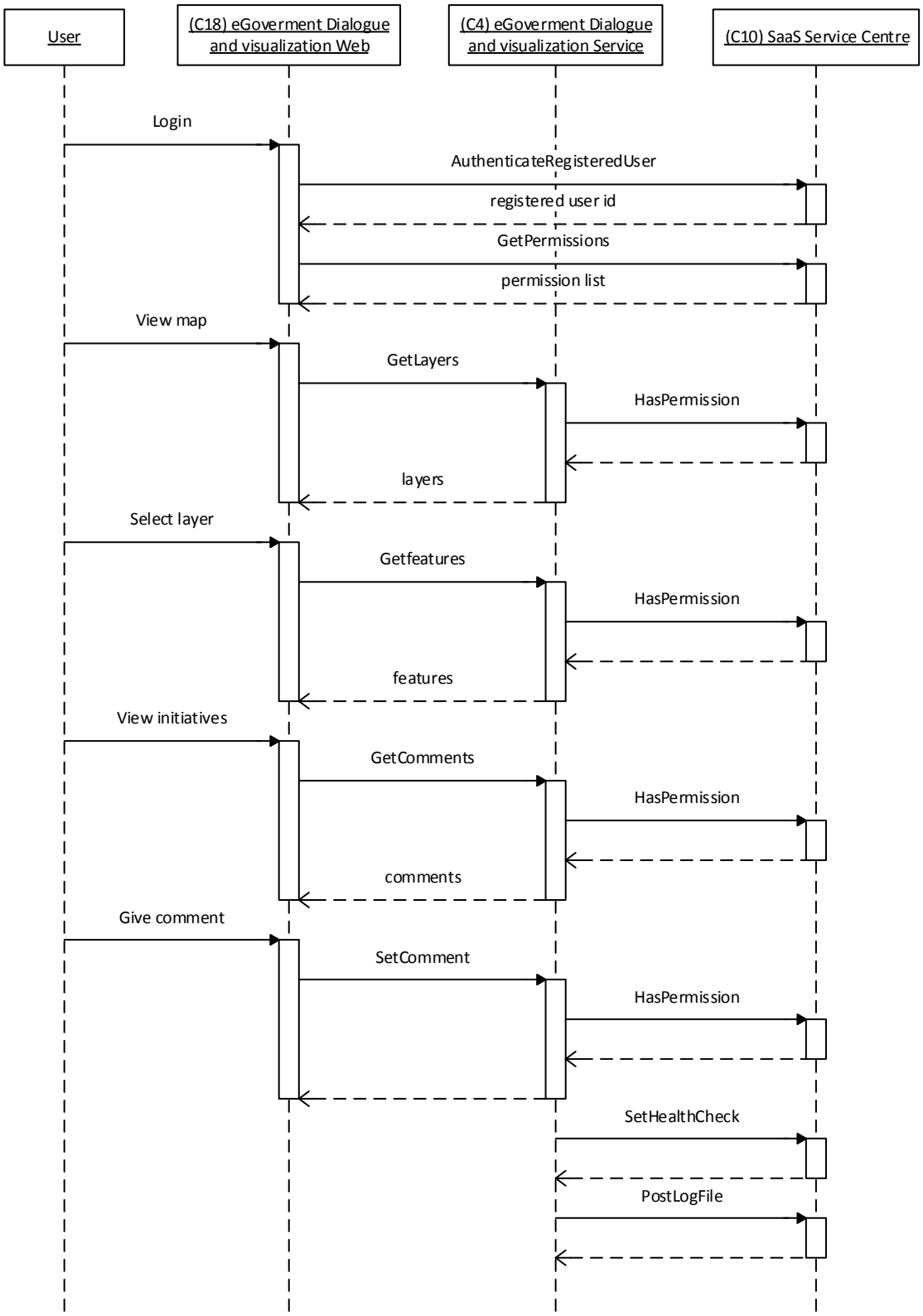

Figure 22: eGovernment Dialogue Service integration

3.6.7 Phase 2 updates

Table 29 describes the updates performed in preparation of the 2nd trials rounds.

Table 29: eGovernment Dialogue and Visualization Service Phase 2 updates

eGovernment Dialogue and Visualization Web application		
Update	**Update description**	**Rationale**
Multilanguage support	Support for multiple languages in a single instance of the web application.	Required for the Urban Planning trial were Spanish and Basque versions are required by Law.
Questionnaire statistics view	Added generic view for the presentation of summarized statistics on submitted questionnaires.	Required for the Urban Planning trial were the government official need to analyse the results of the voting rounds.
Access control	Added login (integrated with service center) to restrict access to the web application.	Required for the Urban Planning trial were the web application is not public, but used by the government officials.
Deeplinks	Added the possibility to create a direct link (deep link) to a specific initiative.	Required for the Urban Maintenance trial. This makes sharing specific initiatives via e.g. social media or mail possible.
Social media sharing	Added the possibility to share an initiative via Twitter, Facebook, LinkedIn or mail.	Required for the Urban Maintenance trial to make promotion of initiatives via other channels easier.
What's new?	Added past/present/future filters to make clear what content is relevant at the moment.	Required for the Urban Maintenance trial to give a better presentation of the dynamic character of the web application.
Tooltips	Added the possibility to show tooltips when a user hoover over a button.	Required for the Urban Maintenance trial to give addition explanation on the different sections and functions of the web application
Search option	Update of the search option. In the first version the search was only location (Street, postal code) oriented, in the second version a generic text search is implemented.	Required for the Urban Maintenance trial to improve utility and efficiency.
Mail notifications	Added mail notifications for initiative owner when new reactions are added to initiatives	Required for the Urban Maintenance trial to improve the responsiveness in the dialogue.
Better mobile support	Multiple refinements have been implemented to improve the support on mobile devices like tablets.	Required for the Urban Maintenance trial to improve utility and reliability.
Cookie Bar	Ask users for permission to use cookies to improve their user experience.	Required by European legislation
Browser check	Detect browser that are too old for viewing the web application in	Improve reliability, usability and stability.

	a proper way, and advise them to install a new browser.	

3.7 Issue Reporting Service (S7)

This chapter describes the Issue Reporting Service.

3.7.1 Description

The Issue Reporting Service is responsible for providing functionality to:

- Post 'rich' issue reports including location, photo, category and contact information.
- Forward issues to external systems (Public Administration systems)
- Provide visualizations and an overview of the reported issues.
- Provide enhanced feedback on issues to citizens

3.7.2 Requirements

The Issue Reporting Service is involved in realization of parts of the requirements described in Table 30 [5]. For each of the requirements the current status is discussed and it is indicated if the requirement is implemented for the first or second trials. Please note that often the combination of multiple components is required to cover the whole requirement.

Table 30: Issue Reporting Requirements

No	Live+Gov system requirements	Feature status	Part of 1st Trials	Part of 2nd Trials
L+G-FR.08	Provide possibilities for textual feedback. For example how his contribution is taken into consideration, including possible benefits for participating and status report on the situation	The Issue Reporting Service provides the possibility for Extended feedback. For municipalities, a specific web interface to provide the feedback has been developed and implemented. This requirement is linked to L+G-FR.23.	Y	Y
L+G-FR.09	Allow a user to view all of the issues that are currently open, and also a history record of "my issues" (where the citizen has participated) and "my municipality" (where all of the past issues from the town can be found) with a short summary of each of the past issues.	The web application for extended feedback is extended with a list overview and geographical overview of all issue reports.	Y	Y
L+G-FR.10	Be able to utilize 'citizens as sensors' for their environment by mining relevant data (e.g. location, context, pictures, reports) from their reality	The mobile issue reporting client for citizens senses the experienced context automatically and by manual user-actions	Y	Y
L+G-FR.13	Be able to connect to different municipality administration systems	In several municipalities Issue reports are forwarded to the public administration system via open standards or proprietary formats.	Y	Y

L+G-FR.19	Collect and mine citizen experience and citizen (implicit) norms	The issue reporting client is able to extract and collect the context dependent citizen experiences. (Semi) automatic classifications to implicit norms is subject for future research and development in cooperation with Data-mining, augmented reality and object recognition features.	Partial	Y
L+G-FR.22	Deliver alerts from users to other users if a number (configurable) of users report the message from the same area	The Issue reporting Service is able to deliver alerts entered in the web application to all users.	Partially	Y
L+G-FR.23	Provide possibilities for Feedback as picture	The first step has been implemented: textual feedback possibilities. The next step will be further advancements on the Extended feedback mechanism in such a manner that apart from status and textual feedback, also visual feedback options are provided.	N	N

3.7.3 Architecture

The Issue Reporting Service consist of four components:

1) The Issue Reporting Service (C4), which offers the following functions that can be integrated using the Web API.

