WP3 – Architecture, Specification and Integration

D3.3.1: Technical Specification

Deliverable Lead: ASC

Contributing Partners: NTUA, TALK, TIE, TPVI, UA, UOR

Delivery Date: 2014-11

Dissemination Level: Public

Final

This deliverable represents the technical specification of the SAM platform. It defines the SAM platform from a technical point of view and shows how the components cooperate with each other in order to provide the desired functionality. It also reaches a consensus on technical decisions regarding the choice of suitable technologies/ tools for the implementation of the SAM components based on the project vision.

Document Status	
Deliverable Lead	ASC
Internal Reviewer 1	Fredrik Kronlid, v0.4, 27.10.2014
Internal Reviewer 2	Marco Tiemann, v0.6, 07.11.2014
Type	Deliverable
Work Package	WP3 – Architecture, Specification and Integration
ID	D3.3.1: Technical Specification
Due Date	10.2014
Delivery Date	11.2014
Status	For Approval

Document History	
Versions	V0.1: ASC First Draft produced by Editor V0.2: ASC Section 3.8 added V0.21: ASC Using new document template V0.22: TIE, TALK, TPVI, UA, UOR, NTUA, ASC Added Section 3 contributions V0.23: ASC Write protected version for discussion and alignment V0.3: ASC Added feedback (1st iteration) V0.4: ASC 1st review version V0.5: TALK 1st review amendment V0.51: ASC Adaptation changes based on 1st review comments V0.52: TIE, TPVI, UA, UOR, NTUA, ASC Added contributions based on 1st internal review V0.6: ASC 2nd review version V0.61: ASC Adaptation changes based on 2nd review comments V0.62: TIE, TPVI, UA, TALK, NTUA, ASC Added contributions based on 1st internal review V0.7: ASC Final version V1.0: Submitted Version
Contributions	ASC: Norman Wessel – Document creation and contributions to all sections Danny Pape – Section 3 contributions TALK: Apostolos Apostolidis – Section 3 contributions TIE: Fran Rodriguez – Section 3 contributions Juan Vte. Vidagany – Section 3 contributions TPVI: Lukasz Kreft – Section 3 contributions Pieter van Loocke – Section 3 contributions UA: Yoan Gutiérrez – Section 3 contributions Isabel Moreno – Section 3 contributions UOR: Marco Tiemann – Section 3 contributions NTUA: Andreas Menychtas – Section 3 contributions Alexandros Psichas – Section 3 contributions

Disclaimer

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

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

Project Partners

TIE Nederland B.V., The Netherlands

Ascora GmbH, Germany

Talkamatic AB, Sweden

TP Vision Belgium NV, Belgium

Institute of Communication and Computer Systems, National Technical University of Athens, Greece

The University of Reading, UK

Universidad de Alicante, Spain

Deutsche Welle, Germany

Bibliographic Data Services Limited, UK

Executive Summary

The purpose of this deliverable is to present the technical specification of the SAM platform. This document takes the deliverable D3.2.2 Function Specification as a basis to highlight and to further elaborate the design of the SAM platform into a concretely implementable solution. The methods used to achieve this are the following related activities, performed for each of the SAM components:

- The major design decisions were specified in order to define the context for the next steps.
- Technology comparison and selection was performed in order to select the most suitable technologies for the implementation of each component. The technology selection process was based on predefined criteria. Based on this process, an analysis of the relevant technologies was made, supported by a parameter evaluation, and the most suitable solution approaches were elaborated. Finally, any missing elements and implementation needs were identified.
- Each of the components was specified with respect to its static structure (subcomponent design) and dynamic behaviour (subcomponent collaborations design).
- The necessary communication protocols, interfaces and formats between the components were specified.

In the deliverable, the component operations, protocols and data exchange formats are specified in order to ensure effective execution, communication and interoperability among the 19 SAM components. The detailed descriptions provided by this technical specification will underpin the further developments of the project. Therefore, this very detailed deliverable serves as the basis and guide for the entire software development in the SAM project.

Table of Contents

1 Introduction

SAM – Dynamic Social and Media Content Syndication for 2nd Screen – is a project funded by the Seventh Framework Programme of the European Commission under Grant Agreement No. 611312. It provides a content delivery platform for syndicated data to be consumed in a contextualised social way through 2nd Screen devices.

1.1 SAM Project Overview

Today's generation of Internet-connected devices has changed the way users are interacting with media, exchanging their role from passive and unidirectional to proactive and interactive. Under this new role, users are able to comment or rate a TV show and search for related information regarding characters, facts or personalities. They do this both with friends and wider social communities through the so-called "2nd Screen".

Another coupled phenomenon is "Content Syndication" which is a field of marketing where digital content is created once and delivered too many different marketing channels (devices, Social Media channels, websites and stakeholders) together and so allowing efficient content control, delivery, and feedback.

However, the 2nd Screen phenomenon has grown in an unordered way. Tools are provided by the media providers companies (e.g. as mobile or tablet apps), which limits outreach and as a result, users are not stimulated and fed with relevant contextual syndicated information. European enterprises wishing to provide services have limited potential to receive feedback, which restricts the business intelligence that can be extracted and applied therefore to profit from and enrich this market.

SAM will change this disorder by developing an advanced Social Media delivery platform based on 2nd Screen and Content Syndication within a Social Media context. This is achieved by providing open and standardised ways of characterizing, discovering and syndicating media assets interactively. Users will be able to consume and prosume digital assets from different syndicated sources and different synchronised devices (e.g. connected TVs), thus creating richer experiences around the original media assets.

SAM's innovation is that instead of users reaching for the data; it is the data, which reaches the user through the syndication approach and their 2nd Screen. This is based on the creation of dynamic social communities related to the user and digital asset context (e.g. profiles, preferences and devices connected). These are dynamic hangouts where people share interests, socialise and build virtual communities. SAM will enable syndication of comments, ratings, facts, recommendations and new information that will enrich and energise the community as well as enhance personalised knowledge and satisfaction.

1.2 Deliverable Purpose, Scope and Context

The goal of this deliverable is to present the technical specification of the SAM platform. It defines and describes the technical view of the SAM architecture that will eventually represent the final SAM software.

As shown in Figure 1, this deliverable is closely related to previous deliverables of the project, namely D2.3 User Stories and Requirements Analysis and D3.2.1 Global Architecture Definition and directly related to D3.2.2 Functional Specification and work packages WP4-7 that involve software development).

This deliverable specifies a design that supports the functionality defined in D3.2.1, consequently satisfying SAM user requirements as elicited in D2.3. The design is also constrained and put into technical context by D3.2.1. It describes how the SAM components are structured and operate technically, in order to provide the necessary functionality to its users.

The technical specification of the components contains the necessary information for the technical design of the platform. Therefore, this deliverable serves as a basis for WP4, WP5, WP6 and WP7, where the software components that implement these functionalities will be developed.

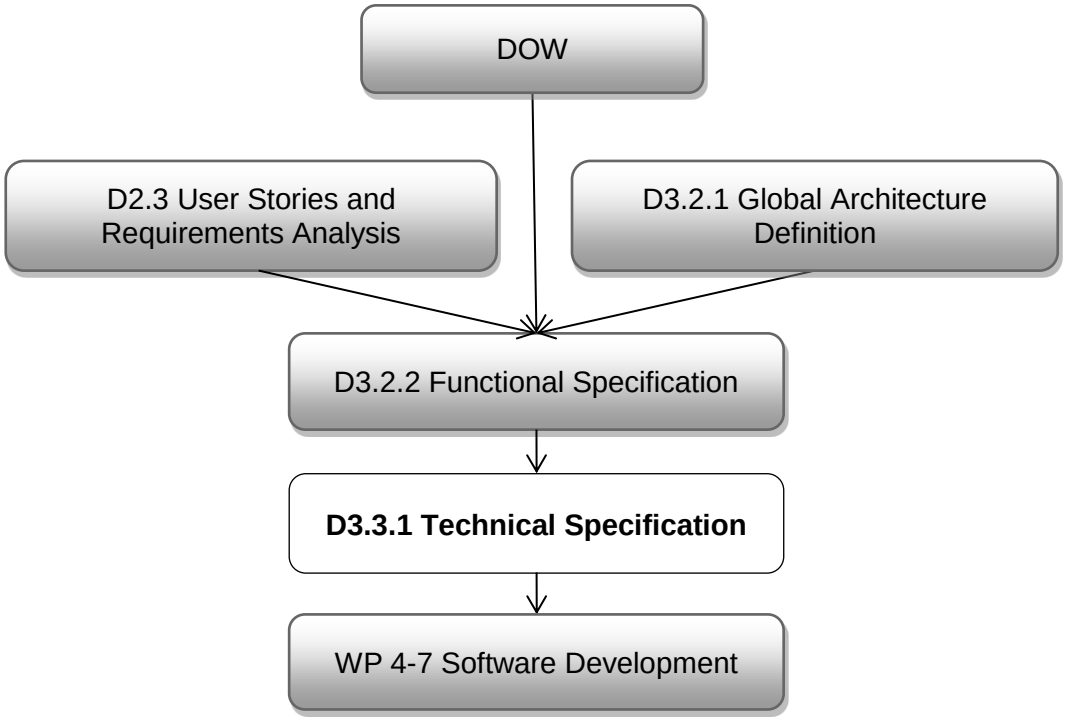

Figure 1: Context of the Deliverable

1.3 Document Status and Target Audience

This document is listed in the DOW as public since the technical specification may be useful for external parties as a reference or as a base for understanding the technical elements of the project. It may also be used as a starting point when integrating external systems with SAM, e.g. for integrating content management systems or content provider applications.

1.4 Abbreviations and Glossary

A definition of common terms and roles related to the realisation of SAM as well as a list of abbreviations is available in the supplementary and separate document “Supplement: Abbreviations and Glossary”.

Further information can be found at
http://wiki.socialisingaroundmedia.com/index.php/Glossary

1.5 Document Structure

This deliverable is structured into the following sections:

- **Section 1 (Introduction):** An introduction to this deliverable that includes a general overview of the project and outlines the purpose, scope, context, status, and target audience
- **Section 2 (General Description of Technical Specification):** Provides an overview of the system and presents a functional analysis that includes the overall architecture design, functional recapitulation and translation from functional to technical specification. The section also contains a general description of the technical specification and explains the generic and the specific parameters used to select the right technology for each of the SAM components.
- **Section 3 (Technical Specification):** Specifies the concrete technical details for each SAM component. This includes the major design decisions, applicable technology comparison and selection, technical component specification and specification of interfaces, protocols and formats.
- **Section 4 (Conclusions):** Concludes the document and highlights the final outcomes from this deliverable
- **References:** Provides details of the references mentioned in the document

1.6 External Annexes and Supporting Documents

There are no external document references in this deliverable.

2 General Description of Technical Specification

This deliverable is closely aligned with previous deliverables of the project, particularly specifying the user requirements, system functionalities and the global architecture. As specified in deliverable D3.2.1 Global Architecture Definition and shown in Figure 2, the functionalities of SAM will be realised by 19 components, organised in four architectural layers.

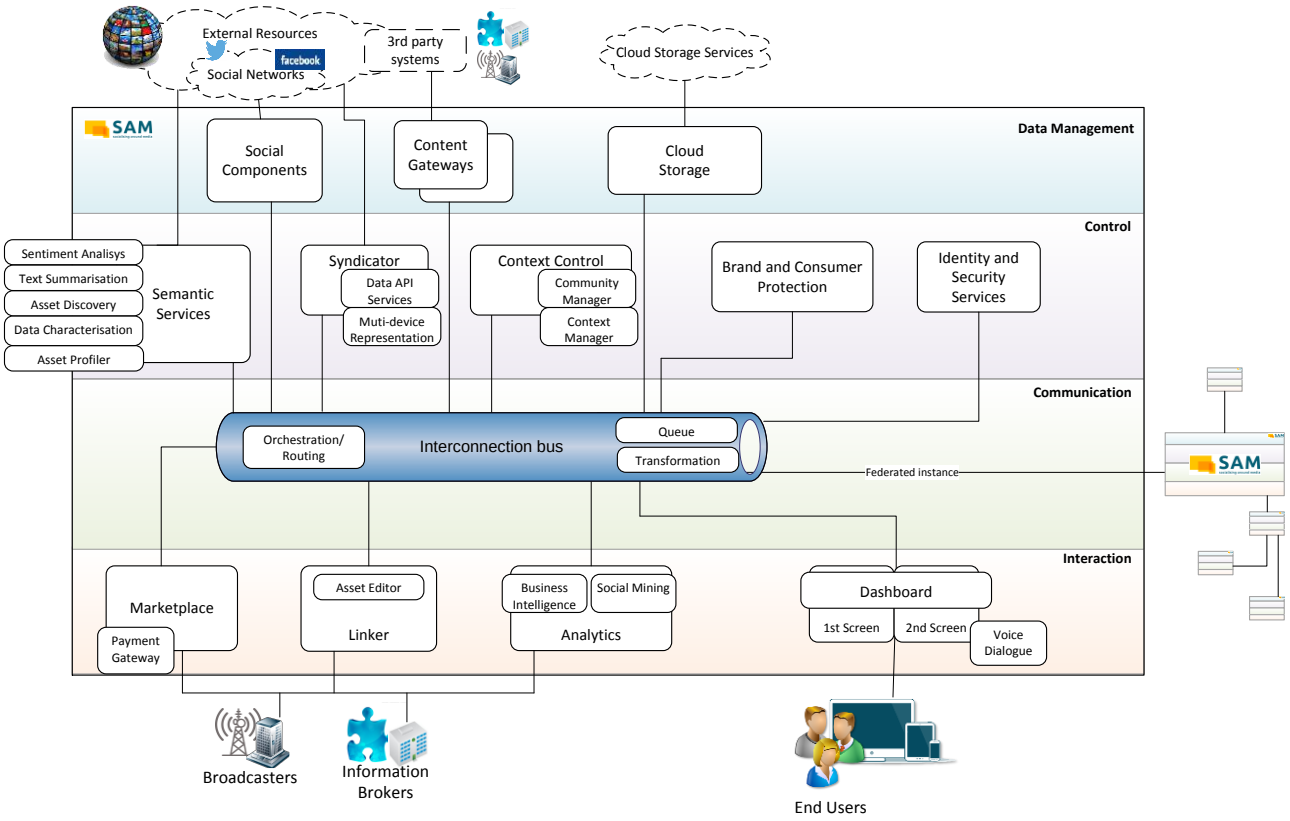

Figure 2: SAM High-Level Architecture

Based on this architecture, the deliverable D3.2.2 Functional Specification presents a system-level analysis of the SAM platform from a functional point of view. An overall functional characterisation of each component explains the main ideas on what the component will do and how this goal will be achieved. Also, for each component, the corresponding use cases are identified and described. The interactions between the components are analysed in detail and the services that each component will provide and consume are specified.

The Technical Specification enhances the focus on functional requirements with technical requirements to provide a technical understanding of the SAM vision with regard to each individual component. It provides a pre-selection of eligible technologies and compares their suitability with respect to users' requirements with the help of a parameter evaluation, in which general and component-specific parameters will be used to evaluate and compare the technologies. Additionally the component's architecture overview will be enriched with the selected technology to provide a quick overview. In total, this approach helps developers understand and define different interfaces and standards to ensure interoperability among various components of the SAM platform and also to external systems.

3 Technical Specification

The following subsections contain the components the SAM platform consists of. Each component is elaborated in a common form that addresses the following topics:

- **Major Design Decisions:** The goal of this subsection is to briefly describe the functionality and scope for the particular component as defined within the D3.2.2 Functional Specification. Also, this subsection discusses the major design decisions which provide the foundation for this component. These major design decisions are technology-independent, i.e., not determined by particular attributes of the technology to be selected. Quite to the contrary, these design decisions considerably influence the later technology comparison and selection.
- **Technology Comparison and Selection:** This subsection contains a short description of possible technologies that are suitable for the tasks of the component and the previously mentioned design decisions. Additionally, the technologies are compared using the generic parameters specified in the deliverable D3.2.2 in Section 4.1 and the specific parameters specified for each component. Finally, the most suitable technology are selected.
- **Technical Component Specification:** This subsection contains additional information about the structure of the component. Internal sequence diagrams can be found in the respective section in the deliverable D3.2.2. However, if a component provides more detailed diagrams, these can be found in this section. The goal is to provide an overview of how and where the selected technology will be deployed.
- **Specification of Interfaces, Protocols and Formats:** In this subsection, the external interfaces, protocols and different data formats provided by this component are listed. The defined interfaces, protocols and formats act as guidelines for the consortium on how to use the specific component.
- **Summary:** The summary briefly recapitulates the most important aspects of the technical specification and planned implementation for the software component considered in the subsection.

3.1 Common Graphical User Interface

In order to provide access to the features of the SAM platform for the different stakeholders such as Content Broadcasters, Content Brokers or Software Providers, a graphical user interface (GUI) has to be provided. In this section, eligible candidates for the GUI technology will be compared and at the end one technology will be selected. Due to this process, this section has a different structure than the following sections.

As seen in D3.3.2 Component Mock-ups, the GUI will be separated in three different sections:

- **Marketplace**, which provides the necessary interface to search/enrich/buy assets and services that a before mentioned stakeholder may need (see D3.3.2 Component Mock-ups, Section 3.4).
- **Administration**, which provides the necessary interface to manage the internal components and connections to external sources (see D3.3.2 Component Mock-ups, Section 3.2).
- **Dashboard**, which provides the user interfaces for End Users (see D3.3.2 Component Mock-ups, Section 3.3).

Since the Dashboard GUI technology will focus more on mobile devices than the Marketplace and Administration GUI, separate technology comparisons have been conducted and can be found in Section 3.14.2.2. The Marketplace and Administration GUI technologies can use a shared technology comparison because of the similar environment and the need to provide a consistent user experience. An exception to this approach are components with already existing implementations: Already existing user interfaces which are compatible to the chosen technology do not need to be implemented again but, if feasible, will be adapted to the common style to provide a consistent user experience.

3.1.1 Technology Comparison and Selection

This subsection outlines technology selection criteria and compares existing technologies – potential candidates for the realisation of the Graphical User Interfaces for the Marketplace and the Administration.

3.1.1.1 Possible Technologies and Comparison

In order to implement the graphical user interfaces several technologies have been considered as base technology candidates. The following section compare the most promising candidates:

- **AngularJS**[1] is an open-source web application framework, maintained by Google, which assists with creating single-page applications, which consist of one HTML page with CSS and JavaScript on the client side. Its goal is to simplify both development and testing of web applications by providing client-side model-view-controller (MVC) capability as well as providing structure for the entire development process, from designing an app through to testing it.
- **Ember**[2] is an open-source client-side JavaScript web application framework based on the MVC software architectural pattern. It allows developers to create scalable single-page applications by incorporating common idioms and best practices into a framework that provides a rich object model, declarative two-way data binding, computed properties, automatically-updating templates and a router for managing application state.
- **Backbone.js**[3] is a MVC-based framework implemented in JavaScript. As all MVC-based frameworks, its structure contains models containing the data, views which represent the graphical user interface and a controller, which contains the business logic. The API can be used through a RESTful JSON interface. Additionally, there is a significant number of extensions available for Backbone.
- **Ruby on Rails**[4] is an open-source web application framework written in Ruby. Rails is a full-stack framework that emphasises the use of well-known software engineering patterns and paradigms, including convention over configuration (CoC), don't repeat yourself (DRY), the active record pattern, and model–view–controller (MVC).

The table below (see Figure 3) provides a comparison containing the technologies introduced previously. This and later comparison evaluate the chosen technologies using different parameters. A technology candidate can receive parameter values between “---“ (bad) and “+++” (good) or “YES” or “NO” values.

[1] https://angularjs.org/
[2] http://emberjs.com/
[3] http://backbonejs.org/
[4] http://rubyonrails.org/

Parameter	Importance	AngularJS	Ember	Backbone.js	Ruby On Rails
Generic Parameters					
Maturity & Stability	+++	++	++	++	+++
Regularly Updated	+	++	+	+	+++
Technical Up-to-Datedness / Appeal	++	+++	++	++	++
Open Source	+	YES	YES	YES	YES
Non-Infecting	++	YES	YES	YES	YES
Code Quality	+	+++	++	++	++
Extensibility	+++	++	++	++	++
Community	+	+++	+	+/-	++
Performance / Scalability	++	+++	++	++	++
Reuse of existing developments	++	YES	NO	NO	NO
EU project origin	--	NO	NO	NO	NO
Platform (Portability)	+++	++	++	++	++
Open Standards Compliance	+	YES	YES	YES	YES
Interoperability (easy integration for all platforms)	+++	+++	+++	+++	+++
HTML / JS-based (Client-side technology)	+++	+++	+++	+++	+++
User-friendly Interface	+++	+++	+++	+++	+++
Testable	+++	++	+	+	+
Design Pattern focused	++	+++	++	++	++

Figure 3: Parameter Evaluation of Technologies for Graphical User Interfaces

3.1.1.2 Technology Selection

As seen in Figure 3, the solution that can be set apart as the most fitting is AngularJS. It can be said that both AngularJS and Backbone fit the project quite well, but the following reasons justify this choice:

- AngularJS is much easier to develop with since it makes more assumptions and therefore supports the developer more compared to Backbone
- AngularJS focuses on testing the production code which can speed up iterating between developing and testing
- AngularJS has been chosen as a suitable technology candidate for graphical user interfaces by multiple partners within the SAM project so a significant amount of knowledge will be shared

3.2 Interconnection Bus

The Interconnection Bus plays a central role in the SAM architecture (see Figure 4) allowing reliable transmission of information between the different SAM components and the federated SAM instances. In order to make the best technology decision, the main functionalities described in the deliverable D3.2.2 Functional Specification is taken into account. The most important goals are the following:

- The Interconnection Bus component will directly interact with all components of the SAM Platform. It will provide advanced queuing mechanisms (allowing event-driven

approaches), service orchestration, message routing, transformations and diverse kind of pluggable Communication Adaptors mechanisms in order to communicate with auxiliary external systems.

- The Interconnection Bus interacts with the Identity and Security Services component in order to provide user authentication features and to establish secure data exchange between SAM federated instances.
- The configuration of the Interconnection Bus will be performed by the system administrator (System Admin) through the Bus Management Console

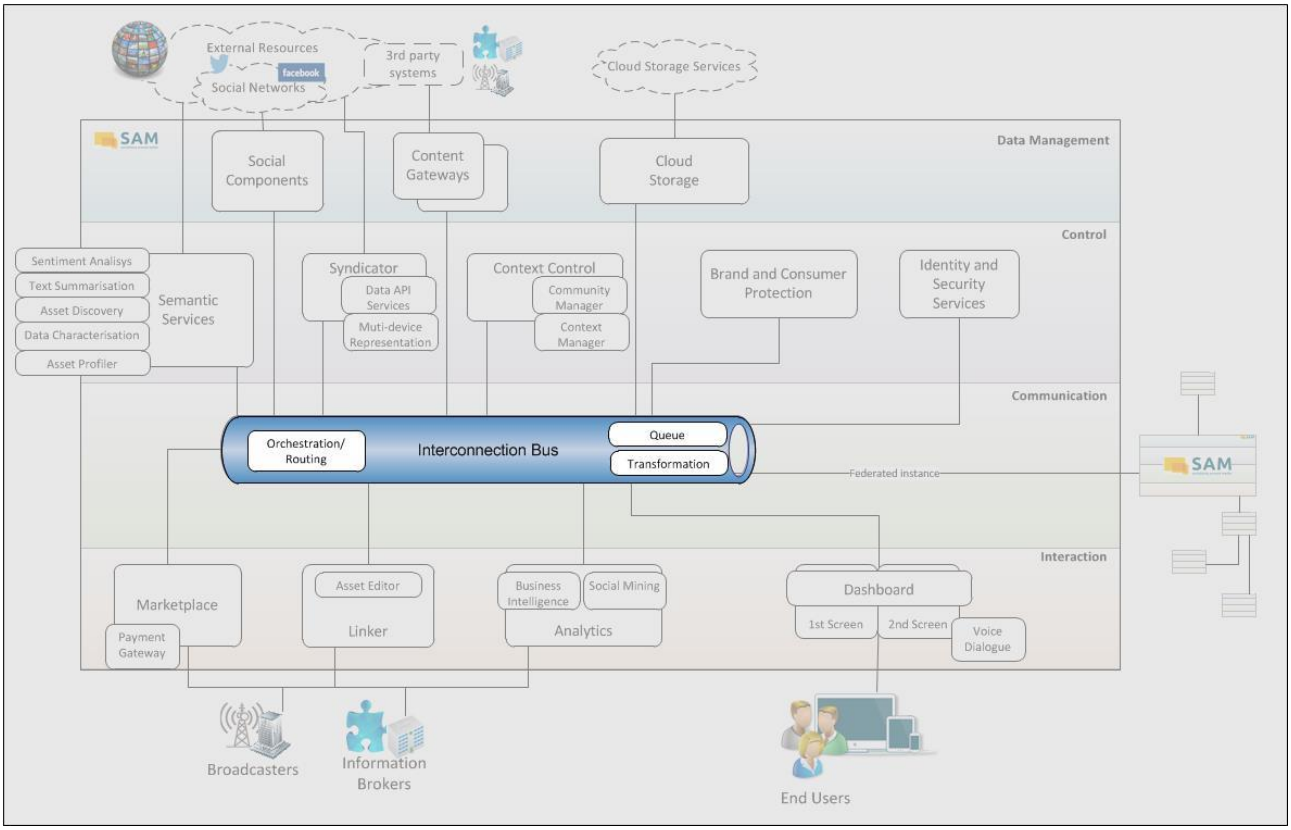

Figure 4: Architecture Overview – Interconnection Bus

3.2.1 Major Design Decisions

The Interconnection Bus is the component that provides the communication infrastructure between all the SAM components in the SAM platform. The most important decision is that TIE Kinetix SmartBridge ESB (TSB) will be used for this component. TSB has been nominated in the DOW as the technology to be used in order to implement the Interconnection Bus functionalities (See DOW Section B1.3.3.4 WP4 Information Management & Interoperability System). However, the major decisions to take into account in the communication module are explained in the following points:

- Each component in the SAM Platform will be developed as an independent service using different technologies. Therefore it is necessary to implement a communication system which allows the interoperability between these heterogeneous services. For this reason, the Interconnection Bus component will be based on an ESB (Enterprise Service Bus) system which supports SOA (Service-Oriented Architecture) and Event-driven SOA (SOA 2.0 or SOA+) approaches. SOA 2.0 combines event-driven architecture with traditional SOA allowing the use of events in the communication between SAM components.

- The selected technology should allow basic features such as routing and forwarding messages, sending the same message to several recipients at the same time (Multi-recipient messaging), controlling the different information queues, where the messages are temporarily stored (Messages/topic queues), and finally synchronous and asynchronous communication.
- Audit Logs about the component communication activity should be available for tracing the messages. Besides, using these logs, it will be possible to get statistics about this activity, providing information for the Business Intelligence component.
- A graphical user interface should be available which allows easy configuration of the different services and processes. It should also allow the monitoring of current activities and management of component logs.
- Orchestration features should be provided in order to define complex operations such as interactions between different technologies and data format transformation. A specialised Workflow Designer and its execution engine will constitute the core mechanism of the Interconnection Bus.
- The Interconnection Bus will offer a pluggable mechanism in order to extend the communication protocols and formats supported.

3.2.2 Technology Comparison and Selection

The objective of this subsection is to demonstrate that TSB provides the necessary features in order to implement the component. For this reason, a comparison with other technologies has been carried out. Within the subsection, the areas that need to be implemented or improved to fully meet SAM specific requirements are also identified and presented. The specific comparison criteria to be used are specified in the deliverable D3.2.2 Functional Specification, Section 4.2.5 and will be used in the next sections.

3.2.2.1 Possible Technologies and Comparison

The following technologies have been selected to be compared with TSB:

- **JBoss Fuse ESB**[5] is an open-source ESB based on Apache Camel, Apache CXF and Apache ActiveMQ. Using these technologies, JBoss Fuse provides reliable messaging, routing, enterprise integration patters, RESTful web services, Open Service Gateway initiative (OGSI) services and a robust SOA infrastructure based on Java Business Integration (JBI) allowing the implementation of a federated architecture. JBoss Fuse ESB has multiple options for connecting to external applications with connectors for JDBC, HTTPS, SalesForce, Twitter, etc. It also provides dynamic configuration and management allowing deploying or updating of services across nodes while the ESB is running. Furthermore, it includes an easy integration with heterogeneous environments hosted anywhere. Finally, the development environment is based on Eclipse and provides support to users through of a community and forums.
- **WSO2 ESB**[6] is an open-source ESB based on SOA pattern design and released under the Apache License v2.0. WSO2 ESB supports the most important transport protocols (HTTPS, POP/IMAP, JMS, etc.) using different formats (JSON, XML, SOAP, WS, EDI, etc.) and adapters to cloud services (Salesforce, PayPal, LinkedIn, Twitter, JIRA, etc.). However, the number of connectors is limited and smaller than other technologies. WSO2 ESB provides key features like routing, messaging and logging and auditing. Another important feature is that this technology enforces and manages security

[5] https://www.jboss.org/products/fuse/overview/
[6] http://wso2.com/products/enterprise-service-bus/

centrally, including authentication and authorisation providing security protocols such as WS-Security, LDAP, Kerberos, OpenID, SAML, XACML. WSO2 supports a high level of concurrency (up to 2,500 concurrent connections on standard hardware, without losing a single message) and horizontal scaling via clustering. Finally, this technology provides a web interface in order to manage and monitor the activity and retrieving performance statistics.

- **TIE Kinetix SmartBridge (TSB)**[7] is a complete ESB integration platform focused on supporting SOA architecture and based on the latest .NET technology. TSB provides reliable and multi-recipient messaging, data transformation, service orchestrations, and exchanges messages fully adjustable to the required communication methods and underlying message queues which temporarily store the messages. By default, it supports the most important communication protocols such as HTTP/HTTPS, SMTP/POP/IMAP, XMPP, etc. but TSB offers the possibility to extend these protocols through add-in implementations. All TSB activity is registered in internal logs, providing better control and reliability in the system. In order to manage services and monitor activity and statistics, TSB provides a user-friendly web user interface built on .NET technologies (ASP.NET MVC 3[8] and MonoRail[9]). TSB is therefore a scalable solution that can be optionally extended with more advanced functionalities.
- **Mule ESB**[10] is an open-source ESB platform. It offers a secure, scalable and reliable platform that is lightweight and extensible with an intuitive installation and Eclipse-based tooling. The core capabilities that Mule ESB offers are message routing, data transformation and event handling. Important advantages are the graphical editors for an efficient implementation of integration scenarios and the available connectors for B2B products such as SAP or Salesforce. Negative aspects of Mule ESB are the small community, a restrictive licensing model and limited availability of the source code.

Parameter	Importance	JBOSS	WSO2	TSB	Mule ESB
Generic Parameters					
Maturity & Stability	+++	+	+	++	+
Regularly Updated	+++	+	+	++	+
Technical Up-to-Datedness / Appeal	+++	+	+	++	+
Open Source	+/-	YES	YES	NO	YES
Non-Infecting	+	YES	YES	NO	YES
Code Quality	++	++	++	+++	++
Extensibility	+++	++	++	+++	++
Community	+	++	++	---	+
Performance / Scalability	+++	+	+	++	+
Reuse of existing developments	+++	NO	NO	YES	NO
EU project origin	+	NO	NO	NO	NO
Platform (Portability)	+	++	++	+	++
Open Standards Compliance	+	++	++	++	++
Interoperability (easy integration for all platforms)	+++	++	++	+++	++
Specific Parameters					
Reliable Messaging /	+	+	+	++	+

[7] http://tiekinetix.com/nl/tie-kinetix-smartbridge
[8] http://www.asp.net/mvc/mvc3
[9] http://www.castleproject.org/projects/monorail/
[10] https://www.mulesoft.com/platform/soa/mule-esb-open-source-esb

Receipt Acknowledgement					
Secure communication protocol	+++	++	++	++	+
Message queues / topic queues	+++	++	++	+++	++
Multi-recipient-messaging	+++	+++	+++	+++	+++
Provide configuration UI for component	+++	NO	YES	YES	YES
Register Logs	+++	+	++	+++	+

Figure 5: Parameter Evaluation of Technologies for ESB (Enterprice Service Bus)

3.2.2.2 Technology Selection

The comparison carried out in the previous section shows how TIE Kinetix SmartBridge ESB (TSB) meets the necessary requirements in order to implement the Interconnection Bus component. The conclusions about the comparison are the following:

- The table above (Figure 5) shows how all technologies have quite similar values in the general and specific parameters. However, TSB's advantage is the maturity and stability of the system; TSB has been offering services for five years, providing support to more than 2000 customers across five countries, thus proving high reliability in production environments
- TSB, in contrast to the open-source technologies, is updated monthly providing extra reliability and improving the behaviour and security in TSB and in consequence in the SAM Platform. T latest version of TSB, version 3.3, has been updated with the latest key .NET features such as asynchronous file operations.
- Finally, TSB is more extensible and scalable than the other technologies since the integration environment is already composed of several independent servers that act together as one system. Therefore, the federated infrastructure needed for the SAM Platform, will be easily implemented due to the fact that TSB is prepared to be extended in order to implement the federated SAM instances.

Additionally, TSB is nominated in the DOW as the technology to be used in order to implement the Interconnection Bus functionalities (See DOW Section B1.3.3.4 WP4 Information Management & Interoperability System). TIE's SmartBridge ESB product will be adapted to support the internal communication features and extended to implement the federation patterns needed.

In conclusion, TSB is selected to carry out the communication module since TSB meets the criteria to be used as technology and it is nominated in the DOW as the technology to be used.

3.2.2.3 User Interface Technology

As described in Section 3.2.2.2, the TIE Kinetix SmartBridge technology has been selected. TSB already includes a graphical user interface which will be extended in order to adapt to the SAM needs. Therefore, the technology comparison in Section 3.1 does not need to be applied since the SAM functionalities will be extended using the current TSB technology.

3.2.3 Technical Component Specification

3.2.3.1 Structure

Figure 6 provides an overview of the Interconnection Bus, which can be found also in D3.2.1 Global Architecture Definition.

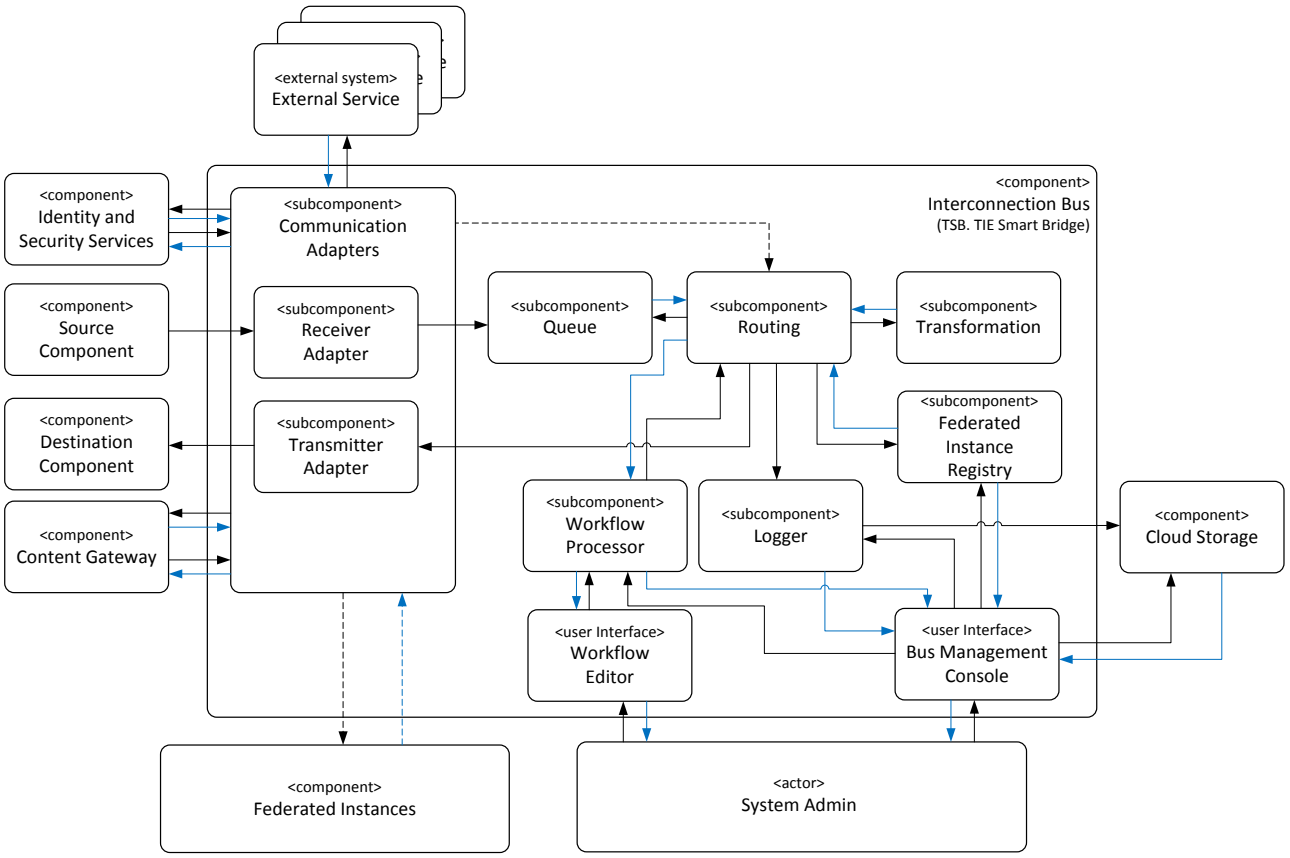

Figure 6: Overview with Selected Technologies – Interconnection Bus

3.2.4 Specification of Interfaces, Protocols and Formats

The Interconnection Bus will provide RESTful interfaces (described in Section 3.2.4.1) as well as protocols (described in Section 3.2.4.2).

3.2.4.1 RESTful Interfaces

In order to describe the RESTful interfaces, each provided service is described in a separate table. Each table will show a description, a URL, request parameters with an example and the response type including an example.

3.2.4.1.1 Method “Get Services Info“

The RESTful interface “GetServicesInfo” (Figure 7), will be used to get a list with related info about the registered services component in the SAM platform. This interface will give back the component name (e.g. Cloud Storage) and the info service necessary to send a message (service name and parameters) to this services (see Section 3.2.4.1.3).

<table>
<tr><th colspan="3">Get Services Info</th></tr>
<tr><td>Description</td><td colspan="2">Interface to get a list with related info about services registered in the SAM platform.</td></tr>
<tr><td colspan="3">Request</td></tr>
<tr><td>Request URL</td><td colspan="2">GET http://example.com/api/ib/GetServicesInfo:componentFilter</td></tr>
<tr><td rowspan="2">Resource Parameters</td><td>Name</td><td>Description</td></tr>
<tr><td>componentFilter
Required Object</td><td>JSON object with the filter parameters. The value -1 means that the filter is not taken into account.
Example:
{
“componentName”: “Cloud Storage”
}</td></tr>
<tr><td colspan="3">Response</td></tr>
<tr><td rowspan="4">HTTP Status Code</td><td>Value</td><td>Description</td></tr>
<tr><td>200</td><td>The request was successful</td></tr>
<tr><td>400</td><td>Bad Request</td></tr>
<tr><td>500</td><td>Error during processing</td></tr>
<tr><td>JSON Attributes</td><td>template
JSON Object</td><td>JSON object with the answer.
Example:
{
“components”:
[
{
“componentName” : “Cloud_Storage”,
“services” :
[
{
“serviceName” : “CreateBucket”,
“parameters”:[‘param1’,‘param2’]
},
{
“serviceName” : “DeleteBucket”,
“parameters”:[‘param1’,‘param2’]</td></tr>
</table>

		}] }] }

JSON Error	JSON Object	Example:
		{
		“errors”:
		[
		{
		“errorId” : 1,
		“description” : “No components found”
		}
		]
		}

Figure 7: RESTful Interface Description – Get Info Components

3.2.4.1.2 **Method “Get Federated Instances“**

The RESTful interface “GetFederatedInstances” (Figure 8), will be used to get a list with related info about federated instances registered in the SAM platform. This interface will be useful in case that you need to know any federated instance identifier or name.

Get Federated Instances		
Description	Interface to get a list with related info about federated instances registered in the SAM platform.	
Request		
Request URL	**GET** http://example.com/api/ib/GetFederatedInstances:componentFilter	
Resource Parameters	Name	Description

	componentFilter
	Required Object
		JSON object with the filter parameters. The value -1 means that the filter is not taken into account.
		Example:
		{
		“instanceId” : -1,
		“name” : “Federated Instance 1”,
		}

Response		

HTTP Status Code	Value	Description
	200	The request was successful
	400	Bad Request
	500	Error during processing

JSON Attributes	template
	JSON Object
		JSON object with the answer.
		Example:
		{
		“instances”:
		[
		{
		“instanceId” : 1,
		“description” : “Federated Instance 1”
		},
		{
		“instanceId” : 3,
		“description” : “Federated Instance 3”
		}
		]
		}

JSON Error	JSON Object	Example:
		{
		“errors” :
		[
		{
		“errorId” : 1,
		“description” : “No components found”
		}
		]
		}

Figure 8: RESTful Interface Description – Get List of Federated Instances

3.2.4.1.3 **Method “Send Message Component“**

The RESTful interface “SendMessageComponent” (Figure 9), will be used to send a message to a component. All messages in SAM will use a common XML structure detailed in Section 3.2.4.3.1. The service information can be obtained using the “Get Services Info” interface described in Section 3.2.4.1.1.