- Handles connections with Mobile Issue Reporting Client (C11).
- Receives issues reports from mobile clients or other toolkit components.
- Stores issues reports in database and photo's on file system.
- Provides required municipality info, issue categories, related images and current issue status.

2) The Issue Distribution Service (C2). Forwards issue reports to specific municipality based on coordinates.

- Offers generic API for requesting or adding issue reports
- Offers Plug-ins for forwarding issues reports to specific Public Administration systems.
- Offers issue mail-forwarding service for 'low-tech' municipalities, without advanced interface options.

3) The Issue Updating Service (C3). Issue Reporting Updating Service. Updates issue report status information.

- Offers generic API for updating issue reports status information
- Offers Plug-ins for updating issues reports from specific Public Administration systems.

4) The Mobile Issue Reporting Client (C11), which offers the following functions that can be integrated in mobile applications using the software API. E.g. the Urban Planning trial application.

- Submit issue report including:
  - Contact information (name, address, telephone, email)
  - Photo of situation
  - Location based on GPS or manual input
  - Issue Label
  - Issue description
- Receive feedback on current status of issue

5) The Web application for Enhanced Issue Report Feedback (C16) enables a 'low-tech' municipality to give feedback to issue reports. They receive an e-mail for the issue reporting distribution service, including a link to this web application. In this web application one can change the status of the issue report and add a free text message for the citizen that submitted the report.

- Show detail issue report information including contact details, issue report details, photo, maps.
- Government official can change issue report status
- Government official can add free text message explaining the current status of the issue report.

Figure 23 show the architecture of the Issue Reporting Service and the relation between the components in this service.

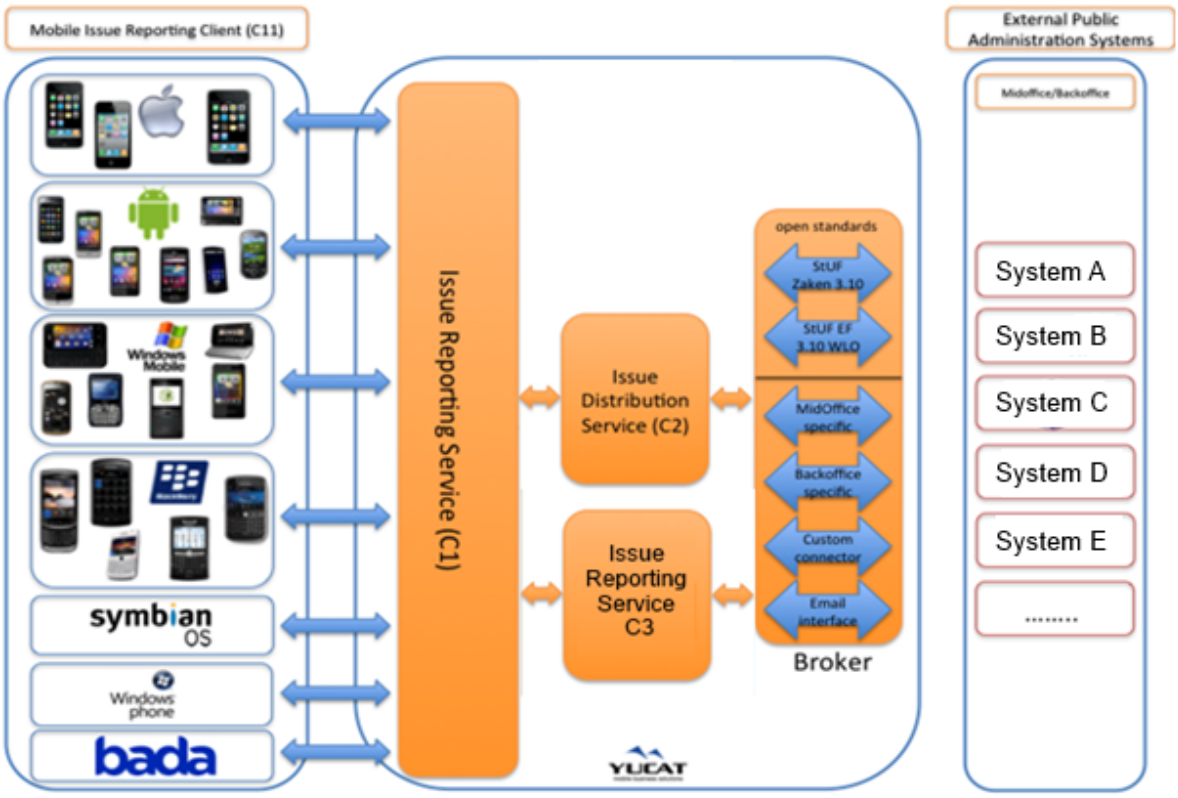

Figure 23: Issue Reporting Service Architecture

3.7.4 Web API

Table 31 shows a summary of the Issue Reporting Web API. A full description of this API, including message examples can be found in [16].

Table 31: Issue Reporting Web API

Function name	Purpose	Request Parameters	Response parameters
Get Municipality Info	Get details (contact info, major, number of inhibitions) of the current municipality based on current location	Location	Municipality info
Get Municipality Logo	Get the logo of the current municipality based on municipality id	Municipality id	Municipality logo (binary)
Get Category List	Get the possible Issue labels of the current municipality based on location	Location	Possible Issue label
Get Label Icon	Get the icon of a label based on the label id	Label id	Label Icon (binary)
Post Issue Report	Post an issue report to the current municipality based on location.	Location info Issue label Remark User info	Issue Report id
Post Issue Report Attachment	Add a photo to the issue report.	Issue Report id Photo	Issue Report Attachment id
Get Issue Report Status	Request a status update of a specified list of issue reports.	List of Issue Report id's	Issue Report is, Issue Report status code Issue report status message

3.7.5 Software Library API

The mobile Issue Reporting Client can be embedded in the trial application. The library offers screens (fragments) for submitting an issue report and an overview of details and status (including feedback) of the submitted issues. Figure 24 shows the interaction between the library and the trial application.

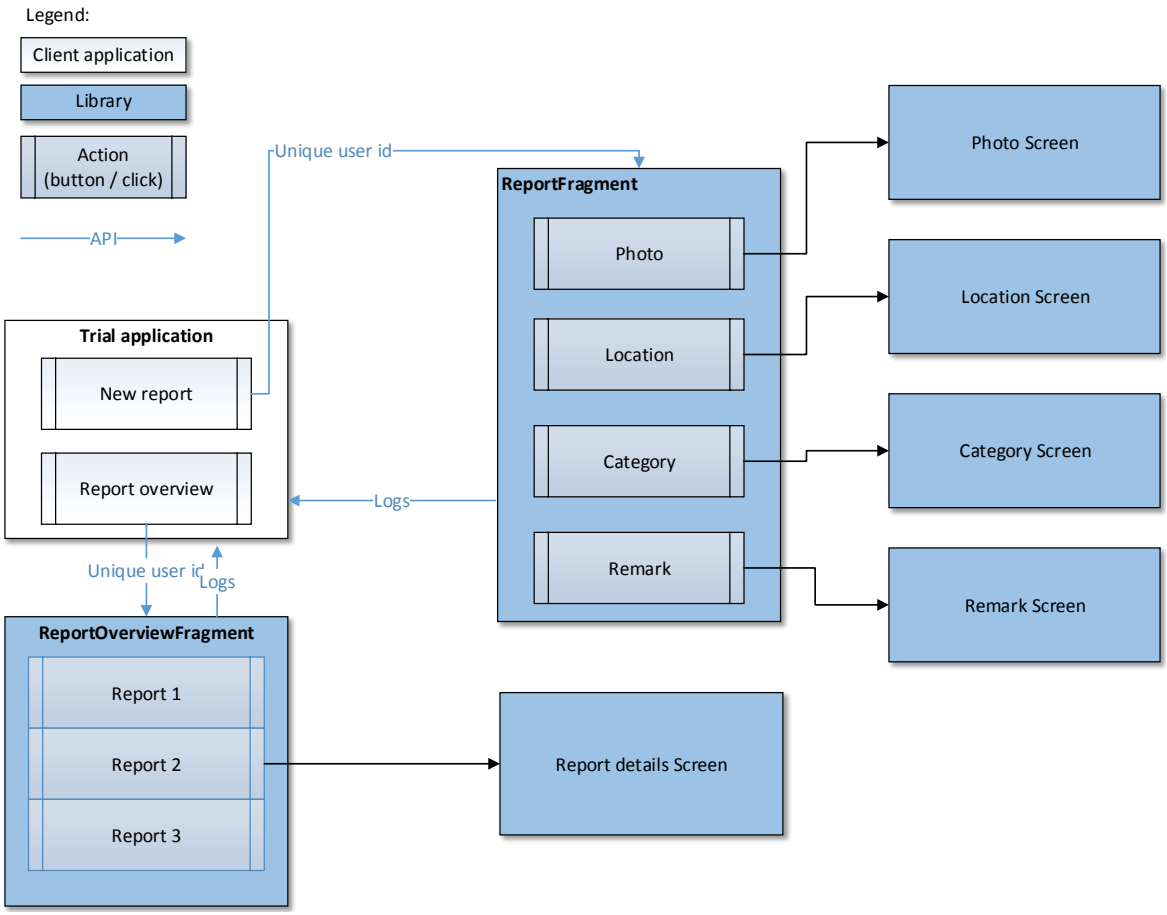

Figure 24: Mobile Issue Reporting Client Software API

3.7.6 Integration

Figure 25 and Figure 26 show the interaction between the different components in the Issue Reporting Service and the foreseen integration with the Service Center.

The first scenario (Figure 25) shows a user using the mobile issue-reporting client. First the anonymous user details are passed to the Service Center. A permission list is received, which determined which functionality is available for this application instance. All actions performed on the service API are checked against the Service Center (has permission). Periodically log files and Health Check updates are provided to the service center.

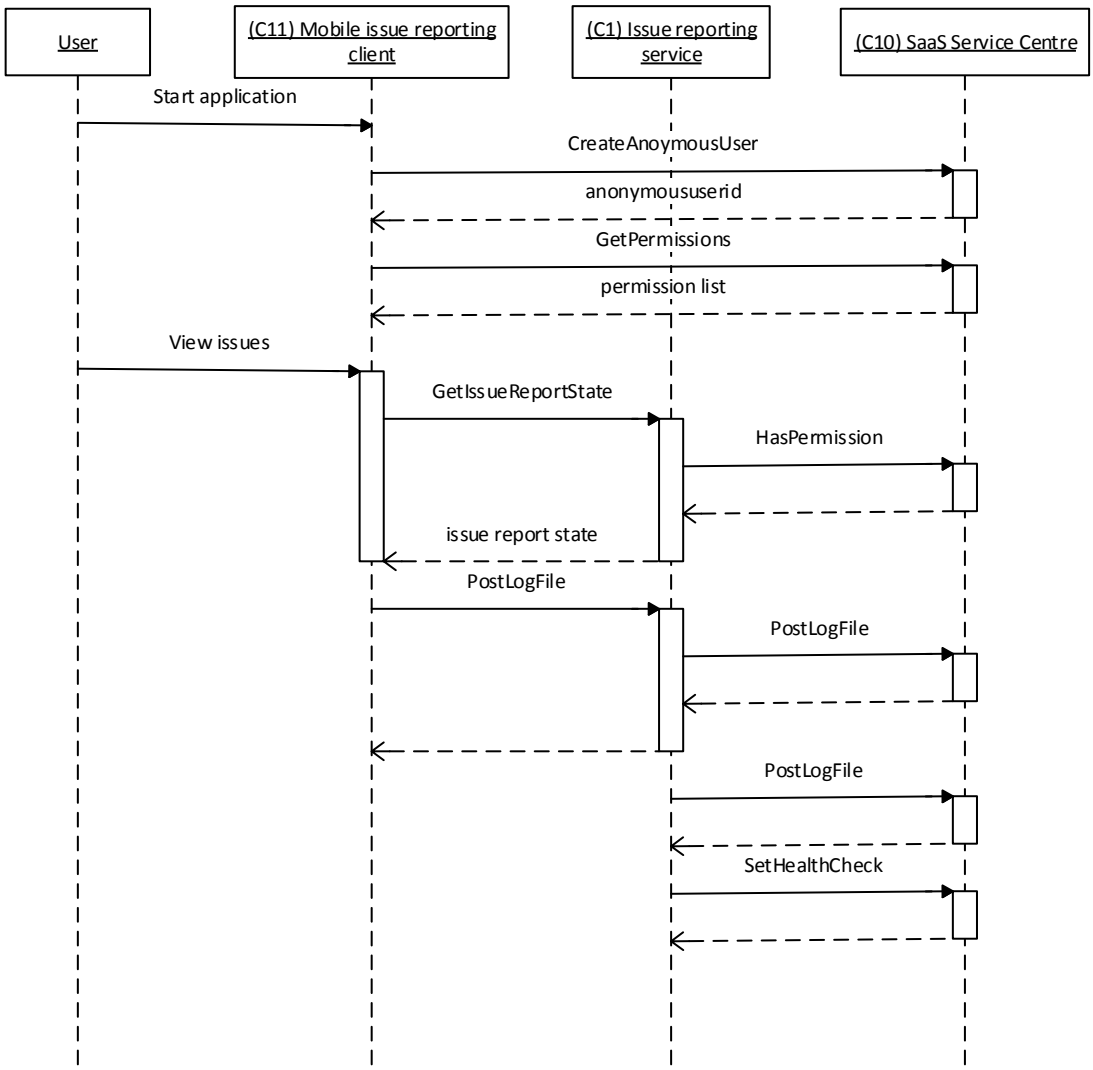

Figure 25: Issue Reporting Service Integration

The second scenario (Figure 26) shows a user that logs into the Web application for enhanced Issue Report Feedback. The user logs in and is authenticated with the Service Center. A permission list is received, which determined which functionality is available for this user. Furthermore all actions performed on the service API are checked against the Service Center (has permission). Periodically log files and Health Check updates are provided to the service center.