<table>
<tr><th colspan="3">Send Message Component</th></tr>
<tr><td>Description</td><td colspan="2">Interface to send a message to a component.</td></tr>
<tr><td colspan="3">Request</td></tr>
<tr><td>Request URL</td><td colspan="2">POST
http://example.com/api/ib/SendMessageComponent:messageDestination/</td></tr>
<tr><td rowspan="2">Resource Parameters</td><td>Name</td><td>Description</td></tr>
<tr><td>messageDestination
Required Object</td><td>Example:
<SAMRequest>
<Header>
<Destination>CloudStorage</Destination>
<Service>CreateBucket</Service>
<Topic> </Topic>
<Parameters>
<Parameter>
<Name>bucketId</Name>
<Value>23</Value>
</Parameter>
</Parameters>
</header>
<Payload> { JSON string or another embedded Xml message </Payload>
</SAMRequest></td></tr>
<tr><td colspan="3">Response</td></tr>
<tr><td rowspan="4">HTTP Status Code</td><td>Value</td><td>Description</td></tr>
<tr><td>200</td><td>The request was successful</td></tr>
<tr><td>400</td><td>Bad Request</td></tr>
<tr><td>500</td><td>Error during processing</td></tr>
<tr><td>JSON Attributes</td><td>template
JSON Object</td><td>JSON object with the answer.
Example 1:
{
“id” : 3,
“description” : “Error. Component 2-Semantic Services not found”
}</td></tr>
</table>

Figure 9: RESTful Interface Description – Send a Message to a Component

3.2.4.1.4 **Method "Send Message External Services"**

The RESTful interface "SendMessageExternalServices" (Figure 10) will be used to send a message to an external resource. All messages to external resources will use a common XML structure detailed in Section 3.2.4.3.2.

Send Message External Services		
Description	Send information to External Services	
Request		
Request URL	**POST** http://example.com/api/ib/SendMessageExternalServices:externalServiceInformation	
Resource Parameters	Name	Description
	externalServiceInformation	XML Structure Example: <SAMRequest> <Header> <Destination>External</Destination> <Service>"http://en.wikipedia.org/wiki/Mike_Nielsen"</Service> <Parameters> <Parameter> <Name>param1</Name> <Value>abc</Value> </Parameter> </Parameters> </Header> <Payload> { JSON string or another embedded Xml message </Payload> </SAMRequest>
Response		
HTTP Status Code	Value	Description
	200	The request was successful
	400	Bad Request
	500	Error during processing
JSON Attributes	template	A JSON object with the information.

	JSON Object	Example:
		[
		{
		"code" : "
		<!DOCTYPE html>
		<html lang="en" dir="ltr" class="client-nojs">
		<head>
		<meta charset="UTF-8" />
		<title>Mike Nielsen - Wikipedia, the free encyclopedia</title>
		<meta name="generator" content="MediaWiki 1.25wmf3" />
		<link rel="alternate" href="android-app://org.wikipedia/http/en.m.wikipedia.org/wiki/Mike_Nielsen" />
		<link rel="alternate" type="application/x-wiki" title="Edit this page" href="/w/index.php?title=Mike_Nielsen&action=edit" />
		<link rel="edit" title="Edit this page" href="/w/index.php?title=Mike_Nielsen&action=edit" />
		<link rel="apple-touch-icon" href="//bits.wikimedia.org/apple-touch/wikipedia.png" />
		<link rel="shortcut icon" href="//bits.wikimedia.org/favicon/wikipedia.ico" />
		…
		</html>”
		}
		]

Figure 10: RESTful Interface Description – Send a Message to an External Service

3.2.4.1.5 **Method “Get List Queues”**

The RESTful interface “GetListQueues” (Figure 11), returns a list of queues/topics. This result contains the necessary information to subscribe to any of the queues/topics or to send messages to a queue/topic. After subscription, the component will receive messages from the queue/topic.

Get List Queues

<table>
<tr><td>Description</td><td colspan="2">Interface to get a list of queues/topics</td></tr>
<tr><td colspan="3">Request</td></tr>
<tr><td>Request URL</td><td colspan="2">POST http://example.com/api/ib/GetListQueues:filter/</td></tr>
<tr><td rowspan="2">Resource Parameters</td><td>Name</td><td>Description</td></tr>
<tr><td>filter
Required Object</td><td>JSON object with the filter parameters. The value -1 means that the filter is not taken into account. Type can be “queue” or “topic”
Example:
{
“componentName” : “Cloud_Storage”,
“type” : “topic”,
“description”:”sports”
}</td></tr>
<tr><td colspan="3">Response</td></tr>
<tr><td rowspan="4">HTTP Status Code</td><td>Value</td><td>Description</td></tr>
<tr><td>200</td><td>The request was successful</td></tr>
<tr><td>400</td><td>Bad Request</td></tr>
<tr><td>500</td><td>Error during processing</td></tr>
<tr><td>JSON Attributes</td><td>template
JSON Object</td><td>JSON object with the answer.
Example:
{
“Queues” :
[
{
“componentName” : “Cloud_Storage”,
“type” : “topic”,
“description” : “sports queue”
},
{
“componentName” : “Cloud_Storage”,
“type” : “queue”,
“description” : “semantic queue”
}
]</td></tr>
</table>

		}

JSON Error	JSON Object	Example:
		{
		“errors” :
		[
		{
		“errorId” : 1,
		“description” : “parameters are not valid”
		}
		]
		}

Figure 11: RESTful Interface Description –Get a List of Queue/Topic

3.2.4.1.6 **Method “Subscribe Queue“**

The RESTful interface “Subscribe to a queue/topic” (Figure 12) will be used to subscribe a component to a queue/topic in order to receive messages from this queue. The XML structure of the message is described in Section 3.2.4.3.2.

Subscribe to a Queue/Topic		
Description	Interface to subscribe to a queue/topic	
Request		
Request URL	**POST** http://example.com/api/ib/SubscribeQueue:subscriptionInfo/	
Resource Parameters	Name	Description
	subscriptionInfo Required Object	XML document. Example: <SAMSubscription> <Subscriptor> Content_Gateways </Subscriptor> <Callback> url </Callback> <Expression> /SAMRequest/destination[text()=’CloudStorage’] </Expression> <Expression> /SAMRequest/topic[text()=’sport’]

		</Expression> <SAMSubscription>
Response		
HTTP Status Code	Value	Description
	200	The request was successful
	400	Bad Request
	500	Error during processing

JSON Attributes	template
	JSON Object
		JSON object with the answer.
		Example 1:
		{
		“Id” : 1,
		“description” : “Subscription successfully”
		}
		Example 2:
		{
		“Id” : 3,
		“description” : “Error. Component 2-Semantic Services not found”
		}

JSON Error	JSON Object	Example:
		{
		“errors” :
		[
		{
		“errorId” : 1,
		“description” : “Bad request”
		}
		]
		}

Figure 12: RESTful Interface Description – Subscribe to a Queue/Topic

3.2.4.1.7 **Method “Unsubscribe Queue”**

The RESTful interface “UnsubscribeQueue” (Figure 13) will be used to unsubscribe a component from a queue/topic. The XML structure of the message is described in Section 3.2.4.3.4.

Unsubscribe Queue

Description	Interface to unsubscribe from a queue/topic	
Request		
Request URL	**POST** http://example.com/api/ib/UnsubscribeQueue:subscriptionInfo/	
Resource Parameters	Name	Description
	subscriptionInfo Required Object	JSON object with the filter parameters. The value -1 means that the filter is not taken into account. XML Structure Example: <SAMUnSubscription> <Subscriptor> Content_Gateways </Subscriptor> <Expression> /SAMRequest/destination[text()='CloudStorage'] </Expression> <Expression> /SAMRequest/topic[text()='sport'] </Expression> </SAMUnSubscription>
Response		
HTTP Status Code	Value	Description
	200	The request was successful
	400	Bad Request
	500	Error during processing

JSON Attributes	template
	JSON Object
		JSON object with the answer.
		Example 1:
		{
		"Id" : 1,
		"description" : "Unsubscribed successfully"
		}
		Example 2:
		{
		"Id" : 3,
		"description" : "Error. Component 2-Semantic

		Services not found” }

JSON Error	JSON Object	Example:
		{
		“errors” :
		[
		{
		“errorId” : 1,
		“description” : “Bad request”
		}
		]
		}

Figure 13: RESTful Interface Description – Unsubscribe to a Queue/Topic

3.2.4.1.8 **Method “Send Message Queue”**

The RESTful interface “SendMessageQueue” (Figure 14) will be used to send a message to a queue/topic. It uses the SAM Component Message structure defined in Section 3.2.4.3.1.

Send Message Queue		
Description	Interface to send a message to a queue/topic	
Request		
Request URL	**POST** http://example.com/api/ib/SendMessageQueue:messageDestination/	
Resource Parameters	Name	Description
	messageDestination Required Object	XML Structure. Example: <SAMUnSubscription> <Subscriptor> Content_Gateways </Subscriptor> <Expression> /SAMRequest/destination[text()=’CloudStorage’] </Expression> <Expression> /SAMRequest/topic[text()=’sport’]

		</Expression> </SAMUnSubscription>
Response		
HTTP Status Code	Value	Description
	200	The request was successful
	400	Bad Request
	500	Error during processing

JSON Attributes	template
	JSON Object
		JSON object with the answer.
		Example 1:
		{
		“Id” : 1,
		“description” : “Message has been sent successfully”
		}
		Example 2:
		{
		“Id” : 3,
		“description” : “Error. Component 2-Semantic Services not found”
		}

JSON Error	JSON Object	Example:
		{
		“errors” :
		[
		{
		“errorId” : 1,
		“description” : “Bad request”
		}
		]
		}

Figure 14: RESTful Interface Description – Send a Message to a Queue/Topic

3.2.4.1.9 **Method “Get Workflow List”**

The RESTful interface “GetWorkflowList” (Figure 15) will be used to get a list with related info about the workflows created by a user.

Get Workflow List

<table>
<tr><td>Description</td><td colspan="2">Interface to get a list with related info about workflows registered in the SAM platform.</td></tr>
<tr><td colspan="3">Request</td></tr>
<tr><td>Request URL</td><td colspan="2">GET http://example.com/api/ib/GetWorkflowList:filter</td></tr>
<tr><td rowspan="2">Resource Parameters</td><td>Name</td><td>Description</td></tr>
<tr><td>filter
Required Object</td><td>JSON object with the filter parameters. The value -1 means that the filter is not taken into account.
Example:
{
“userId” : 20
}</td></tr>
<tr><td colspan="3">Response</td></tr>
<tr><td rowspan="4">HTTP Status Code</td><td>Value</td><td>Description</td></tr>
<tr><td>200</td><td>The request was successful</td></tr>
<tr><td>400</td><td>Bad Request</td></tr>
<tr><td>500</td><td>Error during processing</td></tr>
<tr><td>JSON Attributes</td><td>template
JSON Object</td><td>JSON object with the answer.
Example:
{
“workflows” :
[
{
“workflowId” : 1,
“name” : “Workflow 1”,
“description” : “Complex Workflow 1”
},
{
“workflowId” : 2,
“name” : “Workflow 2”,
“description” : “Complex Workflow 2”
}
]
}</td></tr>
<tr><td>JSON Error</td><td>JSON Object</td><td>Example:</td></tr>
</table>

		{
		“errors” :
		[
		{
		“errorId” : 1,
		“description” : “userId not found”
		}
		]
		}

Figure 15: RESTful Interface Description – Get List of Workflows

3.2.4.2 Protocols

In SAM components, it will be mainly use HTTP, HTTPs and REST protocols. However, TSB additionally supports a wide set of communication protocols as part of its communication infrastructure. Following the different supported protocols in detail:

- **HTTP(s):** Hypertext Transfer Protocol (HTTP) is used to request and transmit files, especially web pages and web page components, over the Internet or other computer networks. The secure version (HTTPS) is a combination of the Hypertext Transfer Protocol with the SSL/TLS protocol to provide encryption and secure identification of the server.
- **REST:** Representational state transfer (REST) is a style of software architecture which is focused on the use of the HTTP and XML standards for the transmission of data. The operations will be requested using GET, POST, PUT and DELETE, so it does not require special implementations to consume these services. In addition it is possible to use JSON instead of XML as a container for the information.
- **SOAP:** Simple Object Access Protocol (SOAP) is a standard protocol that defines how two objects in different processes can communicate through the exchange of XML data, identifying the point of SOAP is that the operations are defined as ports WSDL (Web Services Description Language).
- **XMPP:** Extensible Messaging and Presence Protocol (XMPP) is a communication protocol for message-oriented middleware based on XML (Extensible Markup Language).
- **SMTP:** Simple Mail Transfer Protocol (SMTP) is an Internet standard for electronic mail (e-mail) transmission. TSB supports secured and non-secured communication.
- **IMAP:** Internet Message Access Protocol (IMAP) is a protocol for e-mail retrieval and storage developed as an alternative to POP. IMAP, unlike POP, specifically allows multiple clients simultaneously connected to the same mailbox, and through flags stored on the server, different clients accessing the same mailbox at the same or different times can detect state changes made by other clients.
- **S/MIME:** Secure/Multipurpose Internet Mail Extensions is a widely accepted protocol for sending digitally signed and encrypted messages. S/MIME allows encrypting emails and digitally signing them.
- **FTP:** File Transfer Protocol is a standard network protocol used to transfer computer files from one host to another host over a TCP-based network. TSB also provide secure FTP (SFTP).

- **X.400 / P7:** The P7 protocol allows a mail user agent to send and receive messages through an X.400 message store and the X.400 MTS. It is the X.400 equivalent to the Internet IMAP protocol, providing services to manage the messages within the message store, rather than having to download them locally.
- **GXS:** Managed File Transfer Service enabling the secure, reliable exchange of large or bulk files.AS2. Applicability Statement (AS) 2 uses the same signing, encryption, and MDN conventions used in the original AS1 protocol. AS2 messages are usually sent across the internet using the HTTP or HTTPS protocol.
- **AS3:** Applicability Statement (AS) 3 was developed by the IETF to implement secure and reliable messaging over FTP. AS3 is based upon the secure version of the FTP protocol, rather than HTTP.
- **Oftp:** Odette File Transfer Protocol was developed to offer a standard communication platform for the European automotive industry. The protocol is extremely efficient, allowing large transmission windows to be utilised while incorporating file restart, data compression and security.

3.2.4.3 Format

In order to send messages within the SAM platform, it is necessary to define some common XML formats. These structures will be used by the TSB to control the communication in SAM Platform. The structures to send messages between components, to send message to external resources and to subscribe/unsubscribe a component in a queue/topic will be described in the following subsections.

3.2.4.3.1 **SAM Component Message**

The SAM messages between components will have the following common XML structure:

- **<SAMRequest>:** This root label will be used by TSB in order to identify that the message belongs to the SAM platform. The message is further separated into a header (<Header>) which contains the information about the destination and body (<Payload>) in case extra embedded information is required.
- **<Destination>:** This label will identify the destination component. The value can be extracted using the interface “Get Services Info” (see Section 3.2.4.1.1)
- **<Service>:** This label will identify the service which will receive the information. The value can be extracted using the interface “Get Services Info” (see Section 3.2.4.1.1)
- **<Topic>:** This label is required when sending a message to a topic. The topic identifier can be extracted using the interface “Get List Queues” (see Section 3.2.4.1.5)
- **<Parameters>:** This label will be used to send the necessary service parameters. It will contain a label for parameter (<Parameter>) and within it a label for name (<Name>) and value (<Value>)
- **<Payload>:** This label will be used to send any extra information that the service could need (e.g. JSON structure, XML, etc.).

```
<SAMRequest>
   <Header>
    <Destination>CloudStorage</Destination>
    <Service>CreateBucket</Service>
    <Topic> </Topic>
    <Parameters>
     <Parameter>
       <Name>bucketId</Name>
       <Value>23</Value>
    </Parameter>
   </Parameters>
  </Header>
  <Payload> { JSON string or another embedded Xml message </Payload>
</SAMRequest>
```

Figure 16: SAM Component Message

3.2.4.3.2 **SAM External Message**

In order to send a message to an external resource the same XML structure defined in the previous Section 3.2.4.3.1 will be used, but the destination will be filled with the value “External” and the label services will contain the destination url.

```
<SAMRequest>
   <Header>
    <Destination>External</Destination>
    <Service>“http://en.wikipedia.org/wiki/Mike_Nielsen”</Service>
    <Parameters>
     <Parameter>
      <Name>param1</Name>
      <Value>abc</Value>
     </Parameter>
    </Parameters>
  </Header>
  <Payload> { JSON string or another embedded Xml message </Payload>
</SAMRequest>
```

Figure 17: SAM External Message

3.2.4.3.3 **SAM Subscription Message**

In order to subscribe a component in a queue/topic, it is necessary to send the specific XML message with the following structure:

- **<SAMSubscription>:** This root label identifies the message as a queue/topic subscription message within the SAM Platform.
- **<Subscriber>:** This label will identify the component which wants to subscribe.
- **<Callback>:** This label will contain the url of the service which will receive the messages from the queue/topic.
- **<Component>:** This label will identify the component which registered the queue/topic.
- **<Queue>:** This label will contain the queue/topic information.

```
<SAMSubscription>
 <Subscriber>Content_Gateways</Subscriber>
 <Callback> service_url </Callback>
 <Component> /SAMRequest/destination[text()='CloudStorage'] </Component>
 <Queue> /SAMRequest/queue[text()='q1'] </Queue>
</SAMSubscription>
```

Figure 18: SAM Queue Subscription Message

```
<SAMSubscription>
 <Subscriber>Content_Gateways</Subscriber>
 <Callback> service_url </Callback>
 <Component> /SAMRequest/destination[text()='CloudStorage'] </Component>
 <Queue> /SAMRequest/topic[text()='sport'] </Queue>
</SAMSubscription>
```

Figure 19: SAM Topic Subscription Message

3.2.4.3.4 **SAM Unsubscription Message**

The message structure to unsubscribe a service from a queue/topic is the same as the subscribe message (see Section 3.2.4.3.3) but with the main label changed to "SAMUnSubscription".

```
<SAMUnSubscription>
 <Subscriber>Content_Gateways</Subscriber>
 <Callback> service_url </Callback>
 <Component> /SAMRequest/destination[text()='CloudStorage'] </Component>
 <Queue> /SAMRequest/queue[text()='q1'] </Queue>
</SAMUnSubscription>
```

Figure 20: SAM Queue Unsubscription Message

```
<SAMUnSubscription>
 <Subscriber>Content_Gateways</Subscriber>
 <Component> /SAMRequest/destination[text()='CloudStorage'] </Component>
 <Queue> /SAMRequest/topic[text()='sport'] </Queue>
</SAMUnSubscription>
```

Figure 21: SAM Topic Unsubscription Message

3.2.5 Summary

This section has explained which technology will be used to implement the Interconnection Bus component based on the major design decisions detailed in Section 3.2.1. TIE's SmartBridge ESB has been selected as the basis for the communication module since it meets the criteria to be used as technology and it is nominated in the DOW as the technology to be used (See DOW Section B1.3.3.4 WP4 Information Management & Interoperability System). TSB will be adapted to support the internal communication features and extended to implement federation patterns needed. TSB already includes a graphical user interface which will be extended in order to adapt to the SAM needs. Finally, the different interfaces, protocols and formats, that the Interconnection Bus provides, have been detailed in Section 3.2.4.

3.3 Semantic Services

The Semantic Services constitutes the central component for addressing the textual and semantic processes in the SAM platform, see Figure 22. This component uses different language technologies in order to provide natural language processing and semantic inference functionalities. These operations will be provided as several web methods in a web service to be consumed by different components of the SAM platform. The main characteristics of this component are the following:

- The Semantic Services component will directly interact with other components of the Platform providing semantic functionalities based on linguistic and semantic knowledge
- These functionalities consist of: semantic data analysis, semantic exploration and discovery, sentiment analysis and automatic summarisation
- As part of this component, the Asset Profiler editor will provide a user interface to allow SAM Content Providers to supervise and edit assets imported into the SAM platform, as well as to create new ones

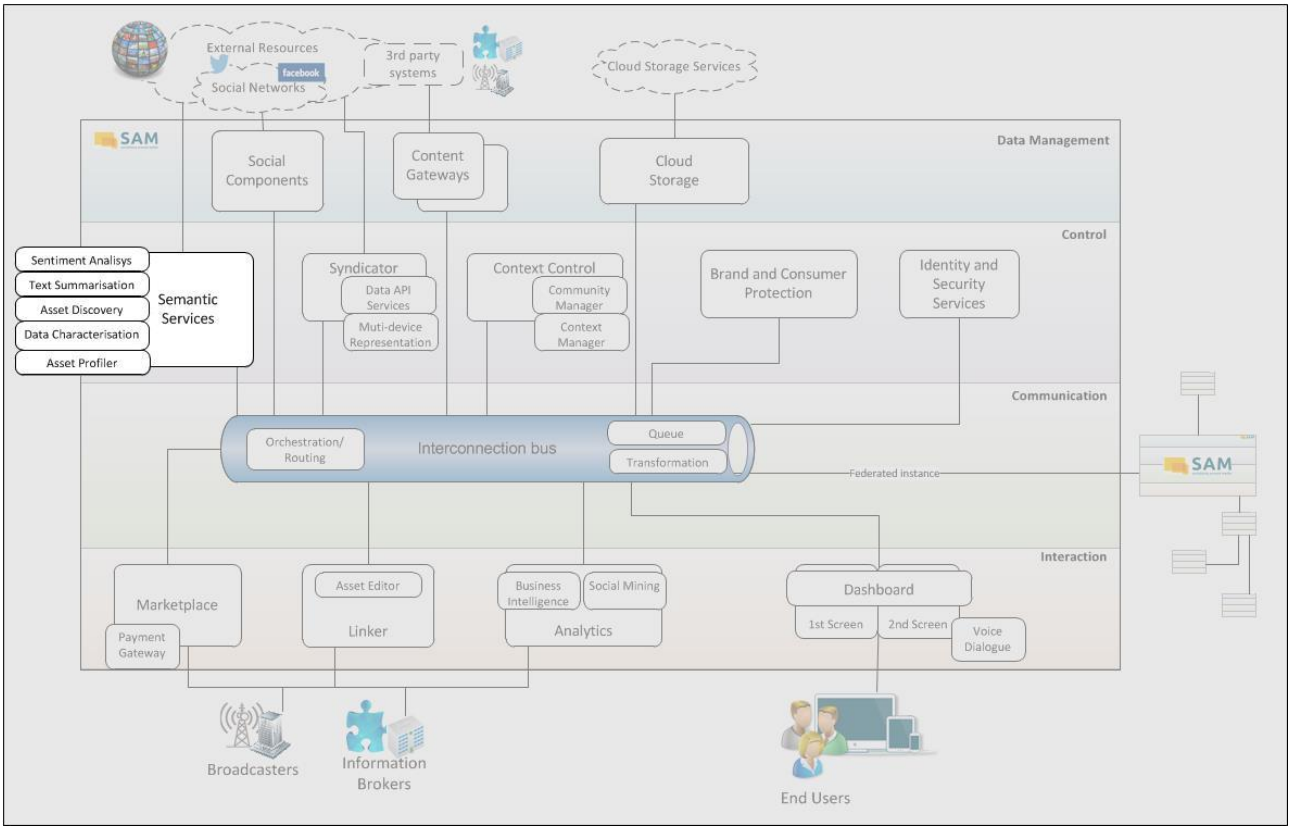

Figure 22: Architecture Overview – Semantic Services

3.3.1 Major Design Decisions

The focus of the Semantic Services is to address the following important tasks:

- Text characterisation
- Sentiment analysis
- Text summarization
- Semantic Inference

Each task has different types of functionalities to consider. However, some of these are common for the Semantic Services task. Therefore, the technologies in discussion were

split into the following four groups for conducting their comparison and proceeding to select the most appropriate for their common usage by all functionalities:

- NLP tools described in Section 3.3.2.1.1
- Semantic APIs or libraries described in Section 3.3.2.1.2
- Semantic repositories described in Section 3.3.2.1.3
- Machine learning tools described in Section 3.3.2.1.4

For each of the above mentioned functionality groups a description is provided:

- **Text characterisation** task involving lexical disambiguation technologies in order to identify the lexical units of the analysed text for linking them with semantic labels. As part of this process, it is necessary to take into account the context (i.e. text adjacent to the target lexical units), lexical units (including tokenisation of words and part of speech tagging), semantic repositories, and corpora for supporting language processing techniques, as well as semantic label dictionary for linking. In the following sections, Natural Language Processing (NLP) tools (Section 3.3.2.1.1), semantic APIs (Section 3.3.2.1.2), semantic repositories (Section 3.3.2.1.3) and machine learning tools (Section 3.3.2.1.4) were compared in order to select the most suitable technologies for conducting the tasks involved in this component.
- For **sentiment analysis** task numerous state-of-the-art resources play an important role in the component. From those, sentiment dictionaries and corpora should be taken into account to train the learning systems involved in this component, which is based on the selected technology identified in Section 3.3.2.1.4. Pre-processing text using NLP tools is also necessary in sentiment analysis so as to identify lexical units into the analysed text (including tokenisation of words and part of speech tagging). Moreover, semantic repositories and machine learning tools are necessary to perform this functionality, since the sentiment analysis information requires a substantial amount of sentiment knowledge which should be acquired by means of these technologies and resources.
- For the **text summarisation** task it is necessary to employ different NLP tools. These tools are required to apply sentence segmentation, tokenisation of words, part of speech tagging and stemming (helping to expand the word context). In Section 3.3.2.1 there is a comparison among different NLP tools that are suitable for this task.
- Providing **semantic inference,** different semantic engines can be used. These inference engines depend on querying RDF-based formal languages, such as SPARQL[11], or DL-based, such as SPARQL-DL[12]. In this respect, semantic APIs for exploiting semantic databases should be considered.

3.3.2 Technology Comparison and Selection

This subsection outlines technology selection criteria and compares existing technologies candidates for the development of the Semantic Services component. Within the subsection the areas that need to be implemented or improved to fully meet SAM specific requirements are also identified and presented. The selected technologies will be used as a base during the development phase in the realisation of the technical design of this component: The specific selection criteria to be used are specified in the document D3.2.2 Functional Specification.

[11] http://www.w3.org/TR/rdf-sparql-query/
[12] http://www.w3.org/2001/sw/wiki/SPARQL-DL

3.3.2.1 Possible Technologies and Comparison

In order to realise the Semantic Services component several options have to be considered as base technology. The assessment of these technologies is presented in the following sections.

3.3.2.1.1 Natural Language Processing Tools

NLP tools constitute all those tools that provide functionalities for processing and analysing textual inputs from lexical, syntactical and semantic point of views. For the case of SAM, sentence segmentation, tokenisation of words, Part-of-Speech tagging and stemming are the main focus:

- **Freeling**[13] is an open-source language analysis tool suite. The Freeling package consists of a library providing language analysis services (e.g., morphological analysis, date recognition, POS tagging and Rule-based dependency parsing).
- **Stanford CoreNLP**[14] provides a set of natural language analysis tools for the English language. Stanford CoreNLP is an integrated framework that combines a set of language analysis tools including a part-of-speech (POS) tagger, a named entity recognizer (NER), a parser, a co-reference resolution system and other tools. It is designed to be highly flexible and extensible.
- The **Apache Open NLP**[15] library is a machine-learning toolkit for the processing of natural language text. It supports the most common NLP tasks, such as tokenisation, sentence segmentation, part-of-speech tagging, named entity extraction, chunking, parsing, and co-reference resolution. OpenNLP also includes learning technologies such as maximum entropy and neuronal networks, which enables processing complex data to make predictions or decisions.
- **Apache Lucene**[16] is a high-performance, full-featured text search engine library which contains NLP functionalities (such as tokenisation and stemming), written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform. Apache Lucene is an open-source project available for free download. Lucene offers powerful features through a simple API: scalable, high-performance indexing; powerful, accurate and efficient search algorithms; and cross-platform functionality.
- **Apache Commons Lang**[17] + **Jexl**[18] are part of Apache's reusable Java components. The first one, Lang, provides additional methods for Java String manipulation (using regular expressions or Levenshtein distance) along with basic numerical methods (for managing big numerical values), among others. The latter library, Jexl, offers functionalities to evaluate logical expressions dynamically.

Parameter	Importance	Freeling	Stanford CoreNLP	Apache Open NLP	Apache Lucene	Apache Commons Lang + Jexl
Generic Parameters						
Maturity & Stability	+++	+++	+++	+++	+++	+++
Regularly Updated	+/-	++	++	++	++	++
Technical Up-to-Datedness /	++	+++	+++	+++	+++	++

[13] http://nlp.lsi.upc.edu/freeling/
[14] http://nlp.stanford.edu/software/corenlp.shtml
[15] https://opennlp.apache.org/
[16] http://lucene.apache.org/core/
[17] http://commons.apache.org/proper/commons-lang/
[18] http://commons.apache.org/proper/commons-jexl/index.html

Appeal						
Open Source	+	YES	YES	YES	YES	YES
Non-Infecting	+	NO	NO	YES	YES	YES
Code Quality	++	N/A	N/A	N/A	N/A	N/A
Extensibility	+++	+	++	++	++	++
Community	-	+/-	+	+++	++	+
Performance / Scalability	+++	+	+	++	+++	+++
Reuse of existing developments	+/-	+	-	-	+++	+
EU project origin	---	NO	NO	NO	NO	NO
Platform (Portability)	+	-	+++	+++	+++	+++
Open Standards Compliance	+	YES	N/A	YES	N/A	N/A
Interoperability (easy integration for all platforms)	++	+	+++	+++	++	+++
Specific Parameters						
Lexical disambiguation	+	+	+	+	-	-
Part of Speech Tagging	+++	+++	+++	+++	-	-
Syntactic analysis	+	++	++	++	-	-
Semantic Analysis	+	+	+	-	-	-
Textual annotation	+	-	-	-	-	-
Sentiment analysis	+	-	+	-	-	-
Text summarisation	-	+	+	+	+	+
Semantic inference	-	-	-	-	-	-

Figure 23: Parameter Evaluation of Technologies of Natural Language Processing Tools

Some summarisation tools were studied. A description of them can be found below:

- **GPLSI Compendium**[19] is a text summarisation system capable of generating generic informative summaries for the English language of a specific length provided by the user. This system makes use of POS Tagging functionalities from third-party tools. The Tree-Tagger tool is set as default POS Tagger, but it has to be replaced since it is licensed using GPL and therefore it can affect other components as well. Therefore, the NLP technology selected from Figure 23 will provide the POS Tagging functionalities necessary for GPLSI COMPENDIUM.
- **MEAD**[20] is a publicly available toolkit for multi-lingual summarisation and evaluation. The toolkit implements multiple summarisation algorithms (at arbitrary compression rates) such as position-based, Centroid, TF*IDF, and query-based methods. Methods for evaluating the quality of the summaries include co-selection (precision/recall, kappa, and relative utility) and content-based measures (cosine, word overlap, bigram overlap).
- **Open Text Summarizer**[21] (OTS) is an open-source tool for summarizing texts. The program reads a text and decides which sentences are important and which are not. OTS is both a library and a command line tool. The program is multilingual and works with UTF-8 encoding. In this approach, keywords are identified by means of word occurrence, and sentences are given a score based on the keywords they contain. Some language-specific resources, such as stemmers and stop word lists are employed. It has been shown that this system obtains better performance than other multi-lingual Text Summarisation systems. Its license is GPL, a viral license.
- **Essential Summarizer**[22] (Essential): is a commercial tool, which relies on linguistic techniques to perform semantic analysis of written text, taking into account discursive

[19] http://gplsi.dlsi.ua.es/gplsi11/en/content/gplsi-compendium
[20] http://www.summarization.com/mead/
[21] http://libots.sourceforge.net/
[22] https://essential-mining.com/summarizer/index.jsp?ui.lang=en

elements of the text. It is able to produce summaries in twenty languages. The licence of this tool has not been found.

Parameter	Importance	GPLSI Compendium	MEAD	OTS	Essential
Generic Parameters					
Maturity & Stability	+++	+++	+++	++	++
Regularly Updated	+/-	+	++	+	++
Technical Up-to-Datedness / Appeal	++	+++	+++	+++	+++
Open Source	+	N/A	YES	YES	NO
Non-Infecting	+	YES	YES	NO	N/A
Code Quality	++	N/A	N/A	N/A	N/A
Extensibility	+++	-	++	++	-
Community	-	+	++	++	++
Performance / Scalability	+++	+++	++	++	+++
Reuse of existing developments	+/-	+++	-	-	-
EU project origin	---	NO	NO	NO	NO
Platform (Portability)	+	+	+	+	+
Open Standards Compliance	+	N/A	N/A	N/A	N/A
Interoperability (easy integration for all platforms)	++	+++	+++	+++	+++
Specific Parameters					
Text summarisation	+++	+++	++	++	++

Figure 24: Parameter Evaluation of Technologies for Semantic APIs – Summarisers

3.3.2.1.2 Semantic APIs

The Semantic APIs are programming libraries for accessing the Semantic Data Storage in an intuitive way. In order to retrieve the semantic information, query languages, such as SPARQL or DL, can be used. Also, semantic reasoners [DCT11] are taken into account since the inference of knowledge constitutes a special part of the Semantic Services. These client APIs are used for querying the semantic data from the Cloud Storage:

- **Sesame**[23] is an open-source de facto standard framework for RDF data format. Sesame supports two query languages, SeRQL and SPARQL. As an open-source product, Sesame is free of charge and can be hosted on any server running Java.
- **Jena**[24] is an open-source RDF framework which also provides APIs for leading with TDB[25] (Triple store) and OWL[26] [27]. It supports SPARQL as query language. It is very similar to Sesame with the exception that it provides support for reasoning.
- **Protégé**[28] is a free, open-source ontology editor and framework for building intelligent systems. Protégé provides an open Java API to query and manipulate models through F-Logic and DM language.

[23] http://rdf4j.org/
[24] https://jena.apache.org/
[25] https://jena.apache.org/documentation/tdb/index.html
[26] http://www.w3.org/standards/techs/owl#w3c_all
[27] https://jena.apache.org/documentation/ontology/
[28] http://protege.stanford.edu/

Parameter	Importance	Sesame	Jena	Protégé
Generic Parameters				
Maturity & Stability	+++	+++	+++	+++
Regularly Updated	+/-	+	+	+
Technical Up-to-Datedness / Appeal	++	++	++	++
Open Source	+	YES	YES	YES
Non-Infecting	+	YES	YES	YES
Code Quality	++	N/A	N/A	N/A
Extensibility	+++	++	++	++
Community	-	++	++	++
Performance / Scalability	+++	+++	+++	++
Reuse of existing developments	+/-	+++	+	-
EU project origin	---	NO	NO	NO
Platform (Portability)	+	+++	+++	+++
Open Standards Compliance	+	NO	NO	YES
Interoperability (easy integration for all platforms)	++	+++	+++	++
Specific Parameters				
Semantic inference	+++	+++	++	++

Figure 25: Parameter Evaluation of Technologies for Semantic APIs – Database Engine

3.3.2.1.3 Semantic Repositories

Semantic Repositories are resources which are similar to databases management systems (DBMS). Some of them have APIs for their exploitation and others only consist of data stored in a structured way. These resources permit storing, querying and handling of data. Additionally, the semantic repository can use ontologies as a semantic schema for automatically reasoning based on query data. Semantic repositories make use of generic and flexible physical data models, such as graphs. This allows them to quickly read and implement new metadata, schemata or ontologies. As a result, semantic repositories provide better incorporation of assorted data as well as more analytical power. For SAM, the data will be stored into a RDF data format and will be managed by the technology selected in Section 3.8.2.2.2, Sesame. Besides, incorporating annotated corpora as well as the lexical dictionaries were considered as semantic repository. Therefore, the following categories are taken into account:

- Corpus, described in the following section
- Lexical database, described in Section 3.3.2.1.3.2
- Knowledge base (for ontologies and semantic networks), described in Section 3.3.2.1.3.3

3.3.2.1.3.1 Corpus

- **EmotiBlog**[BBM09] is a collection of blog posts created and annotated for detecting subjective expressions in the new textual genres born with the Web 2.00 (e.g. social networks, blogging or microblogging).

- **Semeval data sets** are a set of documents in which a large amount of tweets are tagged as Positive, Negative or Neutral. These have been taken from Task 2[29] of Semeval 2013, which contains 12-20k twitter messages, as well as Task 9[30] of Semeval 2014, which has more than 15K twitter messages.

Parameter	Importance	EmotiBlog	Semeval Data Sets
Generic Parameters			
Maturity & Stability	+++	+	+
Regularly Updated	+/-	+	+
Technical Up-to-Datedness / Appeal	++	+++	+++
Open Source	+	N/A	N/A
Non-Infecting	+	YES	YES
Code Quality	++	N/A	N/A
Extensibility	+++	N/A	N/A
Community	-	-	-
Performance / Scalability	+++	N/A	N/A
Reuse of existing developments	+/-	+++	+++
EU project origin	---	NO	NO
Platform (Portability)	+	+++	+++
Open Standards Compliance	+	-	-
Interoperability (easy integration for all platforms)	++	+++	+++
Specific Parameters			
Sentiment analysis	++	+	++

Figure 26: Parameter Evaluation of Technologies for Semantic Repositories – Corpus

3.3.2.1.3.2 Lexical Databases

- **WordNet**[31] is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. The resulting network of meaningfully related words and concepts can be navigated using a browser or the API. WordNet is freely and publicly available for download. WordNet's structure makes it a useful tool for supporting natural language processing tasks.
- **BabelNet**[32] is both a multilingual encyclopaedic dictionary, with lexicographic and encyclopaedic coverage of terms, and a semantic network. It connects concepts and named entities in a very large network of semantic relations, made up of more than 9 million entries with a meaning and all the synonyms which express that meaning.
- **WordNet Domains**[33] **(WND)** is a lexical resource created by augmenting WordNet with domain labels. It includes **WordNet-Affect**[34] **(WNA)** resource which is an additional hierarchy of "affective" domain labels
- **SentiWordNet**[35] is a lexical resource for opinion mining. SentiWordNet assigns to each synset of WordNet three sentiment scores: positivity, negativity, objectivity

[29] http://www.cs.york.ac.uk/semeval-2013/task2/index.php?id=data
[30] http://alt.qcri.org/semeval2014/task9/index.php?id=data-and-tools
[31] http://wordnet.princeton.edu/
[32] http://babelnet.org/
[33] http://wndomains.fbk.eu/
[34] http://wndomains.fbk.eu/wnaffect.html
[35] http://sentiwordnet.isti.cnr.it/

Parameter	Importance	WordNet	BabelNet	WND & WNA	SentiWordNet
Generic Parameters					
Maturity & Stability	+++	+++	++	++	++
Regularly Updated	+/-	+	++	+	+
Technical Up-to-Datedness / Appeal	++	+++	+++	+++	+++
Open Source	+	N/A	N/A	N/A	N/A
Non-Infecting	+	YES	NO	YES	YES
Code Quality	++	N/A	N/A	N/A	N/A
Extensibility	+++	-	+++	-	-
Community	-	+	++	+	+
Performance / Scalability	+++	+++	++	+++	+++
Reuse of existing developments	+/-	+++	+++	+++	+++
EU project origin	---	NO	YES	NO	NO
Platform (Portability)	+	+	+	+	+
Open Standards Compliance	+	N/A	N/A	N/A	N/A
Interoperability (easy integration for all platforms)	++	+++	+++	+++	+++
Specific Parameters					
Semantic Analysis	+++	+	+++	+	-
Sentiment analysis	++	-	-	+	++

Figure 27: Parameter Evaluation of Technologies for Semantic Repositories – Lexical Databases

3.3.2.1.3.3 Knowledge Bases

- **DBpedia**[36] is a project aiming to extract structured content from the information created in Wikipedia. This structured information is known as a knowledge base that allows users to query relationships and properties associated with Wikipedia resources, including links to other related datasets
- **Freebase**[37] is similarly a collaborative knowledge base consisting of metadata composed mainly by its community members. It is an online collection of structured data harvested from many sources, including individual, user-submitted wiki contributions. DBpedia and Freebase are interconnected and complementary knowledge bases.
- **Wikidata**[38] is a collaboratively edited knowledge base operated by the Wikimedia Foundation. It is intended to provide a common source of certain data types (for example, birth dates) which can be used by Wikimedia projects such as Wikipedia

Parameter	Importance	DBpedia	Freebase	Wikidata
Generic Parameters				
Maturity & Stability	+++	+++	++	+
Regularly Updated	+/-	+++	++	++
Technical Up-to-Datedness / Appeal	++	+++	+++	+++

[36] http://www.dbpedia.org/
[37] https://www.freebase.com/
[38] http://www.wikidata.org/

Open Source	+	N/A	N/A	N/A
Non-Infecting	+	YES	YES	YES
Code Quality	++	N/A	N/A	N/A
Extensibility	+++	++	++	+
Community	-	+++	+++	+
Performance / Scalability	+++	+++	+++	++
Reuse of existing developments	+/-	-	-	-
EU project origin	---	NO	NO	NO
Platform (Portability)	+	+	+	+
Open Standards Compliance	+	+++	+	+
Interoperability (easy integration for all platforms)	++	+++	+++	+++
Specific Parameters				
Semantic Analysis	+++	++	+	+
Textual annotation	++	++	+	+
Semantic inference	++	++	+	+

Figure 28: Parameter Evaluation of Technologies for Semantic Repositories – Knowledge Bases

3.3.2.1.4 Machine Learning Tools

In order to combine NLP technologies with Artificial Intelligence, machine learning tools will be employed. In this manner several semantic, sentiment, and lexical features can be combined discovering unseen characteristics of the analysed text. The following tools have been analysed and compared in order to know which ones are suitable for the SAM project.

- **Weka**[39] is a collection of machine learning algorithms for data mining tasks. Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualisation. It is also well-suited for developing new machine learning schemes and can be applied to big data.
- **MALLET**[40] is a Java-based package for statistical natural language processing, document classification, clustering, topic modelling, information extraction, and other machine learning applications to text. In addition to sophisticated Machine Learning applications, MALLET includes routines for transforming text documents into numerical representations that can then be processed efficiently. This process is implemented through a flexible system of "pipes", which handle distinct tasks such as tokenizing strings, removing stopwords, and converting sequences into count vectors.
- **MATLAB**[41] (matrix laboratory) is a multi-paradigm numerical computing environment and fourth-generation programming language. Developed by MathWorks, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, and Fortran.
- **R**[42] is a free software programming language and software environment for statistical computing and graphics. The R language is widely used among statisticians and data

[39] http://sourceforge.net/projects/weka/
[40] http://mallet.cs.umass.edu/
[41] http://mathworks.com/
[42] http://www.r-project.org/

miners for developing statistical software and data analysis. R is an implementation of the S programming language combined with lexical scoping semantics inspired by Scheme.

Parameter	Importance	Weka	MALLET	MATLAB	R
Generic Parameters					
Maturity & Stability	+++	++	++	+++	++
Regularly Updated	+/-	++	++	++	++
Technical Up-to-Datedness / Appeal	++	+++	+++	+++	+++
Open Source	+	YES	YES	NO	YES
Non-Infecting	+	NO	YES	N/A	NO
Code Quality	++	N/A	N/A	N/A	N/A
Extensibility	+++	+++	+++	+	+
Community	-	++	+	+++	++
Performance / Scalability	+++	N/A	N/A	N/A	N/A
Reuse of existing developments	+/-	+	-	-	+++
EU project origin	---	NO	NO	NO	NO
Platform (Portability)	+	+++	+++	+++	+++
Open Standards Compliance	+	N/A	N/A	N/A	N/A
Interoperability (easy integration for all platforms)	++	++	++	++	++
Specific Parameters					
Semantic analysis	+	+	+	+	+
Sentiment analysis	+	+	+	+	+

Figure 29: Parameter Evaluation of Technologies for Machine Learning Tools

3.3.2.2 Technology Selection

After comparing the technologies in the previous section, the selected technologies for each functionality are given and motivated in this section.

3.3.2.2.1 **Natural Language Processing Tools**

Open NLP and Lucene were selected as NLP tools since these tools have suitable licenses, and furthermore they cover most of the NLP tasks necessary for dealing with the development of the Semantic Services.

By using Open NLP tasks such as tokenisation, sentence segmentation, part-of-speech tagging, named entity extraction, chunking, parsing, and co-reference resolution can be addressed. Moreover, this tool offers some machine learning functionalities very suitable for Semantic and Sentiment Analysis.

Lucene offers a very powerful engine for textual searching and indexing. Furthermore, the stemming and tokenisation functionalities provided by this tool will be taken into account.

Apache Commons Lang + Jexl were selected due to the fact that provides extra utilities for text matching and evaluation of logical expressions. These methods are necessary to identify relations between different contents in SAM (e.g., asset descriptions that match with specific user comments).

With respect to dealing with summarising issues, Compendium was selected. It is a system that considers the most novel summarisation techniques published at international relevant journals. This tool can be adjusted to the SAM's particularities if it is necessary since the source code is open for SAM consortium. It is important to remark that the POS-Tagger included into Compendium should be replaced by the one selected in the NLP tools section, since Tree-tagger (the POS-Tagger of Compendium) uses a GPL license.

3.3.2.2.2 **Semantic APIs**

For the task of querying the semantic data from the Cloud Storage, Sesame was chosen. Semantic client API and semantic database (see Section 3.8.2.2.2) are based on the same framework, so a total complementation between both exists. Also, the Sesame API allows exploiting the semantic information stored into DBpedia database (semantic repository chosen in Section 3.8.2.2.3), since this database is described into RDF ontology language.

3.3.2.2.3 **Semantic Repositories**

Different kinds of semantic repositories were chosen. In case of corpora, both Emotiblog and Semeval were selected. These corpora are annotated with opinion tags which are useful for training sentiment analysis models.

As part of lexical databases WordNet and its by-products were selected. WordNet is an English machine readable dictionary organised at sense level. WordNet provides WordNet Domains and WordNet Affect, which represent domain taxonomies that categorise WordNet senses. In terms of sentiment polarities SentiWordNet was selected since it links the WordNet senses with sentiment polarity scores. All of these resources constitute the basis knowledge for the characterisation and sentiment analysis processes.

With respect to the knowledge bases, DBpedia was selected. It constitutes a semantic network where the Wikipedia encyclopaedia information was transformed to a structured format. This structured information offers a large quantity of semantic data necessary for the characterisation process. It should be noticed that all these resources are licensed in a way that matches the SAM exploitation plan.

3.3.2.2.4 **Machine Learning Tools**

For machine learning, MALLET was selected. It is a tool highly adapted to NLP issues allowing the removal of stopwords and tokenisation of sentences. It provides functionalities such as document classification, clustering, topic modelling and information extraction, suitable for the Semantic Services.

Machine learning functionalities such as maximum entropy and perceptron provided by the selected technology Open NLP are taken into account. This tool has been already selected in Section 3.3.2.2.1 but also it involves functionalities suitable for machine learning.

3.3.2.3 User Interface Technology

This component has to provide the Asset Profiler user interface so the Content Provider is able to create, edit or delete the asset involved into the platform. The technology used for realisation has been selected in Section 3.1.

3.3.3 Technical Component Specification

3.3.3.1 Structure

Figure 30 provides an overview of the Semantic Services, which can be found also in D3.2.1 Global Architecture Definition. Additionally the version of the overview in this section has been augmented with the selected technologies.

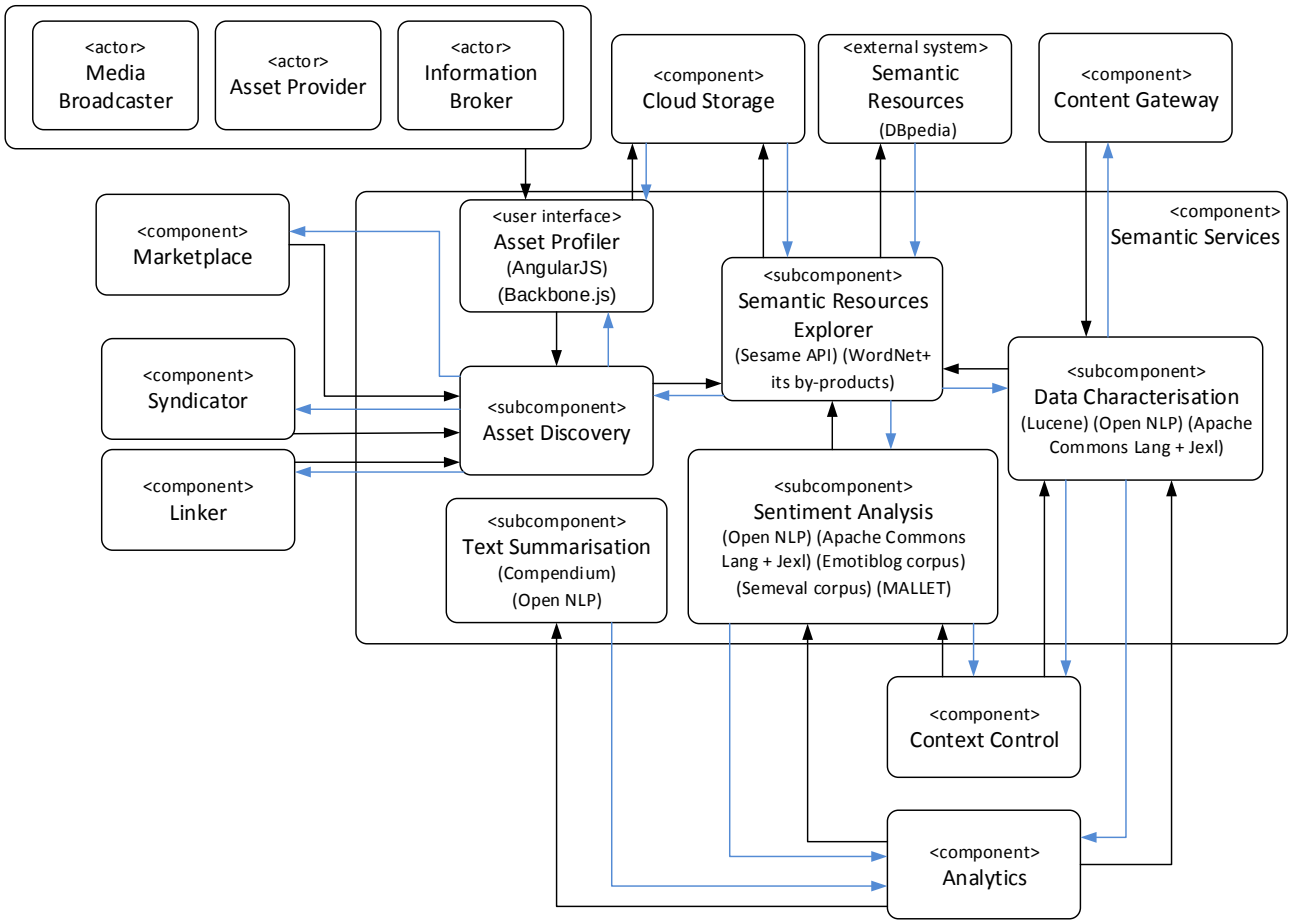

Figure 30: Overview with Selected Technologies – Semantic Services

3.3.4 Specification of Interfaces, Protocols and Formats

The functionalities of the Semantic Services will be provided as web methods of a SOAP Web Service (WS). As result, every functionality of the Semantic Services will provide JSON objects as a standard interchange data type.

3.3.4.1 Web Services Interfaces

In order to describe the interfaces, each web method of this service is shown in a separate table.

3.3.4.1.1 Methods “Characterise Asset” and “Characterise Content”

The “Characterise Content” functionality, part of the Semantic Services WS, is able to identify different semantic features involved in a given text. These features or semantic units are part of the semantic repositories included in the SAM platform. “Characterise

Content" presents two web methods, one for addressing asset characterisation and another for analysing a given User-Generated Comment (UGC).

Characterise Content – Asset Characterisation		
Description	This functionality is able to populate an asset with semantic units and references by means the characterisation process.	
Request		
Request URL	**POST** http://example.com/api/SemanticServices?wsdl	
Web Method	CharacteriseAsset	
	Name	Description

HTTP Parameters	asset
	Required object
		A JSON object structured following the asset data structure.
		Example:
		{
		"labelId": "a2"
		"title": "Casino Royale",
		"description": „Casino Royale is the twenty-first film in the Eon Productions James Bond film series and the first to star Daniel Craig as the fictional MI6 agent James Bond",
		"keywords": ["Casino Royale","film"],
		"resourceType": „ Asset",
		"assetType": "Films"
		"relatedLabelIds": ["",""],
		"relatedLabels": ["",""],
		"relatedLabelResources": ["",""],
		"relationType": ["",""]
		}

	paths Required string list	A string list that contains the asset attribute values for characterising. Example: *["description", "Title"]*
Response		
HTTP Status Code	Value	Description
	100	Successful operation
	200	Object not found
	300	Command waiting in queue
	500	Fatal error

JSON Attributes	list
	Optional asset list
		A list of JSON objects with assets. In case of no related data is found the list is empty.
		Example:
		[
		{
		“labelId”: “a1”,
		“title”: “Mads Mikkelsen”,
		“description”: “Mads D. Mikkelsen born 22 November 1965) is a Danish actor. Originally a gymnast and dancer, he began his career as an actor in 1996.”,
		“resourceType”: “Asset”,
		“relatedLabelIds": [“ wn1”,”db10”],
		“relatedLabels": [“Entertaiment”,”Actor”],
		“relatedLabelResources”: [“WordNet”,”DBpedia”],
		“relationType”: [“hypernym”,” hypernym”]
		},
		{
		“labelId”: “a2”,
		“title”: “Casino Royale”,
		“description”: “Casino Royale is the twenty-first film in the Eon Productions James Bond film series and the first to star Daniel Craig as the fictional MI6 agent James Bond”,
		“keywords”: [“Casino Royale”,”film”],
		“resourceType”: “Asset”,
		“assetType”: “Films”
		“relatedLabelIds”: [“wn50”,” db50”, “wn40”, “ass40”],
		“relatedLabels”: [“film”,” James Bond”, “star”, “Daniel Craig”],
		“relatedLabelResources”: [“WordNet”,”DBpedia”, “WordNet”,”Asset”],
		“relationType” : [“is_a”,”part_of”,”is_a”, “undefined”],
		}
		]

Figure 31: Web Service Interface Description – Characterise Content – Asset Characterisation

Characterise Content – UGC Characterisation		
Description	This functionality is able to identify and retrieve the Semantic information related with the analysed textual content.	
Request		
Request URL	**POST** http://example.com/api/SemanticServices?wsdl	
Web Method	CharacteriseContent	
	Name	Description

HTTP Parameters	UGCs
	Required text list
		A list of text for applying semantic characterisation.
		Example:
		[
		{
		"UGC": "Casino Royale is a great movie. And a PHENOMENAL Bond movie. I don't care what any of you have to say! #DanielCraig #007 #Bond #Poker"
		}
		]

	Contexts Required string list	A string list that contains some keywords related to the context where the user comment was generated. Example:*["film", "actor", "entertainment"]*
Response		
HTTP Status Code	Value	Description
	100	Successful operation
	200	Object not found
	300	Command waiting in queue
	500	Fatal error
JSON Attributes	list Optional assets list	A list of JSON objects with assets. In case of no related data is found the list is empty. Example: [{ "labelId": "a1",

		“title”: “Mads Mikkelsen”,
		“description”: “Mads D. Mikkelsen born 22 November 1965) is a Danish actor. Originally a gymnast and dancer, he began his career as an actor in 1996.”,
		”resourceType”: “Asset”,
		“relatedLabelIds”: [“ wn1”,”db10”],
		“relatedLabels”: [“Entertainment”, ”Actor”],
		“relatedLabelResources”: [“WordNet”,”DBpedia”],
		“relationType”: [“hypernym”,” hypernym”]
		},
		{
		“labelId”: “a2”,
		“title”: "Casino Royale”,
		“description”: “Casino Royale is the twenty-first film in the Eon Productions James Bond film series and the first to star Daniel Craig as the fictional MI6 agent James Bond”,
		"resourceType": " Asset",
		"relatedLabelIds": [“wn50”,” db50”, “wn40”, “ass40”],
		"relatedLabels": [“film”,” James Bond”, “star”, “Daniel Craig”],
		"relatedLabelResources": [“WordNet”,”DBpedia”, “WordNet”,”Asset”],
		"relationType": [“is_a”,”part_of”,”is_a”, “undefined”]
		}
		]

Figure 32: Web Service Interface Description – Characterising Content- UGC Characterisation

3.3.4.1.2 **Method “Discover Assets”**

The web method “Discover Assets”, which belongs to the Semantic Services WS, finds asset information.

Discover Assets	
Description	Retrieves a collection of assets according to the indicated sources.
Request	

Request URL	**POST** http://example.com/api/SemanticServices?wsdl	
Web Method	DiscoverAssets	
	Name	Description

HTTP Parameters	keywords
	Required object list
		A JSON object represents an object of the specified data type which includes a collection of strings. They are the input words for the exploration.
		Example:
		[
		{
		"keyword": "actor"
		},
		{
		"keyword": "film"
		},
		{
		"keyword": "Mads Mikkelsen"
		}
		]

	Contexts Required string list	A string list that contains some keywords related to the context where normally the target assets are used. These context keywords allow refining the search. Example: [“action”, “entertainment”, "science fiction"]
Response		
HTTP Status Code	Value	Description
	100	Successful operation
	200	Object not found
	300	Command waiting in queue
	500	Fatal error
JSON Attributes	list Optional list	A list of JSON objects with semantic attributes. In case of no related data is found the list is empty. Example: [

```
 {
  “labelId”: “a1”,
  “title”: “Mads Mikkelsen”,
  “description”: “Mads D. Mikkelsen born 22 November 1965) is a Danish actor. Originally a gymnast and dancer, he began his career as an actor in 1996.”,
  “resourceType”: “Asset”,
  “assetType”:”Actor”,
  “relatedLabels”: [“ Entertainment”, ”Actor”],
  “relatedLabelIds”: [“wn1”,”db10”],
  “relatedLabelResources”: [“WordNet”,”DBpedia”],
  “relationType": [“hypernym”,” hypernym”]
 },
 {
  „labelId“: "a2",
  "title": "Casino Royale",
  “description”: “Casino Royale is the twenty-first film in the Eon Productions James Bond film series and the first to star Daniel Craig as the fictional MI6 agent James Bond”,
  "resourceType": " Asset",
  “assetType”: “Films”,
  “semanticFeatures” :
  {
   "relatedLabelIds": [“wn50”,” db50”, “wn40”, “ass40”],
   "relatedLabels": [“film”,” James Bond”, “star”, “Daniel Craig”],
   "relatedLabelResources": [“WordNet”,”DBpedia”, “WordNet”,”Asset”],
   "relationType": [“is_a”,”part_of”,”is_a”, “undefined”]
  }
 }
]
```


Figure 33: Web Service Interface Description – Discovery Assets

3.3.4.1.3 **Method "Analyse Sentiment"**

The web method "Analyse Sentiment", part of the Semantic Services WS, is able to identify different sentiment analysis features involved into a given UGC.

Analyse Sentiment		
Description	This functionality retrieves sentiment analysis features regarded to given UGCs.	
Request		
Request URL	**POST** http://example.com/api/SemanticServices?wsdl	
Web Method	AnalyseSentiment	
	Name	Description

HTTP Parameters	UGCs
	Required text list
		A list of User-Generated Content to be processed by the sentiment analysis algorithms.
		Example:
		[
		{
		"UGC": "Casino Royale is a great movie. And a PHENOMENAL Bond movie. I don't care what any of you have to say! #DanielCraig #007 #Bond #Poker"
		}
		]

	Contexts Optional string list	A string list that contains some keywords related to the context where the user comment was generated. Example: *["film", "actor", "entertainment"]*

	subjectList
	Optional list of text
		A list of subjects to evaluate with respect to the UGCs.
		Example:
		{
		[
		"Casino_Royale", "James_Bond"
		]
		}

Response		
HTTP Status Code	Value	Description
	100	Successful operation

	200	Object not found
	300	Command waiting in queue
	500	Fatal error

JSON Attributes	list
	Optional list
		A list of JSON objects with semantic attributes. In case of the attribute "subject" of this output, a "OVERALL" value will be set for the sentiment analysis evaluation of the entire comment, and additional evaluations for every "subject".
		Example:
		[
		{
		"subject": "OVERALL",
		“positiveIntensity”: “0.75”,
		“negativeIntensity”: “0.15”,
		“emotion labels”: [“joy”],
		“sentimentCategory”: “Positive”
		},
		{
		"subject": "Mads_Mikkelsen",
		“positiveIntensity”: “0.75”,
		“negativeIntensity”: “0.15”,
		“emotionLabels”: [“joy”],
		“sentimentCategory”: “Positive“
		},
		{
		"subject": "James_Bond",
		“positiveIntensity”: “1.0”,
		“negativeIntensity”: “0.0”,
		“emotionLabels”: [“joy”],
		“sentimentCategory”: “Positive”
		}
		]

Figure 34: Web Service Interface Description – Analyse Sentiment

3.3.4.1.4 **Method "Summarise Text"**

The web method "Summarise Text", included in the Semantic Services WS, is able to summarise UGC.

Summarise Text		
Description	This functionality allows summarise large quantity of incoming UGC, remaining the most representative information of them.	
Request		
Request URL	**POST** http://example.com/api/SemanticServices?wsdl	
Web Method	SummariseText	
	Name	Description

HTTP Parameters	UGC
	Required text
		A large text (normally UGCs) for summarising.
		Example:
		{
		"UGC":
		"Casino Royale is a great movie. And a PHENOMENAL Bond movie. I don't care what any of you have to say! #DanielCraig #007 #Bond #Poker.
		Some of the recent James Bond themes are about not trusting anyone. I wonder why they've focused on that. Casino Royale + Quantum of Solace.
		You Know My Name - From James Bond - Casino Royale – Movie Sounds Unlimited http://spoti.fi/1ri9Cn2 #NowPlaying"
		}

	Compression Ratio Required numeric value	A numeric value that indicates the reduction of the output text. Example: *{"compressionRatio": "20"}*
Response		
HTTP Status Code	Value	Description
	100	Successful operation
	200	Object not found
	300	Command waiting in queue
	500	Fatal error
JSON Attributes	Summary Optional string	A string that represents the reduced UGCs. Example:

		{
		“Summary”: “Casino Royale is a great movie.
		Some of the recent James Bond themes are about not trusting anyone.
		You Know My Name - From James Bond - Casino Royale“
		}

Figure 35: Web Service Interface Description – Summarise Text

3.3.4.2 Java Interfaces

The Java interfaces act as wrappers for the SOAP interfaces. They provide a convenient way to access the Semantic Services for developers.

3.3.4.2.1 Method “Characterise Asset” and “Characterise Content”

For applying data characterisation, the API provides two web methods. Both methods require two parameters for conducting the characterisation functionalities.

Characterise Content - Asset Characterisation	
Method Header	public String CharacteriseAsset(String anAsset, String paths)
Parameters	anAsset : A JSON object which represents an asset that should be characterised paths: A JSON object which refers the a list of string attributes to be characterised.
Return Value	A JSON object which refers at the incoming asset. This asset has been updated filling the semantic fields according to a characterisation process which takes as analysis target the asset texts field pointed by the input parameter's paths.
Error Handling	In case of an error, a null-value is returned
Remarks	None

Figure 36: Java Interface – Asset Characterisation

In Figure 37 the header of the method is shown.

```
/**
     * Characterise an asset
     * @param anAsset   asset to characterise, in JSON format
     * @param listOfPaths      specific sections of the asset to be characterised, in
JSON format
     * @return  the incoming asset annotated with semantic features related to the
analysed Content, in JSON format
     */
    @WebMethod(operationName = "characteriseAsset")
    public String characteriseAsset(@WebParam(name = "anAsset") String anAsset,
@WebParam(name = "listOfPaths")String listOfPaths) {
        ...
```

```
        return anAsset;
    }
```

Figure 37: Source Code Example – API Method Signature for Characterise Asset

If the API call is successful an asset object will be returned with its semantic attribute filled, otherwise it returns null.

Characterise Content - UGC Characterisation	
Method Header	public String CharacteriseContent(String UGC, String context)
Parameters	UGCs : A JSON object which represents a list of UGCs which should be characterised context: A JSON object which represents a collection of keywords of the context where the user comment was generated.
Return Value	A list of assets.
Error Handling	In case of an error, a null-value is returned
Remarks	None

Figure 38 Java Interface – UGC Characterisation

In Figure 39 the header of the method is shown.

```
    /**
     * characterise User-Generated Content (UGC)
     * @param UGC   phrases, sentences, and metadata labels such as keywords which
     * describe an asset, where named entities, events, etc. are involved (in JSON
     * format)
     * @param listOfContext Collection of context information, in JSON format
     * @return a list of assets, in JSON format
     */
    @WebMethod(operationName = "CharacteriseContent")
    public String characteriseContent(@WebParam(name = "UGC") String UGC,
       @WebParam(name = "listOfContext") String listOfContext) {
       ...
       return assetsList;
    }
```

Figure 39: Source Code Example – API Method Signature for Characterise UGC

If the API call is successful a list of asset objects will be returned with its semantic attributed filled, otherwise it returns null.

3.3.4.2.2 Method “Discover Assets”

For applying asset discovery, the API provides one web method. The method has two parameters.

Discover Asset	
Method Header	public String DiscoverAssets(String keywords, String context)
Parameters	keywords: A JSON object which represents a list of keywords with string format for discovering the assets semantically related with

	them. context: A JSON object which represents a collection of keywords of the context where normally the target assets are used.
Return Value	A JSON object which represents a list of the assets that match or are semantically similar with the incoming keywords.
Error Handling	In case of an error, a null-value is returned
Remarks	None

Figure 40: Java Interface – Discover Asset

In Figure 41 the header of the method is shown.

```
    /**
     * Suggest assets from given keywords
     * @param keywords list of keywords, in JSON format
     * @param context list of context keywords, in JSON format
     * @return a list of suggest assets from given keywords, in JSON format
     */
    @WebMethod(operationName = "DiscoverAssets")
    public String DiscoverAssets(@WebParam(name = "keywords") String keywords,
@WebParam(name = "context") String context) {
        ...
        return assetsList;
```

Figure 41: Source Code Example – API Method Signature for Discovering Asset

If the API call is successful a list of asset objects will be returned, otherwise it returns null.

3.3.4.2.3 Method “Analyse Sentiment”

For applying Sentiment Analysis, the API provides one method, which requires three parameters.

Analyse Sentiment	
Method Header	public String AnalyseSentiment(String UGCs, String context, String subjectList)
Parameters	UGCs: A JSON object that represents a list of text generated by users from which will be extracted sentiment analysis features. context: A JSON object that represents a collection of keywords which identify the context where the UGC was generated. subjectList: A JSON object that represents a list of subjects to evaluate with respect to the UGCs.
Return Value	A JSON object that represents different types of sentiment features. These are sentiment polarity, intensity of the polarities and emotion labels.
Error Handling	In case of an error, a null-value is returned
Remarks	None

Figure 42: Java Interface – Analyse Sentiment

In Figure 43 the header of the method is shown.

```
    /**
     * This functionality retrieves sentiment analysis features regarded to given UGCs
     *
     * @param UGC   Phrases, sentences, and metadata labels such as keywords which
     * describe an asset, where named entities, events, etc. are involved
     * @param context   Collection of context information, in JSON format
     * @param subjectList   Collection of subjects to evaluate with respect to the UGCs,
     * in JSON format
     * @return A list of JSON objects with sentiment analysis information, with the
     * following features:
     * - Subject (i.e. Mads Mikkelsen)
     * - Sentiment polarities (i.e., labels that represent positive, negative, objective
     *   and neutral sentiment classes).
     * - Sentiment intensity scores (i.e., collection of float numbers that represent the
     *   score value of the corresponding sentiment polarities).
     * - Emotion labels (i.e., collection of words concerning emotion labels such as
     *   “fear”, “sad”, “joy”, etc.)
     */
    @WebMethod(operationName = "AnalyseSentiment")
    public String AnalyseSentiment(@WebParam(name = "UGC") String UGC, @WebParam(name =
"listOfContext") String  context, @WebParam(name = "subjectList") String  subjectList) {
        ...
        return listOfSentimentAnalysisInformation;
    }
```

Figure 43: Source Code Example – API Method Signature for Analyse Sentiment

If the API call is successful a sentiment analysis object will be returned with its attributes filled, otherwise it returns null.

3.3.4.2.4 Method “Summarise Text”

For applying Text Summarisation, the API provides one web method, which requires two parameters.

Summarise Text	
Method Header	public string Summarise(string UGC, int compressionRatio)
Parameters	UGC: A JSON object that represents a text generated by users from which will be summarised. compressionRatio: An integer value for establishing the percentage that the UGC will be reduced.
Return Value	The UGC reduced
Error Handling	In case of an error, a null-value is returned
Remarks	None

Figure 44: Java Interface – Summarise Text

In Figure 45 the header of the method is shown.

```
    /**
     * Summarisation process
     * @param UGC              Text to summarise
     * @param compressRatio    a number which indicates the desired reduction percentage of
     * the text
     * @return Summarised UGC
     */
    @WebMethod(operationName = "Summarise")
    public String Summarise(@WebParam(name = "UGC") String UGC,@WebParam(name =
```

```
    "compressRatio") int compressRatio) {
        ...
        return summariserUGC;
    }
```

Figure 45: Source Code Example – API Method Signature for Summarise text

If the API call is successful a string will be returned, otherwise it returns null.

3.3.4.3 Content Formats

3.3.4.3.1 Java Data Schema

This section lists the Java data schemas which will be used by the Semantic Services information infrastructure internally as well as for the communication with other components via Web Services.

3.3.4.3.1.1 Semantic Data Objects

All Semantic Data Objects (SDO) will be used for the transfer of structured data for the Semantic Services (both internally and with other components). The list of objects described in this section is not complete and will be subject to changes in order to accommodate the special needs of external components which will use the Semantic Services.

Semantic Services will transfer SDO as JSON objects, which are structured based the following types of Java data schemas. Asset transferring, sentiment analysis transferring, and summaries transferring are shown in Figure 46, Figure 47 and Figure 48 respectively.

```
public class Asset {

    public enum AssetType { FILMS}
    public enum Resource { WORDNET, DBPEDIA, ASSET}
    public enum Relation {IS_A, PART_OF, UNDEFINED}

    private String labelid;
    private String title;
    private String description;
    private List<String> keywords;
    private String resourceTypes;
    private AssetType assetType;
    private SemanticFeatures semanticFeatures;

    public class SemanticFeatures {
        private List<String> relatedLabelIds;
        private List<Resource> relatedLabelResources;
        private List<Relation> relationType;
    }
```

Figure 46: Java Data Schema – Asset

```
public class SentimentFeatures {
    private enum Sentiment {POSITIVE, NEGATIVE}
    private double positiveIntensity;
    private double negativeIntensity;
    private List<String> emotionLabels;
    private Sentiment sentiment;
}
```

Figure 47: Java Data Schema – Sentiment Analysis Features

```
public class Summary {
    private String summary;
}
```

Figure 48: Java Data Schema – Summary

3.3.5 Summary

In this section, the Semantic Services component has been technically specified. During the technology selection (Section 3.3.2.2), the following tools were selected (organised by categories).

- **NLP tools:** The selected NLP tools should be able to apply the basic language processing functionalities for addressing SAM necessities. Open NLP and Lucene were selected since the Open NLP allows using different NLP functionalities such as tokenisation, sentence segmentation, part-of-speech tagging, named entity extraction, chunking, parsing, and co-reference. These functionalities provide support to the following basic parameters: syntactic analysis, semantic analysis, textual annotation and sentiment analysis.
  Lucene is a powerful engine for textual searching and indexing which is very necessary for semantic and sentiment analysis. Furthermore, Apache Commons Lang + Jexl which provide extra methods for text matching and evaluation of logical expressions were selected. These libraries are necessary for aligning content inputs with the SAM knowledge. For addressing the summarisation task, Compendium was selected. This summarisation system involves the most novel summarisation techniques nowadays. Its functionalities can be adapted to SAM scenarios since its source code is available for the SAM consortium.
- **Semantic APIs:** The chosen Semantic APIs should be able to consume the semantic data involved in the SAM platform, and also provide a summary function for reducing large quantity of text generated into the platform. In order to query the semantic data from the Cloud Storage, Sesame client tool was chosen, since it supports the expected semantic API skills and share the same framework with the semantic database engine.
- **Semantic Repositories:** The selected Semantic Repositories should have enough information for supporting the semantic and sentiment analysis processes. Emotiblog and Semeval were selected since they involve many examples of sentences and words annotated with opinion tags. WordNet and its by-products WordNet Domains , WordNet Affect, and SentiWordNet, were also selected because they allow obtaining lexical units with their respective domains, emotions and sentimental scores. DBpedia was also chosen as semantic network due to the fact that it is strongly related with Wikipedia and also that is a huge knowledge base annotated with the semantic data format, RDF. By using RDF, different kinds of semantic inferences can be applied.
- **Machine Learning Tools:** The selected Machine Learning tools should be able to provide the the SAM platform with intelligent automatic learning functionalities. MALLET, which is highly adapted to NLP tasks, as well as Open NLP were selected since they provide the necessary learning functionalities.

In this section the interfaces of the Semantic Services component have been described with the aim of describing the characterisation, sentiment analysis, asset discovery and summarisation functionalities. As a result of these functionalities, JSON objects will be transferred where all necessary data will be included in a simple structure of understanding.

3.4 Social Components

The Social Components provide an interface to connect the SAM Platform with social networks (see Figure 49). This component connects SAM users with each other. User-Generated Content is important for building social communities and recommendations. Statements in social networks from SAM users are processed, analysed and forwarded in the Social Components, to connect people with same interests. The main tasks of the Social Components are presented in the following list:

- The Social Components will provide Social Media content by querying different social networks
- The Social Components will enable the sharing of user-generated content such as comments and ratings, both for the SAM platform and different social networks
- The Social Components will enable the usage of social information by other SAM components for Social Mining and Business Intelligence purposes

The Social Components component satisfies the requirement to perform actions in chosen social networks using the SAM End User's credentials. The provided connection to social networks complies with the login requirements from the respective social network. The credentials from the social network are saved in the Cloud Storage like every User-Generated Content for later accesses. Furthermore recent responses from social networks are stored in the local cache, to have faster access for repeated requests.

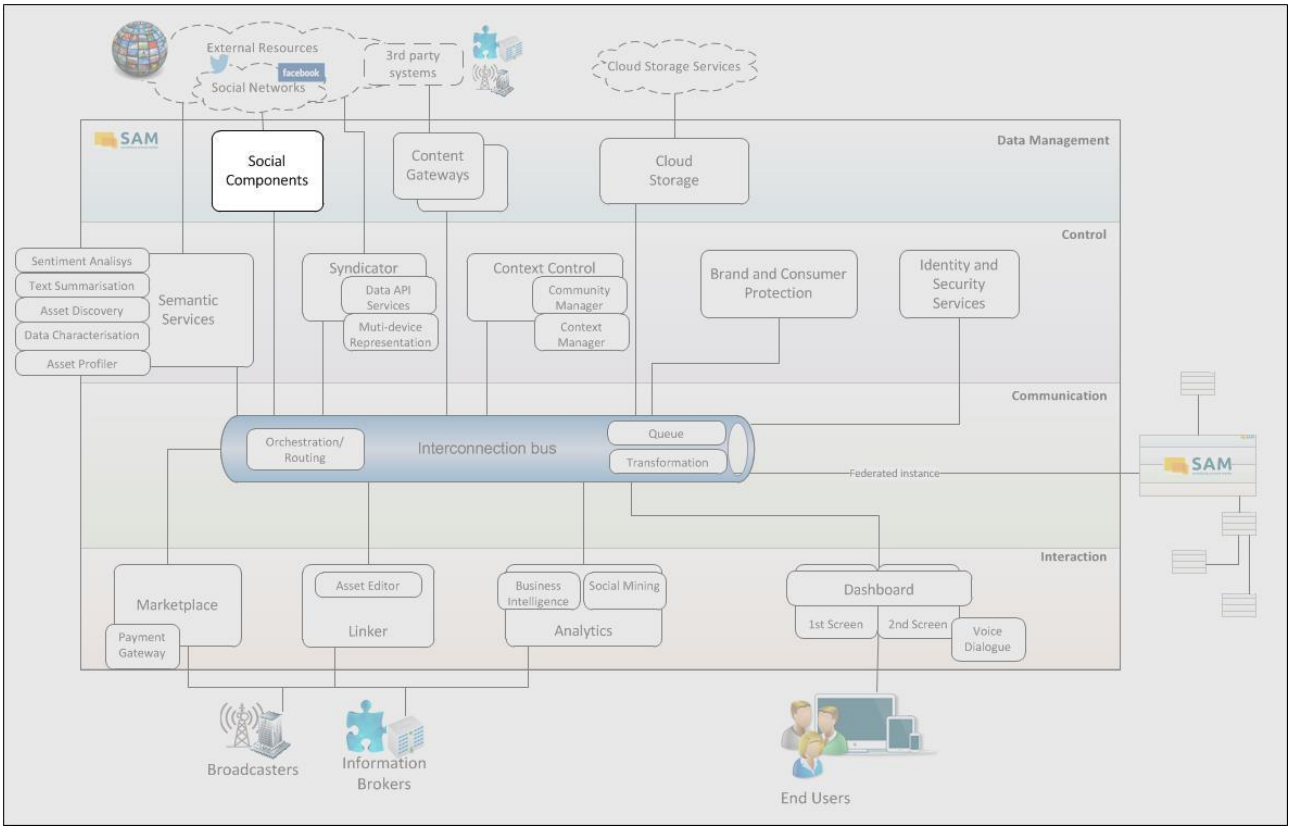

Figure 49: Architecture Overview – Social Components

3.4.1 Major Design Decisions

The focus of the Social Components is forwarding Social Media content from SAM to social networks and vice versa.

Several activities must be considered while performing these actions. Firstly it is important to handle the authentication of each social network provider. Therefore the Access Service is defined to manage the authentication. The Access Service complies with the guidelines from the social networks and is able to use different authentication methods. The biggest social networks like Twitter, Facebook, Google+ and Instagram are using OAUTH and OAUTH2.0 as authentication method. To perform actions (e.g. post a message, receive a message) it is necessary to have the access token from the respective social network.

Once the access token is available, it will be stored in the cloud storage.

User-Generated content is getting forwarded from the Generic Dashboard, where it is generated, to the Social Network API, the Context Control and the Cloud Storage. The following core subcomponents of the Social Components component allow to process Social Media content internally.

- Social Components Service
- Social Interaction Logic
- Social Feedback Mining
- Social Network Communication

The Social Components Service provides an interface to forward user-generated content to the Social Interaction Logic. The Interface provides functionalities to submit content and request content from the social network. To retrieve information from the social networks, at first the Local Cache Information will be queried for existing results, after that the social network will be queried if no satisfying results are found in the Local Cache.

The Social Interaction Logic distributes the generated content. Firstly the data will be forwarded to the Social Feedback Mining and secondly to the Social Network Communication which is the most important internal subcomponent. It communicate constantly with the external components Cloud Storage, Context Control and the different Social Networks APIs.

The implementation of the Social Network API is based on native APIs from the respective social networks. The SAM Platform doesn´t use multiple third-party APIs for the following reasons:

- **Update time**: Often, it takes too long to get an updated third-party API, if the original social network API is updated.
- **Implementation and maintenance effort**: It is not reasonable and produces a lot of overhead to adapt several different third-party API of social networks to the SAM Platform.
- **Documentation:** Third-party APIs are often not thoroughly and completely documented.

To not handle several, different APIs it is planned to use a social network framework, which uses the native social network APIs and coordinates the requests for different social networks in a unified way.

3.4.2 Technology Comparison and Selection

As mentioned in Section 3.4.1, a social network framework will be used to communicate with several different social networks. There are several providers of such frameworks. In the following Section 3.4.2.1 possible technologies are explained and compared and in Section 3.4.2.2 one of the possible technologies is selected and substantiated.

3.4.2.1 Possible Technologies and Comparison

The internal core component Social Network Communication is accessed by three external Components:

- Cloud Storage,
- Context Control and
- Social Network API

The communication with the Cloud Storage and the Context Control proceeds via the Interconnection Bus. The communication with the different social networks proceeds via the external resources of the social networks. To reduce the amount of work with several Social Network APIs, a framework will be used which provides a unified access to several different Social Network APIs. The following frameworks represent the most appropriate choices:

- **SocIoS**[43] presents a framework for application developers to combine content and services from a wide range of social networks. SocIoS provides a SOA infrastructure that will act as a virtualisation layer on top of social networking containers, an API that will grant a single point access to the underlying functionality of social networks and a toolset for third-party services support, for more effectively delivering applications that exploit the User-Generated Content and social graph. Also SocIoS was developed within an EU Project where two partners of the SAM consortium were involved.
- **OpenSocial**[44] is a set of APIs for building social applications that run on the web. OpenSocial is providing a common API that can be used in many different contexts. Developers can create applications, using standard JavaScript and HTML that run on social websites that have implemented the OpenSocial APIs.
- **GNIP**[45] was acquired by Twitter in 2010. It is one of the biggest providers for social content. It provides access to several social network APIs and it supports a wide range of social networks and sends social network content to the consumer application as soon as it is posted into the social network. Of course it is also able to request specific historical data at any time.
- **Native API Access** means to develop a framework to access several different social network APIs. This approach ensures that the framework is up-to-date and corresponds to the defined requirements.

Parameter	Importance	SOCIOS	Open Social	GNIP	Native API Access
Generic Parameters					
Maturity & Stability	++	++	+++	+++	+++
Regularly Updated	+++	++	++	++	++
Technical Up-to-Datedness / Appeal	++	++	++	++	+
Open Source	+/-	YES	NO	YES	YES
Non-Infecting	++				
Code Quality	+	+	N/A	N/A	N/A
Extensibility	+++	+++	---	---	+++
Community	+/-	++	+	+++	+++
Performance / Scalability	+	++	+	+	N/A

[43] http://www.sociosproject.eu/
[44] http://opensocial.org/
[45] http://gnip.com/

Reuse of existing developments	+	++	-	-	-
EU project origin	+/-	YES	NO	NO	NO
Platform (Portability)	---	++	N/A	N/A	++
Open Standards Compliance	+	Apache 2.0	Apache 2.0	NO	Apache 2.0
Interoperability (easy integration for all platforms)	++	+++	+++	+++	+
Specific Parameters					
Multiple SNS support	++	+++	+++	+++	++
Native SNS access	+++	+++	++	++	++
Authentication support	+/-	+++	+	+++	++
Application authentication on SNSs	+++	++	+	++	++

Figure 50: Parameter Evaluation of Technologies for Social Network Frameworks

Furthermore, a storage technology for the Local Information Cache must be selected to get fast and secure access to the data. The storage should be independent, so that no connection to another component is needed. The comparison of those technologies can be found at 3.8.2.1.1.

3.4.2.2 Technology Selection

SocIoS will be used to access the several external social network APIs. Because of the involvement of two partners of this Consortium, the source code of SocIoS is available. Thereby it is possible for the consortium to extend the framework in the way that it is needed. Furthermore all specific parameters of Figure 50 are satisfied. In contrast to building a custom implementation with all necessary native social network APIs, using an existing framework will save a lot of time and effort, especially if the source code is available. The other possibility to get unified access for several social networks is to build a native framework from the scratch. This is rejected because of a huge implementation and maintenance effort.

MongoDB has been selected as the storage solution for the Local Information Cache. Section 3.8.2.2.1 provides more information on this decision.

3.4.3 Technical Component Specification

3.4.3.1 Structure

Figure 51 provides an overview of the Social Components, which can be found also in D3.2.1 Global Architecture Definition. Communication Interfaces for external components can be found at Social Components and Social Network Communication. Another important internal component of the Social Components is the Local Information Cache, which reduces the amount of requests to social networks and to speed up the response time of the Social Components.