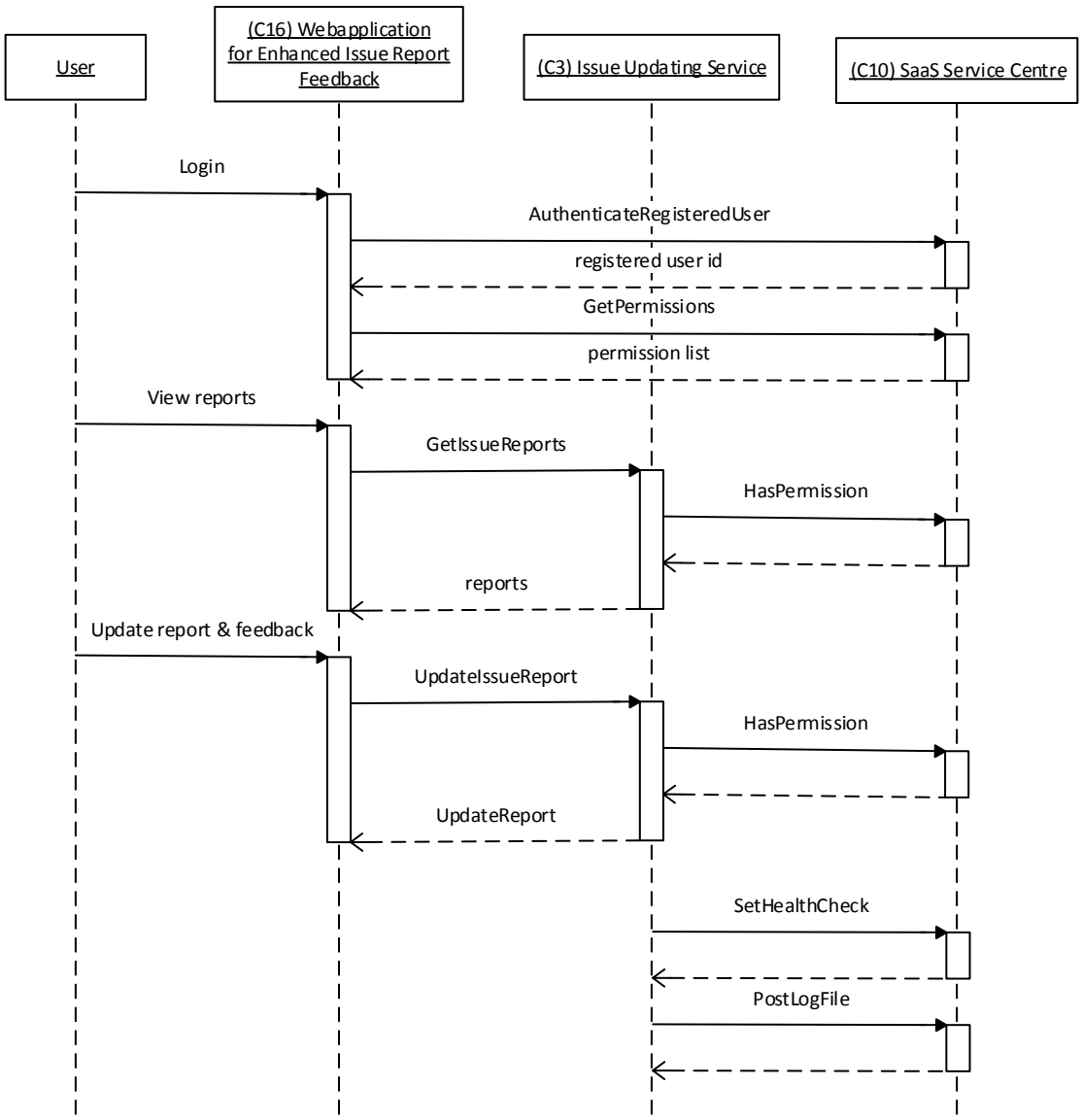

Figure 26: Issue Reporting Service Integration

3.7.7 Phase 2 updates

Table 36 describes the updates performed in preparation of the 2nd trials rounds.

Table 32: Issue Reporting Service Phase 2 updates

Web application for issue reporting		
Update	**Update description**	**Rationale**
Heat Map View	Added a heat map view to present issue report distribution	Useful for quick assessments of concentration of issue reports and user experiences of urban space
Photo Gallery View	Added a photo gallery view to present issue report photos	A great tool for quick impressions of the user experiences of urban space

Time slider	Added a time slider for easy selection of a time period to filter the different view	More friendly user interface
Server side clustering for issue report map view and heat map	Improved performance by calculation clusters server side instead of client side	Performance improvement to be able to present large amount of issues
Multi-tenant	Running multiple customers on the same instance	Required for SaaS based exploitation.

3.8 Traffic Service (S8)

This chapter describes the Traffic Service from functional, technical and integration viewpoints. The Traffic Service provides solutions, services and programming interfaces for applications interested in mobility data.

3.8.1 Description

The Traffic Service (S8) contains the components Traffic Jam Detection (C19), External Connector for Public Transport System (C21) and Route Analysis Web Component (C22).

The Traffic Jam Detection uses the External Connector for Public Transport System to collect the real time public transportation vehicle locations and based on that information it calculates the traffic congestion status.

The Route Analysis Web Component is a rich Internet application that the public transportation authorities utilize to analyse how the public transportation network is used by the citizens.

3.8.2 Requirements

The Functional Requirements are listed in D5.1 [7]. The following table refers to the numbering used in D5.1.

Table 33: Requirements related to the Traffic Service

No	Live+Gov system requirements	Feature status	Part of 1st Trials	Part of 2nd Trials
L+G-FR.03	Be able to show information in a context aware manner (location, person, time specific, situational) to create greater awareness and create mutual citizen-government understanding.	The Route Analysis Web Component provides various reports in the user interface for public transportation authorities.	N	Y
L+G-FR.04	Provide possibilities to group, aggregate, filter large amounts of data to discover knowledge.	The Route Analysis Web Component uses the Sensor Data Storage Service for the analysis.	N	Y
L+G-FR.06	Provide possibilities to visualise knowledge and patterns from data by data mining techniques.	The Route Analysis Web Component uses the Sensor Data Storage Service for the analysis. The visualization is done drawing the information on a map.	N	Y

L+G-FR.10	Be able to utilise 'citizens as sensors' for their environment by mining relevant data (e.g. location, context, pictures, and reports) from their reality.	The Route Analysis Web Component provides the user interface for public transportation authorities for the recorded citizen data.	N	Y
L+G-FR.13	Be able to connect to different municipality administration systems.	The External Connector for Public Transport System to get real time traffic information is created.	Y	Y
L+G-FR.15	Be able to analyse a recorded route in public transport with respect to relevant criteria like 'waiting times' and 'number of changes'.	The Route Analysis Web Component provides the needed analysis for the end user. The component uses the Sensor Data Storage Service as the data source.	N	Y
L+G-FR.17	Detect traffic jams.	Traffic Jam Detection component uses the External Connector for Public Transport System to get real time traffic information and detect traffic jams in Helsinki region area.	Y	Y

3.8.3 Architecture

The Traffic Service consist of three components:

3.8.3.1 External Connector for Public Transportation System

The public transport system connector is responsible for exchanging transport related data with the government transport systems. Due to a low degree of standardization of information exchange with these transport systems, one potentially has to deal with a lot different system interfaces. To hide this complexity the public transport system connector offers a single interface hiding the external system complexity.

Traffic System include external connector implementation Public Transportation Information Reader (visualized in Figure 27). The Public Transportation Information Reader has an implementation to read real time and static (planned) public transportation traffic data from Helsinki area.

3.8.3.2 Traffic Jam Detection

The main functionality of the component is to provide information about traffic jams in a specific area. The simplified architecture is visualized in Figure 27. The public transportation systems generally differ in each city or area in multiple ways in the available data and data formats. The different public transportation systems are isolated in the module using the external connector and it is configurable using the Spring Framework.

A generic API is offered to the Live+Gov toolkit for requesting the current traffics jams in a certain area. Figure 28 shows the message structure of the generic API offered to the Live+Gov Toolkit. The jam information includes information about involved vehicles (e.g. trams) and nearest stop locations, so the jam location can be approximated either from the participating vehicle locations or stop locations.

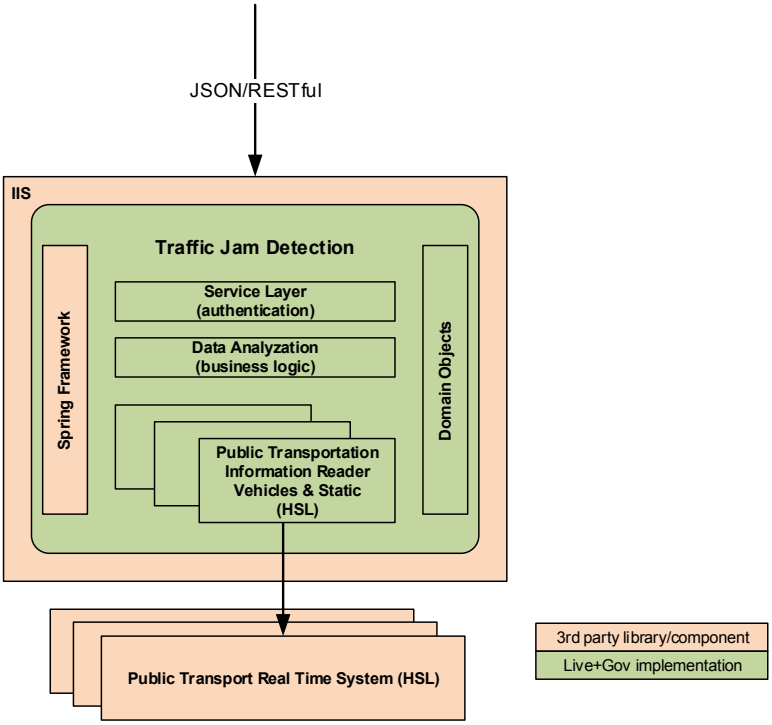

Figure 27: Traffic Jam Detection Architecture

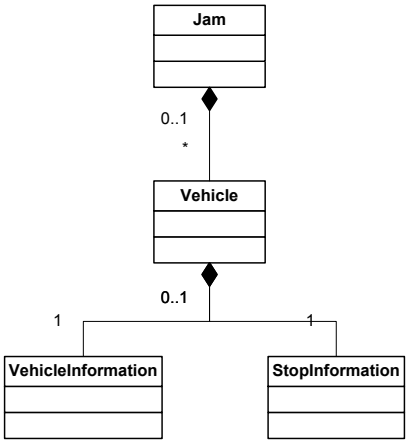

Figure 28: Traffic Jam Message Structure

3.8.3.3 Route Analysis Web Component

The Route Analysis Web Component (Figure 29) is a rich Internet application. The server side of the application uses the Live+Gov Sensor Data Storage Service as the data source to obtain the citizen sensor data. The user interface runs on the web browser and visualizes the data over the map. The application uses advanced GIS based SQL queries to create needed reports for the end user.

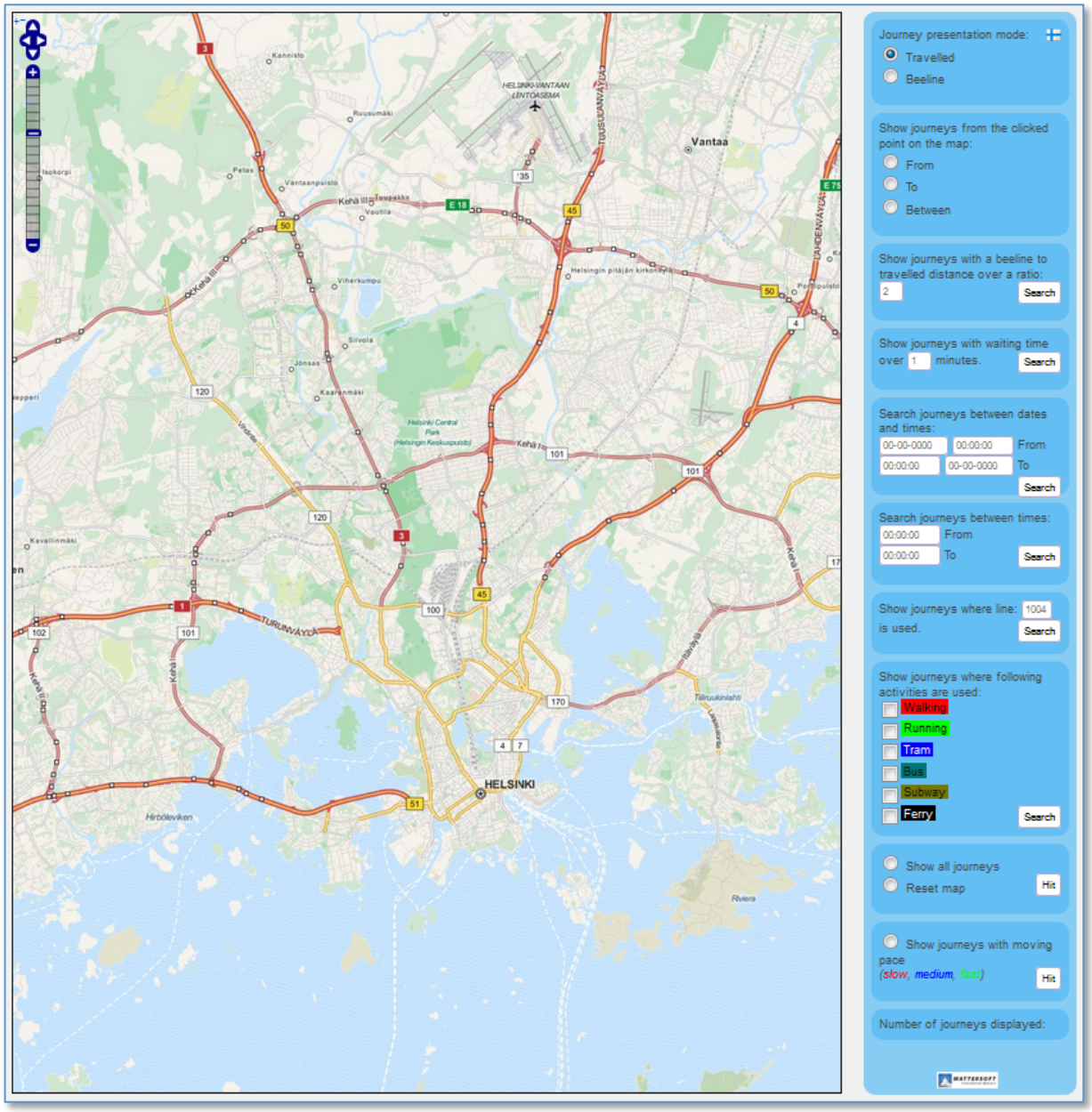

Figure 29: Route Analysis Web Component

The simplified architecture and the most relevant used technologies are visualized in Figure 30.

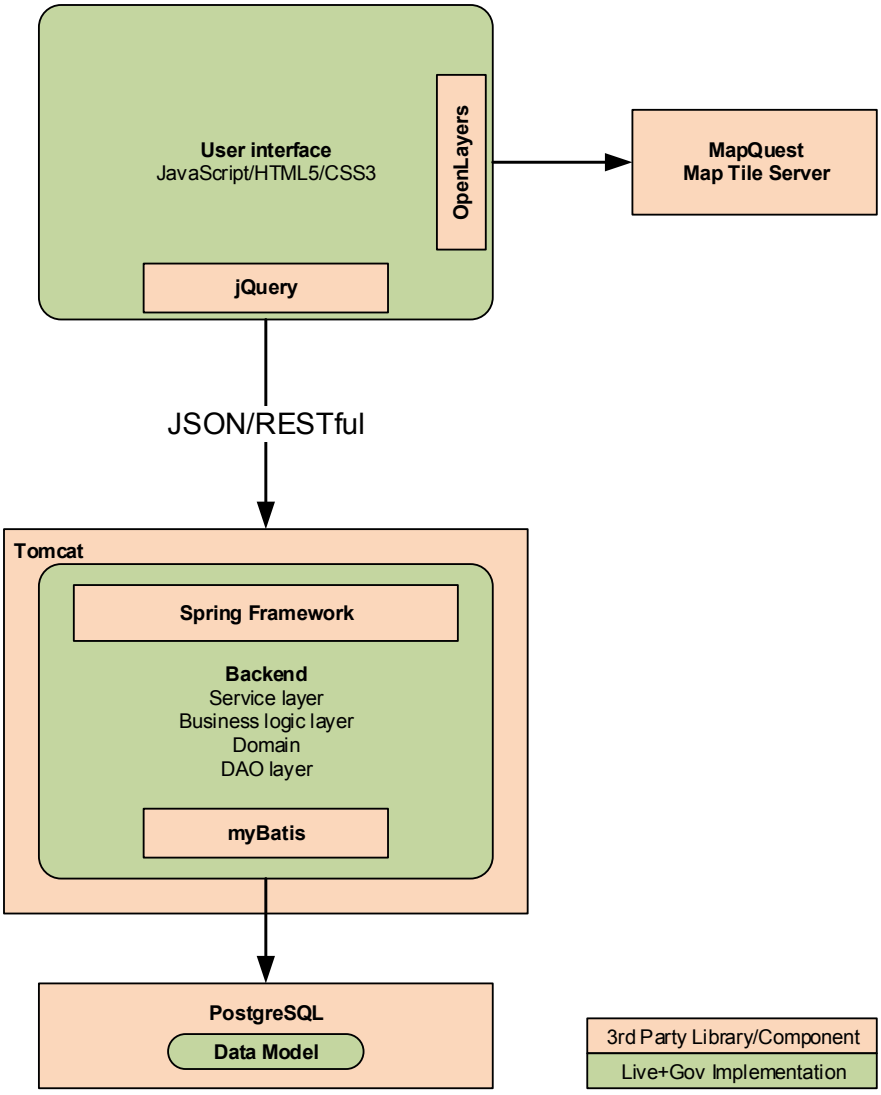

Figure 30: Route Analysis Web Component architecture

3.8.4 Web API

The Traffic Jam Detection component provides a Web API, which is described in the Table 34. The currently supported area is Helsinki.