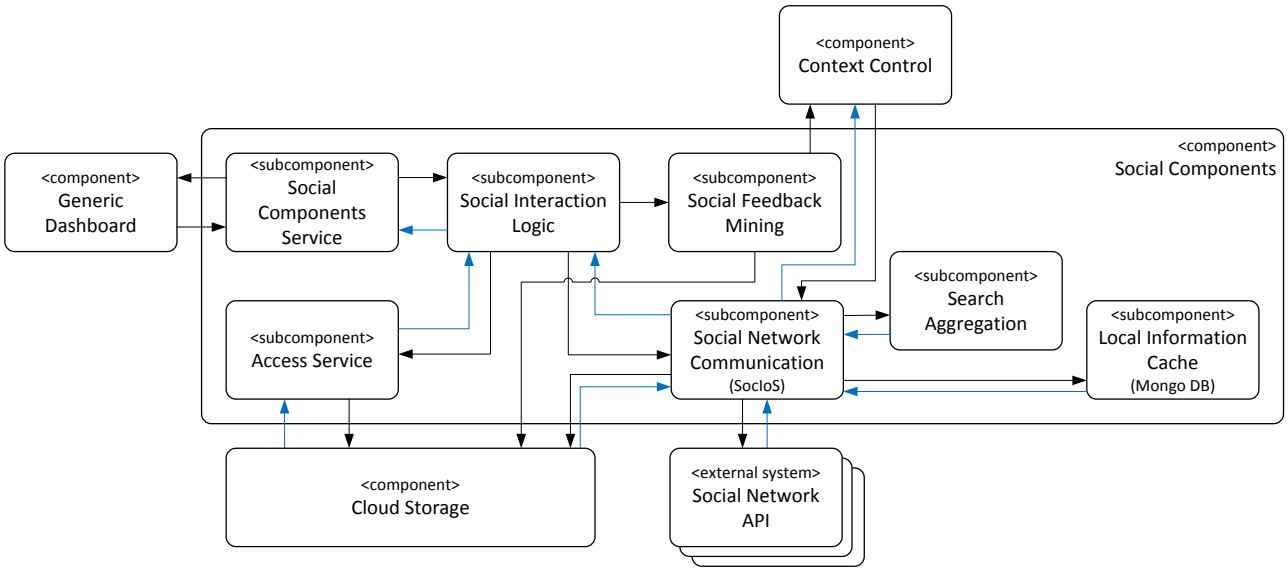

Figure 51: Social Components Structure

3.4.3.1.1 Local Information Cache

To reduce the requests to external social networks, a Local Information Cache stores recently requested data. The options for implementing the Local Information Cache are either use a structured data storage or a semi-structured data storage. Due to the fact that responses from various social networks can be different, the data is most likely semi-structured. Since NoSQL databases manage semi-structured data without a fixed data schema, these databases can be used to store such data in a document-oriented way.

SQL databases are excluded because they are designed to store data in a certain scheme. Though different social network responses are unstructured data, so they cannot be stored in SQL databases.

The comparison and selection of the used NoSQL database are equal to the Cloud Storage technology and is located in Figure 95 on page 129.

3.4.4 Specification of Interfaces, Protocols and Formats

The services of the Social Components will be provided as RESTful interfaces (described in Section 3.4.4.1).

3.4.4.1 RESTful Interfaces

In order to describe the RESTful interface, some of the provided services are described in a separate table. This table is followed by a listing showing an example for the JSON parameter (if applicable) as well as an example for the return value (if applicable).

In order for the 2nd Screen to communicate with the Social Components component, the Social Components component provides RESTful interfaces to forward the social events to the internal logic component.

3.4.4.1.1 Social Components Service

This component enables other SAM components to communicate with extern social networks. It provides a RESTful interface to request and submit User-Generated Content.

For every request or submit the login credentials, respectively the access token of the authenticated user, is used to authenticate at the social networks.

3.4.4.1.1.1 Method “Find User”

The RESTful interface “findUser” (Figure 52) gets a list of user, which matches the provided username. This list can then be used to select a specific user.

Find User		
Description	This interface will help to retrieve user information based on a provided username.	
Request		
Request URL	**GET** http://example.com/api/sc/useSN:socialNetwork/username:name	
Resource Parameters	Name	Description
	name Required string	Message of the Status. Example: *“Fred”*
	socialNetwork Required String	The social network target Example: “*Twitter”*
Response		
HTTP Status Code	Value	Description
	200	Status created
	401	Not authorised – wrong credentials
	409	Error in further processes
	502	Database error
JSON Attributes	list Optional User List	A list of JSON objects with Users. Example: [{ “userId” : “1230” “screenName” : ”exampleFred”, “name” : ”Fred Example”, “birthdate” : ”01.01.2014”, “location” : ”Berlin” “description” : ”I am an open minded person, who is interested in sports”, “profileImageURL” : “http://example.com/fred.jpg” “friendsCount” : 239, },

		{
		“username” : ”ferdinandFred”,
		“firstName” : ”Fred”,
		“lastName” : ”Ferdinand”,
		“birthdate” : ”03.03.2014”,
		“location ” : ”London”
		“description” : ”I am an open minded person, who is interested in culture”,
		“profileImageURL” : “http://example.com/fred2341.jpg”
		“friendsCount” : 21,
		}
		]

Figure 52: RESTful Interface Description – Find User

3.4.4.1.1.2 Method “Get User Profile”

The RESTful interface “getUserProfile” (Figure 53), gets a specific user profile by unique userID.

Get User Profile		
Description	Returns the user profile of the provided user ID.	
Request		
Request URL	**GET** http://example.com/api/sc/useSN:socialNetwork/userId:userId	
Resource Parameters	Name	Description
	userId Required integer	ID of the User Example: *“1230”*
	socialNetwork Required String	The social network target Example: “*Twitter”*
Response		
HTTP Status Code	Value	Description
	200	Status created
	401	Not authorised – wrong credentials
	409	Error in further processes
	502	Database error
JSON Attributes	list Optional User List	A list of JSON objects with Users. Example:

		{
		“userId” : “1230”
		“screenName” : ”exampleFred”,
		“name” : ”Fred Example”,
		“birthdate” : ”01.01.2014”,
		“location” : ”Berlin”
		“description” : ”I am an open minded person, who is interested in sports”,
		“profileImageURL” : “http://example.com/fred.jpg”
		“friendsCount” : 239,
		}

Figure 53: RESTful Interface Description – Get User Profile

3.4.4.1.1.3 Method “Create Status”

The RESTful interface “Create Status” (Figure 54), creates a new Status for the user. A Status is a User-Generated Content, which appears in the timeline of the logged in user.

Create Status		
Description	Creates a new status entry which appears in the timeline of the logged in user.	
Request		
Request URL	**PUT** http://example.com/api/sc/useSN:socialNetwork/createStatus:message	
Resource Parameters	Name	Description
	message Required string	Message of the Status. Example: “*SAM is awesome!”*
	socialNetwork Required String	The social network target Example: “*Twitter”*
Response		
HTTP Status Code	Value	Description
	200	Status created
	401	Not authorised – wrong credentials
	409	Error in further processes
	502	Database error

JSON Attributes	Status
	Optional Status Object
		The status which was posted
		Example:
		{
		“statusId” : “1234”,
		“creationDate” : “01.01.2014”,
		“statusText” : “SAM is awesome!”,
		“source” : “Twitter”,
		“geoLocation” : “Berlin, Germany”,
		“language” : “English”,
		“retweetCount” : 0,
		“favoriteCount” : 0
		}

Figure 54: RESTful Interface Description – Create Status

3.4.4.1.1.4 Method “Create Comment”

The RESTful interface “Create Comment” (Figure 55), creates a new Comment of a Status for the user. A Comment is a reply to an existing status.

Create Comment		
Description	Creates a new comment entry which appears in the comments of a status of the logged in user.	
Request		
Request URL	**PUT** http://example.com/api/sc/useSN:socialNetwork/createComment:comment	
Resource Parameters	Name	Description
	comment Required string	Message of the comment. Example: “*SAM is awesome!”*
	socialNetwork Required String	The social network target Example: “*Twitter”*
Response		
HTTP Status Code	Value	Description
	200	Rank was successful
	401	Not authorised – wrong credentials
	409	Error in further processes
	502	Database error
JSON	Status	The comment which was posted, The response may differ from social network to

Attributes	Optional Status Object	social network.
		Example:
		{
		“statusId” : “1234”,
		“creationDate” : “01.01.2014”,
		“statusText” : “SAM is awesome!”,
		“source” : “Twitter”,
		“geoLocation” : “Berlin, Germany”,
		“language” : “English”,
		“retweetCount” : 0,
		“favoriteCount” : 0
		}

Figure 55: RESTful Interface Description – Create Comment

3.4.4.1.1.5 Method “Rank Status”

The RESTful interface “Rank Status” (Figure 56), ranks a status if the user likes the content.

Rank Status		
Description	Ranks a status to communicate that the user likes this post	
Request		
Request URL	**POST** http://example.com/api/sc/useSN:socialNetwork/useStatus:statusID/rank:rank	
Resource Parameters	Name	Description
	rank Required bool	True is equivalent as an like Example: *true*
	statusId Required string	ID for the status Example: “*1234”*
	socialNetwork Required String	The social network target Example: ”*Twitter”*
Response		
HTTP Status Code	Value	Description
	200	Rank was successful
	401	Not authorised – wrong credentials
	409	Error in further processes
	502	Database error

JSON Attributes	Status
	Optional Status Object
		The status which was posted with an increased favoriteCount
		Example:
		{
		“statusId” : “1234”,
		“creationDate” : “01.01.2014”,
		“statusText” : “SAM is awesome!”,
		“source” : “Twitter”,
		“geoLocation” : “Berlin, Germany”,
		“language” : “English”,
		“retweetCount” : 0,
		“favoriteCount” : 1
		}

Figure 56: RESTful Interface Description – Rank Status

3.4.4.1.1.6 Method “Share Status”

The RESTful interface “Share Status” (Figure 57), shares an existing status. Typically, sharing statuses indicates other users that the content is recommended.

Share Status		
Description	Shares an existing status of a user.	
Request		
Request URL	**POST** http://example.com/api/sc/useSN:socialNetwork/useStatus:statusId/share:/	
Resource Parameters	Name	Description
	statusId Required string	ID for the status Example: “*statusId”*
	socialNetwork Required String	The social network target Example: ”*Twitter”*
Response		
HTTP Status Code	Value	Description
	201	Recommend created
	401	Not authorised – wrong credentials
	409	Error in further processes
	502	Database error
JSON	Status	The status which was posted with an

Attributes	Optional Status Object	increased retweetCount.
		Example:
		{
		“statusId” : “1234”,
		“creationDate” : “01.01.2014”,
		“statusText” : “SAM is awesome!”,
		“source” : “Twitter”,
		“geoLocation” : “Berlin, Germany”,
		“language” : “English”,
		“retweetCount” : 1,
		“favoriteCount” : 0
		}

Figure 57: RESTful Interface Description – Recommend Status

3.4.4.1.1.7 Method “Get Personal Timeline”

The RESTful interface “Get Personal Timeline” (Figure 58) requests the personal timeline of the logged-in user.

Get Personal Timeline		
Description	Request the timeline of the logged-in user.	
Request		
Request URL	**GET** http://example.com/api/sc/useSN:socialNetwork	
Resource Parameters	Name	Description
	socialNetwork Required String	The social network target Example: *“Twitter”*
Response		
HTTP Status Code	Value	Description
	200	Request was successful
	401	Not authorised – wrong credentials
	409	Error in further processes
	502	Database error
JSON Attributes	list Optional Status Object	The status list which was requested Example: [{ “statusId” : “1234”,

```
    “userId” : “12”
    “creationDate” : “01.01.2014”,
    “statusText” : “This was my first tweet”,
    “source” : “Twitter”,
    “geoLocation” : “Berlin, Germany”,
    “language” : “English”,
    “retweetCount” : 0,
    “favoriteCount” : 0
  },
  {
    “statusId” : “1235”,
    “userId” : “12”,
    “creationDate” : “02.01.2014”,
    “statusText” : “This was my second tweet”,
    “source” : “Twitter”,
    “geoLocation” : “Berlin, Germany”,
    “language” : “English”,
    “retweetCount” : 2,
    “favoriteCount” : 1
  },
  {
    “statusId” : “1236”,
    “userId” : “12”,
    “creationDate” : “03.01.2014”,
    “statusText” : “This was my third tweet”,
    “source” : “Twitter”,
    “geoLocation” : “Berlin, Germany”,
    “language” : “English”,
    “retweetCount” : 3,
    “favoriteCount” : 5
  }
]
```


Figure 58: RESTful Interface Description – Get Personal Timeline

3.4.4.1.1.8 Method “Get User Timeline”

The RESTful interface “Get User Timeline” (Figure 59), requests a timeline by a given user ID.

Get User Timeline		
Description	Gets a timeline from a given user ID.	
Request		
Request URL	**GET** http://example.com/api/sc/useSN:socialNetwork/userId:id	
Resource Parameters	Name	Description
	id Required int	A user ID from the social network Example: *12*
	socialNetwork Required String	The social network target Example: ”*Twitter”*
Response		
HTTP Status Code	Value	Description
	200	Request successful
	401	Not authorised – wrong credentials
	409	Error in further processes
	502	Database error
JSON Attributes	list Optional Status Object	The status list which was requested Example: [{ “statusId” : “1234”, “userId” : “12” “creationDate” : “01.01.2014”, “statusText” : “This was my first tweet”, “source” : “Twitter”, “geoLocation” : “Berlin, Germany”, “language” : “English”, “retweetCount” : 0, “favoriteCount” : 0 }, { “statusId” : “1235”, “userId” : “12”,

		“creationDate” : “02.01.2014”,
		“statusText” : “This was my second tweet”,
		“source” : “Twitter”,
		“geoLocation” : “Berlin, Germany”,
		“language” : “English”,
		“retweetCount” : 2,
		“favoriteCount” : 1
		},
		{
		“statusId” : “1236”,
		“userId” : “12”,
		“creationDate” : “03.01.2014”,
		“statusText” : “This was my third tweet”,
		“source” : “Twitter”,
		“geoLocation” : “Berlin, Germany”,
		“language” : “English”,
		“retweetCount” : 3,
		“favoriteCount” : 5
		}
		]

Figure 59: RESTful Interface Description – Get User Timeline

3.4.4.1.1.9 Method “Get Comments”

The RESTful interface “Get Comments” (Figure 60), requests comments from a status.

Get Comments		
Description	Gets comments from an existing status.	
Request		
Request URL	**GET** http://example.com/api/sc/useSN:socialNetwork/status:statusId/comments	
Resource Parameters	Name	Description
	statusId Required string	ID for the status Example: *“statusId”*
	socialNetwork Required String	The social network target Example: ”*Twitter”*

Response		
HTTP Status Code	Value	Description
	200	Comment created
	401	Not authorised – wrong credentials
	409	Error in further processes
	502	Database error
JSON Attributes	list Optional Comments Object	The list of statuses/comments which were requested Example: [{ “statusId” : “1234”, “userId” : “12” “creationDate” : “01.01.2014”, “statusText” : “This was my first tweet”, “source” : “Twitter”, “geoLocation” : “Berlin, Germany”, “language” : “German”, “retweetCount” : 0, “favoriteCount” : 0 }, { “statusId” : “1235”, “userId” : “12”, “creationDate” : “02.01.2014”, “statusText” : “This was my second tweet”, “source” : “Twitter”, “geoLocation” : “Berlin, Germany”, “language” : “English”, “retweetCount” : 2, “favoriteCount” : 1 }, { “statusId” : “1236”, “userId” : “12”,

		“creationDate” : “03.01.2014”, “statusText” : “This was my third tweet”, “source” : “Twitter”, “geoLocation” : “Berlin, Germany”, “language” : “English”, “retweetCount” : 3, “favoriteCount” : 5 }]

Figure 60: RESTful Interface Description – Get Comments

3.4.4.1.1.10 Method “Get Status”

The RESTful interface “Get Status” (Figure 61), requests a single Status.

Get Status		
Description	Gets all information of a single status	
Request		
Request URL	**GET** http://example.com/api/sc/useSN:socialNetwork/status:statusId/	
Resource Parameters	Name	Description
	statusId Required string	ID of the status Example: *“statusId”*
	socialNetwork Required String	The social network target Example: ”*Twitter”*
Response		
HTTP Status Code	Value	Description
	200	Request successful
	401	Not authorised – wrong credentials
	409	Error in further processes
	502	Database error
JSON Attributes	Status Optional Status Object	The status which was posted Example: [{ “statusId” : “1234”, “userId” : “12”

		“creationDate” : “01.01.2014”, “statusText” : “This was my first tweet”, “source” : “Twitter”, “geoLocation” : “Berlin, Germany”, “language” : “English”, “retweetCount” : 0, “favoriteCount” : 0 }]

Figure 61: RESTful Interface Description – Get Status

3.4.4.1.1.11 Method “Get Statuses by Hashtag”

The RESTful interface “Get Status by Hashtag” (Figure 62), requests a list of statuses, which contains the requested hashtag.

Get Status by Hashtag		
Description	Gets all statuses identified by a hashtag	
Request		
Request URL	**GET** http://example.com/api/sc/useSN:socialNetwork /Hashtag:hashtag	
Resource Parameters	Name	Description
	hashtag Required string	Hashtag as a filter for the result list Example: *“#SAM”*
	socialNetwork Required String	The social network target Example: *“Twitter”*
Response		
HTTP Status Code	Value	Description
	200	Request successful
	401	Not authorised – wrong credentials
	409	Error in further processes
	502	Database error
JSON Attributes	list Optional Status List	The status which was posted Example: [{ “statusId” : “1234”,

```
    “userId” : “12”
    “creationDate” : “01.01.2014”,
    “statusText” : “#SAM is my new favourite”,
    “source” : “Twitter”,
    “geoLocation” : “Berlin, Germany”,
    “language” : “English”,
    “retweetCount” : 0,
    “favoriteCount” : 0
  },
  {
    “statusId” : “249685”,
    “userId” : “14”,
    “creationDate” : “02.01.2014”,
    “statusText” : “#SAM is great”,
    “source” : “Twitter”,
    “geoLocation” : “Berlin, Germany”,
    “language” : “English”,
    “retweetCount” : 2,
    “favoriteCount” : 1
  },
  {
    “statusId” : “983895”,
    “userId” : “9894”,
    “creationDate” : “03.01.2014”,
    “statusText” : “#SAM is awesome”,
    “source” : “Twitter”,
    “geoLocation” : “Berlin, Germany”,
    “language” : “English”,
    “retweetCount” : 3,
    “favoriteCount” : 5
  }
]
```


Figure 62: RESTful Interface Description – Get Status by Hashtag

3.4.4.1.2 **Social Network Communication**

The SocIoS API is used in order to get connected with Social Network APIs. The RESTful requests from the Social Components Services will be forwarded to the Social Network Communication. But also the Social Network Communication collects social content from the Local Information Cache. This component provides some requested data from recently requests. If the Cache History is outdated, new requests will be sent to the social networks. The response contains the requested social content and will be forwarded to the Local Information Cache, Cloud Storage and finally to the 2nd Screen and the generic dashboard.

3.4.4.1.3 **Dynamic Communities**

Additionally to external social networks this component enables other SAM components to communicate with so-called Dynamic Communities provides by the SAM platform itself.

3.4.4.1.3.1 Method "Get Dynamic Community"

The RESTful interface "GetDynamicCommunity" (Figure 63), returns an existing Dynamic Community provided by the Context Control component.

Get Dynamic Community		
Description	Returns an existing Dynamic Community	
Request		
Request URL	**POST** http://example.com/api/sc/GetDynamicCommunity:dynCommunityId	
Resource Parameters	Name	Description
	dynCommunityId Required Integer	Message of the status. Example: *12*
Response		
HTTP Status Code	Value	Description
	200	Community successful forwarded
	401	Not authorised – wrong credentials
	409	Error in further processes
	502	Database error
JSON Attributes	Dynamic Community Optional DynamicCommunity Object	The Dynamic Community which was created for the user Example: { "dynCommunityId" : 12, "dynCommunityName" : "sportfanatics", "dynCommunityKeywords" : [{sports, fanatic, fan}] "dynCommunityDescription" : "users are

		talking about sports” “creationDate” : “01.01.2014”, “userCount” : 25, “statuses”:[{Status}, {Status2}, {Status3}] “language” : “English”, }

Figure 63: RESTful Interface Description – Get Dynamic Community

3.4.4.1.3.2 Method “Change Dynamic Community”

The RESTful interface “ChangeDynamicCommunity” (Figure 64), returns an existing Dynamic Community provided by the Context Control component.

Change Dynamic Community		
Description	Change a Dynamic Community	
Request		
Request URL	**POST** http://example.com/api/sc/ChangeDynamicCommunity:dynCommunityId	
Resource Parameters	Name	Description
	dynCommunityId Required Integer	Message of the status. Example: *14*
Response		
HTTP Status Code	Value	Description
	201	Community successful forwarded
	401	Not authorised – wrong credentials
	409	Error in further processes
	502	Database error
JSON Attributes	Dynamic Community Optional DynamicCommunity Object	The Dynamic Community which was changed for the user Example: { “dynCommunityId” : 14, “dynCommunityName” : “Action”, “dynCommunityKeywords” : [{Action, fanatic, fan}] “dynCommunityDescription” : “users are talking about Action Movies”, “creationDate” : “05.06.2014”,

		“userCount” : 137, “statuses”:[{Status}, …, {StatusX}] “language” : “English”, }

Figure 64: RESTful Interface Description – Change Dynamic Community

3.4.4.1.3.3 Method “Find Dynamic Community”

The RESTful interface “FindDynamicCommunity” (Figure 65), returns an existing Dynamic Community provided by the Context Control component.

Find Dynamic Community		
Description	Finds Dynamic Communities which are suitable for provided keywords	
Request		
Request URL	**GET** http://example.com/api/sc/DynamicCommunity/FindCommunity/keywords:keywords	
Resource Parameters	Name	Description
	keywords Required List<String>	Keywords to find suitable Dynamic Communities Example: *“Fanatic”, “Fan”*
Response		
HTTP Status Code	Value	Description
	201	Community successful forwarded
	401	Not authorised – wrong credentials
	409	Error in further processes
	502	Database error
JSON Attributes	Dynamic Community Optional DynamicCommunity List	The Dynamic Community which was changed for the user Example: [{ “dynCommunityId” : 12, “dynCommunityName” : “sportfanatics”, “dynCommunityKeywords” : [{sports, fanatic, fan}] “dynCommunityDescription” : “users are talking about sports” “creationDate” : “01.01.2014”,

		“userCount” : 25,
		“statuses”:[{Status}, {Status2}, {Status3}]
		“language” : “English”,
		},
		{
		“dynCommunityId” : 14,
		“dynCommunityName” : “Action”,
		“dynCommunityKeywords” : [{Action, fanatic, fan}]
		“dynCommunityDescription” : “users are talking about Action Movies”,
		“creationDate” : “05.06.2014”,
		“userCount” : 137,
		“statuses”:[{Status}, …, {StatusX}]
		“language” : “English”,
		}
		]

Figure 65: RESTful Interface Description – Find Dynamic Community

3.4.5 Summary

In this section, the Social Components have been technically specified. During the technology selection (see Section 3.4.2) the SOCIOS Framework was chosen to communicate with external social networks. SOCIOS provides unified access to several different social networks. The responses from the social network requests are forwarded to the Generic Dashboard and the Local Information Cache component. MongoDB was chosen to store recently requests from the social networks. It was chosen for the same arguments reasons as the Cloud Storage Technology for NoSQL Databases (see also Figure 95).

According to this specification, the following components have to be implemented:

- RESTful Interface to provide the Generic Dashboard with social content and an interface for the Context Control to send information about the current dynamic community
- A service to distribute and forward social content requests and User-Generated Content to the Context Control, Cloud Storage Local Information Cache and SOCIOS
- Local Information Cache with an accompanying service to get access to the storage

3.5 Syndicator

The Syndicator component provides the information to be consumed by the End Users through the SAM Dashboard or through 3rd Party Apps built by external Software Developers (See Figure 66). In order to provide the right data at the right moment, the Syndicator will take into account the user’s context and usage in the 1st and 2nd Screen.

This component is split into Data API Services and Multi-Device Representation subcomponents and the technical decisions about both are described in the next subsections.

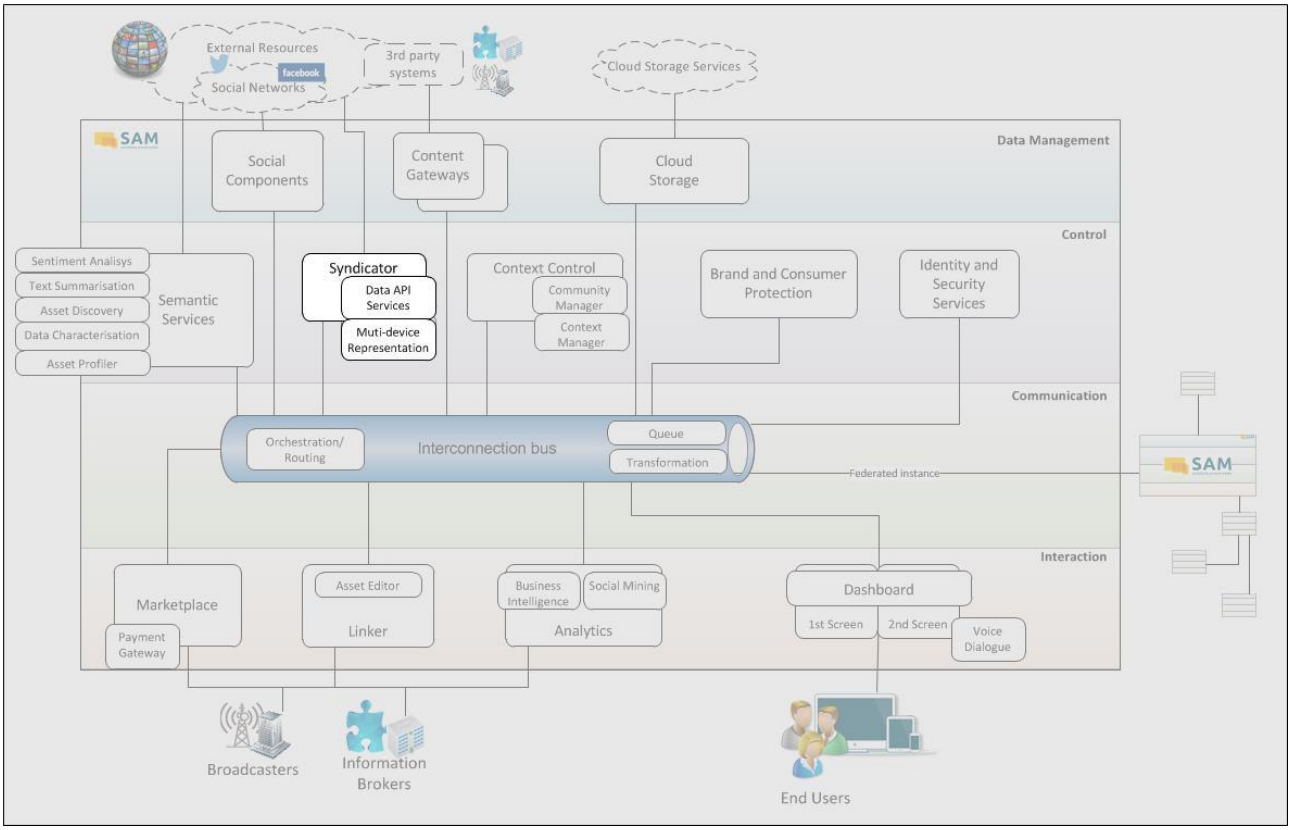

Figure 66: Architecture Overview – Syndicator

3.5.1 Data API Services

This subcomponent is in charge of delivering the information to the Dashboard or 3rd Party Apps (see Figure 67) based on the assets stored in the SAM Platform, related assets discovered through Semantic Services component and linked information retrieved from external services. The most important goals are the following:

- Providing information stored in SAM to the End User, taking into account the user context, the assets business constraints, etc.
- Interacting with the Multi-device Representation component to provide the information in the correct format
- Providing access to 3rd Party in order to access SAM information to be used e.g. by mobile apps, external web sites, etc.

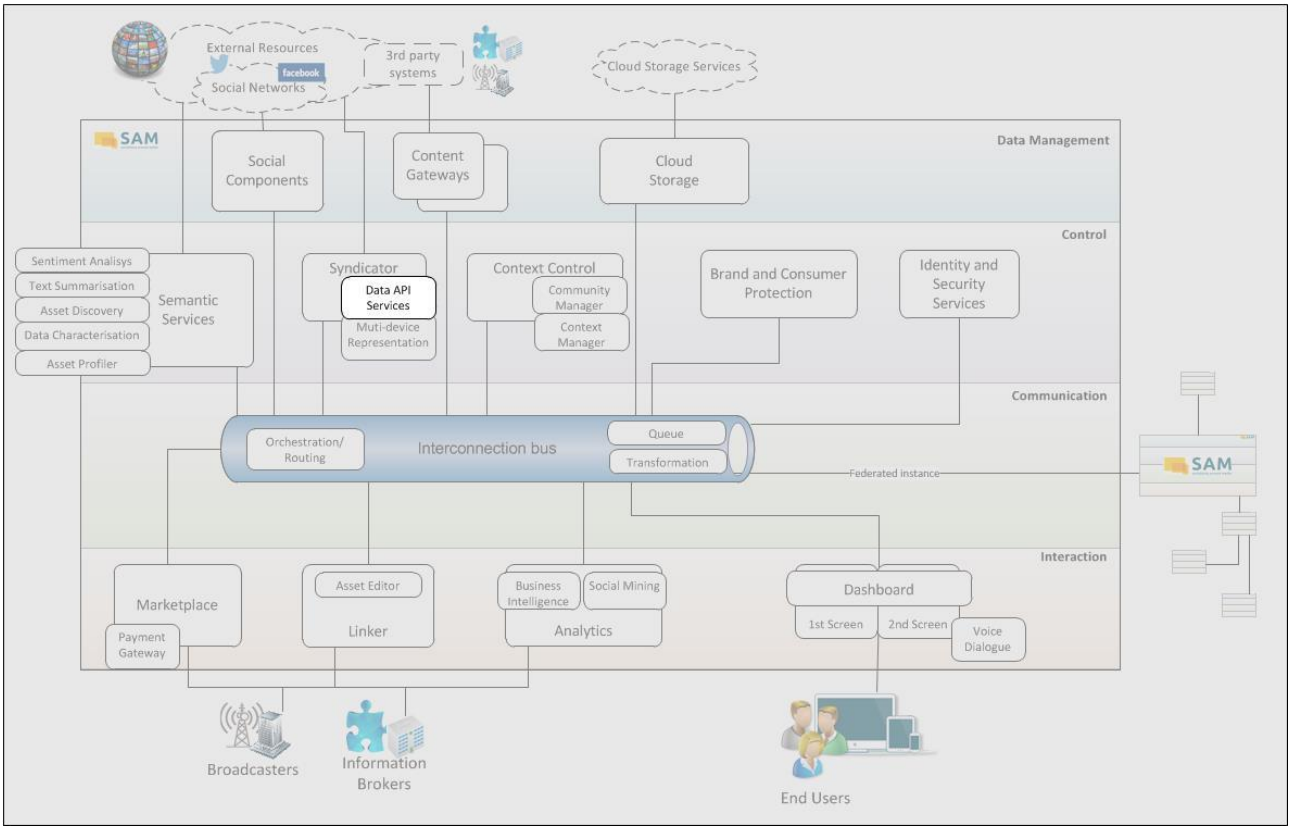

Figure 67: Architecture Overview – Data API Services

3.5.1.1 Major Design Decisions

TIE Kinetix™ Content Syndication Platform (TIE CSP) has been previously selected as technology to use in SAM Platform (See DOW Section 1.3.3.5 WP5 Content Syndication and Delivery), thus it will be necessary to enhance and adapt for its usage in SAM. The major decisions to take into account in the Data API Services are explained in the following points:

- Data API Services will be used mainly in the prosumption scenario (run-time scenario), thus the selected technology should be able to provide a very fast and light answer to the multiple concurrent users in different kinds of devices. Therefore, the maturity and stability of the technology is a key feature to take into account.
- The Data API services will provide rendered data in the form of microsites, videos, advertisements, etc. to be used in the client side (Dashboard or 3rd party apps). The component will generate executable code instead of static structures improving the interactivity at the client. The generated code will be navigable, so that internal calls in the client side allow the navigation through the syndicated content.
- The selected technology will allow SAM to distribute the same content to multiple destinations simultaneously, personalising the information depending on the context (user profile, device, etc.). Server and client logging features will be implemented, augmenting the granularity of the possible feedback to be provided back to the user.

3.5.1.2 Technology Comparison and Selection

Due to the fact that TIE CSP has been already selected as Content Syndication technology, this subsection aims to show that TIE CSP meets the necessary requirements to implement Data API services component. Within the subsection, the areas that need to

be implemented or improved to fully meet SAM specific requirements are identified and presented. The selected technologies will be used as a basis for the development phase in the realisation of the technical design of this component: The specific selection criteria to be used are specified in the document D3.2.2 Functional Specification, Section 4.5.1.4 and will be used in the next sections.

3.5.1.2.1 Possible Technologies and Comparison

TIE Kinetix™ Content Syndication Platform [46](TIE CSP) is a platform based on the latest .NET technologies and state-of-the-art client technologies such as AngularJS. It uses the publish/subscribe pattern to provide syndication of multiple kinds of content in several destinations and devices at the same time. The system generates navigable code on the client side allowing control over the content consumed by the user. One of TIE Kinetix main business activities is Content Syndication orchestrated through TIE CSP, serving content to more than 50 countries in more than 30 languages, creating more than 12 million impressions and more than 3 million unique visitors a month and used by companies such as Siemens, Avaya or Lenovo. Therefore, it has proven track record of maturity and stability. TIE CSP is able to syndicate all kind of digital content in a fast and secure way and provide tailored information depending on the end user. In order to monitor end user activity (number of clicks, time in watching the content, geo position, etc.), the system registers all this activity in logs which can be used to generate reports and statistics.

3.5.1.2.2 Technology Selection

TIE CSP looks like a suitable solution to use for developing the component since it offers a cutting-edge technology in the syndication area, which offers high value in key requirements such as maturity, scalability, interoperability and specific parameters.

Besides, TIE CSP has been selected due to the lack of similar technologies available in the market and the fact that it was previously nominated as technology to use in SAM Platform (See DOW Section 1.3.3.5 WP5 Content Syndication and Delivery).

In conclusion, TIE CSP is selected to carry out the Data API services component since it meets the criteria to be used as technology and it is nominated in the DOW as the technology to be used.

3.5.1.3 Technical Component Specification

3.5.1.3.1 Structure

Figure 68 provides an overview of the Data API Services component, which can be found also in D3.2.1 Global Architecture Definition.

[46] http://contentsyndication.tiekinetix.com

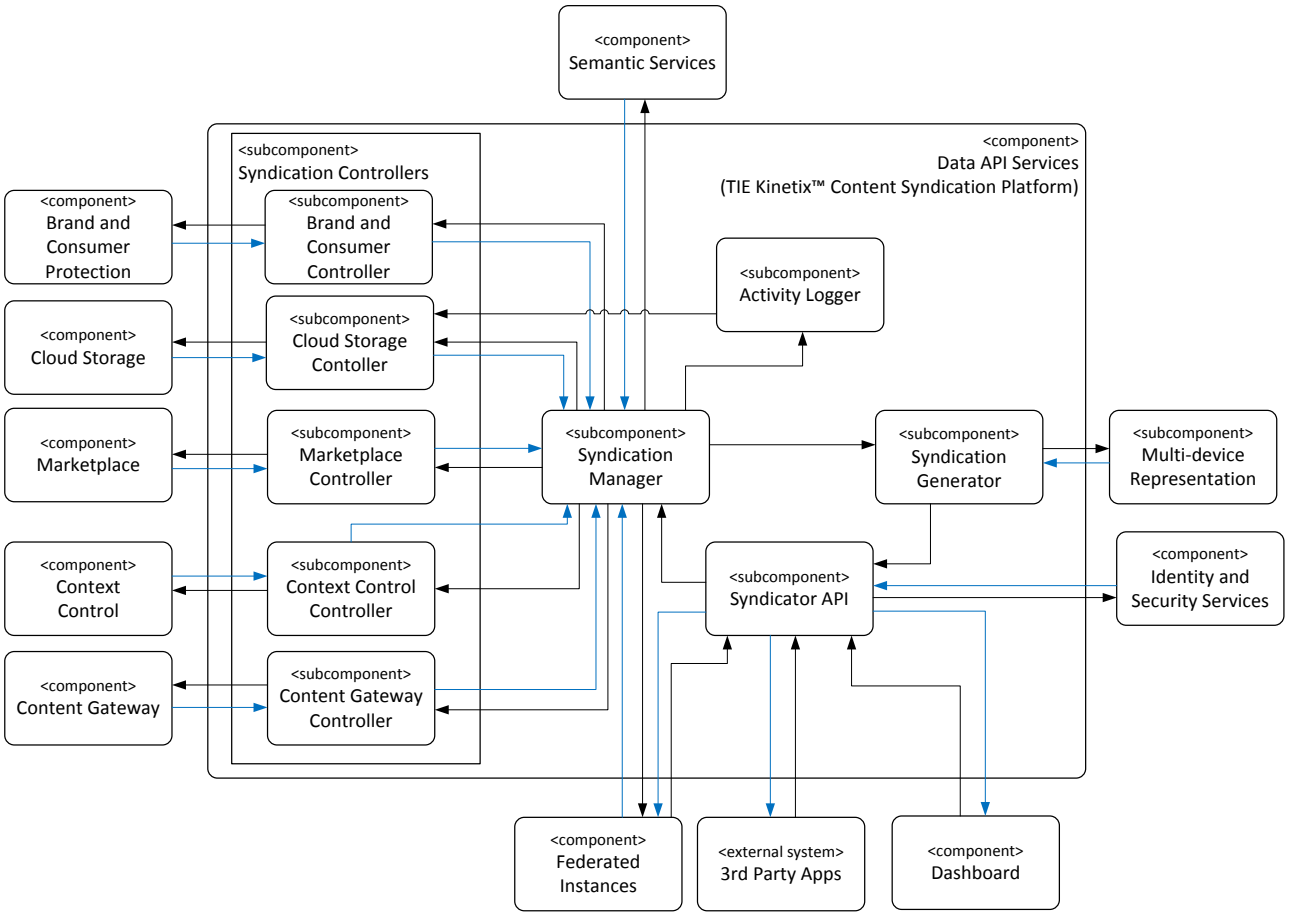

Figure 68: Overview with Selected Technologies – Data API Services

3.5.1.4 Specification of Interfaces, Protocols and Formats

The services of the Data API Services will be provided as RESTful interfaces (described in Section 3.5.1.4.1) in order to allow the communication with the Dashboard and 3rd Party Apps.

3.5.1.4.1 RESTful Interfaces

In order to describe the RESTful interfaces, each provided service is described in a separate table. Each table will show a description, URL, request parameters with an example and response type including an example.

Data API services component provide interfaces to allow the communication with Dashboard and 3rd party apps.

3.5.1.4.1.1 Method “Get Content to Syndicate Dashboard”

The RESTful interface “GetContentToSyndicateDashboard” (Figure 73), sends the information to be syndicated in the Dashboard. This is the core functionality of the Generic Dashboard. By receiving content from the Syndicator, the Widgets can visualise this information in an attractive way.

Get Content to Syndicate Dashboard	
Description	This interface is used to send the syndicated information to the Dashboard.

Request		
Request URL	**GET** http://example.com/api/synd/GetContentToSyndicateDashboard:syndicationFilter	
Resource Parameters	Name	Description

	syndicationFilter
	Required Object
		JSON object with the filter parameters. The video ID is a required value
		Example:
		{
		“videoId”: 2345,
		“time”: “#8”,
		“assetId”: 7896
		}

Response		
HTTP Status Code	Value	Description
	200	The request was successful
	400	Bad Request
	500	Error during processing
JSON Attributes	template JSON Object	JSON object with the widgets. Example: { “widgets”: [{ “widgetId”: 1, “html”: “<b:widget id='1' locked='false' title='Widget1' type='HTML'> Widget Information </b:widget>”, }, { “widgetId”: 2, “html”: “<b:widget id='2' locked='false' title='Widget2' type='HTML'>

		 Widget Information </b:widget>”, }] }

JSON Error	JSON Object	Example:
		{
		“errors”:
		[
		{
		“errorId”: 1,
		“description”: “videoId is required value”
		}
		]
		}

Figure 69: RESTful Interface Description – Get Content to Syndicate Dashboard

3.5.1.4.1.2 Method “Syndicator Authentication”

The RESTful interface “SyndicatorAuthentication” (Figure 70) will be used to validate the user credentials in the syndicator system. This interface will use for authentication internally the Identity and Security Services component (see Section 3.10).

Syndicator Authentication		
Description	Authenticates a user using the provided credentials.	
Request		
Request URL	**GET** https://example.com/api/synd/SyndicatorAuthentication/user:userId/pass:password	
Resource Parameters	Name	Description
	userId Required string	User ID Example: “*testuser123”*
	password Required string	User password Example: *“abcdef”*
Response		
HTTP Status	Value	Description

Code	200	Authentication success
	401	Authentication failure
	500	Error during processing
JSON Attributes	list Required list	A JSON object containing the session token Example: [{ "token": "abcdef", time_validation:"60days" }]

JSON Error	JSON Object	Example:
		{
		"errors":
		[
		{
		"errorId": 1,
		"description": "user and password are required value"
		}
		]
		}

Figure 70: RESTful Interface Description – Syndicator Authentication

3.5.1.4.1.3 Method "Get Content to Syndicate"

The RESTful interface "GetContentToSyndicate" (Figure 73), sends the information to syndication to 3rd party apps.