Table 34: Public Transport System Connector API

Function name	Purpose	Request Parameters	Response
Get Traffic Jams	Get current traffic jams in a certain area.	Area	Traffic jam information
Get Slow Vehicles	Get current slow vehicles in a certain area.	Area	List of slow vehicles in the area

/jamdetector/JamService.svc/jams/hsl

Example request to get traffic jams:

The response is provided in JSON format. An example of a response is:


```
[ { "Id" : 0,
    "IsJam" : true,
    "SlowVehicles" : [ { "CumulativeDelay" : 13,
          "NextStop" : { "Code" : "1301453",
              "Latitude" : 60.197929999999999,
              "Longitude" : 24.876580000000001,
              "Name" : "Laajalahden aukio"
            },
          "PreviousStop" : { "Code" : "1301455",
              "Latitude" : 60.195390000000003,
              "Longitude" : 24.873429999999999,
              "Name" : "Tiilimäki"
            },
          "Stop" : { "Code" : "1301455",
              "Latitude" : 60.195390000000003,
              "Longitude" : 24.873429999999999,
              "Name" : "Tiilimäki"
            },
          "Vehicle" : { "Delay" : 13,
              "Id" : "RHKL00076",
              "IsOnStop" : false,
              "Latitude" : 24.873405999999999,
              "LineDirection" : 2,
              "LineId" : "1004",
              "Longitude" : 60.195535999999997,
              "NextStopIndex" : 2,
              "Time" : "20140116-102700",
              "Timestamp" : "/Date(1389860820413+0200)/"
            }
        },
        { "CumulativeDelay" : 72,
          "NextStop" : { "Code" : "1240419",
              "Latitude" : 60.203020000000002,
              "Longitude" : 24.96584,
              "Name" : "Kyläsaarenkatu"
            },
          "PreviousStop" : { "Code" : "1230410",
              "Latitude" : 60.204529999999998,
              "Longitude" : 24.96998,
              "Name" : "Toukoniitty"
            },
          "Stop" : { "Code" : "1230410",
              "Latitude" : 60.204529999999998,
              "Longitude" : 24.96998,
              "Name" : "Toukoniitty"
            },
          "Vehicle" : { "Delay" : 82,
              "Id" : "RHKL00065",
              "IsOnStop" : false,
              "Latitude" : 24.968769999999999,
              "LineDirection" : 2,
              "LineId" : "1006",
              "Longitude" : 60.203980999999999,
              "NextStopIndex" : 3,
              "Time" : "20140116-102700",
              "Timestamp" : "/Date(1389860820405+0200)/"
            }
        }
      ],
    "SlowVehiclesInJamCount" : 2
  } ]
]
```

In this example the detector has detected one jam and there are two vehicles in the jam. In addition to the vehicle information the next, previous and nearest stop information is returned. The jam location can be approximated either from the participating vehicle locations or stop locations.

The interface contains more information than the current clients require. This is for debugging and possible visualization purposes. For example vehicle's line information is such information.

If there are no jams currently detected, the interface returns:

[]

Example request to get slow vehicles:

/jamdetector/JamService.svc/slowvehicles/hsl

The response is provided in JSON format. An example of a response is:


```
[
  {
    "CumulativeDelay": 53,
    "NextStop": null,
    "PreviousStop": null,
    "Stop": {
      "Code": "9212218",
      "Latitude": 60.2491,
      "Longitude": 25.49161,
      "Name": "Kitön silta"
    },
    "Vehicle": {
      "Delay": 73,
      "Id": "RHKL00115",
      "IsOnStop": false,
      "Latitude": 24.901162,
      "LineDirection": 2,
      "LineId": "1010",
      "Longitude": 60.198366,
      "NextStopIndex": 5,
      "Time": "20141024-145005",
      "Timestamp": "\/Date(1414151405652+0300)\/"
    }
  },
  {
    "CumulativeDelay": 18,
    "NextStop": null,
    "PreviousStop": null,
    "Stop": {
      "Code": "9212218",
      "Latitude": 60.2491,
      "Longitude": 25.49161,
      "Name": "Kitön silta"
    },
    "Vehicle": {
      "Delay": 194,
      "Id": "RHKL00057",
      "IsOnStop": false,
      "Latitude": 24.923647,
      "LineDirection": 1,
      "LineId": "1002",
      "Longitude": 60.172226,
      "NextStopIndex": 10,
      "Time": "20141024-145005",
      "Timestamp": "\/Date(1414151405665+0300)\/"
    }
  }
]
```


In this example there are two vehicles that would potentially participate the jam.

If there are no slow vehicles currently detected, the interface returns:

[]

3.8.5 Software Library API

External Connector for Public Transportation System is implemented according to the observer design pattern. The Traffic Jam Detection component registers itself as an observer to the vehicle information and the vehicle information is provided to the Traffic Jam Detection component as the new vehicle information is available.

Table 35: Public Transportation Information Reader API

Function name	Purpose	Request Parameters	Response
New Vehicle Info	Provide new information about a vehicle in the area.	Vehicle location information	-

3.8.6 Integration

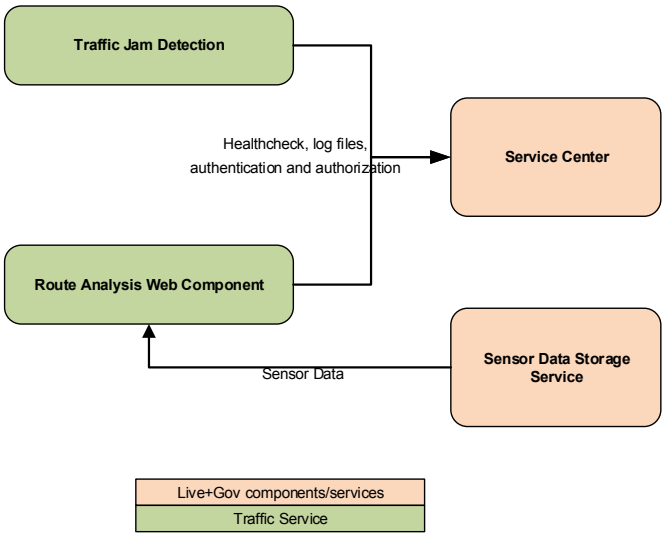

Figure 31: Traffic Service integration to other Live+Gov services.

Figure 31 shows the integration between the different components in the Live+Gov service architecture from the Traffic Service perspective. The components use the Service Center for user authentication and authorization. The component heart beat is sent to the Service Center periodically and log files are sent every midnight for further analysis to the Service Center.

Route Analysis Web Component does not have a database of its own; it uses the Sensor Data Storage to create reports for the end user about the passenger movements.

3.8.7 Phase 2 updates

Table 36 describes the updates performed in preparation of the 2nd trials rounds.

Table 36: Traffic Service Phase 2 updates

Traffic Jam Detector		
Component	**Change**	**Rationale**
New web interface	New web interface was created in order to visualize the component state. Specifically it was necessary find out how the component state changes and for example why jams were not detected in certain situations.	New interface was created for debugging, testing and visualization purposes.
Refactoring & performance improvements	Parts of the component were refactored in order to make the code more readable. At the same time e.g. some of the inner data structures were changed for better performance. Also the component parameterizing was fine-tuned during the testing for better user experience for the field trial.	The basic functionality of the component was implemented for the first trial and major changes were not needed. The update work can be regarded as general maintenance.
Web application for Route Analysis		
Component	**Change**	**Rationale**
Route Analysis Web Component	This new component was implemented.	The component was targeted to be available in the second field trial.

4 Testing Strategy

This chapter summarizes the Assembly-, Integration- and Verification (AIV)-plan that is followed by the Live+Gov consortium during the development of the Live+Gov toolkit and the development and customization of the three SaaS-based mobile governance solutions.

During the Live+Gov project two Assembly-, Integration- and Verification cycles have been performed in a highly complex distributed environment with multiple companies developing and deploying reusable and integrated SaaS-based toolkit services.

The described strategy proved to provide the required quality assurance for the development and deployment of three different mobile eGovernance solutions on different trial sites. Future toolkit developers and users can reuse this strategy when developing their own toolkit extensions or SaaS solutions.

A full description of the Technical verification and testing strategy can be found in Deliverable D4.4 [6].

4.1 Scope and goals

The scope of the AIV plan is to describe how the Live+Gov toolkit and the three SaaS-based mobile governance solutions are verified ("Are we building the product right?"). The goals of this AIV-plan are:

- To ensure that the developed Live+Gov toolkit and the three SaaS-based mobile governance solutions meet the requirements that guided its design and development;
- To ensure the toolkit and solutions work as expected after the tests described in the AIV-plan have been executed;
- To minimize the efforts in integrating the partners' different components (by eliminating errors in the components in an early stage);
- To align the different partners in the testing process to gain the necessary quality level in the developed software;
- To describe the tests such that after successful testing the software satisfies the needs of all stakeholders.

4.2 V-model

Verification of the Live+Gov toolkit is performed using the V-model displayed in Figure 32. The V-model is a simple variant of the traditional waterfall model of software development with an emphasis on the verification and validation of the software.

The V-model identifies different testing activities or phases in which the deliverables of the associated design phases are analysed or tested. The horizontal axis represents time and project completeness, while the vertical axis represents the level of abstraction.

Following the V-model, software development starts with describing the use cases and defining the requirements. The requirements lead to a high-level design or architecture. The different system components within this architecture are further elaborated in detailed technical designs. Based on the technical designs the developers start coding, the lowest point on the V-model.

From this point, the testing part of the software development begins, represented by the right part of the V-model. Testing starts with low-level testing of the components, followed by service testing, integration testing and system (or user acceptance) testing.

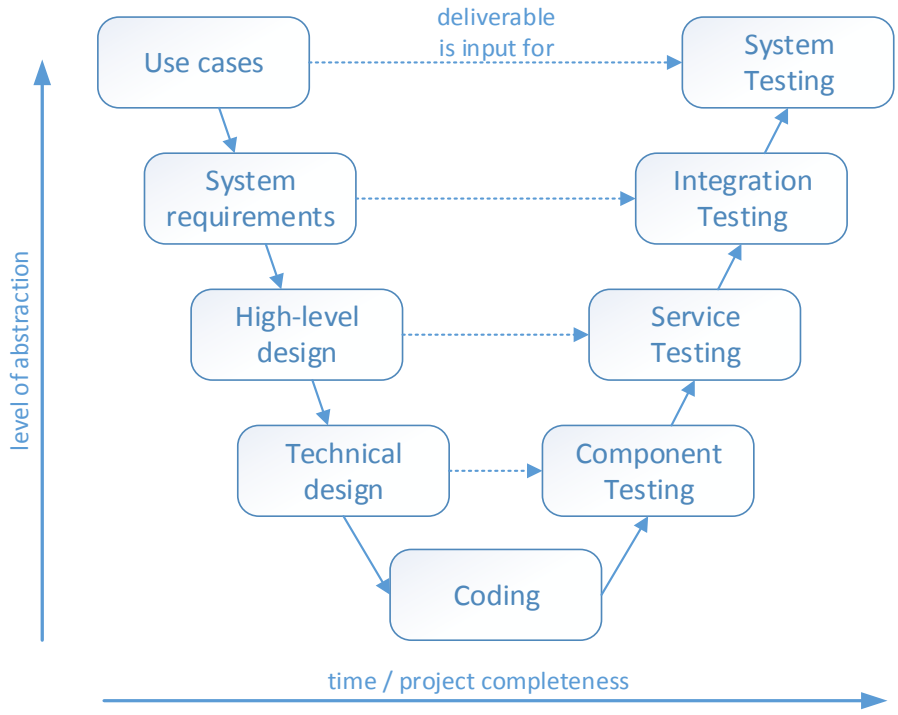

Figure 32: V-model [6].

The Live+Gov project's result is a prototype consisting of a toolkit and three SaaS-solutions. During the project two trials are performed in a real environment. These trials provided feedback on both the usage of the toolkit itself and on the mobile governance solutions created using the toolkit. This feedback is used to improve the toolkit architecture, toolkit components and mobile governance solutions for the second trial run. In preparation of each trial, the V-model as a whole is executed once, while some parts of the V-model are executed multiple times, e.g. when retesting specific components.

4.3 Test phases

Table 37 summarizes the testing-phases, responsible partners and deliverables in a single development cycle.

Table 37 : List of test phases [6].

Test phase	Subject	Responsible WP/Partner	Deliverables
Component Level Testing Phase	Individual Components	Developers (WP1, WP2, WP3, WP4)	Focus on the isolated software components, verifies these components compile and if the basic functions work as expected. All individual components are tested before they are embedded into a service.

Service Level Testing Phase	Individual Services	Service owners (WP1, WP2, WP3, WP4)	Testing the services within the Live+Gov toolkit and the interaction between the components within each service. Verifies if the basic functions work as excepted (functional testing) and that the service meets the performance and security requirements.
Integration Level Testing Phase	Integrated Live+Gov Toolkit	Service owners (WP1, WP2, WP3, WP4)	Testing the integration of the different services with the Service Center.
System Level Testing Phase	Mobile eGovernance Solutions	Use case partners (WP5)	Testing the solution against the functional requirements to validate if the system meets the key business requirements.

D4.4 [6].

Table 38 summarizes which types of testing are advised in a specific test phase, a description of these tests and guidelines for testing can be found in D4.4 [6].

Table 38: Overview of test types required in the test phases [6].

Test phase	Functional	Inter-operability	Compliance	Performance	Availability	Security
Component Level Testing Phase	v	-	-	v	-	v
Service Level Testing Phase	v	-	v	v	-	v
Integration Testing Phase	v	v	v	v	-	v
System Testing Phase	v	v	-	v	v	v

5 Test Results

This chapter gives a summary of the test results in preparation of the second trial rounds. Section 5.1 summarizes the results of the system test, section 5.2 summarizes the results of the service level & integration tests. As defined in *'D4.4 Technical verification and testing strategy'* [6] partners are not required to report the individual Component level test to the consortium. An example of the test template used during these tests can be found in Appendix A.

5.1 System testing

During the system tests the solution is tested against the functional requirements to validate if the system meets the key business requirements. Multiple test rounds have been performed by the use-case partners on the different mobile- and web applications for the Urban Planning, Urban Maintenance and Mobility SaaS solutions. Table 39 shows an overview of the performed system tests and their exit criteria.

Table 39: System test summary

TEST DESCRIPTION					EXIT CRITERIA						
Test no.	Service & integration test	Performed by	Planned date	Actual date	# test rounds	# Issues 1st test round	# Issues last test round	# High priority Issues left	Source code submitted	Reports	All exit criteria met
ST-1	Urban Planning SaaS Solution	BIZ	M30	M33	3	23	0	0	Yes	Appendix B ST-1	Yes
ST-2	Urban Maintenance SaaS Solution	YCT	M28	M28	3	11	0	0	Yes	Appendix B ST-2	Yes
ST-3	Mobility SaaS Solution	MTS	M30	M31	2	9	0	0	Yes	Appendix B ST-3	Yes

All test have successfully met their exit criteria. Maximum three test rounds were required before all high priority issue were solved and the use-case partners were confident to start the trial.