Get Content to Syndicate		
Description	This interface is used to send the syndicated information to 3rd party apps.	
Request		
Request URL	**GET** http://example.com/api/synd/GetContentToSyndicate:syndicationFilter	
Resource Parameters	Name	Description
	syndicationFilter Required Object	JSON object with the filter parameters. The value -1 means that the filter is not taken into account. The video ID is required value Example: { "token": "abcdef",

		“device”: “nexus5”, “resolution”: “1080px”, “videoId”: 2345, “time”: “#8”, “assetId”: 7896 }
Response		
HTTP Status Code	Value	Description
	200	The request was successful
	400	Bad Request
	500	Error during processing

JSON Attributes	template
	JSON Object
		JSON object with the widgets.
		Example:
		{
		“widgets”:
		[
		{
		“widgetId”: 1,
		“html”: “<b:widget id='1' locked='false' title='Widget1' type='HTML'>
		
		Widget Information
		
		</b:widget>”,
		},
		{
		“widgetId”: 2,
		“html”: “<b:widget id='2' locked='false' title='Widget2' type='HTML'>
		
		Widget Information
		
		</b:widget>”,
		}
		]

		}

JSON Error	JSON Object	Example:
		{
		“errors”:
		[
		{
		“errorId”: 1,
		“description”: “token is required value”
		}
		]
		}

Figure 71: RESTful Interface Description – Get Content to Syndicate 3rd Party

3.5.1.5 Summary

TIE CSP has already been proposed as technology to use in the SAM Platform (See DOW Section 1.3.3.5 WP5 Content Syndication and Delivery) and it will be enhanced and adapted for use in SAM. It offers a cutting-edge technology in the syndication area and high value in all key parameters such as maturity, scalability, interoperability and behaviour in the client side. Therefore, TIE CSP has been selected to provide SAM Platform with a technology in 2nd Screen multi-device syndication.

3.5.2 Multi-Device Representation

Multi-Device Representation (MDR) is a component that houses two different subcomponents, Graphical Editor and Format Converter. Figure 72 shows how the component is placed within the overall SAM Platform environment.

During Design Time, MDR’s task is to enable the Content Owner to create, edit or delete presentation templates. For this purpose, the Graphical Editor needs to contain logic that will handle interaction with the templates and the place where they will be stored (Cloud Storage). Additionally, it will contain a UI, which will enable the user to take advantage of underlying functionalities.

During Runtime, MDR’s task is to enable SAM to display the relative assets optimised for the End User’s current 2nd Screen device and with the right subset of functionalities for it. It will do this by providing the correct presentation template for the device. The Format Converter will function as an underlying mechanism; no UI is needed.

The code used by the Format Converter while in Runtime will be a subset of the code used in Design Time.

Thus, in general, MDR will provide the following:

- During **Design Time** it will serve as a template builder for a given asset. It will allow Users/ Content Owners to instruct how they want their content to be displayed given the device that the content will be shown on.

- During **Runtime** it will serve as a presentation template retriever for a given asset. It will allow the 2nd Screen to receive the related assets in a form optimised for the device they will be displayed on.
- It will provide an editor so that the content providers could design specific appearance or themes to be associated to their specific assets.

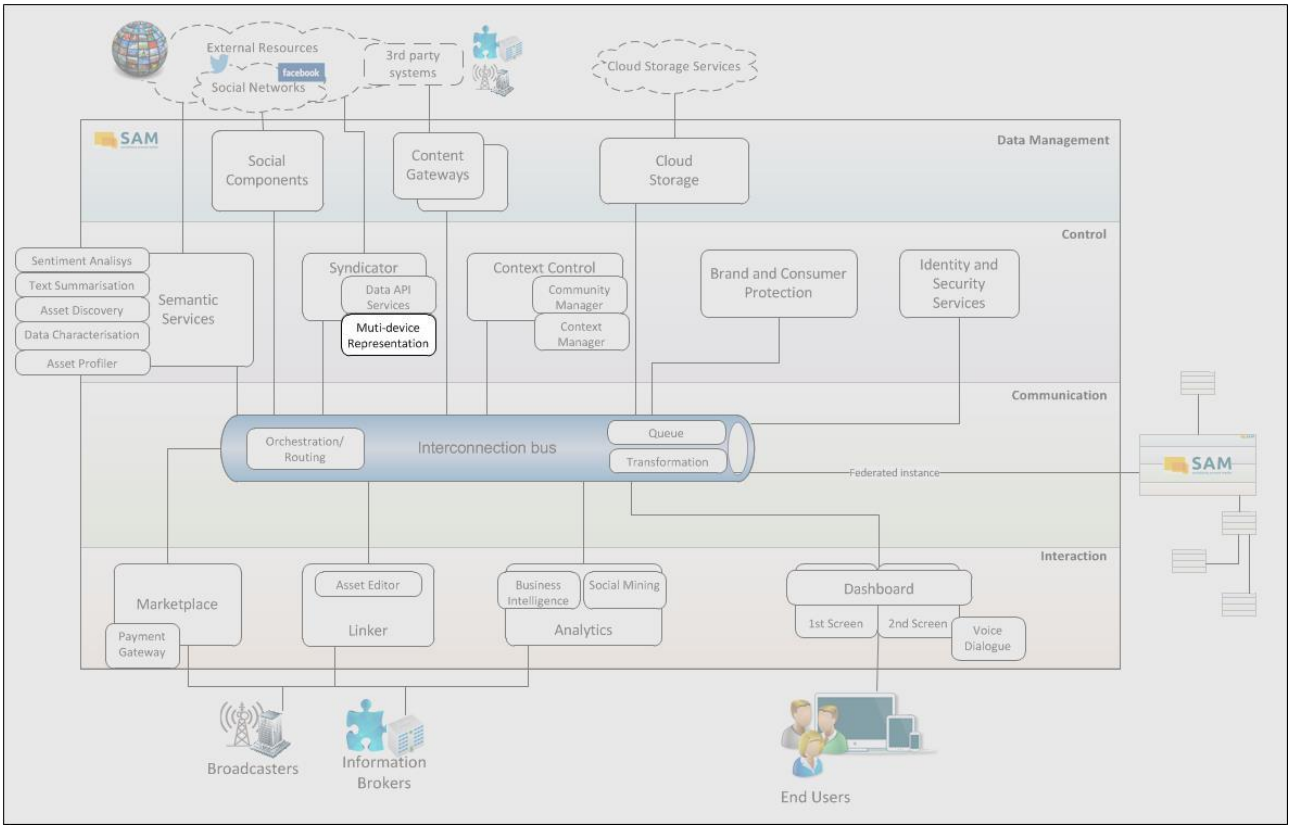

Figure 72: Architecture Overview – Multi-Device Representation

3.5.2.1 Major Design Decisions

Various major decisions have to be done for the Design Time and Runtime subcomponents:

- In **Design Time**, the Model-View-Controller (MVC) paradigm will be followed. This paradigm is ideal for applications which enable the user to edit data objects. It entails that the user need not and should not have knowledge of the underlying structure and mechanisms that are applied when handling the data:
  - The Design Time component, **Graphical Editor**, needs to be built in HTML5 since the rest of the Marketplace will be built with it and all the Marketplace components need to be tightly integrated. By following the MVC paradigm, the UI can be built in HTML5 while different tools can be used for the business logic.
  - **Graphical Editor's business logic will partly overlap with** the Runtime component**, Format Converter**. So, in essence, Format Converter will be used both as a stand-alone component and as part of the backend of the Graphical Editor. Consequently, from a component level view and as it is visible in the architecture diagram, the Data API Services call Format Converter directly when in Runtime. During Design Time, the Content Provider indirectly accesses Format Converter through the Graphical Editor.

- Data API Services will contact the Format Converter through the **Intercommunication Bus**. The communication between Graphical Editor and Format Converter will be done through **REST**. This will be a fitting solution since Graphical Editor will essentially be a web interface.

- In **Runtime**, it has been decided that the range of the frameworks to select from should be limited to Python-based ones. The reason is that the specific language is well within the area of knowledge of Talkamatic, the partner who will develop it. Also, Python has excellent support for RESTful interfaces and it is a popular and well supported cross-platform language. Finally, Python's popularity results in a multitude of web frameworks that can be used for the purposes of MDR.

3.5.2.2 Technology Comparison and Selection

This subsection outlines technology selection criteria and compares existing technologies – potential candidates for the realisation of Multi-Device Representation. Within the subsection, the areas that need to be implemented or improved to fully meet SAM-specific requirements are also identified and presented. The selected technologies will be used as a basis for the development phase in the realisation of the technical design of this component: The specific selection criteria to be used are specified in the document D3.2.2 Functional Specification, Section 4.14.5 and will be used in the next sections.

3.5.2.2.1 Possible Technologies and Comparison

In order to realise the MDR component several options have to be considered as base technology. The assessment of these technologies is presented in the following sections.

3.5.2.2.1.1 Format Converter

The framework that will be chosen will allow the creation of a software component that will accept device specifications as input and produce HTML5 files as output. Therefore, all following web-frameworks have been selected based on their ability to handle web content.

- **Django**[47] is a Python framework of a sufficiently high level of abstraction, which enables rapid development and easy design. It includes its own template language system, which can also be used as a stand-alone component. It enforces the separation of the Python (business) logic and the presentation (HTML) logic.
- **Flask**[48] is another Python framework that offers high-level design of web applications. Its focus is on small-scale applications.
- **Pyramid**[49] is another Python web framework that is increasing in popularity. It intends to be a versatile framework. It supports templates by using template-language add-ons such as Jinja2 or Mako.
- **Jinja2**[50] is a template tool. It has the advantages of being built exclusively for handling templates and being quite fast

Parameter	Importance	Django	Flask	Pyramid	Jinja2
Generic Parameters					
Maturity & Stability	+++	+++	++	++	+

[47] https://www.djangoproject.com/
[48] http://flask.pocoo.org/
[49] http://docs.pylonsproject.org/en/latest/docs/pyramid.html
[50] http://jinja.pocoo.org/docs/dev/

Regularly Updated	+	+++	++	++	++
Technical Up-to-Datedness / Appeal	++	+++	++	++	++
Open Source	++	YES	YES	YES	YES
Non-Infecting	++	YES	YES	YES	YES
Code Quality	++	+++	+++	+++	+++
Extensibility	+++	++	+++	+++	++
Community	++	+++	++	++	+
Performance / Scalability	++	+++	+++	+++	+++
Reuse of existing developments	-/+	NO	NO	NO	NO
EU project origin	--	NO	NO	NO	NO
Platform (Portability)	++	+++	+++	+++	+++
Open Standards Compliance	+++	YES	YES	YES	YES
Interoperability (easy integration for all platforms)	++	+++	+++	+++	+++
Specific Parameters					
HTML5 Support	+++	+++	+++	+++	+++
CSS3 Support	+++	+++	+++	+++	+++
Automatic Device and Feature Detection	++	YES	YES	YES	YES
Design and Runtime Consistency	+	++	++	++	++
Ease of Development	+++	+++	++	++	++

Figure 73: Parameter Evaluation of Technologies for Format Converter

3.5.2.2.2 Technology Selection

After comparing the technologies in the previous section, the selection of technology for each subcomponent is explained below.

3.5.2.2.2.1 Format Converter

As seen in Figure 73, Django is slightly more fitting to our purposes than Jinja2 and considerably better than Flask and Pyramid. The main reason for this outcome is Django's popularity, robustness and size that makes it more preferable even compared to domain-specific solutions such as Jinja2.

Even though Django is a full-fledged web framework with template handling being only one of its subcomponents, its community and ubiquity among Python developers renders it more fitting for the purposes of SAM.

3.5.2.2.3 User Interface Technology

The Graphical Editor subcomponent will function as the MDR user interface during Design Time. Through it, the content owner is able to manage templates for the representation of assets. The technology used for the implementation of the user interface has been selected in Section 3.1.

3.5.2.3 Technical Component Specification

3.5.2.3.1 Structure

Figure 74 provides an overview of the Multiple Device Representation, which can be found also in D3.2.1 Global Architecture Definition. Additionally this version of the overview in this chapter has been edited to include the selected technologies.

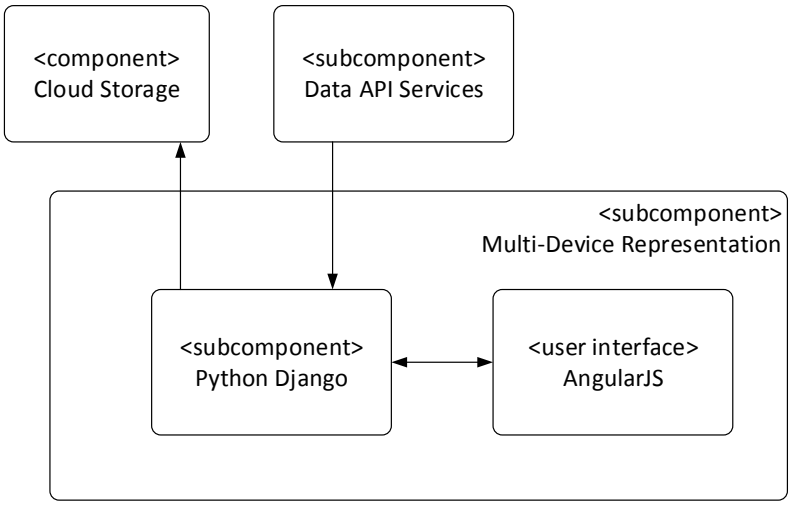

Figure 74: Overview with Selected Technologies – Multi-Device Representation

3.5.2.4 Specification of Interfaces, Protocols and Formats

The services of MDR and specifically the Format Converter subcomponent will be accessed by the following RESTful interface.

3.5.2.4.1 Restful Interfaces

In order to describe the RESTful interfaces, each provided service is described in a separate table. Each table will show a description, URL, request parameters with an example and response type including an example.

3.5.2.4.1.1 Method “Format Converter”

Request Formatting		
Description	Requests a matching operation from the Format Converter. Depending on the data passed with the request, Format Converter returns with the appropriate template.	
Request		
Request URL	**POST** http://example.com/api/mdr/formatConverter:dataType	
HTTP Parameters	dataType Required String	Data type of the transferred object Example: “*JSON”*
	deviceSpecs Required object	JSON object representing a query object for the specified data type. Example:

		{
		“model” : “nexus5”,
		“res” : “1080p”
		}

Response		
HTTP Status Code	Value	Description
	200	Object found
	204	Object not found
	400	Bad Request

JSON Attributes	template
	JSON Object
		A list of JSON objects matching the query.
		Example:
		{
		"templateId": "nexus5_template",
		"templateContent": "<html>…</html>”
		}

Figure 75: Request Formatting Method Specification

3.5.2.5 Summary

In this section, Multiple Device Representation was technically specified. During the technology selection in Section 3.5.2.2.2, Python Django was selected to implement the Format Converter. This was due to the fact that it is built in a familiar and productive language, Python, and that it also has a status of an industry standard.

For the rest of SAM components to access MDR a RESTful Interface will be implemented which will expose the Format Converter to Data API Services.

3.6 Content Gateways

The Content Gateways is the component in charge of data gathering from external data sources, including 3rd party systems (see Figure 76). The objective is to implement strategies, tools and techniques to allow easy integration of heterogeneous content sources into the SAM Platform. The most important goals are the following:

- Define the mapping between the data source structures and the data destination structures and store it in the Mapping Repository to make it available for re-use.
- Importing or linking the necessary information into the SAM Platform from Content Providers’ 3rd party systems
- Extracting data from external resources, such as Social Media services or Wikipedia, implementing internal mechanisms such as API wrappers, web crawlers or scraping techniques

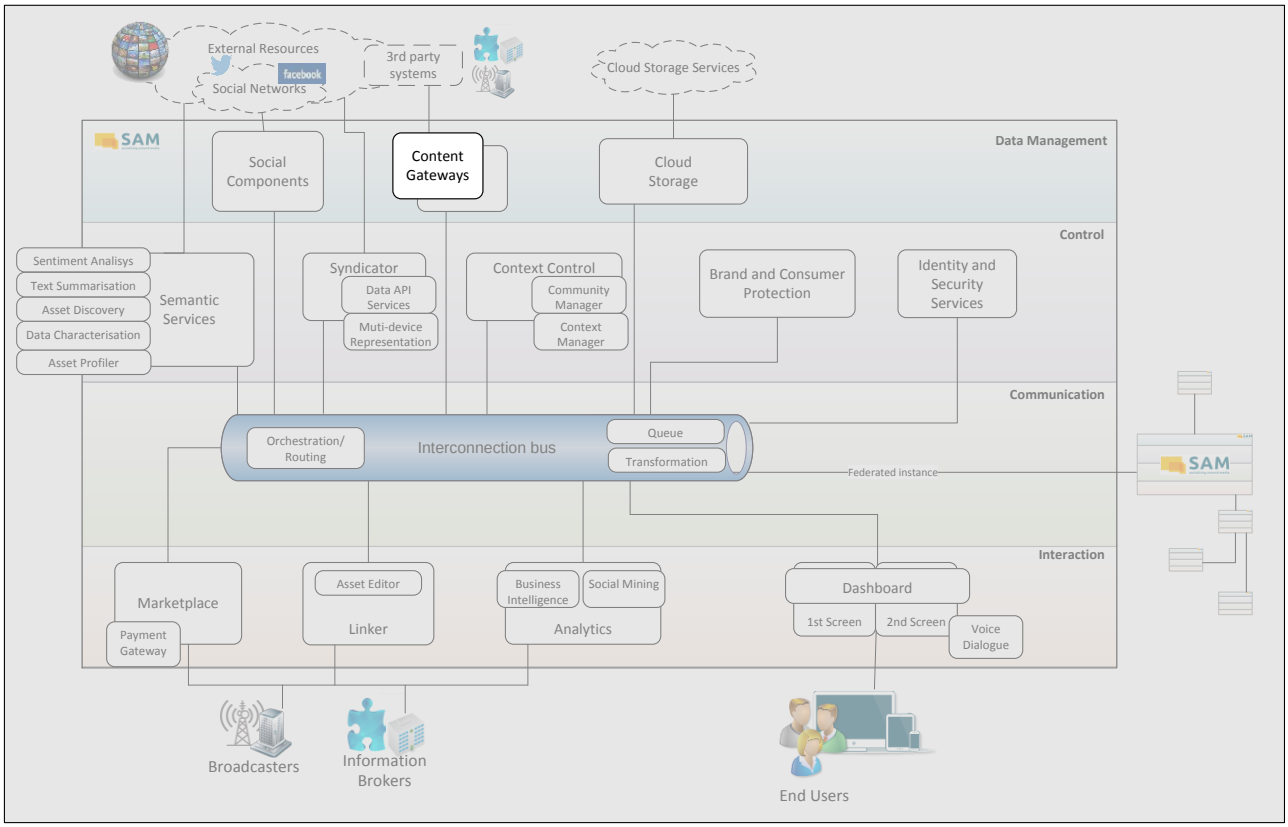

Figure 76: Architecture Overview – Content Gateways

3.6.1 Major Design Decisions

Similar to SAM, the STASIS project relies on semantics for data annotation, storage and transformation. SAM intends to use the STASIS approach of semantic mapping through TIE TSI (TIE Semantic Integrator) – the product TIE has built based on STASIS project and that will be the base for SAM gateways developments (See DOW Section B 1.2.10.2).

TSI is a tool, which offers a user-friendly user interface with a wide range of features in order to define the mapping between the data source structures and the data destination structures. It supports different kinds of transformations and different data type formats based on the most common data file structures such as XML, CSV or RDF. TSI will also suggest mappings based on the semantic entities inferred from the data source and data destination formats definitions (Database structure, XML schema, etc.). The mapping definition is stored in the Cloud Storage so that it can be used for further import or data access operations.

The Content Gateways component will have a web interface providing the management and control of importations provided by Content Providers. Using the Interconnection Bus component (TIE SmartBridge), the Content Gateways component is able to perform complex data import workflows. The execution of the importations can be started manually or by using a scheduled task.

Finally, new screen-scraping techniques should be tested in order to accelerate the development of content extraction mechanisms.

3.6.2 Technology Comparison and Selection

This subsection outlines technology selection criteria and compares existing technologies – potential candidates for the realisation of Data API Services. Within the subsection the areas that need to be implemented or improved to fully meet SAM specific requirements are also identified and presented. The selected technologies will be used as a base for the development phase in the realisation of the technical design of this component: The specific selection criteria to be used are specified in the document D3.2.2 Functional Specification, Section 4.6.4 and will be used in the next sections.

3.6.2.1 Possible Technologies and Comparison

The Content Gateway will use different technologies: An integration tool in order to import/link data from 3rd party systems, a web user interface in order to manage the importations and the mapping repository and finally, it will use a web scraping tool in order to extract information from external resources. The Section 3.6.2.1.1 will describe the comparison about integrations tools and the Section 3.6.2.1.2 about different web scraping open-source tools. The web user interface will follow the consortium decision about user interfaces explained in Section 3.6.2.3.

3.6.2.1.1 Integration Tool

In order to implement the integration tool, several options have to be considered as base technology. The following selection will compare the most promising technologies.

- **TIE Semantic Integrator (TSI)** is a powerful semantic-based format characterisation, mapping and transformation tool that maps to and from different data sources like XML, XSD, RDF, Flat Files (FF), CSV, EXCEL, Databases (MS SQL 2008/2012, MySQL). The aim of TSI is to provide intelligent automatic mapping suggestion based on smart semantic algorithms and vocabularies to further transform the data from one format to another TSI creates and manages semantic relationship between entities based on a Logical Data Model (LDM). Each schema format (XSD, EXCEL, RDB, FF, RDF) imported into TSI is transformed to an internal neutral format based on the LDM meta-model. TSI is an independent solution written in Java and provides a state-of-the-art user interface to import, map and transform data improving the usability of the tool. This user interface is based on Graphical Modelling Framework (GMF)[51] using Standard Widgets Toolkit (SWT)[52] as a graphical technology. The mapping suggestions between elements of schemas are auto-generated based on semantic algorithms and these maps are persisted and can be re-used and shared.
- **Apache Camel**[53] is one of the most popular EAI frameworks at the moment. Apache Camel implements EIP (Enterprise Integration Patterns). It provides up to 100 integration components, including cloud based data services, SQL and NoSQL databases, REST, Web services, and tools for dealing with different data formats (XML, JSON, CSV, flat files, etc.). Beyond this, it provides the ability to execute it in a standalone environment, making it easier for adoption by, e.g. SMEs. Camel is one of the most dynamic Apache projects, having a broad community both contributing to and using it for wider developments (Camel is a core part of the Apache ESB ServiceMix).

[51] http://www.eclipse.org/modeling/gmp/
[52] http://www.eclipse.org/swt/
[53] http://camel.apache.org/

- **Spring Integration** [54] provides an extension of the Spring programming model to support Enterprise Integration Patterns. It supports integration with external systems via declarative adapters. Spring integrations provide enterprise orchestration and adapters for distributed applications and batch applications. It supports traditional RDBMS as well as new NoSQL solutions map-reduce frameworks and cloud based data services. Spring Integration provides a comprehensive integration framework oriented to Spring-based applications. Spring Integration is a very active project in the SpringSource Community.
- **OpenAdaptor**[55] provides many ready-built connectors that include JMS, JDBC, IBM MQ Series, TIBCO Rendezvous, TCP/IP sockets, SOAP, HTTP and Files. OpenAdaptor provides exception management and scriptable components for data filtering, simple transformation and validation. It was originally conceived in 1997, to facilitate a large financial organisation's requirement to integrate its very large application suite using Message Oriented Middleware. It was released to the Open Source Community in 2001. OpenAdaptor is providing a couple of new versions per year.

Parameter	Importance	TSI	Apache Camel	Spring Integration	OpenAdaptor
Generic Parameters					
Maturity & Stability	++	+++	++	++	++
Regularly Updated	+	+++	+++	+++	++
Technical Up-to-Datedness / Appeal	+	+++	+++	++	+
Open Source	+	NO	YES	YES	YES
Non-Infecting	++	YES	YES	YES	YES
Code Quality	+++	+++	++	++	+
Extensibility	+++	+++	++	++	++
Community	++	NO	YES	YES	YES
Performance / Scalability	++	+++	++	++	++
Reuse of existing developments	+	+++	++	++	++
EU project origin	+	NO	NO	NO	NO
Platform (Portability)	+	+	+	+	+
Open Standards Compliance	+	YES	YES	YES	YES
Interoperability (easy integration for all platforms)	+	+++	+++	+++	+++
Specific Parameters					
Reliability in the Communication with External Services	+++	++	++	++	++
Transformation based on standard	+++	+++	+++	+++	+
XML, CSV, RDF supported	+++	+++	+++	+++	+++
Database, Web Service access supported	+++	YES	YES	YES	YES
Allow semantic annotation of input/output formats.	+++	YES	NO	NO	NO
Web Scraping techniques	+	NO	N/A	N/A	N/A

[54] http://www.springsource.org/spring-integration
[55] https://www.openadaptor.org/

supported					

Figure 77: Parameter Evaluation of Technologies for Content Gateways

3.6.2.1.2 Web Scraping Tool

Web scraping and web crawling are different concepts which are necessary to be explained in order to have a better understanding of web scraping tools. Web scraping is a technique which extracts information from websites and stories it in databases or local machines to exploit it in further processes. The scripts and applications will simulate a person viewing a web site with a browser, thus it is possible to connect to web pages and request their information.

Web crawling is the process of iteratively finding and fetching web links starting from a list of seed URL's. It goes around a website looking at links and building a database of the layout of that site and sites it links to.

In order to implement a web scraping tool, several web scraping and web crawling open-source tools has been considered as base technology. The selected technologies will be extended to conform to the SAM objectives. This comparison takes into account the generic parameters for technology comparison only. The specific parameters will not be taken into account since they relate to the integration tool. The following selection will compare the most promising technologies:

- **Apache Nutch**[56] is an open-source web search engine based on Java. It has a loosely coupled architecture allowing the development of add-ins for media-type parsing, data retrieval, querying and clustering. Nutch is also highly scalable and robust (can run on a cluster of up to 100 machines),can fetch several billion pages per month, maintain an index of these pages, search that index up to 1000 times per second, provide very high quality search results and operate at minimal cost. It also has an active community and is distributed under the Apache License, version 2.0.
- **PHPCrawl**[57] is a framework for crawling websites written in PHP. PHPCrawl crawls websites and passes information about all found documents (pages, links, files and so on) for further processing to users of the library. It provides several options to specify the behaviour of the crawler such as URL- and Content-Type-filters, cookie-handling, robots.txt-handling, limiting options, multiprocessing and much more. PHPCrawl is completely free open-source software and is licensed under the GPL V3. At least PHP 5.2.1 or later version and PHP with OpenSSL support for encrypted connections (https) are necessary to run PHPCrawl in basic single-process-mode.
- **Scrapy**[58] is a fast high-level screen scraping and web crawling framework, used to crawl websites and extract structured data from their pages. It can be used for a wide range of purposes, from data mining to monitoring and automated testing**.** It is an open-source and collaborative framework developed in Python that starts to crawl with a root URL and specific constraints, such as the number of URLs should to fetch. It requires Python 2.7 and works on Linux, Widows and Mac OSX and is available under a BSD license[59]. It provides a substantial documentation[60] and is easy to grasp, making new developments easier.

[56] http://nutch.apache.org/
[57] http://phpcrawl.cuab.de/
[58] http://scrapy.org/
[59] http://www.linfo.org/bsdlicense.html
[60] https://media.readthedocs.org/pdf/scrapy/0.24/scrapy.pdf

- **Web-Harvest 2.0**[61] is an open-source web data extraction tool written in Java. It offers a way to collect desired Web pages and extract useful data from them. In order to do that, it leverages established techniques and technologies for text/xml manipulation such as XSLT, XQuery and Regular Expressions. Web-Harvest mainly focuses on HTML/XML based web sites which still make vast majority of the Web content. On the other hand, it could be easily supplemented by custom Java libraries in order to augment its extraction capabilities. The main goal behind Web-Harvest is to improve the usage of existing extraction technologies. Its purpose is not to propose a new method, but to provide a way to easily use and combine the existing ones. Processors could be combined in a pipeline, making the chain of execution. For easier manipulation and data reuse Web-Harvest provides a variable context where named variables are stored. The result of extraction could be available in files created during execution or from the variable context if Web-Harvest is programmatically used. Web-Harvest is distributed under the BSD License.
- **Snoopy**[62] is a PHP class that simulates a web browser. It automates the task of retrieving web page content and posting forms. It is distributed under GNU General Public License version 2.0 (GPLv2). Snoopy implements features that must be used when imitating a web browser, such as HTTP redirects using regular expressions instead of HTML parsing.

Parameter	Importance	Apache Nutch	Scrapy	PHPCrawl	Web-Harvest	Snoopy
Generic Parameters						
Maturity & Stability	+	+	++	+	+	+
Regularly Updated	+	+	+	+	+	+
Technical Up-to-Datedness / Appeal	+	+	+	+	+	+
Open Source	+++	YES	YES	YES	YES	YES
Non-Infecting	+++	YES	YES	YES	YES	YES
Code Quality	+++	+	++	+	+	+
Extensibility	+++	+	++	+	+	+
Community	++	+/-	+	+/-	+/-	+/-
Performance / Scalability	+	+/-	+/-	+/-	+/-	+/-
Reuse of existing developments	+	NO	NO	NO	NO	NO
EU project origin	+	NO	NO	NO	NO	NO
Platform (Portability)	++	++	++	++	++	++
Open Standards Compliance	++	YES	YES	YES	YES	YES
Interoperability (easy integration for all platforms)	++	++	++	++	++	++

Figure 78: Parameter Evaluation of Web Scraping Tools

[61] http://web-harvest.sourceforge.net/index.php
[62] http://sourceforge.net/projects/snoopy/

3.6.2.2 Technology Selection

3.6.2.2.1 Integration Tool

SAM intends to use the STASIS approach of semantic mapping through TIE TSI (TIE Semantic Integrator), the product TIE has built based on STASIS project and that will be the base for SAM gateways developments (see DOW Section B 1.2.10.2).

After comparing the technologies in the previous section, it can be inferred that all technologies are very similar in almost all parameters with the exception of TIE Semantic Integrator (TSI). The key difference is that it provides semantic mapping services. This feature allows the creation of a mapping between different formats in a semi-automatic way. Based on the semantic entities of the source and destination maps, it provides automatic weighted link suggestions that the user can confirm or discard. Besides, TSI provides a user interface providing a superior usability when compared to the rest of the compared tools.

TSI is completely integrated with TSB (TIE Smart Bridge), the technology selected to implement the Interconnection Bus component (see Section 3.2.2.2), offering a complete semantic data and operational (communications) interoperability framework providing SAM Platform with a complete (data and communication) interoperability solution.

3.6.2.2.2 Web Scraping Tool

Several scrapping tools have been compared in Section3.6.2.1.2, and based on the comparison table (see Figure 78), the best candidate is Scrapy because it combines crawling and scraping techniques. However, it is necessary to experiment more with the different tools in order to verify which one provides the better results. In order to improve the efficiency of the development, it may prove beneficial to combine crawling and scrapping techniques in order to extract information from semi-structured and non-structured information. These tests will be carried out in the task T5.2 Content Gateways.

3.6.2.3 User Interface Technology

As it was described in the functional specification, Section 4.6.1, Content Gateways will implement the Semantic Integrator Editor and Mapping Repository user interfaces. The editor graphical interface will be extended using the tool that TSI already provides. However, the interface of the mapping repository will be implemented as a web interface. Thus, it will be implemented following the technology decision about user interfaces described in Section 3.1.

3.6.3 Technical Component Specification

3.6.3.1 Structure

Figure 79 provides an overview of the Content Gateways, which can also be found in D3.2.1 Global Architecture Definition.

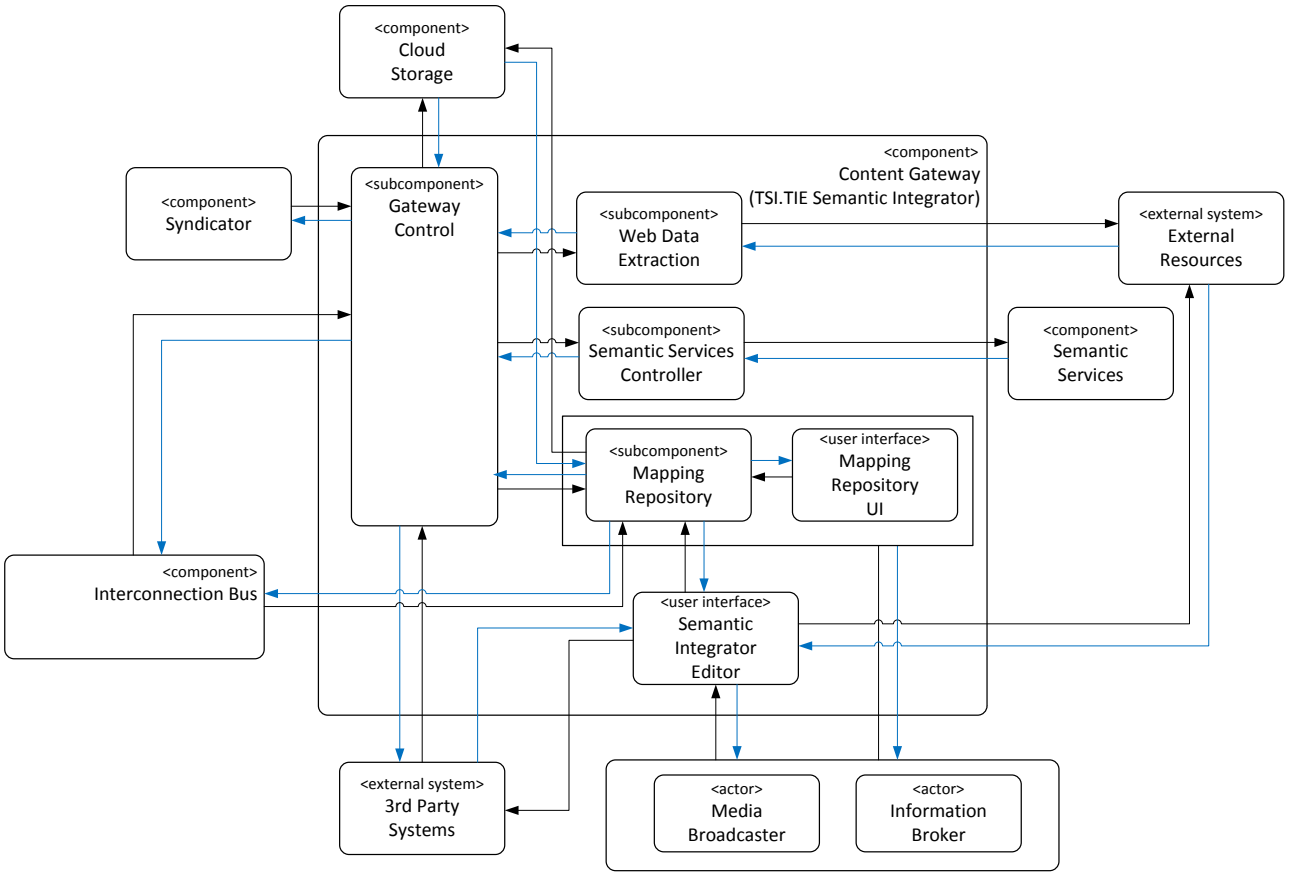

Figure 79: Overview with Selected Technologies – Content Gateways Services

3.6.4 Specification of Interfaces, Protocols and Formats

The services of the Content Gateways will be provided as RESTful interfaces (described in Section 3.6.4.1).

3.6.4.1 RESTful Interfaces

In order to describe the RESTful interfaces, each provided service is described in a separate table. Each table will show a description, URL, request parameters with an example and response type including an example.

3.6.4.1.1 Method “Get External Information”

The RESTful interface “GetExternalInformation” (Figure 80) is used to provide the interaction with Data API Services in order to send information that is linked from external services, such as Wikipedia.

Get External Information		
Description	Provide information that is linked from external services.	
Request		
Request URL	**GET** http://example.com/api/cg/GetExternaInformation:externalLinkInformation	
Resource Parameters	Name	Description
	externalLinkInformation	An object with the necessary external link

		information to extract the information, for instance the URL.
		Example:
		{
		“url”: http://en.wikipedia.org/wiki/Mike_Nielsen,
		”section”: ”references”
		}

Response		
HTTP Status Code	Value	Description
	200	The request was successful
	400	Bad Request
	500	Error during processing

JSON Attributes	template
	JSON Object
		A JSON object with the information.
		Example:
		[
		{
		"description": " 'Live Reviews', Journal of Music",
		“url”: “http://journalofmusic.com/criticism/live-reviews-mike-nielsen-and-louis-winsberg”
		"date": " Retrieved 26 May 2012.”
		},
		{
		"description": " Mike Nielsen', The Irish Times"",
		“url”: “http://www.irishtimes.com/premium/loginpage?destination=http://www.irishtimes.com/culture/music/album-reviews/mike-nielsen-1.563579”
		"date": " Retrieved 26 May 2012.”
		}
		]

Figure 80: RESTful Interface Description – Get External Information

3.6.4.1.2 **Method “Search Importations”**

The RESTful interface “SearchImportations” (Figure 80) is used to search all available importations for a user.

<table>
<tr><th colspan="3">Search Importations</th></tr>
<tr><td>Description</td><td colspan="2">This interface provides a list of available importations for a user.</td></tr>
<tr><td colspan="3">Request</td></tr>
<tr><td>Request URL</td><td colspan="2">GET http://example.com/api/cg/SearchImportations:importationFilter</td></tr>
<tr><td rowspan="2">Resource Parameters</td><td>Name</td><td>Description</td></tr>
<tr><td>importationFilter
Required string</td><td>JSON object with the filter parameters. The value -1 means that the filter is not taken into account. The user ID is a required value.
Example:
{
“userId”: 2345,
“description”: “Import Games Data”,
“date1”: “01/04/2014”,
“date2”: “10/01/2015”,
“active”: “true”
}</td></tr>
<tr><td colspan="3">Response</td></tr>
<tr><td rowspan="4">HTTP Status Code</td><td>Value</td><td>Description</td></tr>
<tr><td>200</td><td>The request was successful</td></tr>
<tr><td>400</td><td>Bad Request</td></tr>
<tr><td>500</td><td>Error during processing</td></tr>
<tr><td>JSON Attributes</td><td>JSON Object</td><td>JSON object with the importations.
Example:
{
“importations”:
[
{
“importationId”: 1,
“description”: “CMS import films”,
“mappingId”: 34,
“mappingDescription”: “mapping1.jse”,
“idWorkflow”: 56
“workflowDescription”: “Worflow 1”,
“schedule”: “Mon-10.00”,</td></tr>
</table>

		“active”: “true”,
		},
		{
		“importationId”: 45,
		“description”: “Import Game datas”,
		“mappingId”: 56,
		“mappingDescription”: “mapping2.jse”,
		“idWorkflow”: 78
		“workflowDescription”: “Worflow 4”,
		“schedule”: “Tue-00.00”,
		“active”: “false”,
		}
		]
		}

JSON Error	JSON Object	Example:
		{
		“errors”:
		[
		{
		“errorId”: 1,
		“description”: “userId is a required value”
		}
		]
		}

Figure 81: RESTful Interface Description – Search Importation

3.6.4.1.3 **Method “Save Importation”**

The RESTful interface “SaveImportation” (Figure 80) is used to save or update in Cloud Storage the information related to an importation.

Save Importation		
Description	Save the importation information in Cloud Storage	
Request		
Request URL	**POST** http://example.com/api/cg/SaveImportation:importationInfo	
Resource Parameters	Name	Description
	importationInfo	JSON object with the importation information.

	Required string	The user ID is a required value.
		If the importation ID is -1means that it is a new importation. In other case means that is an edition and the importation ID should be valid.
		Example:
		{
		“importationId: -1,
		“userId”: 1,
		“description”: “CMS import films”,
		“mappingId”: 34,
		“mappingDescription”: “mapping1.jse”,
		“idWorkflow”: 56
		“workflowDescription”: “Worflow 1”,
		“schedule”: “Mon-10.00”,
		“active”: “true”,
		}

Response		
HTTP Status Code	Value	Description
	200	The request was successful
	400	Bad Request
	500	Error during processing

JSON Attributes	template
	JSON Object
		JSON object with the assets.
		Example 1: Information stored successfully
		{
		“answer”:
		[
		{
		“code”: 1,
		“description”: “Information stored successfully”,
		}
		]
		}
		Example 2: Errors
		{
		“answer”:

		[
		{
		“code”: 2,
		“description”: “userId is a required value”,
		}
		]
		}

Figure 82: RESTful Interface Description – Save Importation

3.6.4.1.4 **Method “Delete Importation”**

The RESTful interface “DeleteImportation” (Figure 80) is used to delete an importation in the Cloud Storage.

Delete Importation		
Description	Delete an importation in the Cloud Storage	
Request		
Request URL	**POST** http://example.com/api/cg/DeleteImportation:importationId	
Resource Parameters	Name	Description
	importationId Required string	String with the importation ID Example: *“34”*
Response		
HTTP Status Code	Value	Description
	200	The request was successful
	400	Bad Request
	500	Error during processing

JSON Attributes	template
	JSON Object
		JSON object with the answer.
		Example 1: Importation deleted successfully
		{
		“answer”:
		[
		{
		“code”: 1,
		“description”: “Importation deleted successfully”,
		}
		]

		}
		Example 2: Errors
		{
		“answer”:
		[
		{
		“code”: 2,
		“description”: “importation ID is a required value”,
		}
		]
		}

Figure 83: RESTful Interface Description – Delete Importation

3.6.4.1.5 **Method “Run Importation”**

The RESTful interface “RunImportation” (Figure 80) is used to run an importation.

Run Importation		
Description	Run an importation	
Request		
Request URL	**POST** http://example.com/api/cg/RunImportation:importationId	
Resource Parameters	Name	Description
	importationId Required string	String with the importation ID Example: *“34”*
Response		
HTTP Status Code	Value	Description
	200	The request was successful
	400	Bad Request
	500	Error during processing
JSON Attributes	template JSON Object	JSON object with the answer. Example 1: Importation executed successfully { “answer”: [{ “code”: 1,

		“description”: “Importation deleted successfully”,
		}
		]
		}
		Example 2: Errors
		{
		“answer”:
		[
		{
		“code”: 2,
		“description”: “importationId is a required value”,
		}
		]
		}

Figure 84: RESTful Interface Description – Run Importation

3.6.4.1.6 **Method “Search Mapping”**

The RESTful interface “SearchMapping” (Figure 80) is used to search all available mappings for a user.

Search Mapping		
Description	This interface provides a list of available mappings for a user.	
Request		
Request URL	**GET** http://example.com/api/cg/SearchMappings:importationFilter	
Resource Parameters	Name	Description

	importationFilter
	Required string
		JSON object with the filter parameters. The value -1 means that the filter is not taken into account. The user ID is a required value
		Example:
		{
		“userId”: 2345,
		“description”: “Import Games Data”,
		“date1”: “01/04/2014”,
		“date2”: “10/01/2015”,
		}

Response		
HTTP Status Code	Value	Description
	200	The request was successful
	400	Bad Request
	500	Error during processing

JSON Attributes	JSON Object	JSON object with the importations.
		Example:
		{
		“mappings”:
		[
		{
		“mappingId”: 1,
		“description”: “CMS import films”,
		“mappingFile”: “film2SAM.jse”,
		“inputFormat”: “MediaBuay Excel”,
		“outputFormat”: “SAM Asset Format”
		},
		{
		“mappingId”: 2,
		“description”: “Games Interna Oracle DB”,
		“mappingFile”: “games2SAM.jse”,
		“inputFormat”: “Oracle Games DB structure”,
		“outputFormat”: “SAM Asset Format”
		}
		]
		}

JSON Error	JSON Object	Example:
		{
		“errors”:
		[
		{
		“errorId”: 1,
		“description”: “user ID is a required value”
		}

		] }

Figure 85: RESTful Interface Description – Search Mapping

3.6.4.1.7 **Method "Upload Mapping"**

The RESTful interface "UploadMapping" (Figure 80) is used to upload a new mapping to the Cloud Storage.

Upload Mapping		
Description	Upload a new mapping to the Cloud Storage.	
Request		
Request URL	**POST** http://example.com/api/cg/UploadMapping:mappingInfo	
Resource Parameters	Name	Description

	mappingInfo
	Required string
		JSON object with the mapping information. The user ID and mappingFile are required values.
		Example:
		{
		"userId": 1,
		"mappingFile": "mapping1.jse"
		}

Response		
HTTP Status Code	Value	Description
	200	The request was successful
	400	Bad Request
	500	Error during processing

JSON Attributes	template
	JSON Object
		JSON object with the assets.
		Example 1: Information stored successfully
		{
		"answer":
		[
		{
		"code": 1,
		"description": "Information stored successfully",
		}
		]

		}
		Example 2: Errors
		{
		“answer”:
		[
		{
		“code”: 2,
		“description”: “userId is a required value”,
		}
		]
		}

Figure 86: RESTful Interface Description – Upload Mapping

3.6.5 Summary

The Content Gateway will use different technologies: An integration tool in order to import/link data from 3rd party systems, a web user interface in order to manage the importations and the mapping repository and finally, it will use a web scraping tool in order to extract information from external resources.

TIE Semantic Integrator (TSI) has been selected as for the integration tool implementation. It is a very mature and reliable solution, which will provide the SAM Platform with a powerful semantic mapping and transformation tool with an advanced user interface providing superior usability when compared to the rest of the available tools. The graphical editor interface will be based on TSI tool but the mapping repository will be implemented as a web interface. Thus, it will be implemented following the technology decision about user interfaces described in Section 3.1.

Finally, several scrapping tools have been compared in Section 3.6.2.2.2 of which the best candidate is Scrapy because it combines crawling and scraping techniques. However, it is necessary to experiment more with the different tools in order to verify which one provides better results.

3.7 Context Control

The Context Control component provides context management and representation functionalities and dynamic community management functionalities as part of the SAM Platform. Figure 87 illustrates the Context Control component embedded within the overall

SAM Platform environment.

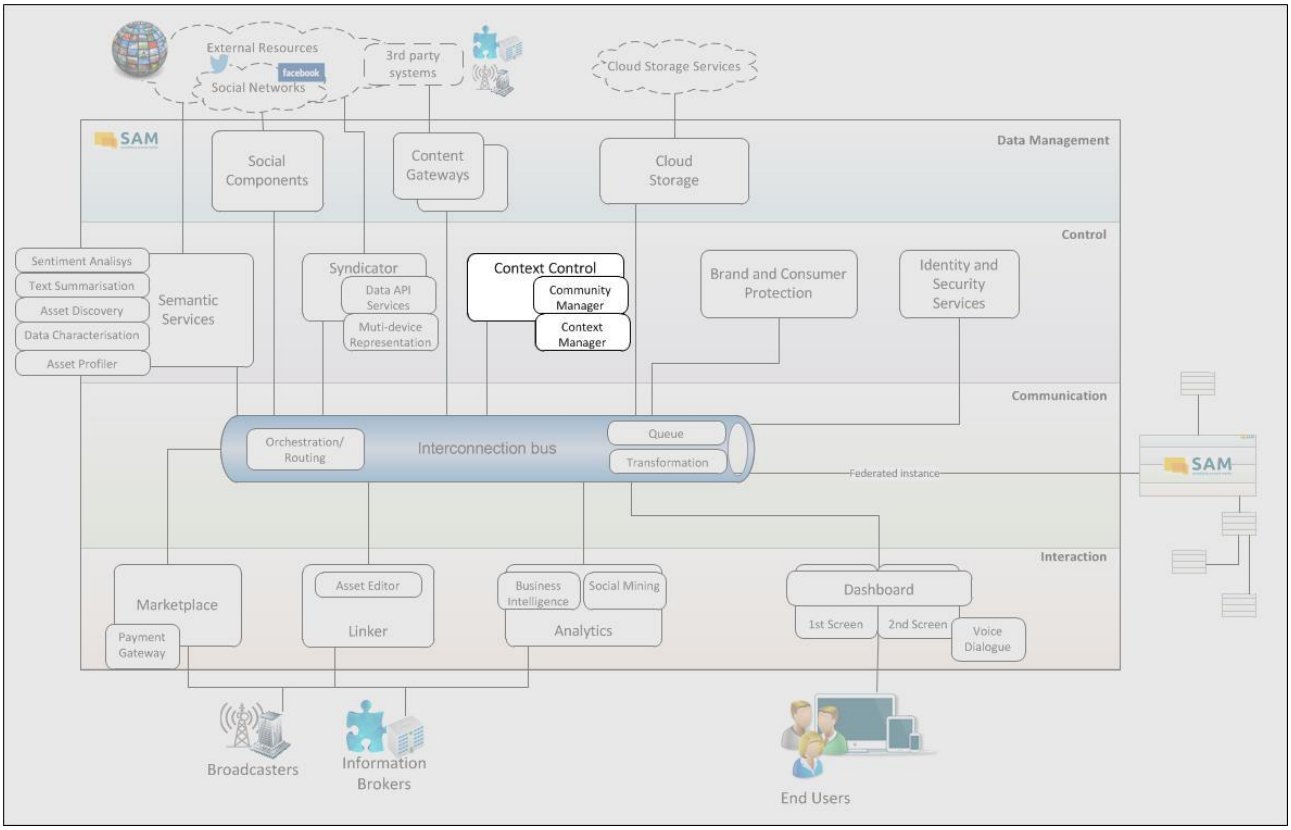

Figure 87: Architecture Overview – Context Control

The Context Control component provides a number of major functionalities: first, the processing, aggregation and representation of context data in context models; second, the analysis and management of dynamic social communities used in SAM. Functionally, these functionalities can be grouped as follows:

- The component receives and processes data provided to it from other relevant components in SAM, most notably the Dashboard and the social components
- The component analyses data that was received and uses relevant data to update context models for the user community members
- The component uses the created context models and other relevant data in order to create and maintain contextual communities of users

The generated context model representation will be used both as input for dynamic community management and as a service accessible to other SAM Platform components. Dynamic community management will interface with the Social Components. The SAM dynamic communities will be integrated as a social networking platform in the same manner as externally connected Social Media platforms will be so that the same API calls can be used regardless of whether the dynamic community or an external one is involved. This ensures that end users will be able to use the same interface components in order to interact with all types of Social Media available directly through SAM.

3.7.1 Major Design Decisions

Major design decisions for the Context Control component concern three distinct design areas: separation of data processing and representation for context modelling, methods to

be applied for dynamic community generation and means by which dynamic communities are managed and presented to users.

Context model data are accessed both by subcomponents within the Context Control component and by other SAM components, most notably the Syndicator component. In order to simplify the interface structure for accessing context model data, both component-internal and external data requests will be handled through the same interface. This interface should be easily accessible via the SAM ESB component and should use a data exchange format that is both flexible and can be used across a wide range of programming language environments.

Concerning methods for the generation and management of dynamic communities, the technical design should facilitate the easy use of different algorithm implementations so that the project developers can evaluate and deploy different types of community analysis algorithms without changing other parts of the component. This is facilitated via a specialised subcomponent “Community Structure Analyser”, for which a common external interface is specified that is to be implemented by different community analyser algorithm implementations.

Since the aim of creating dynamic communities is to make these available to connect users of the SAM Platform, users must be provided with means to interact through dynamic communities. In order to present End Users of SAM with a unified user interaction for Social Media, the presentation of dynamic communities is integrated with the presentation of external third-party Social Media.

3.7.2 Technology Comparison and Selection

3.7.2.1 Possible Technologies and Comparison

The Context Control component and the subcomponents within this component are new developments specifically for the SAM project. To the best of the authors’ knowledge, no existing solutions are available for comparison as far as the overall Context Control component is concerned.

An existing graph library will be used to enable the modelling of Social Media graphs and communities. The following graph libraries have been considered for the Context Control component:

- **JUNG**[63] (Java Universal Network/Graph Framework) is a general graph representation framework that can be used with the Java programming language and supports both graph visualisation and graph modelling.
- **JGraphT**[64] is a library that implements graph representation, analysis and manipulation functionalities in the Java programming language; the focus of JGraphT lies on the logical representation, analysis and manipulation of graph data structures.
- **GraphStream**[65] is a graph management library for the representation and manipulation of graphs, with focus on dynamic graphs. It has been implemented in Java.
- **Grph**[66] is a high-performance library for the representation and manipulation of graphs, implemented in Java.

[63] http://jung.sourceforge.net/
[64] http://jgrapht.org/
[65] http://graphstream-project.org/
[66] http://www.i3s.unice.fr/~hogie/grph/

The table below shows a comparison of these graph libraries concerning the relevant generic and specific parameters of interest.

Parameter	Importance	JUNG	JGraphT	GraphStream	Grph
Generic Parameters					
Maturity & Stability	+++	+++	+	+/-	+
Regularly Updated	-	++	+	+/-	+/-
Technical Up-to-Datedness / Appeal	+/-	+/-	+/-	+/-	+/-
Open Source	+++	YES	YES	YES	YES
Non-Infecting	+++	YES	YES	NO	YES
Code Quality	+++	++	++	+/-	++
Extensibility	+++	+++	++	+	++
Community	+/-	+++	+/-	+/-	+/-
Performance / Scalability	+++	+	+	+	+++
Reuse of existing developments	+/-	+/-	+/-	+/-	+/-
EU project origin	+/-	No	No	No	No
Platform (Portability)	-	-	-	-	-
Open Standards Compliance	-	No (n/a)	No (n/a)	No (n/a)	No (n/a)
Interoperability (easy integration for all platforms)	-	-	-	-	-
Specific Parameters					
Representation of directed graphs	+++	+++	+++	+++	+++
Representation of undirected graphs	+	+++	+++	+++	+++
Integration of graph analysis algorithms	+++	+++	+++	+	++
Integration of graph analysis clustering algorithms	++	+++	---	---	++
Graph manipulation functionalities	++	+++	+++	+	+

Figure 88: Parameter Evaluation of Technologies for Context Control Graph Representation

3.7.2.2 Technology Selection

The system implementation will be carried out using the Java programming language, which has been successfully applied in similar development tasks and for which a high level of expertise is available at the developer organisations.

The graph representation and manipulation will be carried out using the JUNG library and in Java, as the library supports all required functionalities and the developers involved in the work are already familiar with the library.

3.7.3 Technical Component Specification

3.7.3.1 Structure

Figure 89 provides an overview of the Context Control. A more elaborate description of the component can be found in the SAM Deliverable D3.2.1, Section 4.7. The diagram has remained unchanged as no changes to the Context Control component have been made since the specification in Deliverable D3.2.1.

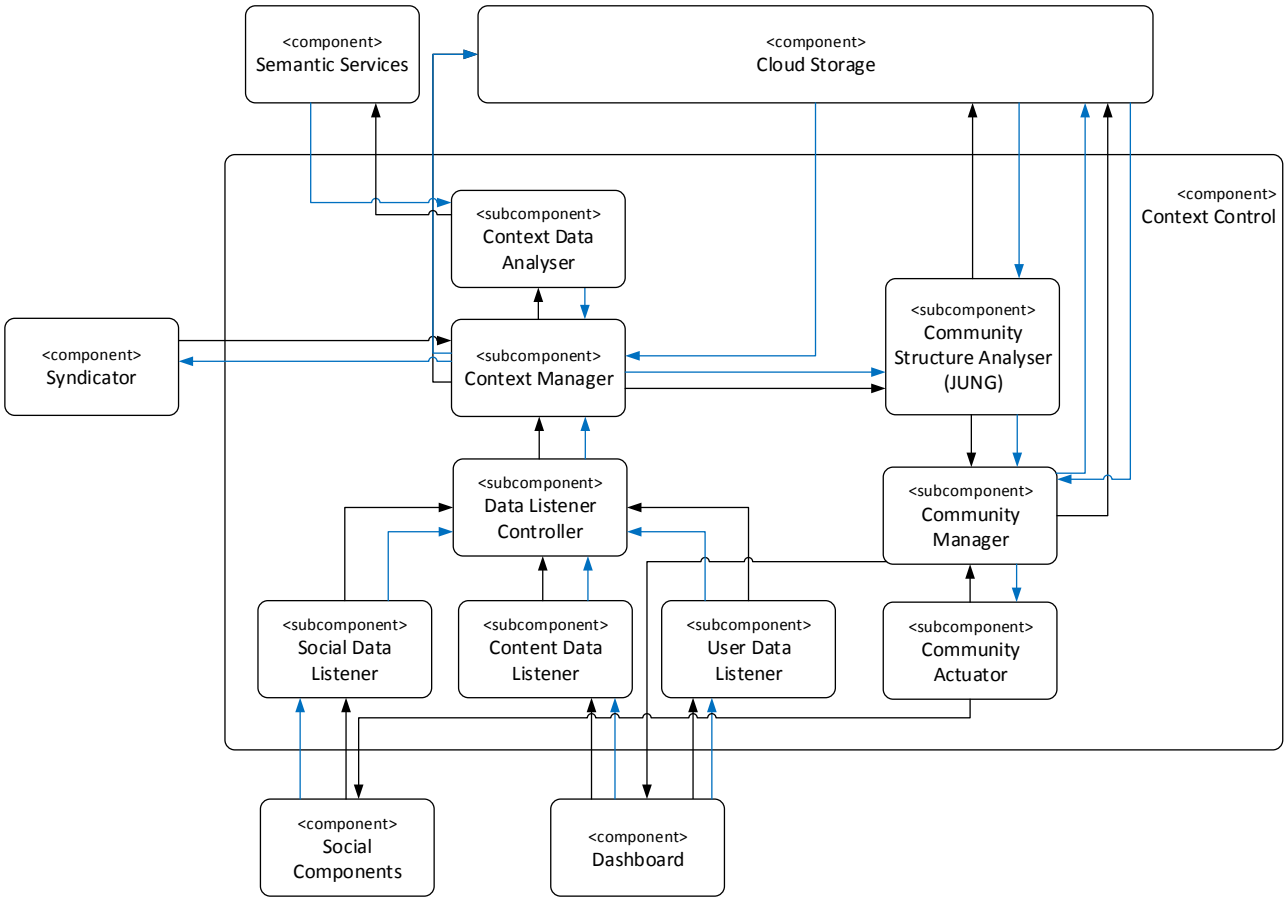

Figure 89: Overview with Selected Technologies – Context Control

3.7.4 Specification of Interfaces, Protocols and Formats

The functionalities of the Context Control component will be exposed to other SAM components through a RESTful Web Service API (described in Section 3.7.4.1).

3.7.4.1 RESTful Interfaces

In order to describe the RESTful interfaces, each provided service is described in a separate table. Each table will show a description, a URL, request parameters with an example and the response type including an example.

3.7.4.1.1 Method “Ingest Social Data”

The RESTful interface “Ingest Social Data” (Figure 100) receives and processes Social Media data provided to the Context Control component. Whether and how the data that is received is used depends on the nature of the input data and is determined solely within the Context Control component.

Ingest Social Data	
Description	Evaluates Social Media data submitted to the method; determines whether and how relevant context models and dynamic social community activities need to be updated or used otherwise.
Request	
Request	**POST**

URL	http://example.com/api/cc/source/social/source:sourceId/channel:channelId	
Resource Parameters	Name	Description
	sourceId Required string	ID of the communication source Example: “*twitter”*
	channelId Optional string	ID of the channel or user from which the message was received Example: “*@testuser123”*
HTTP Parameters	dataType Required string	Data type of the transferred object; the data type must be specified and it must be specified as JSON (preparation for possible support of other input formats) Example: “*JSON”*

	object
	Required object
		JSON object representing the relevant message data.
		Example:
		{
		"messageType" : "tweet",
		"messageContent" : "Hello! My name is Elder Price."
		}

Response		
HTTP Status Code	Value	Description
	200	Object received and processed
	400	Object does not conform to content message formatting conventions
	403	Communication source is unknown
	500	Component encountered an error while processing object

Figure 90: RESTful Interface Description – Process Social Data

3.7.4.1.2 **Method “Ingest Content Data”**

The RESTful interface “Ingest Content Data” (Figure 91) receives and processes content data provided to the Context Control component. Whether and how the data that is received is used depends on the nature of the input data and is determined solely within the Context Control component.

Ingest Content Data	
Description	Evaluates content data submitted to the method; determines whether and how relevant context models and dynamic social community

	activities need to be updated or used otherwise.	
Request		
Request URL	**POST** http://example.com/api/cc/source/content/asset:assetId/user:userId	
Resource Parameters	Name	Description
	assetId Required string	ID of the asset of concern Example: “*videoABC”*
	userId Required string	ID of the user who accesses the asset Example: “*testuser123”*
HTTP Parameters	dataType Required string	Data type of the transferred object; the data type must be specified and it must be specified as JSON (preparation for possible support of other input formats) Example: “*JSON”*

	object
	Required object
		JSON object representing the relevant content data and metadata.
		Example:
		{
		"assetType": "video",
		"assetTitle": "Hello! My name is Elder Price."
		}

Response		
HTTP Status Code	Value	Description
	200	Object received and processed
	400	Object does not conform to content message formatting conventions
	403	Communication source is unknown
	500	Component encountered an error while processing object

Figure 91: RESTful Interface Description – Process Content Data

3.7.4.1.3 **Method “Ingest User Data”**

The RESTful interface “Ingest User Data” (Figure 92) receives and processes user data provided to the Context Control component. Whether and how the data that is received is used depends on the nature of the input data and is determined solely within the Context Control component.

Ingest User Data	
Description	Evaluates user data submitted to the method; determines whether and how relevant context models and dynamic social community activities

	need to be updated or used otherwise.	
Request		
Request URL	**POST** http://example.com/api/cc/source/user/user:userId/comp:compId	
Resource Parameters	Name	Description
	userId Required string	ID of the user from whom the data originates Example: “*testuser123”*
	compId Optional string	ID of the component from which the user data were received Example: “*componentXYZ”*
HTTP Parameters	dataType Required string	Data type of the transferred object; the data type must be specified and it must be specified as JSON (preparation for possible support of other input formats) Example: “*JSON”*

	object
	Required object
		JSON object representing the relevant user data and metadata.
		Example:
		{
		"actionType": "join",
		"actionTarget": "dynamicCommunity"
		"actionValue": "Fans of Orlando"
		}

Response		
HTTP Status Code	Value	Description
	200	Object received and processed
	400	Object does not conform to content message formatting conventions
	403	Communication source is unknown
	500	Component encountered an error while processing object

Figure 92: RESTful Interface Description – Process User Data

3.7.4.1.4 **Method “Query User Context”**

The RESTful interface “Query User Context” (Figure 93) exposes a functionality to retrieve the context model of a user identified by a user ID in JSON format.

Query User Context	
Description	Returns a user context model for a given user ID.

Request		
Request URL	**GET** http://example.com/api/cc/query/user:userId	
Resource Parameters	Name	Description
	userId Required string	ID of the user for whom the context model is to be retrieved Example: “*testuser123”*
Response		
HTTP Status Code	Value	Description
	200	request received and processed
	403	User ID is unknown
	500	Component encountered an error while processing object

JSON Attributes	list
	Optional list
		A JSON object matching the query.
		Example:
		[
		{
		"key": "myKey",
		"value": "TestValue”
		}
		]

Figure 93: RESTful Interface Description – Query User Context

3.7.5 Summary

This section, which describes the Context Control component, focuses on the external interfaces for the component and describes the key interfaces that need to be made available to third parties as part of the component API.

The API focuses on the submission of data to the component and on a basic method for retrieving data from the component. External means through which the component effects changes to dynamic social communities are not part of this API, because these activities are carried out by using functionalities of the Social Components.

The component will be implemented in Java and will use the well-known JUNG graph library for many of the basic dynamic community functionalities.

3.8 Cloud Storage

The Cloud Storage is the central point for all data in the SAM platform (see Figure 94). It abstracts different storage technologies to provide a unified storage service to other components. These services will be provided as a web services to any legitimate user of the SAM platform.

- The Cloud Storage will offer persistent storage for all SAM components. These components will have access to different types of data storage systems, so the data will be saved in the best possible manner. Therefore the Cloud Storage is the central storage point of all data in the SAM platform.
- The Cloud Storage will support different types of databases, namely semi-structured, semantic, binary and relational. These databases can be added to the Cloud Storage as required using the DB Management user interface.
- The Cloud Storage will also provide concrete implementations of API Wrapper for different target platforms in order to easily access functionality provided.

Since the SAM platform will handle and process sensitive data (such as user credentials and rights protected data) data privacy and security is of the utmost importance. The Cloud Storage will provide complete separated storage areas as so called Buckets. Each Bucket can be utilised to create isolated isles of data. By using different Buckets for each component, the confidential data of one component is stored completely separated from the data of another component.

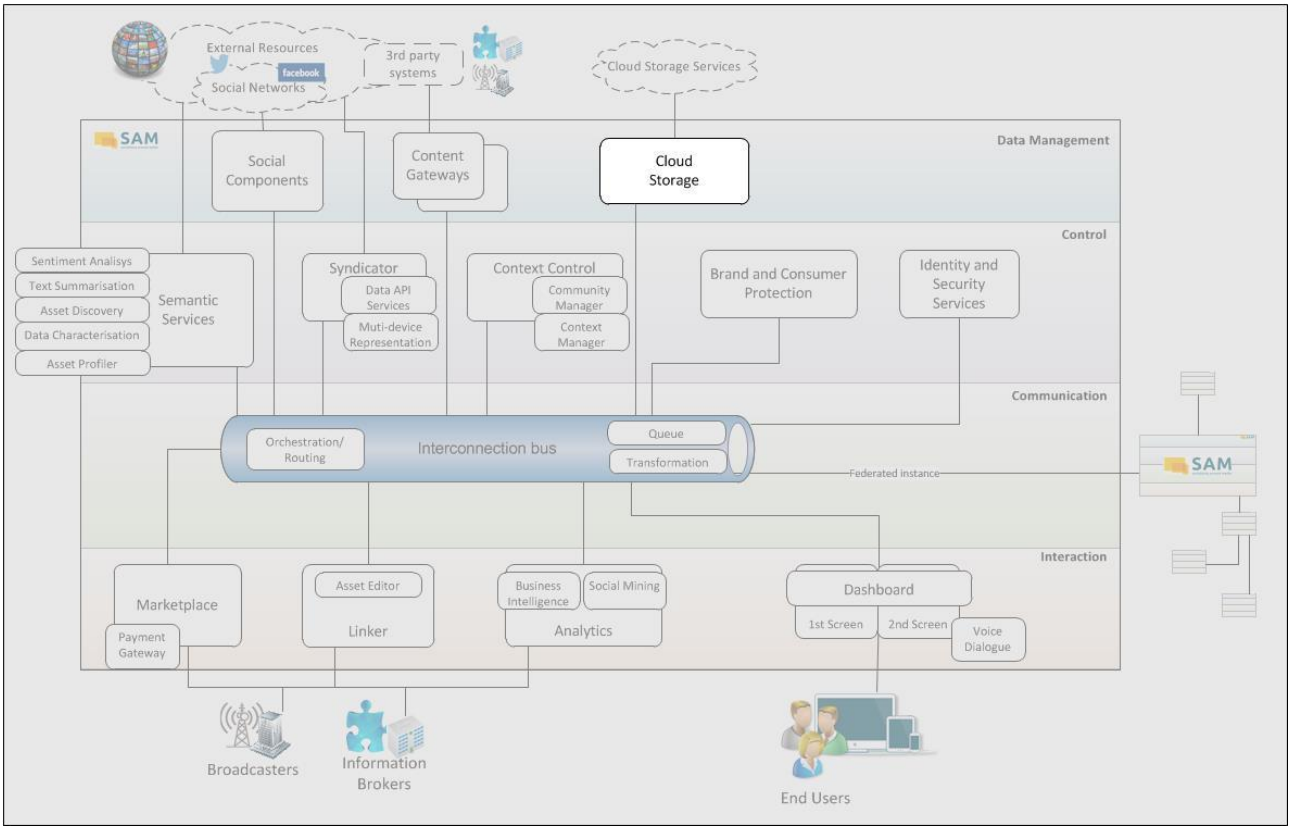

Figure 94: Architecture Overview – Cloud Storage

3.8.1 Major Design Decisions

The focus of the Cloud Storage is to provide all SAM components with secure access to a persistent storage and the therein saved data. Due to the possible diversity of data, the data will be stored encapsulated using data types wrappers. Providing different data type wrapper the Cloud Storage will offer simple CRUD (Create, Read, Update, Delete) operations for all different data types e.g., returning values of a data set or updating properties of a data entry. Additionally, the Cloud Storage will provide more advanced query capabilities based on specific data types as different data types have different requirements. For example, storing large amounts of binary data files requires optimal data

throughput while a semantic database needs to enable graph-based queries, but data throughput is a negligible factor. Consequently, several technologies will be utilised to support the different data types.

The sensitivity level of the Buckets will be implemented with Access Control List (ACL). Every actor of the SAM platform can create Buckets in order to store data and utilise the ACL to specify exactly which other user or user groups can access their Bucket. One component may use several different Buckets and may specify their access rights (Denied, Read, Write, Super, see also Section 3.8.3.1.1) to share data with other components.

Each component may create multiple Buckets for storing data, which are fully separated and not influenced by activities of other Buckets.

In order to handle the different data types in the best way, four different concepts to store data are foreseen. Each of them will utilise a different technology:

- **Semi-Structured Data** will be used to store data in a document-oriented way without a fixed data schema. It is usually referred to as “NoSQL” (Not only Structured Query Language) storage and may be realised by technologies such as CouchDB or Apache Cassandra. A typical data entry in this storage will have an ID and a set of key-value entries, sometimes being hierarchical.
- **Binary Data** will allow a file-centric storage for binary data. It may be used to, e.g., store videos, PDFs or any other type or binary data. Queries will be based on the file name or ID, e.g., by requesting the content of document “brochure.pdf”. Potential base technologies include Amazon S3 or (distributed) file systems.
- **Structured Data** will allow the storage of data which follow a fixed schema. Relational database management systems (RDMS) are using tables where the schema will control the type of data stored in each column and the rows will contain the stored data. The Structured Query Language (SQL) will be used to query this kind of databases.
- **Semantic Data** will allow the storage of semantic information. Queries can be based on a semantic query language such as SPARQL. Possible base technologies include Jena or Sesame.

3.8.2 Technology Comparison and Selection

This subsection outlines technology selection criteria and compares existing technologies – potential candidates for the realisation of Cloud Storage. Within the subsection, the areas that need to be implemented or improved to fully meet SAM specific requirements are also identified and presented. The selected technologies will be used as a base for the development phase in the realisation of the technical design of this component: The specific selection criteria to be used are specified in the document D3.2.2 Functional Specification, Section 4.8.5 and will be used in the next sections.

3.8.2.1 Possible Technologies and Comparison

In order to realise the Cloud Storage component, several options have to be considered as base technology. The assessment of these technologies is presented in the following sections.

3.8.2.1.1 Semi-Structured Data Storage Technologies

Semi-structured data can be used to store data in a document-oriented way without a fixed data schema. It is often referred to as “NoSQL” (Not only Structured Query Language) storage. A typical data entry in this storage will have an ID and a set of key-value entries,

sometimes being hierarchical. The following databases can be utilised to store semi-structured data:

- **Amazon DynamoDB**[67] **and SimpleDB**[68] are NoSQL database services. They are easy to use and have a very good scalability. In comparison to SimpleDB, DynamoDB is faster, more flexible and there are no restrictions on the capacity. For this reason, DynamoDB will be used for further comparisons. The core idea of these services is based on distributed hash tables to ensure a high availability as well an incremental scalability. As an externally hosted solution, the use of DynamoDB would lead to additional cost.
- **Microsoft Azure Table Storage Service**[69] is a NoSQL database service. It serves access over a RESTful interface or libraries for several programming languages such as .Net, Java and PHP. It allows having multiple values for an attribute; the attributes are differentiated by a timestamp, allowing the storage of temporal data. As an externally hosted solution, the use of the Azure Table Storage would lead to additional cost.
- **MongoDB**[70] is schema-less open-source database that is document-oriented. The documents are represented internally with the help of the BSJON specification, which leads to a good performance. MongoDB is written in C++ and is available for Windows, Linux, OS X and Solaris. It offers libraries for many programming languages such as .Net, C++, Java, PHP. It contains a master-slave replication mechanism and can be run on Microsoft Azure or other hosted servers and is a cost-free solution. MongoDB is a good option for applications that require operation intelligence, flexibility in data management and applications that focus on content such as event logging, metadata management, inventory management, product catalogues and e-commerce applications. Key features of MongoDB include indexing, ad-hoc queries, replication, load balancing, auto-sharding, MapReduce jobs and high availability.
- **Apache CouchDB**[71] is a document-based database that can be accessed by a RESTful interface. It is written in Erlang and has a plug-in-architecture, which allows adding extensions. It contains a master-master merge replication mechanism. It can be hosted on-premises servers to avoid third-party hosting costs. CouchDB is a good option for applications that require aggregation of frequently changing data such as contacts, invoices, receipts, etc. Key features of CouchDB include on-the-fly document transformation, real-time change notifications, high availability, partition tolerance, eventual consistency, ACID semantics, MapReduce jobs, master set-ups and automatic conflict detection.
- **Apache Cassandra**[72] is an open-source, distributed database, which can be considered as a hybrid of key-value and column-oriented nature. Cassandra is a good option for applications that deal with mission-critical data and require failure tolerance and high write throughput while not sacrificing read efficiency. Cassandra can run on top of multiple commodity servers, offering high availability, low latency for operations and no single point of failure. The key features of Cassandra include linear-scale performance, ease of data storage and distribution, elastic scalability for read/write operations, fault tolerance and simplicity in its configurability.

[67] http://aws.amazon.com/de/dynamodb/
[68] http://aws.amazon.com/de/simpledb/
[69] http://www.windowsazure.com/en-us/
[70] http://www.mongodb.org/
[71] http://couchdb.apache.org/
[72] http://cassandra.apache.org/

Parameter	Importance	Amazon DynamoDB	MS Azure Table	MongoDB	Apache CoucheDB	Apache Cassandra
Generic Parameters						
Maturity & Stability	+++	+++	+++	+++	+++	+++
Regularly Updated	+/-	++	++	++	++	++
Technical Up-to-Datedness / Appeal	++	++	++	++	+	+
Open Source	+	NO	NO	YES	YES	YES
Non-Infecting	+	NO	NO	YES	YES	YES
Code Quality	++	N/A	N/A	N/A	N/A	N/A
Extensibility	+++	---	---	+++	++	+++
Community	-	+/-	+	+++	++	++
Performance / Scalability	+++	+++	+++	+++	+++	+++
Reuse of existing developments	+/-	+++	+++	+++	+++	+++
EU project origin	---	NO	NO	NO	NO	NO
Platform (Portability)	+	N/A	N/A	+++	++	+++
Open Standards Compliance	+	N/A	N/A	NO	NO	NO
Interoperability (easy integration for all platforms)	++	+++	+++	+++	++	+/-
Specific Parameters						
Availability	++	+++	+++	++	++	++
Reliable Access Rights	+++	+++	+++	+++	++	++
Backup	+/-	N/A	N/A	N/A	N/A	N/A
ACID compliant	+++	N/A	N/A	N/A	N/A	N/A
CRUD operations	+++	+++	+++	+++	+++	+++

Figure 95: Parameter Evaluation of Technologies for Semi-Structured Data Storage

3.8.2.1.2 Semantic Data Storage Technologies

Semantic data represent information stored using the RDF data model. This data model uses a subject-predicate-object structure to represent specific information. An example could be “SAM is a project”, where “SAM” denotes the subject, “is a” denotes the predicate and “project” denotes the object. Using this structure complex relations can be expressed.

Semantic databases can be queried by semantic query languages such as SPARQL. The following technologies are considered to store the semantic data:

- **Sesame**[73] is an open-source de facto standard framework for RDF data. Its supports a variety of different storage devices due to a generic architecture, which abstracts the storage device layer, called SAIL (Storage And Interface Layer). It runs in a Java virtual machine and supports the following relational DBMS to store its data: PostgreSQL, MySQL, Microsoft SQL Server or Oracle. Sesame supports two query languages, SeRQL and SPARQL. As an open-source product, Sesame is free of charge and can be hosted on any server running Java.
- **Virtuoso**[74] is a multi-model cross-platform data server. Noteworthy of Virtuoso is that it stores its data in tuples. This allows also RDF data management with SPARQL support. Successfully used in various projects and products, this data server can store

[73] http://www.openrdf.org/
[74] http://virtuoso.openlinksw.com/

different types of data. As it is cross-platform and a free community edition is provided, it can be hosted on on-premise servers.

- **AllegroGraph**[75] is a commercial RDF database. It exists as a free version with a limitation of 5 million triples. It runs on Windows, Linux and OS X and provides libraries for several programming languages including C# and Java. So it can be hosted on a local server to reduce the cost of the storage system.
- **Jena**[76] is an open-source RDF framework. It supports SPARQL as query languageJena provides a good support for reasoning, but therefore the scalability suffers. For data storage, it can use various storage solutions like SQL databases. As an open-source product, it can be used free of charge and installed on local servers.
- **BrightstarDB**[77] is a schema-free RDF store for the .Net platform. It supports SPARQL 1.1 and OData and it is ACID (Atomicity, Consistency, Isolation, and Durability) compliant. It can be hosted on a local Windows server and is downloadable for free.

Parameter	Importance	Sesame	Virtuoso	Allegro Graph	Jena	Brightstar DB
Generic Parameters						
Maturity & Stability	+++	+++	+++	+++	+++	+
Regularly Updated	+/-	+	+	+	+	+
Technical Up-to-Datedness / Appeal	++	++	++	++	++	++
Open Source	+	YES	YES	NO	YES	YES
Non-Infecting	+	YES	NO	N/A	YES	YES
Code Quality	++	N/A	N/A	N/A	N/A	N/A
Extensibility	+++	++	+	---	++	++
Community	-	++	+	+/-	++	-
Performance / Scalability	+++	+++	+++	+++	+++	+++
Reuse of existing developments	+/-	+++	++	+	+	++
EU project origin	---	NO	NO	NO	NO	NO
Platform (Portability)	+	+++	+++	+++	+++	-
Open Standards Compliance	+	NO	NO	NO	NO	NO
Interoperability (easy integration for all platforms)	++	+++	++	+++	+++	-
Specific Parameters						
Availability	++	++	+	+	+	+
Reliable Access Rights	+++	++	+	++	+	++
SparQL support (Semantic storage)	+++	YES	YES	YES	YES	YES
Backup	+/-	-	-	+	-	-
ACID compliant	+++	N/A	N/A	N/A	N/A	+++
CRUD operations	+++	+++	+++	+++	+++	+++

Figure 96: Parameter Evaluation of Technologies for Semantic Data Storage

3.8.2.1.3 Structured Data Storage Technologies

Structured data will represent data that is manageable in tables and rows, which are typical to relational databases. As such, the Cloud Storage will reuse a relational database

[75] http://www.franz.com/agraph/
[76] http://jena.apache.org/
[77] https://brightstardb.com/

technology as the basis for its data management. For managing structured information, the following technologies are compared:

- **MySQL**[78] is the world's most-used open-source relational database management system. It is very stable and runs on several operation systems such as Mac OX, Linux, Windows, i5/OS and even Symbian. Additionally, many tools exist to manage it.
- **Amazon Relational Database Service (RDS)**[79] is a public cloud installation of MySQL offered by Amazon. It is installed at an Amazon EC2 instance and differs from a local installed version only in support, which is carried by Amazon.
- **Google Cloud SQL**[80] is part of Google's App Engine, which offers a MySQL database with JDBC support. All administration and maintenance activities are done by Google, which ensures high reliability and availability by data replication to multiple data centres.
- **Microsoft SQL Server**[81] is a commercial relational database management system. Several editions are available, including a free Express Edition with limited functionality. It additionally supports ANSI SQL Transact-SQL, an extended version of SQL that supports procedural programming.
- **Microsoft Azure SQL database**[82] is a cloud-based rational database offered by Microsoft. It is based on MS SQL server technology. The data is replicated to multiple servers and the storage scales to the user needs.
- **PostgreSQL**[83] is an object-relational database management system, which is available for several systems.

Parameter	Importance	MySQL	Amazon RDS	Google Cloud SQL	MS SQL Server	MS Azure SQL DB	Postgre SQL
Generic Parameters							
Maturity & Stability	+++	+++	+++	+++	+++	+++	+++
Regularly Updated	+/-	+++	+++	+++	+++	+++	+++
Technical Up-to-Datedness / Appeal	++	+++	+++	+++	+++	+++	++
Open Source	+	YES	YES	YES	NO	NO	YES
Non-Infecting	+	NO	NO	NO	NO	NO	YES
Code Quality	++	N/A	N/A	N/A	N/A	N/A	N/A
Extensibility	+++	+/-	+/-	---	---	---	++
Community	-	+++	+++	+++	+++	++	+++
Performance / Scalability	+++	+++	+++	+++	+++	+++	++
Reuse of existing developments	+/-	+++	+++	+++	+++	+++	+++
EU project origin	---	NO	NO	NO	NO	NO	NO
Platform (Portability)	+	+++	N/A	+++	---	N/A	+++
Open Standards Compliance	+	+++	N/A	+++	+	N/A	++
Interoperability (easy integration for all platforms)	++	+++	+++	+++	+	+	++
Specific Parameters							
Availability	++	++	+++	+++	++	+++	++

[78] http://www.mysql.de/
[79] http://aws.amazon.com/de/rds/
[80] https://developers.google.com/cloud-sql/
[81] http://www.microsoft.com/en-gb/server-cloud/products/sql-server/default.aspx
[82] http://azure.microsoft.com/en-gb/
[83] http://www.postgresql.org/

Reliable Access Rights	+++	+++	+++	+++	+++	+++	+++
SQL support (relational storage)	+++	+++	+++	+++	+++	+++	+++
Backup	+/-	+/-	++	+/-	+/-	+/-	+/-
ACID compliant	+++	+++	+++	+++	+++	+++	+++
CRUD operations	+++	+++	+++	+++	+++	+++	+++

Figure 97: Parameter Evaluation of Technologies for Structured Data Storage

3.8.2.1.4 Binary Data Storage Technologies

Binary data contains data stored in files such as documents, images or configuration files of applications. For that reason, a technology is needed which offers an easy to use and scalable storage. For storing binary data, the following technologies are compared:

- **Amazon Simple Storage Service (S3)**[84] is a key-value storage which is accessible through web service interfaces like REST and SOAP. Amazon provides scalability and a high availability. Amazon even promises an availability of 99.99%. It is a fully managed and externally hosted service and so the usage would lead to additional cost for the project.
- **Microsoft Azure Blob Storage Service**[85] offers a service to store large binary files in the Cloud. All stored files are replicated automatically. The storage is available via web service interfaces. The usage is not free of charge since this is a fully managed and hosted solution.
- A **File Server** is a computer which provides a shared access to storage resources. It is attached to a network and accessed by attached workstations. It could be installed in a dedicated or in a non-dedicated manner. In the first case, the dedicated server is specially configured as file server and does not offer any other functionality. In the second case, the server is used in parallel for other tasks. This can lead to worse performance. There are several ways and protocols to access the data of a file server regardless of the manner it is installed, e.g., FTP, SFTP, or HTTP. As a technology and not a product, this is free of charge and can be hosted externally or internally.
- **Apache SVN**[86] (as representative for the wide range of versioning systems) is an open-source versioning system which is distributed under an Apache License. It allows maintaining different versions of source code or documents. A SVN repository can also installed on an Amazon EC2 instance, so it could be used in local and cloud environments. Apache SVN is open-source software and so can be used for free on the SAM server.