Although not all test were performed according to the original planning, this did not cause any problems in the trial planning. There was a two month windows available for performing the trials. The actual start date not only depended on passing the tests, but also on practical planning in combination with political obligation or delays. For example in the Urban Planning use-case the trial start was postphoned, in order to be able to combine the launch with local festivities in Gordexola, which offered a lot extra publicity options. The full test reports can be found in Appendix B. Note that these test focus on the technical verification. A full evaluation of the trial and software solutions will be given in *'D5.5 – End Results of trials and Live+Gov Methodology'.*

5.2 Service level & integration testing

The Service Level test focussed on the services within the Live+Gov toolkit and the interaction between the components within each service. One verifies if the required functions work as excepted (functional testing) and that the service meets the performance and security requirements. During the integration test it is verified that the service is integrated with the service center according to the integration guidelines and concepts [11]. The first test round showed that the service level & integration testing could be easily combined, since there are performed by the same partners. Table 40 shows an overview of the performed system tests and their exit criteria.

Table 40: Service level & integration test summary

TEST DESCRIPTION					EXIT CRITERIA						
Test no.	Service & integration test	Performed by	Planned date	Actual date	# test rounds	# Issues 1st test round	# Issues last test round	# High priority Issues left	Source code submitted	Reports	All exit criteria met
SLT-1	Sensor Data Capturing Service	UKob	M29	M29	2	4	0	0	Yes	Appendix C SLT-1	Yes
SLT-2	Reality Mining Service	UKob	M29	M29	3	10	4	0	Yes	Appendix C SLT-2	Yes
SLT-3	Augmented Reality Service	CERTH	M29	M29	3	13	0	0	Yes	Appendix C SLT-3	Yes
SLT-4	eGoverment Dialogue and Visualisation Service	YCT	M28	M28	3	26	1	0	Yes	Appendix C SLT-4	Yes
SLT-5	Issue Reporting Service	YCT	M29	M29	2	17	0	0	Yes	Appendix C SLT-5	Yes
SLT-6	SaaS Service Center	YCT	M29	M29	3	13	0	0	Yes	Appendix C SLT-6	Yes
SLT-7	Traffic Service	MTS	M29	M29	1	0	0	0	Yes	Appendix C SLT-7	Yes
SLT-1	Sensor Data Capturing Service	UKob	M29	M29	2	4	0	0	Yes	Appendix C SLT-1	Yes

All test have successfully passed their exit criteria. Maximum three test rounds were required to make sure all high and medium priority issues were solved. The tests were performed according to the original planning. The full test reports can be found in Appendix C. Note that these test focus on the technical verification. A full evaluation of the trial and software solutions will be given in *'D5.5 – End Results of trials and Live+Gov Methodology'.*

6 Summary and Conclusions

This deliverable presents the Live+Gov Toolkit which in one of the end-products of the Live+Gov project. This toolkit consist of several applications, software libraries and back-end services for creating SaaS based mobile governance solutions. Advanced services are included for building SaaS solution with Sensor Data Capturing, Reality Mining, Augmented Reality, Personalized Content Delivery, eGovernment Dialogue and Visualization and Issue Reporting functionality.

The architecture of the toolkit is based on the principles of Service-Oriented Architecture (SOA). In this way a generic architecture is offered, allowing generic services to be applied in multiple scenarios and solutions. This approach proved to be effective in a consortium with a multiple development partners, offering a clear distinction of responsibilities by the definition of public interfaces (API's) to combine the different services into SaaS solution for the use-case trials.

The central SaaS Service Center offers SaaS specific functionalities for account management, access control, billing and multi-tenancy. Toolkit services utilize these services instead of including their own fragmented mechanisms. Integration guidelines and a technical verification strategy [11] are described to make sure future extensions be created reusing the same development and test strategy and architecture.

This also gives a powerful tool to the service provider, enabling him to configure access profiles with different combinations of services in the mobile eGovernance solutions. For exploitation this means that different functional packages (Service Levels) can be offered depending on the actual needs of the customers and the commercial marketing strategy. This will be further elaborated in the final exploitation plan D6.5.

Three example solutions have been developed, tested, deployed and trialled in real life environments: A Mobility solution in Helsinki (Finland), an Urban Maintenance solution in Utrecht (The Netherlands) and an Urban Planning solution in Gordexola (Spain). These solutions show that the toolkit is generic, cross platform and can be applied in different scenarios and environment. Also the AIV test strategy proved to provide the required quality assurance for the development and deployment of three different mobile eGovernance solutions on different trial sites. These examples can be customized and deployed for other municipalities or used as an example by developers when creating their own solution.

The final evaluation results of the trials, experiences and best practices will be described in detail in '*D5.5 - End results of trials and Live+Gov Methodology'.*

7 References

[1] C. Scheafer, H. Hartmann, “Sensor Data Application”, Live+Gov Deliverable D1.1, July 2013.

[2] L. Kovats, et al, “Conceptual documentation on issues, organisation and stakeholder assessment”, Live+Gov Deliverable D2.1, July 2013.

[3] M. Thimm, H. Hartmann, L. Nittylä, M. de Arana Agiretxe, M.J. Terpstra, P.M. Minnigh, L. Kovats, “Report on strategies of mobile sensing in eParticipation” Live+Gov Deliverable D2.2, July 2013.

[4] D. Ververidis, E. Chatzilari, G. Liaros, S. Nikolopoulos, Y. Kompatsiaris, P.A. Minnigh, and F. Thiele, “Platform and prototype application for augmented reality,” Live+Gov Deliverable D3.1, July 2013.

[5] F.Thiele et al, “Report on Live+Gov toolkit requirements and architecture”, Live+Gov Deliverable D4.1, July 2013.

[6] F. Kramer, F, Thiele et al, “D4.4 –Technical verification and testing strategy”, Live+Gov Deliverable D4.4, November 2013.

[7] P.A. Minnigh, et al. “Detailed Use Case description”, Live+Gov Deliverable D5.1, July 2013.

[8] J.A.M. Zubiaur, J. Escion, J. Cassempere, M. de Arana Agiretxe, S. Nikolopoulos, H, Keuchel, L. Nittylä, J.M. Steensma, P.A. Minnigh, S. Sizov, M. Thimm, “Initial Exploitation Plan”, Live+Gov Deliverable 6.3, July 2013.

[9] “StUF”, Kwaliteits Instituut Nederlandse Gemeenten (KING), GEMMA Community, visited 24th Jun 2013. https://new.kinggemeenten.nl/gemma/stuf

[10] “Metaio – The New Metaio SDK”, Metaio Incorporated, visited Oct 1st 2013. http://www.metaio.com/sdk

[11] F. Thiele, et al. “Integration concepts and guidelines”, Live+Gov Deliverable D4.2, November 2013.

[12] P. van Tol, A.J. Krommendijk, “Service Center API Description”, Live+Gov, October 2013.

[13] A.J. Krommendijk, “Service Center Installation Manual”, Live+Gov, November 2013.

[14] Pekka Kaarela, “Traffic Jam Detector API Description”, Live+Gov, November 2013.

[15] “Augmented Reality Service Documentation”, http://augreal.mklab.iti.gr/doc/, visited Nov 27th 2013

[16] F. Kramer, “eGovernment Dialogue and Visualization API”, Live+Gov, May 2012.

[17] Minnigh, P.A., et al., (2014) “Prototype / demonstrator for second trials”, Live+Gov Deliverable D5.4, July 2014.

[18] Live+Gov. “End results of trials and Live+Gov Methodology”, Live+Gov Deliverable D5.5, forthcoming.

[19] L. Kovats, et al., “Visualization of data injection from mobile sensing”, Live+Gov Deliverable D2.3, January 2014.

[20] L. Kovats, H. Keuchel, J. Hoffmann, “Applied Policy Modelling Training Package”, Live+Gov Deliverable D2.4, August 2014

[21] Live+Gov. “eGovernance augmented reality application”, Live+Gov Deliverable D3.2, forthcoming.

8 Appendices

The following appendices are enclosed in a separate document.

Appendix	Description	Dissemination Level
Appendix A	Test scenario's & results template	Public
Appendix B	System test plan and results	Confidential
Appendix C	Service level & integration test plans and results	Confidential