Parameter	Importance	Amazon S3	MS Azure Blob Storage Service	File Server
Generic Parameters				
Maturity & Stability	+++	+++	+++	++
Regularly Updated	+/-	+++	+++	++
Technical Up-to-Datedness / Appeal	++	+++	+++	+++
Open Source	+	NO	NO	N/A
Non-Infecting	+	N/A	N/A	N/A
Code Quality	++	N/A	N/A	N/A
Extensibility	+++	+++	+++	+/-

[84] http://aws.amazon.com/en/s3/
[85] http://www.windowsazure.com/en-us/
[86] http://subversion.apache.org/

Community	-		+	+/-
Performance / Scalability	+++	++	++	-
Reuse of existing developments	+/-	+++	+	++
EU project origin	---	NO	NO	NO
Platform (Portability)	+	+++	+/-	+
Open Standards Compliance	+	+++	+++	+/-
Interoperability (easy integration for all platforms)	++	+++	++	+/-
Specific Parameters				
Availability	++	+++	+++	+
Reliable Access Rights	+++	+++	++	+
Backup	+/-	+++	+++	-
ACID compliant	+++	N/A	N/A	N/A
CRUD operations	+++	+++	+++	+++

Figure 98: Parameter Evaluation of Technologies for Binary Data Storage

3.8.2.2 Technology Selection

After comparing the technologies in the previous section, the selected technology for each data backend is explained in this section.

3.8.2.2.1 Semi-Structured Data Storage

MongoDB was selected as semi-structured data storage, because it is very stable and has a good runtime performance. It is internally in use by at least one project partner that is involved in the development of the Cloud Storage. Thus, it is well-tested and the necessary know-how is already available. MongoDB is an open-source solution and uses the GNU Affero General Public License (AGPL), a non-infecting license. It is available for Windows, Linux, OS X and Solaris and drivers exist for many programming languages. The number of drivers continues to grow, because the community develops new ones for further programming languages. It can be run on its own server, in its own private cloud or on Microsoft Azure as a public cloud. The database supports replications by itself, so that the synchronisation is fully integrated.

3.8.2.2.2 Semantic Data Storage

Sesame was selected to store semantic data, because it is a de-facto standard to store RDF data. It works with several relational DBMS and is platform-independent. It has better performance than Jena for large amounts of data and better stability than BrightstarDB. It is open-source and uses the BSD 3-Clause license, a non-infecting license.

3.8.2.2.3 Structured Data Storage

MySQL will be used to store structured data, because it is very stable, well tested and open-source. Additionally, it is possible to easily migrate the data between different Cloud Providers and local systems.

3.8.2.2.4 Binary Data Storage

Amazon S3 was selected as binary storage, because it offers a good interoperability through standardised interfaces like REST and SOAP. It offers high availability and scalability. The account on the services is performed in a pay-as-you-use manner and thus

there are no up-front costs for initial hardware purchase. Also, through the managed service, costs for storage extensions and administrative work can be avoided, which is a big advantage over a self-hosted “free” network storage system. Furthermore, it is a broadly established solution for binary data storage and is well documented and features a large community.

3.8.2.3 User Interface Technology

This component has to provide the DB Management user interface so the administrator is able to manage connected databases and containing data. The technology used for realisation has been selected in Section 3.1.

3.8.3 Technical Component Specification

3.8.3.1 Structure

Figure 99 provides an overview of the Cloud Storage, which can be found also in D3.2.1 Global Architecture Definition. Additionally, this version of the overview in this chapter has been augmented with the selected technologies. Each Bucket can contain data objects of the required data type. Each SAM component can create additional Buckets to store new data.

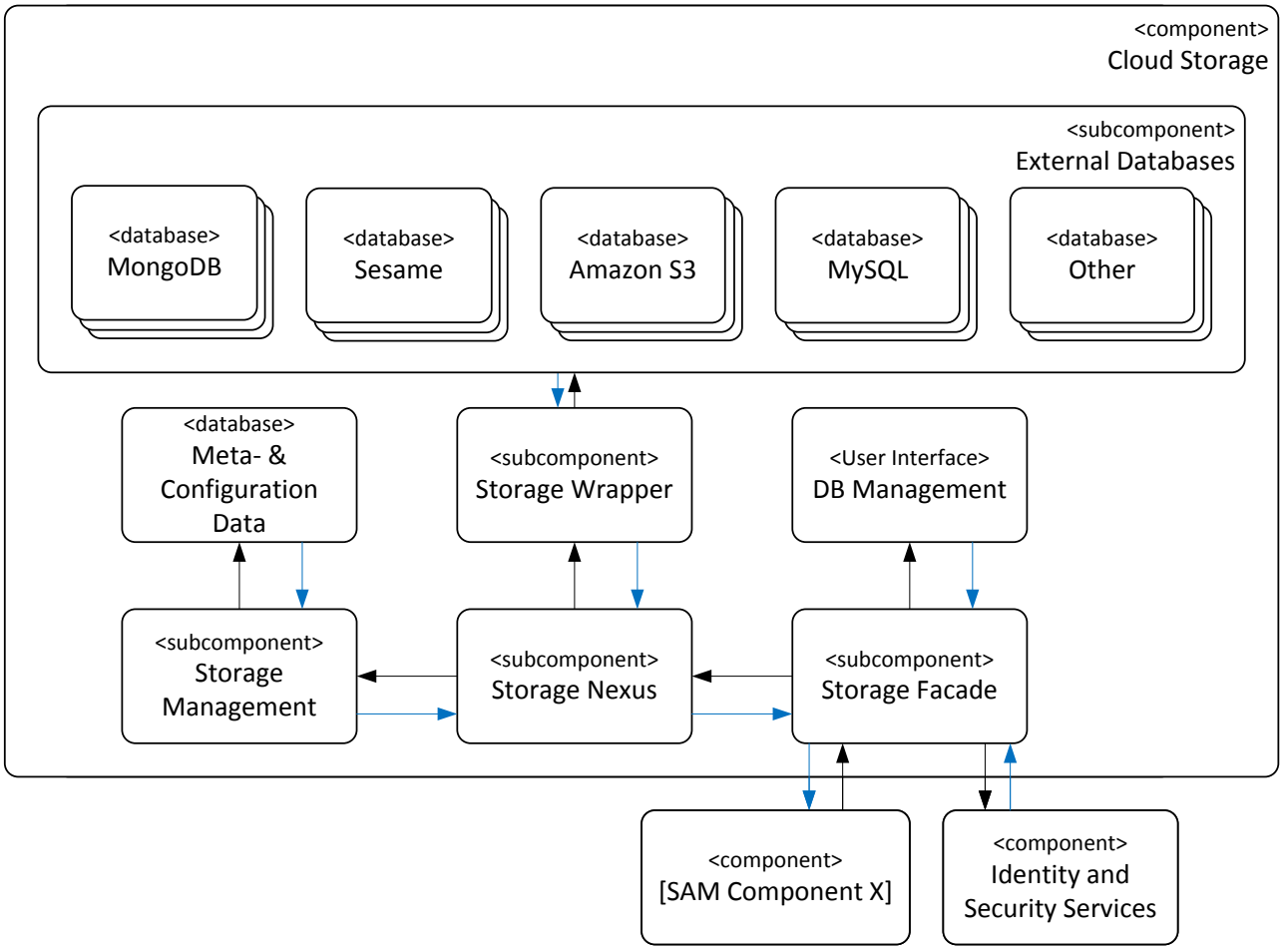

Figure 99: Overview with Selected Technologies – Cloud Storage

3.8.3.1.1 Access Control List

Each Bucket will be protected by an Access Control List (ACL). By default, only the owner of a Bucket has full access to that Bucket. With the ACL the owner takes control over which data can be accessed by whom. The access control of specific storage back ends will be abstracted so that developers do not have to care about system peculiarities. With this system, it will be possible to provide an access control even for storage types which do not provide such functionality with the required granularity. The following access rights will be supported:

- **NotSet:** Delete an access right
- **Denied:** Deny a user or group any access to a bucket
- **Read:** Allow a user or group to read the contents of a bucket
- **Write:** Allow a user or group to read and write the contents of a bucket
- **Super:** Allow a user or group the same rights as the owner. This includes the right to grand or restricts access rights to the Bucket as well as to delete the Bucket

Any entity can be used as identifier to receive an access right. The authentication of an user or his membership to a user group has to be validated during the access to the Cloud Storage.

3.8.4 Specification of Interfaces, Protocols and Formats

The services of the Cloud Storage will be provided as RESTful interfaces (described in Section 3.8.4.1) as well as a Java API for Android (described in Section 3.8.4.2).

3.8.4.1 RESTful Interfaces

In order to describe the RESTful interface, each provided service is described in a separate table. This table is followed with a listing showing an example for the JSON parameter (if applicable) as well as an example for the return value (if applicable). In the following subsection the term “user” describes components which will use this method/interface.

3.8.4.1.1 Method “Create Bucket”

The RESTful interface “Create Bucket” (Figure 100), creates a new Bucket for the user. A Bucket is unique in the context of the user, who created it.

Create Bucket		
Description	Creates a new Bucket for the user.	
Request		
Request URL	**POST** http://example.com/api/cs/bucket:bucketId	
Resource Parameters	Name	Description
	bucketId Required string	ID of the Bucket Example**:** “*testId”*
Response		
HTTP Status	Value	Description

Code	201	Bucket created
	409	Bucket exists already
	502	Database error

Figure 100: RESTful Interface Description – Create Bucket

3.8.4.1.2 **Method "Delete Bucket"**

The RESTful Interface "Delete Bucket" (see Figure 101), deletes the specific Bucket for the user. This will also delete all data stored in the Bucket. Only the owner of the Bucket or a user with "Super" access right can delete a Bucket.

Delete Bucket		
Description	Deletes the specific Bucket for the user. This will also delete all data stored in the Bucket. Only the owner of the Bucket or a user with "Super" access right can delete a Bucket.	
Request		
Request URL	**DELETE** http://example.com/api/cs/bucket/:bucketId	
Resource Parameters	Name	Description
	bucketId Required string	ID of the Bucket Example**:** "*testId"*
Response		
HTTP Status Code	Value	Description
	200	Bucket deleted
	401	Insufficient rights
	404	Bucket not found

Figure 101: RESTful Interface Description – Delete Bucket

3.8.4.1.3 **Method "Add Access Right to Bucket"**

The RESTful interface for "Add Access Right" (see Figure 102), Modifies the ACL of a Bucket by adding a specific access right to a specific user. The access rights provided in Section 3.8.3.1.1 are supported.

Note: The ownership of a Bucket can't be modified.

Add Access Right to Bucket	
Description	Modifies the ACL of a Bucket by adding a specific access right to a specific user.
Request	
Request URL	**PUT** http://example.com/api/cs/bucket/:bucketId/acl/user/:userId/right/:right **PUT**

	http://example.com/api/cs/bucket/:bucketId/acl/right/:right/user/:userId	
Resource Parameters	Name	Description
	bucketId Required string	ID of the Bucket Example: “*testId”*
	userId Required string	ID of the user to receive the access right Example: “*testUser”*
	right Required string	Access right to be granted to the user Example: “*READ”*
Response		
HTTP Status Code	Value	Description
	200	Access right for user updated
	201	Access right for user created
	401	Insufficient rights
	404	Bucket not found

Figure 102: RESTful Interface Description – Add Access Right to Bucket

3.8.4.1.4 **Method “Remove Access Right for Bucket”**

The RESTful Interface “Remove Access Right for Bucket” (see Figure 103), modifies the ACL of a Bucket by removing:

- All access rights to the Bucket
- All access rights for a specific user to the Bucket
- All access rights of a specific kind to the Bucket

Remove Access Right from Bucket		
Description	Modifies the ACL of a Bucket by removing: • All access rights to the Bucket • All access rights for a specific user to the Bucket • All access rights of a specific kind to the Bucket	
Request		
Request URL	**DELETE** http://example.com/api/cs/bucket/:bucketId/acl/user/:userId **DELETE** http://example.com/api/cs/bucket/:bucketId/acl/right/:right **DELETE** http://example.com/api/cs/bucket/:bucketId/acl	
Resource Parameters	Name	Description
	bucketId Required string	ID of the Bucket Example: “*testId”*
	userId Optional string	ID of the user to receive the access right Example: “*testUser”*

	right Optional string	Access right to be granted to the user Example: “*READ”*
Response		
HTTP Status Code	Value	Description
	200	OK
	401	Insufficient rights
	404	Bucket not found

Figure 103: RESTful Interface Description – Remove Access Right from Bucket

3.8.4.1.5 **Method “Get Access Right for Bucket”**

The RESTful Interface “Get Access Right for Bucket” (see Figure 104), get the access controls for a specific Bucket. The result can be filtered by user or access right.

Get Access Right for Bucket		
Description	Get the access controls for a specific Bucket. The result can be filtered by user or access right.	
Request		
Request URL	**GET** http://example.com/api/cs/bucket/:bucketId/acl/ **GET** http://example.com/api/cs/bucket/:bucketId/acl/user/:userId **GET** http://example.com/api/cs/bucket/:bucketId/acl/right/:right	
Resource Parameters	Name	Description
	bucketId Required string	ID of the Bucket Example: “*testId”*
	userId Optional string	Filter by user Example: “*testUser”*
	right Optional string	Filter by access right Example: “*READ”*
Response		
HTTP Status Code	Value	Description
	200	OK
	204	Entry not found
	404	Bucket not found
JSON Attributes	list Optional list	A list of JSON objects with id and right attributes. Both attributes are required for every object in the list. Example: [

		{
		"id": "testUser",
		"right": "Read”
		}
		]

Figure 104: RESTful Interface Description – Get Access Right for Bucket

3.8.4.1.6 **Method “CRUD-Operation Create”**

The RESTful interface “CRUD-Operation Create” (see Figure 105), executes a create operation on an existing Bucket. This will store the data of the transferred data object. The storage details are based on the data type of the transferred data object.

Create Data Object in Bucket		
Description	Executes a create operation on an existing Bucket. This will store the data of the transferred data object. The storage details are based on the data type of the transferred data object.	
Request		
Request URL	**POST** http://example.com/api/cs/bucket/:bucketId/object/:objectId	
Resource Parameters	Name	Description
	bucketId Required string	ID of the Bucket Example: “*testId”*
	objectId Required string	ID of the data object. Unique in the context of the Bucket. Example: “*testObjectId*”
HTTP Parameters	dataType Required string	Data type of the transferred object Example: “*JSON”*

	object
	Required object
		An object representing an object of the specified data type.
		Example:
		{
		"key": "myKey",
		"value": "TestValue"
		}

Response		
HTTP Status Code	Value	Description
	201	Object created
	401	Insufficient rights

	404	Bucket not found
	409	Object with specific ID already exists in the Bucket

Figure 105: RESTful Interface Description – CRUD-Operation Create

3.8.4.1.7 **Method "CRUD-Operation Read"**

The RESTful interface "CRUD-Operation Read" (see Figure 106), executes a read operation on an existing Bucket. This will retrieve all data objects of a specific type matching the query object. The matching criteria's are based on the data type.

Read Data Object from Bucket		
Description	Executes a read operation on an existing Bucket. This will retrieve all data objects of a specific type matching the query object. The matching criteria's are based on the data type.	
Request		
Request URL	**GET** http://example.com/api/cs/bucket/:bucketId/object/:objectId	
Resource Parameters	Name	Description
	bucketId Required string	ID of the Bucket Example: "*testId"*
	objectId Required string	ID of the data object. Unique in the context of the Bucket Example: *"testObjectId"*
HTTP Parameters	dataType Required String	Data type of the transferred object Example: *JSON*
	query Required object	An object representing a query object for the specified data type. Example: *{ "key": "myKey"}*
Response		
HTTP Status Code	Value	Description
	200	Object found
	204	Object not found
	401	Insufficient rights
	404	Bucket not found
JSON Attributes	list Optional list	A list of objects matching the query. Example: [{

		"key": "myKey", "value": "TestValue" }]

Figure 106: RESTful Interface Description – CRUD-Operation Read

3.8.4.1.8 **Method "CRUD-Operation Update"**

The RESTful interface "CRUD-Operation Update" (see Figure 107), an update operation on an existing Bucket. This will update all found data objects found with the query object in the Bucket with the provided values in the parameter.

Update Data Object in Bucket		
Description	Executes an update operation on an existing Bucket. This will update all found data objects found with the query object in the Bucket with the provided values in the parameter.	
Request		
Request URL	**PUT** http://example.com/api/cs/bucket/:bucketId/object/:objectId	
Resource Parameters	Name	Description
	bucketId Required string	ID of the Bucket Example: "*testId"*
	objectId Required string	ID of the data object. Unique in the context of the Bucket Example: *"testObjectId"*
HTTP Parameters	dataType Required string	Data type of the transferred object Example: *"JSON"*
	query Required object	JSON object representing a query object for the specified data type Example: *{ "key": "myKey"}*
	object Required object	An object representing an object of the specified data type with attributes set, which will be updated in the data objects found with the query. Example: *{ "value": "OtherTestValue",}*
Response		
HTTP Status Code	Value	Description
	200	Found objects updated
	204	Object not found
	401	Insufficient rights

	404	Bucket not found

Figure 107: RESTful Interface Description – CRUD-Operation Update

3.8.4.1.9 **Method "CRUD-Operation Delete"**

The RESTful interface "CRUD-Operation Delete" (see Figure 108), executes a delete operation on an existing Bucket. This will delete all data objects of a specific type matching the query object. The matching criteria's are based on the data type.

Delete Data Object from Bucket		
Description	Executes a delete operation on an existing Bucket. This will delete all data objects of a specific type matching the query object. The matching criteria's are based on the data type.	
Request		
Request URL	**Delete** http://example.com/api/cs/bucket/:bucketId/object/:objectId	
Resource Parameters	Name	Description
	bucketId Required string	ID of the Bucket Example: *"testId"*
	objectId Required string	ID of the data object. Unique in the context of the Bucket. **Example**: testObjectId
HTTP Parameters	dataType Required String	Data type of the transferred object Example: *JSON*
	query Required object	An object representing a query object for the specified data type Example: { "key": "myKey"}
Response		
HTTP Status Code	Value	Description
	200	Object deleted
	204	Object not found
	401	Insufficient rights
	404	Bucket not found

Figure 108: RESTful Interface Description – CRUD-Operation Delete

3.8.4.1.10 **Method "Execute Generic Query"**

The RESTful Interface "Execute Generic Query" (see Figure 109), executes a generic query on an existing bucket in the Cloud Storage.

Information of an App

Description	Executes a generic query on the Bucket based on the provided data type. Generic queries will not be tested and can thereby destroy all data in the Bucket.	
Request		
Request URL	**GET** http://example.com/api/cs/bucket:bucketId/Query/:dataType	
Resource Parameters	Name	Description
	bucketId Required string	ID of the Bucket. Example: *"testId"*
	dataType Required string	The data type to define on, which data objects the query will be executed. Example: *"semantic"*
Response		
HTTP Status Code	Value	Description
	200	OK
	401	Insufficient rights
	404	Bucket not found
JSON Attributes	Name	Description
	result Required object	Generic JSON object

Figure 109: RESTful Interface Description – Generic Query

3.8.4.2 Java Interfaces

The Java interfaces act as wrappers for the RESTful interfaces (see Section 3.8.4.1). They provide a convenient way to access the Cloud Storage for Android and Java developers.

3.8.4.2.1 **Method "Create Bucket"**

For the creation of Buckets, the API provides the method "createBucket". The method has one parameter, the user-defined ID for the Bucket. The ID must be unique in the context of the user.

Create Bucket	
Method Header	public Bucket createBucket(String bucketId)
Parameters	bucketId: A user defined ID for the bucket. The ID must be unique in the context of the user.
Return Value	A bucket object, representing the created bucket
Error Handling	In case of an error, a null value is returned
Remarks	None

Figure 110: Java Interface Description – Create Bucket

In Figure 111, the signature of the method for “createBucket” is shown as well as an example for the usage of this method.

```
/**
* @param bucketId, a user defined ID for the bucket. The ID must be unique in
the
* context of the user
* @return on success a Bucket object, otherwise null
*/
public Bucket createBucket(String bucketId)

…

//Creates a semi-structured bucket under with the id BUCKET_TEST_ID
Bucket bucket = api.createBucket(BUCKET_TEST_ID);
```

Figure 111: Source Code Example – API Method Signature for Creating a Bucket and the Usage of this Method

The example usage of the method at the bottom of Figure 111 will create a new Bucket with the ID string of the constant BUCKET_TEST_ID. If the API call is successful a bucket object will be returned, otherwise null.

3.8.4.2.2 Method “Delete Bucket”

For the deletion of Buckets, the API provides the method “deleteBucket”. The method has one parameter, the ID of the Bucket identifying the Bucket, which shall be deleted.

Delete Bucket	
Method Header	public boolean deleteBucket(String bucketId)
Parameters	bucketId: The ID of the bucket identifying the Bucket to be deleted.
Return Value	True on success
Error Handling	In case of an error, false is returned
Remarks	All data saved in the Bucket will be deleted with the Bucket. For this reason, only the owner and user with the AccessRight Super (see also Section 3.8.3.1.1) can delete a Bucket.

Figure 112: Java Interface Description – Delete Bucket

In Figure 113, the signature of the method to delete a Bucket is shown as well as an example for the usage of this method.

```
/***
* @param bucketId, the ID of the bucket which shall be deleted
* @return true on success
*/
public boolean deleteBucket(String bucketId)

…

//Deleted the bucket with the ID BUCKET_TEST_ID
api.deleteBucket(BUCKET_TEST_ID);
```

Figure 113: Source Code Example – API Method Signature for Deleting a Bucket and the Usage of the Method

The example usage of the method at the bottom of Figure 113 deletes the bucket with the provided ID. If the API call is successful true is returned, otherwise false.

3.8.4.2.3 Method "Add Access Rights for User"

To grant users access rights for a specific Bucket the API provide the method "addAccessRight". The method has three parameters, the ID of the bucket to which the access rights shall be granted, an ID, which identifies the user or a group of users, and third the right, which shall be granted.

Add Access Rights for User	
Method Header	public boolean addAccessRight(String bucketId, String id, Rights right)
Parameters	bucketId: The ID of the bucket to which the right will be granted. id: The user or group ID to whom the access right should be granted. The value NotSet will delete the access right. right: The access right which will be granted to the user.
Return Value	True on success
Error Handling	In case of an error, false is returned
Remarks	Only the owner of the bucket or a user with the "Super" access right can add access rights to a bucket.

Figure 114: Java Interface Description – Add Access Rights for User

In Figure 115, the signature of the method to add access rights for a user is shown as well as an example for the usage of this method.

```
/***
* @param bucketId, the ID of the bucket to which the right will be granted
* @param id, the user or group ID to whom the access right should be granted
*       the value NotSet will delete the access right
* @param right, the right to grand to the user or group for the bucket
* @return true on success
*/
public boolean addAccessRight(String bucketId, String id, Rights right)

…

//adds the Read-AccessRight to the specific bucket for the specific user
boolean success = api.addAccessRight(BUCKET_TEST_ID, SECOND_USER_NAME,
Rights.READ);
```

Figure 115: Source Code Example – API Method Signature to Grant Access Rights to a User for a Bucket and the Usage of this Method

The example usage of the method at the bottom of Figure 115 grants the access right "READ" to the specific user for the bucket with the provided ID. If the API call is successful, true is returned, otherwise false.

3.8.4.2.4 Method “Get Access Rights for User”

In order to retrieve the access rights for a specific user, the API provides the method “getAccessRights”. The method has one parameter, the ID of the user of whom the access rights shall be queried.

Get Access Rights for User	
Method Header	public ArrayList<Rights> getAccessRights(String id)
Parameters	id: The user or group ID of whom the access right should be queried.
Return Value	An ArrayList of right values
Error Handling	In case of an error, the list is empty
Remarks	None

Figure 116: Java Interface Description – Get Access Rights for User

In Figure 117, the signature of this method is shown as well as an example of its usage.

```
/***
* @param id, the user or group ID of whom the access right should be queried
* @return a list of Rights for the user or group ID
*/
public ArrayList<Rights> getAccessRights(String id)

…

//Retrieves the access rights for the specific user
ArrayList<Rights> rights = api.getAccessRights(SECOND_USER_NAME);
```

Figure 117: Source Code Example – API Method Signature to Get Access Rights for a User and the Usage of this Method

The example usage of the method at the bottom of Figure 117 retrieves the access rights for the user with the provided ID. All found access rights are stored in the list.

3.8.4.2.5 Method “CRUD-Operation Create”

To create a new data object in a bucket, the API provides the method “createBucket”. As example for a data object for this and all following examples, which require a data object Figure 118 shows a factory method. This method creates an instance of the test class MyTestClass. This class has valid JAXB annotations as well as a corresponding XSD file (see Figure 119).

```
//Creates an instance of MyTestClass. A class with valid JAXB-Annotations and
an apt XSD file
private MyTestClass getTestClass()
{
            MyTestClass test = new MyTestClass();
            test.setTestAttributeBoolean(true);
            test.setTestAttributeInt(INT_BEFORE);
            test.setTestAttributeString(STRING_BEFORE);
            return test;
}
```

Figure 118: Source Code Example – Factory Method to Create a Test Data Object

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs=”http://www.w3.org/2001/XMLSchema”
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">
<xs:element name="MyTestClass"
           xmlns="http://www.ascora.de/xmlSchema/CloudStorage/CloudStorage.xs
           d">
           <xs:complexType >
            <xs:sequence>
                  <xs:element name="testAttributeString"
                                          type="xs:string"
                                          minOccurs="0" />
                  <xs:element name="testAttributeInt"
                                          type="xs:int"
                                          minOccurs="0" />
                  <xs:element name="testAttributeBoolean"
                                          type="xs:boolean"
                                          minOccurs="0" />
            </xs:sequence>
           </xs:complexType>
</xs:element>
</xs:schema>
```

Figure 119: Source Code Example – XSD Description for a Test Class

The “crudCreate” method of the API has two parameters. The first identifies the bucket in which the data object shall be stored. The second is the data object to be stored in the bucket.

CRUD-Operation Create	
Method Header	public boolean crudCreate(String bucketId, DTO object)
Parameters	bucketId: The ID of the bucket in which the data object shall be created. object: A Data Transfer Object (DTO, see Section 3.8.4.3.1.1), e.g., a generic (Java) object, which shall be stored in the bucket.
Return Value	True on success
Error Handling	In case of an error, false is returned
Remarks	The Java interface manipulates with the CRUD-operations generic Java objects.

Figure 120: Java Interface Description –CRUD-Operation Create

In Figure 121, the signature of the method for the CRUD-operation “crudCreate” is shown as well as an example for the usage of this method.

```
/***
* @param bucketId, the ID of the bucket in which the data object shall be
created
* @param object, the data object which shall be stored in the bucket
* @return true on success
*/
public boolean crudCreate(String bucketId, DTO object)

…

//Creates test data
MyTestClass testData = getTestClass();

//Call of a Helper Method. The Helper method is a factory method which
creates
//a DTO object. The Following arguments are required:
//    -the URI to a valid XSD file describing the class of the object
//    -the object instance itself
//    -the package name of the class of the object
DTO test = Helper.GenerateGenericObject("res/TestSchema.xsd", testData,
"junit.test.vo");

//Creates the data object "test" in the (with the ID) specific bucket
boolean success = api.crudCreate(BUCKET_TEST_ID, test);
```

Figure 121: Source Code Example – API Method Signature for the CRUD-Operation crudCreate and the Usage of this Method

The example usage of the method at the bottom of Figure 121 will store the created test data object in the bucket for semi-structured data with the ID BUCKET_TEST_ID. If the API call is successful, the variable success is set to true.

3.8.4.2.6 Method “CRUD-Operation Read”

To retrieve an existing data object from the Cloud Storage component, the API provides the method “crudRead”. For this, a query object is necessary. This object must be of the same type as the data objects that should be retrieved. As mentioned before, for this the factory method shown in Figure 118 will be used.

The “crudRead” method of the API has three parameters. The first identifies the bucket from which the data objects shall be retrieved. The second is the query data object used to find the data objects in question. The last parameter is for the Java object class of the objects that shall be retrieved.

CRUD-Operation Read	
Method Header	public ArrayList<DTO> crudRead(String bucketId, DTO object)
Parameters	bucketId: The ID of the bucket from which the data objects shall be retrieved. object: A DTO, e.g. a generic (Java) object, as query with attributes set to search for in the bucket.
Return Value	An ArrayList of objects which have matched the attributes of the query.
Error Handling	If no matching data object is found, the returned list is empty.

Remarks	The Java interface manipulates generic Java objects with the CRUD-operations.

Figure 122: Java Interface Description – CRUD-Operation Read

In Figure 123, the signature of the method for the CRUD-operation “crudRead” is shown as well as an example for the usage of this method.

```
/***
* @param bucketId, the ID of the bucket from which the data objects shall be
* retrieved
* @param object, query object with attributes set to search for in the bucket
* @return a ArrayList of object which matches the attributes of the query
object
*/
public ArrayList<DTO> crudRead(String bucketId, DTO object)

…

//Creates test data to query for data objects in the cloud storage
MyTestClass testQuery = new MyTestClass();
//Changes the integer attribute of the test data object
testQuery.setTestAttributeInt(INT_AFTER);

//Call of a Helper Method. The Helper method is a factory method which
creates
//a DTO object. The Following arguments are required:
//    -the URI to a valid XSD file describing the class of the object
//    -the object instance itself
//    -the package name of the class of the object
DTO test = Helper.GenerateGenericObject("res/TestSchema.xsd", testQuery,
"junit.test.vo");

//Creates a list for the results of the CRUD operation
ArrayList<MyTestClass> list = new ArrayList<MyTestClass>();
//Retrieve the data objects from a specific bucket matching the query object
ArrayList<Object> resultList = api.crudRead(BUCKET_TEST_ID, test ,
testQuery.getClass());
//Cast the generic result list of objects to the test class
for (Object object : resultList)
      list.add((MyTestClass) object);
```

Figure 123: Source Code Example – API Method Signature for the CRUD-Operation crudRead and the Usage of this Method

The example usage of the method at the bottom of Figure 123 creates a query object, which is used for the API call. The result list of objects is cast to the expected type. If no matching objects could be found in the specific bucket the result list is empty.

3.8.4.2.7 Method “CRUD-Operation Update”

To update attributes of existing data object in the Cloud Storage component, the API provides the method “crudUpdate”. For this, a query object is necessary. This object must be of the same type as the data objects that should be updated as well as a data object with new values for the objects in question. As mentioned before, for this the factory method shown in Figure 118 will be used. This method creates an instance of the test class MyTestClass. This class has valid JAXB annotations as well as a corresponding XSD file (see Figure 119).

The “crudUpdate” method of the API has three parameters. The first identifies the bucket in which the data objects shall be updated. The second is the query data object used to find the data objects in question. The last parameter is the data object with the new values for the found data objects in the Cloud Storage component.

CRUD-Operation Update	
Method Header	public boolean crudUpdate(String bucketId, DTO query, DTO newValues)
Parameters	bucketId: The ID of the bucket from which the data objects shall be retrieved. query: A DTO, e.g., a generic (Java) object, with attributes set to identify the data objects to be updated. newValues: A DTO, e.g., a generic (Java) object, with attributes set to be updated.
Return Value	True on success
Error Handling	In case of an error, false is returned
Remarks	The Java interface manipulates generic Java objects. In Figure 21, the signature of the method for the CRUD-operation crudUpdate is shown as well as an example for the usage of this method.

Figure 124: Java Interface Description – CRUD-Operation Update

```
/***
* @param bucketId, the id of the bucket in which the datao bjects shall be
updated
* @param object, query object with attributes set to identify the data
objects to * updated
* @param newValues, a data object with new values
* @return true on success
*/
public boolean crudUpdate(String bucketId, DTO query, DTO newValues)


//Creates test data to query for data objects in the cloud storage
MyTestClass testQuery = getTestClass();
testQuery.setTestAttributeString(null);

//Creates test data with updated valued
MyTestClass testData = getTestClass();
testData.setTestAttributeInt(INT_AFTER);
testData.setTestAttributeString(STRING_AFTER);

//Double call of a Helper Method. The Helper method is a factory method which
creates
//a DTO object. The Following arguments are required:
//    -the URI to a valid XSD file describing the class of the object
//    -the object instance itself
//    -the package name of the class of the object
DTO query = Helper.GenerateGenericObject("res/TestSchema.xsd", testQuery,
"junit.test.vo");
DTO newValues = Helper.GenerateGenericObject("res/TestSchema.xsd", testData,
"junit.test.vo");

//Updates all attributes of objects found with the query object with the
//attributes of the newValues object
boolean success = api.crudUpdate(BUCKET_TEST_ID, query , newValues);
```

Figure 125: Source Code Example – API Method Signature for the CRUD-Operation crudUpdate and the Usage of this Method

The example usage of the method at the bottom of Figure 125 creates a query object and an object with new values for the data objects which shall be updated in the Cloud Storage component. If the API call is successful, true is returned, otherwise false.

3.8.4.2.8 Method “CRUD-Operation Delete”

To delete data objects in the Cloud Storage component, the API provides the method “crudDelete”. For this, a query object is necessary. This object must be of the same type as the data objects that should be. As mentioned before, for this the factory method shown in Figure 118 will be used.

The “crudDelete” method of the API has two parameters. The first identifies the bucket from which the data objects shall be deleted. The second is the query data object used to find the data objects in question.

CRUD-Operation Delete	
Method Header	public boolean crudDelete(String bucketId, DTO query)

Parameters	bucketId: The ID of the bucket in which the data object shall be deleted. query: A DTO, e.g. a generic (Java) object, with attributes set to identify the data objects to be updated.
Return Value	True on success
Error Handling	In case of an error, false is returned
Remarks	The Java interface manipulates generic Java objects. In Figure 22, the signature of the method for the CRUD-operation crudDelete is shown as well as an example for the usage of this method.

Figure 126: Java Interface Description – CRUD-Operation Delete

```
/***
* @param bucketId, the ID of the bucket in which the data object shall be
deleted
* @param object, query object with attributes set to identify the data
objects to * be deleted
* @return true on success
*/
public boolean crudDelete(String bucketId, DTO query)

…

//Creates test data to query for data objects which shall be deleted
MyTestClass testQuery = new MyTestClass();

//Call of a Helper Method. The Helper method is a factory method which
creates
//a DTO object. The Following arguments are required:
//    -the URI to a valid XSD file describing the class of the object
//    -the object instance itself
//    -the package name of the class of the object
DTO query = Helper.GenerateGenericObject("res/TestSchema.xsd", testQuery,
"junit.test.vo");
boolean success = api.crudDelete(BUCKET_TEST_ID, query);
```

Figure 127: Source Code Example – API Method Signature for the CRUD-Operation crudDelete and the Usage of this Method

The example usage of the method at the bottom of Figure 127 creates first a data object with attributes to identify the data objects in the Cloud Storage component, which shall be deleted. If the API call is successful, true is returned, otherwise false.

3.8.4.2.9 Method “Execute Query on Bucket”

In order to execute a generic query to utilise specific features of one the database backends, the API provides the method “executeQuery”. The method has two parameters, the bucketId identifying the bucket on which the query should be executed and the query itself as a string.

Execute Query on Bucket	
Method Header	public String executeQuery (String bucketId, String query)

Parameters	bucketId: The ID of the bucket on which the query will be executed. query: The query as string.
Return Value	The result of the query as string.
Error Handling	In case of an error, a null value is returned
Remarks	The queries will not be parsed nor evaluated before executing, allowing the possible destruction of all data in a database.

Figure 128: Java Interface Description – Execute Query on Bucket

In Figure 129, the signature this method is shown as well as an example of its usage.

```
/***
* @param bucketId, the ID of the bucket on which the query will be executed
* @param query, the query as string
* @return the result of the query as string
*/
public String executeQuery (String bucketId, String query)

…

//Retrieves the result of a generic query for the specific bucket
String result = api.executeQuery(BUCKET_TEST_ID, QUERY_STRING);
```

Figure 129: Source Code Example – API Method Signature to Execute a Generic Query on a Specific Bucket and the Usage of this Method

The example usage of the method at the bottom of Figure 129 retrieves the access rights for the user with the provided ID. All access rights that are found are stored in the list.

3.8.4.3 Content Formats

3.8.4.3.1 JSON Schema

This section lists the JSON schemas which will be used by the Cloud-based Information Infrastructure internally as well as for the communication with other components via the RESTful and Java interfaces.

3.8.4.3.1.1 Data Transfer Objects

All Data Transfer Objects (DTO) will be used for the transfer of structured data for the Cloud-Storage (internally and externally). The list of objects described in this section is not complete and subject to changes to accommodate the special needs of external components which will use the Cloud Storage.

The DTO Schema acts as the superclass for all objects which will be used to transfer data. In the context of Java, this will be utilised to support generic objects without losing all advantages of typed objects.

```
{
      "type": "object",
      "id": "http://example.com/CS/JSON-Schema/DTO",
      "properties": {
            "description": {
                  "type": "string",
                  "id": "http://example.com/CS/JSON-Schema/description",
                  "required": false
            },
            "link": {
                  "type": "string",
                  "id": "http://example.com/CS/JSON-Schema/link",
                  "required": false
            }
      }
}
```

Figure 130: JSON Schema – Data Transfer Objects

3.8.5 Summary

In this section, the Cloud Storage have been technically specified. During the technology selection (Section 3.8.2), MongoDB, Sesame and Amazon S3 have been selected to store different types of data. An abstraction of these specific storage systems will enable each component and each backend service to create a Bucket, which can store data as different data types without knowledge of the backend systems. The access to a Bucket can be managed with an Access List Control (ACL) feature to share data with several actors of the SAM platform. To manipulate the ACL, an interface is given, where rights can be added, removed and queried. So each actor can use the ACL to query the access rights for his Buckets. The management of Buckets and the transmission of the according data to the backend systems will be handled internally by the Cloud Storage.

According to this specification, the following components have to be implemented:

- RESTful Interface to expose the Cloud Storage Facade to other SAM components
- Cloud Storage Facade to provide CRUD and query functionality
- Cloud Storage Backend with Access Control List feature
- Bucket Abstractors for the four mentioned Storage Backend Systems
- Java API for implementation ease, e.g., for the Android operating system

3.9 Brand and Consumer Protection

Brand and Consumer Protection (BCP) constitutes the central component for approving or disapproving all those assets that should be distributed in the SAM platform, see Figure 131. Different rules are set by the Platform Administrator and the Content Providers in order to be executed over the SAM assets for ensuring the protection functionality of this component. A web service will serve as interaction interface with different distribution components of the SAM platform such as the Syndicator and the Marketplace. The main characteristics of this component are the following:

- The BCP component will directly interact with different components of the Platform providing asset filtering functionalities based on enabled SAM rules.
- The technologies involved in the filtering process will be evaluations of logical expressions, text matching, part of speech tagging, lexical analysis (e.g. tokenisation of

words, and stemming) and semantic analysis. These technologies are necessary for extracting the lexical units from the text and establishing alignments between rules and asset texts.

- As part of this component, the Rules Editor will provide a user interface to allow SAM Content Providers and Platform Administrator to set and manage the SAM rules.

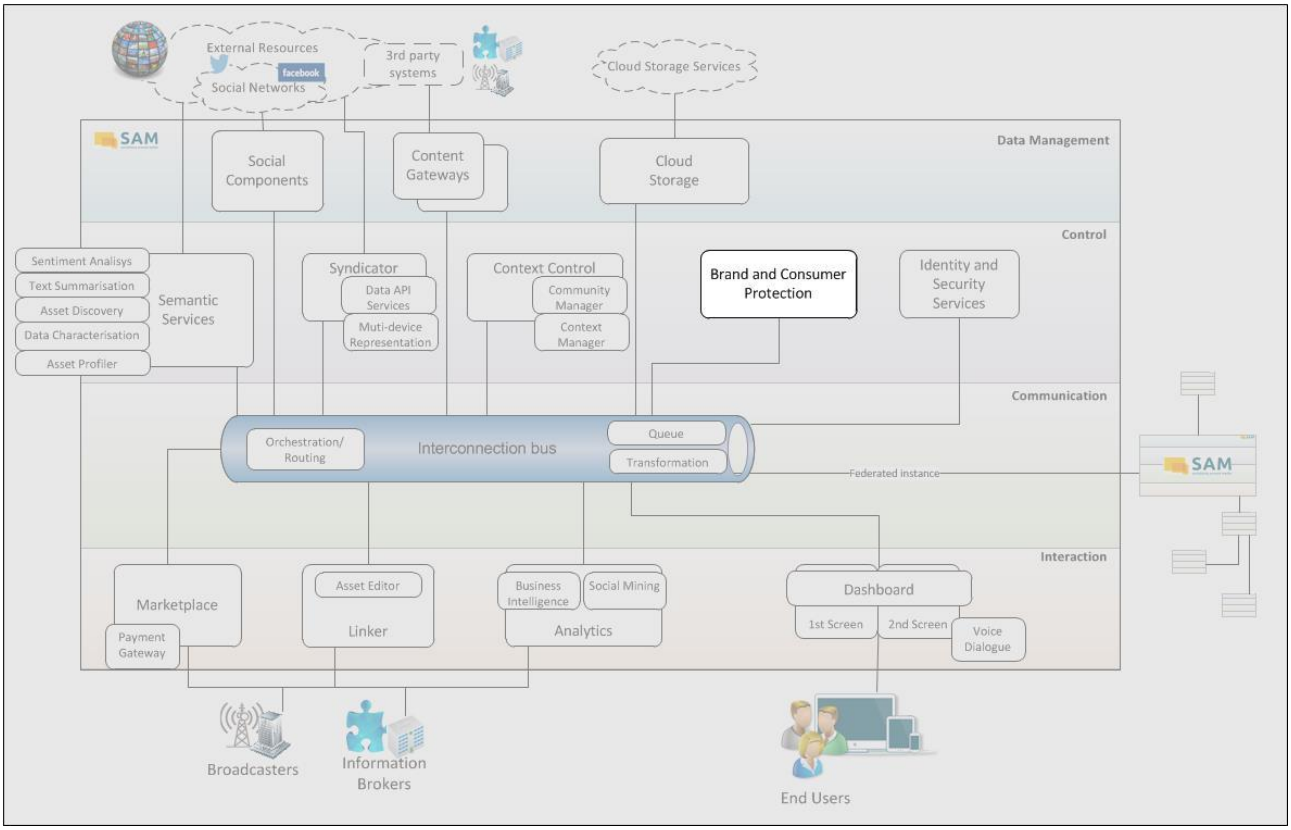

Figure 131: Architecture Overview – Brand and Consumer Protection

3.9.1 Major Design Decisions

The BCP component will be responsible for addressing aspects related to asset content such as brand protection mechanism and inappropriate content filtering. This component will apply content protection filters for the purpose of avoiding that unexpected content could affect both brands and End Users' integrity. In this respect, different filtering rules will be set by the Platform Administrator and Content Providers ensuring the supervision of all assets demanded by the SAM users. Based on these rules and on the procedures described in the deliverable D2.5 Brand and Consumer Integrity and the target user profile, the demanded assets will be approved, or disapproved, for the distribution to the SAM End User.

3.9.2 Technology Comparison and Selection

This subsection outlines technology selection criteria and compares existing technologies as potential candidates for addressing the development of the BCP component. The selected technologies will be used as a basis during its development for conducting text matching and the resulting content filtering. The specific selection criteria to be used are specified in the document D3.2.2 Functional Specification.

Eligible technologies are already covered in previous sections (Section 3.3.2.1.1 and Section 3.3.2.1.3.2 regarding NLP tools and lexical databases respectively).

3.9.2.1 Possible Technologies and Comparison

In order to implement the BCP component several options have to be considered as base technologies. For ensuring the protection of the End Users, different functionalities should be applied on SAM assets which filter assets that could harm brands and consumers as well. Internal functionalities, such as filtering, text matching, part of speech tagging, lexical analysis and semantic analysis should be taken into account for this process. For example, semantic word variations, such as synonyms, hyponyms and hypernyms can expand the range (set of words that provide different ways to express the same meaning), which helps to apply the filtering rules; the lexical analysis allows converting a sequence of characters into a sequence of tokens, i.e. meaningful character strings; and consequently these tokens constitute a step necessary for applying text matching and rule-content alignment. The assessment of these technologies will be presented in the following sections.

The NLP tools have been described in Section 3.9.2 and Section 3.3.2.2.1, which contain tools that deal with the subtasks of the Natural Language Processing. In case of SAM, text matching, part of speech tagging, lexical analysis (e.g. tokenisation of words, and stemming) and semantic analysis are the main focus. The evaluation of the most appropriate technology was based on the NLP tool analysis performed in Section 3.3.2.1.1, since the language processing goals for both BCP and Semantic Services coincide. In some cases, for filtering content it is necessary to identify the words involved or their synonyms. In this regard, in Section 3.3.2.1.3.2 different lexical databases have been described. Below, two comparison tables can be found, where all referred technologies are evaluated according to the specific parameters of the Brand and Consumer Protection component.

Parameter	Importance	Freeling	Stanford CoreNLP	Open NLP	Lucene	Apache Commons Lang
Generic Parameters						
Maturity & Stability	+++	+++	+++	+++	+++	+++
Regularly Updated	+/-	++	++	++	++	++
Technical Up-to-Datedness / Appeal	++	+++	+++	+++	+++	++
Open Source	+	YES	YES	YES	YES	YES
Non-Infecting	+	NO	NO	YES	YES	YES
Code Quality	++	N/A	N/A	N/A	N/A	N/A
Extensibility	+++	+	++	++	++	++
Community	-	+/-	+	+++	++	+
Performance / Scalability	+++	+	+	++	+++	+++
Reuse of existing developments	+/-	+	-	-	+++	+
EU project origin	---	NO	NO	NO	NO	NO
Platform (Portability)	+	-	+++	+++	+++	+++
Open Standards Compliance	+	YES	N/A	YES	N/A	N/A
Interoperability (easy integration for all platforms)	++	+	+++	+++	++	+++
Specific Parameters						
Filtering	+++	NO	NO	NO	++	++
Text matching	++	NO	NO	NO	++	++
Part of Speech Tagging	+	+++	+++	+++	NO	NO
Lexical analysis	+	++	++	++	++	+
Semantic Analysis	-	-	-	-	-	-

Figure 132: Parameter Evaluation of Technologies of Natural Language Processing Tools

Parameter	Importance	WordNet	BabelNet	WND & WNA	SentiWordNet
Generic Parameters					
Maturity & Stability	+++	+++	++	++	++
Regularly Updated	+/-	+	++	+	+
Technical Up-to-Datedness / Appeal	++	+++	+++	+++	+++
Open Source	+	N/A	N/A	N/A	N/A
Non-Infecting	+	YES	NO	YES	YES
Code Quality	++	N/A	N/A	N/A	N/A
Extensibility	+++	-	+++	-	-
Community	-	+	++	+	+
Performance / Scalability	+++	+++	++	+++	+++
Reuse of existing developments	+/-	+++	+++	+++	+++
EU project origin	---	NO	YES	NO	NO
Platform (Portability)	+	+	+	+	+
Open Standards Compliance	+	N/A	N/A	N/A	N/A
Interoperability (easy integration for all platforms)	++	+++	+++	+++	+++
Specific Parameters					
Semantic Analysis	+	+	-+	-+	--+

Figure 133: Parameter Evaluation of Technologies for Semantic Repositories – Lexical Databases

3.9.2.2 Technology Selection

Once the technologies in the previous section have been compared, the selection of the most appropriate technology according to BCP requirements has to be done. From the point of view of the NLP tools, Open NLP, Apache Commons Lang + Jexl and Lucene are the most suitable for dealing with BCP tasks. These tools coincide with the selection made for the Semantic Services. The main features of these tools are:

- All of them have free licenses
- Open NLP provides all the language processing functionalities that BCP needs, including stemming and tokenisation as the most useful ones;
- Lucene offers a very powerful engine for text searching, text matching and indexing;
- Finally, Apache Commons Lang + Jexl provides several functionalities for string management that are suitable to conduct text matching, as well as utilities based on logical expressions evaluation.

These utilities are necessary for aligning SAM content with the filtering rules. The semantic resources selected in Section 3.3.2.2.3 provide much more semantic information than required for addressing the BCP process, so only the use of WordNet will be required in this case.

3.9.2.3 User Interface Technology

This component has to provide the Rules Editor user interface so Content Provider and the administrator are able to create, edit or delete rules. The technology used for realisation has been selected in Section 3.1.

3.9.3 Technical Component Specification

3.9.3.1 Structure

Figure 134 provides an overview of the BCP which can be found also in D3.2.1 Global Architecture Definition. Additionally this version of the overview in this chapter has been augmented with the selected technologies.

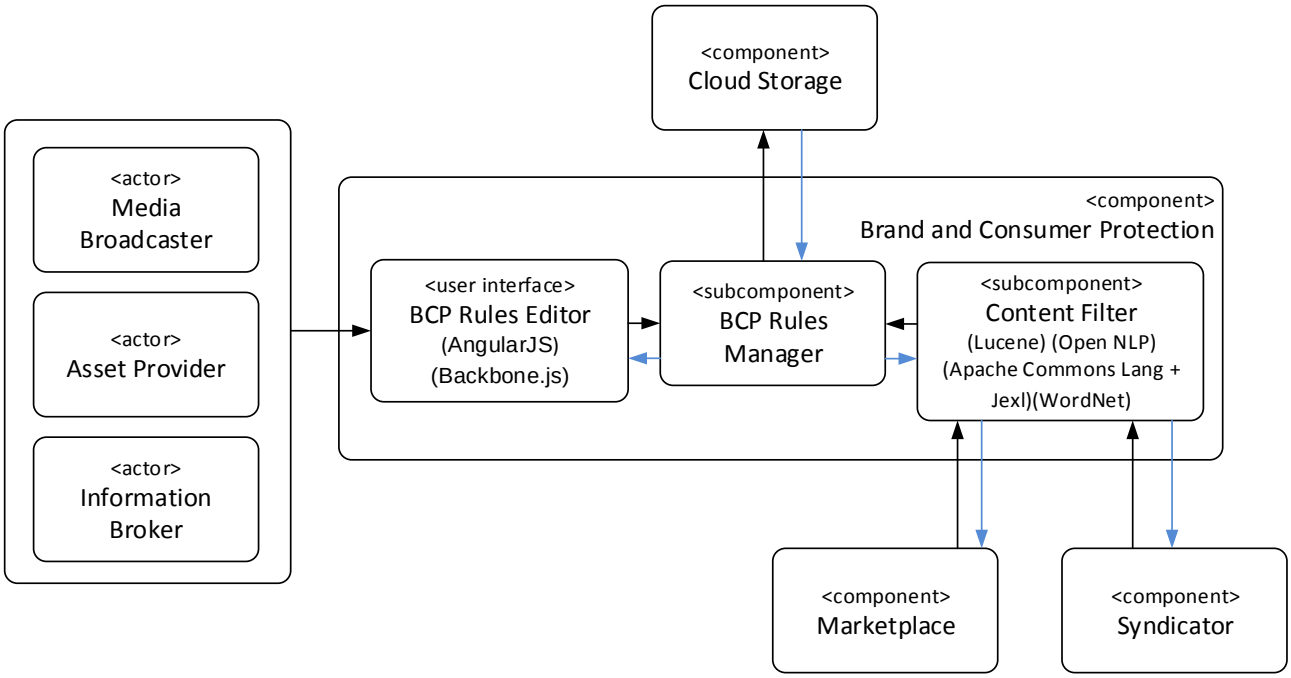

Figure 134: Overview with Selected Technologies – Brand and Consumer Protection

3.9.4 Specification of Interfaces, Protocols and Formats

The functionalities of the BCP will be provided as web methods of a SOAP WS (see Section 3.9.4.1) as well as a Java API (see Section 3.9.4.2). As result, each functionality of the BCP will provide JSON objects, which will serve as standard communication objects for all SAM components (see 3.9.4.3.1.1).

3.9.4.1 Web Services Interfaces

In order to describe the interfaces, each web method from the WS is shown in a separate table.

3.9.4.1.1 Method “Filter Content”

The web method interface “Filter Content” is able to filter inappropriate assets following the instructions set by the SAM rules. The assets for filtering are supplied by the Marketplace and Syndicator on-demand. For that, BCP will explore the assets and extract all the

attribute values from their structure. These values will then be compared with the filtering rules that have been enabled.

Filter Content		
Description	This functionality is able to approve the assets that satisfy the enabled rules.	
Request		
Request URL	**POST** http://example.com/api/BCP?wsdl	
Web Method	FilterContent	
	Name	Description
HTTP Parameters	asset Required object list	A JSON object that represents an object of the specified data type. Example: [{ "labelId": "a2" "title": "Casino Royale", “description”: “Casino Royale is the twenty-first film in the Eon Productions James Bond film series and the first to star Daniel Craig as the fictional MI6 agent James Bond”, “keywords”: [“Casino Royale”,”film”], “resourceType": " Asset", “assetType”: “Films” “minimumPermittedAges”: “18”, “descriptionFeatures”: [“sex”, “violence”], “categories”: [“action”], “status”: “draft”, “validSince”: “2014-09-01”, “validUntil”: “2015-09-01”, “licenseType”: “Creative Commons”, “BCP approval”: [ ] "relatedLabelIds": [“wn50”,” db50”, “wn40”, “ass40”], "relatedLabels": [“film”,” James Bond”, “star”, “Daniel Craig”], "relatedLabelResources": [“WordNet”,”DBpedia”, “WordNet”,”Asset”],

		"relationType": [“is_a”,”part_of”,”is_a”, “undefined”], }]

	target
	Required destination Object
		A JSON object that represents the information of the distribution target.
		Example:
		{
		“profileId”: ””123”,
		“userType”: “”end user”,
		“targetDevice”: “”smatphone”,
		“targetInterface”: “”dashboard”,
		“userAge”: “13”,
		“location”: “Spain”,
		“userPreference”: [“avoid violence”, “avoid advertisement”]
		}

Response		
HTTP Status Code	Value	Description
	100	Successful operation
	200	Object not found
	300	Command waiting in queue
	500	Fatal error
JSON Attributes	list Optional asset list	A list of JSON objects with assets. Example: [{ "labelId": "a2" "title": "Casino Royale", “description”: “Casino Royale is the twenty-first film in the Eon Productions James Bond film series and the first to star Daniel Craig as the fictional MI6 agent James Bond”, “keywords”: [“Casino Royale”,”film”], "resourceType": " Asset", “assetType”: “Films”

		“minimumPermittedAge”: “18”,
		“descriptionFeatures”: [“sex”, “violence”],
		“categories”: [“action”],
		“status”: “draft”,
		“validSince”: “2014-09-01”,
		“validUntil”: “2015-09-01”,
		“licenseType”: “Creative Commons”,
		“BCPapproval”:
		[
		{
		“profileId”: “123”,
		“distributionStatus“: “disapproved”
		}
		]
		"relatedLabelIds": [“wn50”,” db50”, “wn40”, “ass40”],
		"relatedLabels": [“film”,” James Bond”, “star”, “Daniel Craig”],
		"relatedLabelResources": [“WordNet”,”DBpedia”, “WordNet”,”Asset”],
		"relationType": [“is_a”,”part_of”,”is_a”, “undefined”]
		}
		]

Figure 135: Web service Interface Description – Filtering Content

3.9.4.2 Java Interfaces

The Java interfaces act as wrappers for the web service interfaces. They provide a convenient way to access the BCP for developers.

3.9.4.2.1 **Method “Filter Content”**

For applying BCP, the API provides one method. The method has two parameters.

Filter Content	
Method Header	public String filterContent(String assets, String target)
Parameters	assets : A JSON object that represents a list of assets target: A JSON object which refers the profile of the user type which should receive the assets.

Return Value	A JSON object that represents a list of assets modified internally identifying whether they were filtered or not.
Error Handling	In case of an error, a null value is returned
Remarks	None

Figure 136: Java Interface Description – Filter Content

In Figure 137 the header of the method is shown.

```
    /**
     * This functionality is able to approve the assets that satisfy the enabled rules for an
user profile.
     * @param listOfAssets  a list of assets to be filtered, in  JSON format
     * @param targetUser   refers the profile of the user type which should receive the
assets, in JSON format.
     * @return a list of assets filtered, in JSON format
     */
    @WebMethod(operationName = "filterContent")
    public String filterContent(@WebParam(name = "listOfAssets") String listOfAssets,
@WebParam(name = "targetUser")String targetUser) {
        String filteredListOfAssets = null;
        return filteredListOfAssets;
    }
}
```

Figure 137: Source Code Example – API Method Signature for Filter Content

If the API call is successful, an asset labelled as approved or disapproved will be returned for a given target user, otherwise it returns null.

3.9.4.3 Content Formats

3.9.4.3.1 Java Data Schema

This section lists the Java data schemas which will be used by the BCP information infrastructure internally as well as for the communication with other components via Web Services.

3.9.4.3.1.1 Brand and Consumer Protection Objects

All Brand and Consumer Protection Objects (BCPO) will be used for transferring the structured data for the BCP (both internally and externally). This component will transfer BCPOs as JSON objects, which are structured based the following types of Java data schemas. They are not complete and are subject to changes to accommodate the special needs of external components which will make use of the BCP.

For filtering content the representation is shown in Figure 138 and Figure 139.

```
public class TargetUser {

    public  enum UserType {ENDUSER, BRAND};
    public enum TargetDevice {SMARTPHONE, TABLET, TV};
    public enum TargetInterface {DASHBOARD, FIRSTSCREEN, SECONDSCREEN};

    private String profileId;
    private UserType userType;
    private TargetDevice targetDevice;
    private TargetInterface targetInterface;
    private int userAge;
    private String location;
    private List<String> userPreference;
}
```

Figure 138: Java Data Schema – Target User Including Protection Features

```
public class FilteredAsset extends Asset{

    public enum DescriptionFeatures {SEX, VIOLENCE};
    public enum Categories {ACTION, DRAMA, COMEDY, FICTION};
    public enum Status {DRAFT, FINAL};
    public enum Approval {APPROVED, DISAPPROVED};

    private int minimumPermittedAge;
    private List<DescriptionFeatures> descriptionFeatures;
    private List<Categories> categories;
    private Status status;
    private Date validSince;
    private Date validUntil;
    private String licenseType;
    private List<PuntualBCPapproval> puntual;


    public class PuntualBCPapproval {
        private String profileId;
        private Approval distributionStatus;
    }

}
```

Figure 139: Java Data Schema – Asset Including Protection Features

3.9.5 Summary

In this section, the Brand and Consumer Protection component has been technically specified. The technologies finally selected are Open NLP, Lucene, Apache Commons Lang + Jexl and WordNet and have been documented in Section 3.3.2.2. This choice covers the tools necessary to apply tokenisation, text searching, text matching and evaluation of logical expressions.
Additionally, the interface of the BCP component and its functionalities have been described in this section. The outcome of this process is “approved” or “disapproved” assets will be delivered as JSON objects to the corresponding SAM components.

3.10 Identity and Security Services

The Identity and Security Services component provides services to protect the SAM Platform and SAM Platform users against unauthorised system usage. Figure 140 depicts the embedding of Identity and Security Services as part of the overall SAM Platform.

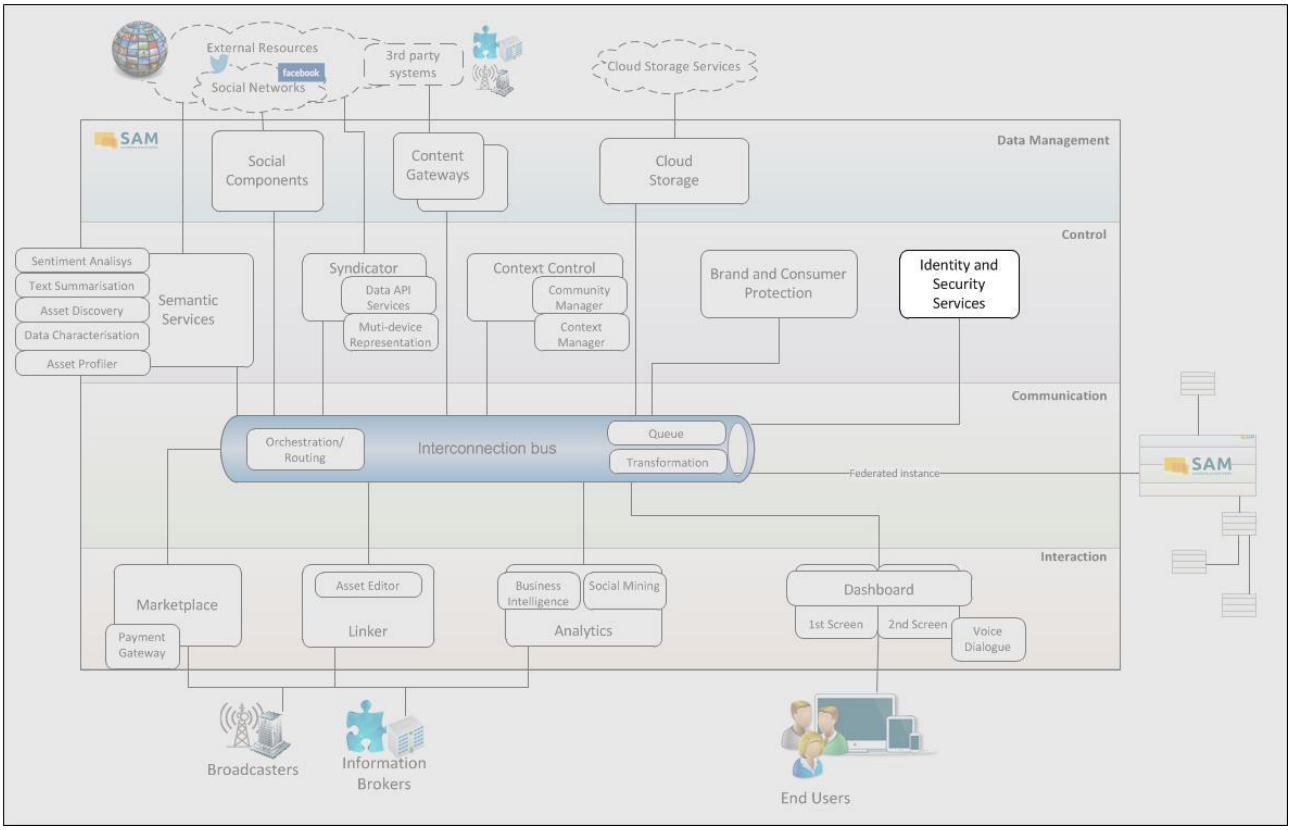

Figure 140: Architecture Overview – Identity and Security Services

The implementation for the SAM Platform prototypes focuses on the protection of SAM users and assets across federated SAM platform instances. The functionalities of the Identity and Security Services component can be characterised as follows:

- The component handles user authentication and uses an authentication token to maintain authentication session information.
- The component provides functionalities for user access authorisation to services and data available in the SAM platform.
- The component allows other SAM components to access these functionalities through an API so that SAM components have access to security functionalities in order to implement security features as a cross-cutting concern.

As indicated in SAM deliverable D3.2.1, the prototype implementation of Identity and Security Services focuses on identity management and authorisation within the framework of the SAM Platform.

3.10.1 Major Design Decisions

A key decision to be made when securing federated and fully distributed computing ecosystems is how to determine the authenticity of entities (both software and users) in particular across system federation boundaries. As for many other federated systems, federation in SAM should not rely on a central party or service for authentication or other

services unless absolutely necessary. Hence the use of decentralised infrastructure and protocols that can interact as federated system is desirable.

With this in mind it is possible to either base a solution for SAM on existing standards and implementations or to develop a custom solution for use with the SAM platform. The solution envisioned for SAM is to use an existing solution and apply and adapt it for SAM as needed in order to both build on established and well-known standards and initiatives and in order to be able to provide working security solutions to the partners in the SAM project early on during system development to facilitate integration of security features as early as possible during project development. The following section will discuss technology options and the selected solution approach.

3.10.2 Technology Comparison and Selection

3.10.2.1 Possible Technologies and Comparison

Identity management in distributed environments requires establishing mechanisms for handling the identity management and authentication of users across and between systems with separated identity management infrastructures and identity management databases.

One trend that can in particular be observed for identity management in the distributed environment of the World Wide Web is the delegation of identity management. Online service providers frequently allow users to authenticate via third-party identity management infrastructures, in particular account infrastructures of very large online services such as Google, Facebook or Twitter. Using publicly available services by these and other providers, online service providers can effectively outsource the authentication of users to a third party.

Two key standards for the implementation of such identity management mechanisms are OpenID Connect and SAML:

- **OpenID Connect**[87] is a relatively new standard for distributed user authentication. It is based on OAuth 2.0 and defines a model through which authentication requests can be routed to third-party authentication providers. OpenID Connect and OAuth 2.0 are used by a wide range of online services, in particular for End Users.
- **SAML**[88] is a standard for distributed user authentication and authorisation. SAML defines data exchange and messaging formats, and SAML implementations provide configurable means for distributed authentication via trusted third-party services. SAML is most frequently employed for implementing single sign-on at Intranet level.

The table below (Figure 141) compares these standards for the implementation of distributed identity management.

Parameter	Importance	OpenID Connect	SAML
Generic Parameters			
Maturity & Stability	+++	++	+++
Regularly Updated	+/-	+/-	+/-
Technical Up-to-Datedness / Appeal	+/-	++	+
Open Source	+++	+++	+/-

[87] http://openid.net/connect/
[88] http://saml.xml.org/

Non-Infecting	+++	+++	+++
Code Quality	+/-	+/-	+/-
Extensibility	+++	++	++
Community	+/-	+	+/-
Performance / Scalability	+++	+/-	+/-
Reuse of existing developments	+++	+++	++
EU project origin	---	---	---
Platform (Portability)	+++	+++	+++
Open Standards Compliance	+++	+++	+++
Interoperability (easy integration for all platforms)	+++	+++	+ (end user domain)
Specific Parameters			
Authentication against remote user database	+++	Supported	Supported
Automatic routing of user accounts to account identity providers	++	Supported	Not supported
Support for authorisation mechanisms	++	Supports OAuth authorisation / UMA enhancements	Authorisation is integrated as part of SAML
Reuse of existing user accounts of End Users	+/-	Likely possible (OpenID supported by Google, Facebook)	Like not possible (solution focuses on enterprise usage)
System management complexity	+/-	Low	Medium – high
Support for Browser-based authentication	++	Yes	Yes
Availability of Open Source / free Identity management server software	+++	Several implementations available	Some implementations available, SAML is mostly a data format specification

Figure 141: Parameter Evaluation of Technologies for Distributed Identity Management Standards

3.10.2.2 Technology Selection

The comparison shows that both OpenID (in its latest version OpenID Connect) and SAML are suitable candidates for use in SAM. For SAM, OpenID has been selected as the most suitable Identity Management infrastructure, because it is already being used in the online and Social Media domain, it is simpler to use than SAML, and it has easier and better documented support for RESTful Web Services than SAML.

SAM will use OAuth 2.0 and likely UMA in order to enable authorisation, since it is easy to integrate OpenID for authentication and OAuth for authorisation in the target application scenarios (OpenID Connect is based on OAuth 2.0).

3.10.3 Technical Component Specification

3.10.3.1 Structure

Figure 142 provides an overview of the Identity and Security Services component with the relevant subcomponents. This diagram and further information pertaining to the diagram can be found in the SAM Deliverable D3.2.1, Section 4.10. The diagram remains unchanged from its original version in Deliverable D3.2.1, as no changes to the subcomponent structure have been necessary.

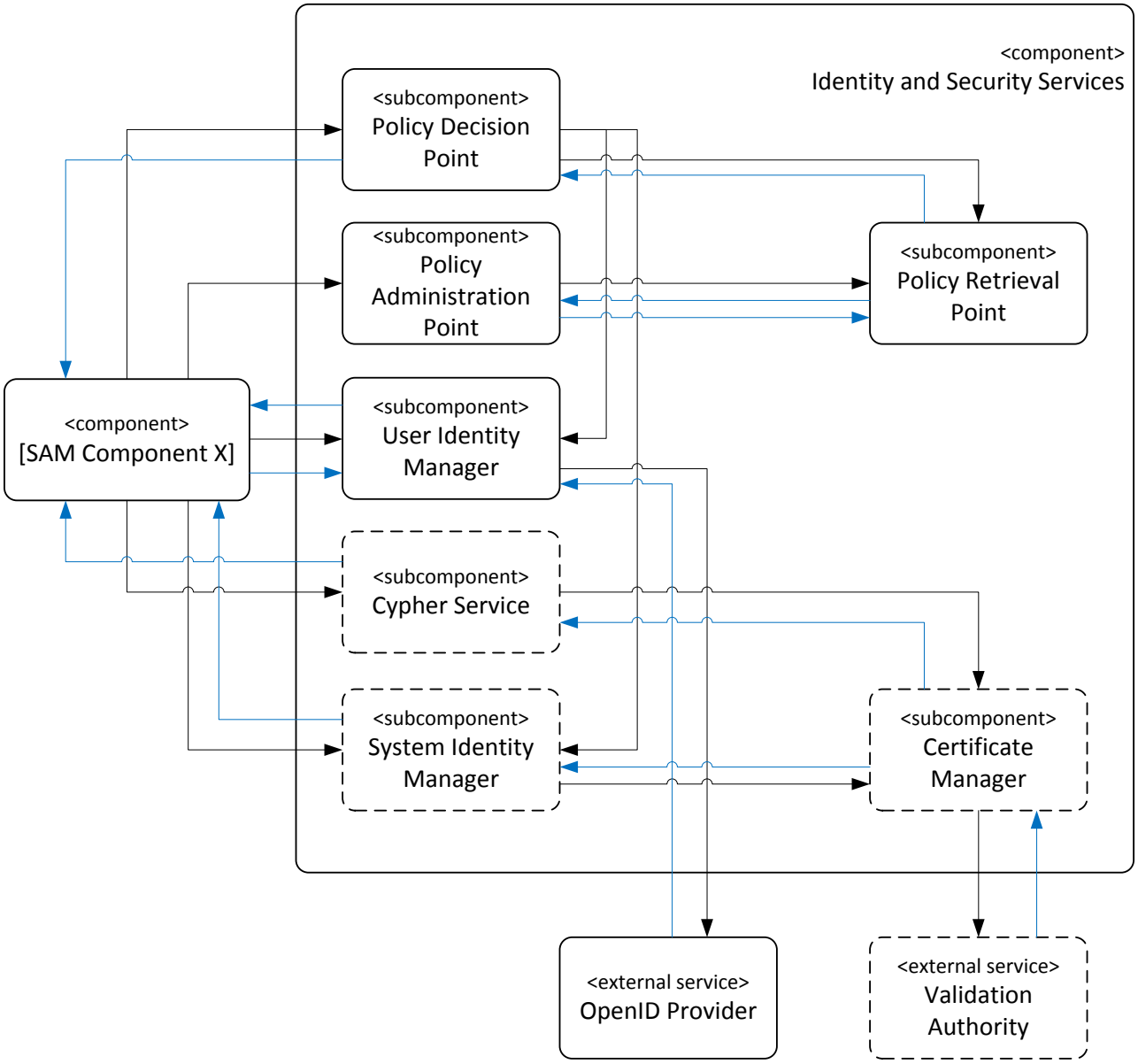

Figure 142: Overview with Selected Technologies – Identity and Security Services

3.10.4 Specification of Interfaces, Protocols and Formats

The API functionalities for Identity and Security Services will be provided as RESTful Web Services which connect to the SAM Integration Bus. If necessary, a subset of RESTful Web Services using the standard OpenID Connect protocols will be provided for interacting with relevant services without using the SAM Integration Bus; these functionalities – if used – will be limited to authentication only.

3.10.4.1 Restful Interfaces

In order to describe the RESTful interfaces, each provided service is described in a separate table. Each table will show a description, URL, request parameters with an example and response type including an example.

3.10.4.1.1 **Method “Authenticate”**

Note that as for all identity and security services methods, communication for this method must be secured at transport layer level.

Authenticate		
Description	Authenticates a user using the provided credentials.	
Request		
Request URL	**GET** https://example.com/api/iss/auth/user:userId/pass:password	
Resource Parameters	Name	Description
	userId Required string	User ID Example: *“testuser123”*
	password Required string	User password Example: *“abcdef”*
Response		
HTTP Status Code	Value	Description
	200	Authentication success
	401	Authentication failure
	500	Error during processing
JSON Attributes	list Required list	A JSON object containing the session token Example: [{ "token": "abcdef” }]

Figure 143: RESTful Interface Description – Authenticate

3.10.4.1.2 **Method “Verify Authentication Token”**

Note that as for all identity and security services methods, communication for this method must be secured at transport layer level.

Verify Authentication Token		
Description	Verifies an authentication token provided by a user.	
Request		
Request URL	**GET** https://example.com/api/iss/auth/user:userId/token:token	
Resource Parameters	Name	Description
	userId Required string	User ID Example: *“testuser123”*

	tokenId Required string	ID of the user's authentication token Example: *"abcdef"*
Response		
HTTP Status Code	Value	Description
	200	Verification successful
	401	Authentication failure
	500	Error during processing

Figure 144: RESTful Interface Description – Verify Authentication Token

3.10.4.1.3 **Method "Evaluate Access Request"**

Note that as for all identity and security services methods, communication for this method must be secured at transport layer level.

Evaluate Access Request		
Description	Evaluates an access request against a repository of access control policies.	
Request		
Request URL	**GET** https://example.com/api/iss/acc/user:userId/token:token/resource:resourceId/access:accessType	
Resource Parameters	Name	Description
	userId Required string	User ID Example: *"testuser123"*
	tokenId Required string	ID of the user's authentication token Example: *"abcdef"*
	resourceId Required string	ID of the resource to be access Example: *"resource456"*
	accessType Required string	ID of the access type requested
Response		
HTTP Status Code	Value	Description
	200	Evaluation successful
	401	Evaluation unsuccessful
	500	Error during processing

Figure 145: RESTful Interface Description – Evaluate Access Request

3.10.4.1.4 **Method "Manage Access Control Configuration"**

Submitting an empty configuration erases any existing configuration.

Note that as for all identity and security services methods, communication for this method must be secured at transport layer level.

Manage Access Control Configuration		
Description	Modifies an access control configuration.	
Request		
Request URL	**POST** https://example.com/api/iss/acc/ user:userId/token:tokenId/resource:resourceId	
Resource Parameters	Name	Description
	userId Required string	User ID Example: “*testuser123”*
	tokenId Required string	ID of the user’s authentication token Example: *“abcdef”*
	resourceId Required string	ID of the resource to be regulated Example: *“resource456”*
HTTP Parameters	dataType Required string	Data type of the transferred object; the datatype must be specified and it must be specified as JSON (preparation for possible support of other input formats) Example: *“JSON”*

	configuration
	Required object
		JSON object representing the relevant access control policy
		Example:
		{
		"userId": "user123",
		"accessType": "rw"
		}

Response		
HTTP Status Code	Value	Description
	201	Update persisted successfully
	401	Update not persisted, because credentials not valid
	500	Error during processing

Figure 146: RESTful Interface Description – Manage Access Control Configuration

3.10.5 Summary

The SAM Identity and Security Services provide a set of functionalities that are available to SAM components and allow them to protect users and assets throughout SAM. The

technology selection for the Identity and Security Services has taken into consideration the need to support syndication of potentially independent SAM Marketplace instances.

As discussed in this section, authentication and authorisation in SAM will be implemented based on OAuth 2.0, OpenID Connect and UMA standards, which allows other components, in particular the SAM Dashboard components, to use existing client library implementations for authentication and authorisation while at the same time providing a security framework with well-known properties concerning the requirements for the implementation of the mentioned standards.

3.11 Marketplace

The Marketplace offers an area where content users and resellers can find different content assets that are published in the platform by Content Providers (see Figure 147). These assets could be obtained free of charge or via a payment scheme, depending on the business model defined by the Content Provider. The most important goals are the following:

- Providing an area where Information Brokers and Content Providers can publish and access existing assets (use/enrich/buy/sell)
- The Marketplace will be the main User Interface for the business user roles such as the Content Providers, Content Brokers and 3rd party Software Providers. It will provide a unified interface to access to the main editors in SAM such as the Linker component and the Ontology Editor.
- From the Marketplace, it is also possible to browse and extract information from other Marketplace instances. The information and services in the different instances can be provided to the Linker as well, in order to compose more complex or different assets.

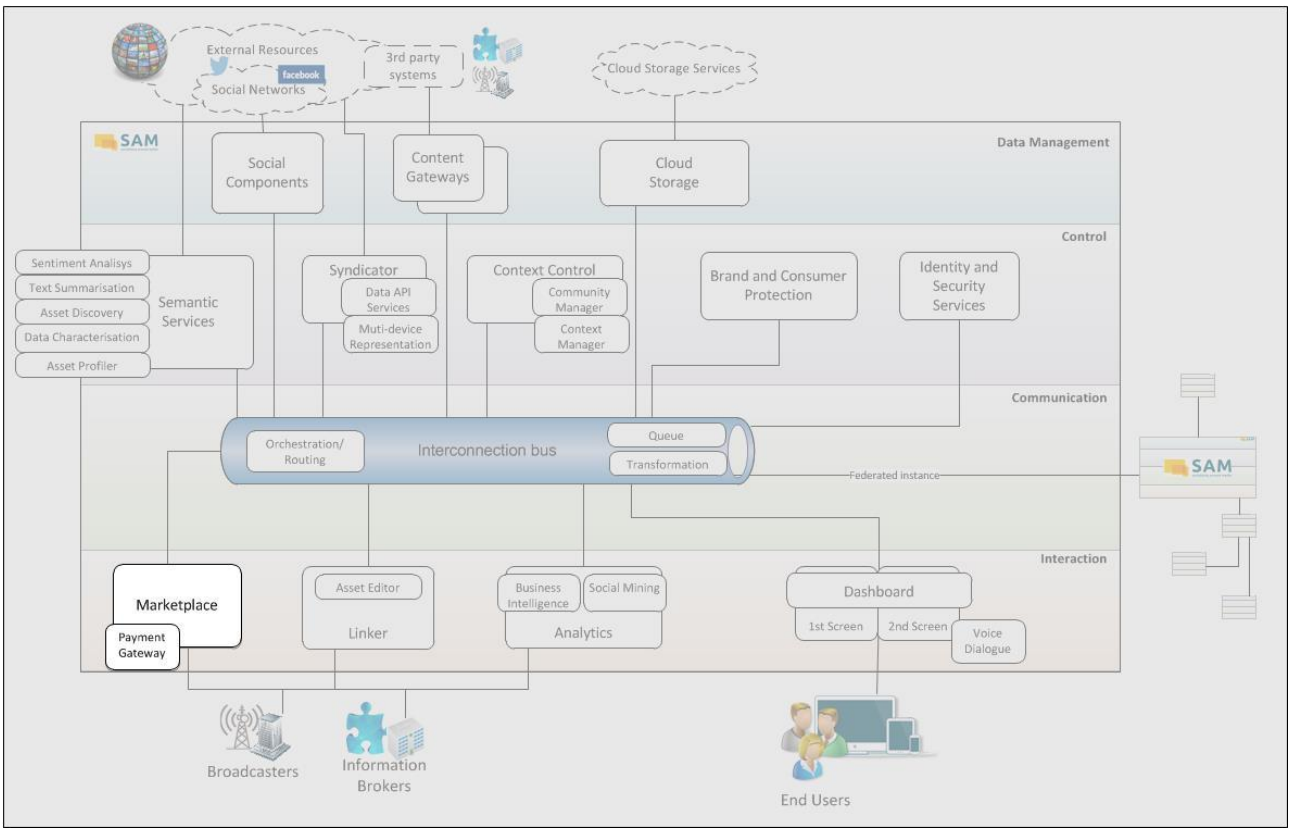

Figure 147: Architecture Overview – Marketplace

3.11.1 Major Design Decisions

The Marketplace component aims to provide an online web application where Information Brokers and Content Providers can publish and access existing assets. Therefore, it is very important that the client-side will be clear, concise, intuitive and efficient. The selected technology should offer support for the latest HTML5 and CSS3 features and provide an asynchronous communication with the server side in order to facilitate the development of user-friendly interfaces.

The Marketplace will be the entry point for the user interfaces of several components in the production scenario. Through the Marketplace, the end user will be able to access the Linker, Content Gateways, Analytics and Multi-Device Representation user interfaces (See D3.2.2 Component Mock-ups section 3.4)

The technology should incorporate features to support the main design patterns such as MVC (Model-View-Controller), MVVM (Model-View-ViewModel) or DI (Dependency Injection) and should be easily testable by unit tests, taking into account that testing is a critical success factor in application development.

Also, the technology in the server side should allow the development of applications in order to provide communication through web services with the client side and the possibility to connect with a wide set of databases sources including NoSQL Systems such as MongoDB or using search engines such as ElasticSearch.

3.11.2 Technology Comparison and Selection

This subsection outlines technology selection criteria and compares existing technologies – potential candidates for the realisation of Marketplace. Within the subsection the areas that need to be implemented or improved to fully meet SAM specific requirements are also identified and presented. The selected technologies will be used as a basis for the development phase in the realisation of the technical design of this component: The specific selection criteria to be used are specified in the document D3.2.2 Functional Specification, Section 4.11.5 and will be used in the next sections.

3.11.2.1 Possible Technologies and Comparison

The Marketplace technologies are split in frontend technologies and backend technologies. For the frontend, the Marketplace will follow the general consortium decision to use AngularJS; the comparison between the different possible technologies can be found in Section 3.1.1. For the backend development, there are a lot combinations and possibilities. One combination for every important technologies (.NET, Scala and Java) has however been selected. The following selection will describe the comparison about backend technologies.

- **IIS7 + WCF Services**[89] Windows Communication Foundation (WCF) is a framework for building service-oriented applications. Using WCF, it is possible to send data as asynchronous messages from one service endpoint to another. A service endpoint can be part of a continuously available service hosted by IIS7[90] (Internet Information Server), or it can be a service hosted in an application. An endpoint can be a client of a service that requests data from a service endpoint. The messages can be as simple as a single character or word sent as XML, or as complex as a stream of binary data.

[89] http://msdn.microsoft.com/en-us/library/ms731082(v=vs.110).aspx
[90] http://technet.microsoft.com/en-us/library/cc732976(v=ws.10).aspx

WCF enables the creation of SOA applications allowing the design of loosely-coupled applications. Security can be implemented using well-known standards such as SSL or WS-SecureConversation and the messages can be encrypted to protect privacy. WCF offers AJAX and REST support. It can be configured to process "plain" XML data that is not wrapped in a SOAP envelope. WCF can also be extended to support specific XML formats, such as ATOM (a popular RSS standard), and non-XML formats, such as JavaScript Object Notation (JSON).

- **Play Framework + Scala**[91] Play is an open-source web application framework, written in Scala and Java, which follows the Model–View–Controller (MVC) architectural design pattern. It aims to optimise developer productivity by using convention over configuration, hot code reloading and display of errors in the browser. Play supports REST, JSON/XML handling, non-blocking I/O, WebSockets and NoSQL.
- **JBOSS**[92] **+ Spring**[93] JBoss Enterprise Application Platform is an Open-Source platform used for building, deploying, and hosting highly-transactional Java applications and services. The JBoss Enterprise Application Platform is part of the JBoss Enterprise Middleware portfolio of software. Because it is Java-based and it operates cross-platform, it can be used on any operating system that supports Java. The Spring Framework is an open-source application framework and inversion of control container for the Java platform. The framework's core features can be used by any Java application.

Parameter	Importance	IIS7 + WCF Services	Play Framework + Scala	JBOSS + Spring
Generic Parameters				
Maturity & Stability	++	++	++	++
Regularly Updated	+	++	++	++
Technical Up-to-Datedness / Appeal	+	++	++	++
Open Source	+	NO	NO	YES
Non-Infecting	++	YES	YES	YES
Code Quality	+++	++	++	++
Extensibility	+++	++	++	++
Community	++	++	++	++
Performance / Scalability	++	++	++	++
Reuse of existing developments	+	++	++	++
EU project origin	+	NO	NO	NO
Platform (Portability)	+	+	+	+
Open Standards Compliance	+	YES	YES	YES
Interoperability (easy integration for all platforms)	+	++	++	++
Specific Parameters				
Testable	+++	++	++	++
Design Pattern focused	+++	YES	YES	YES
Compatible with NoSQL systems	+++	+	++	+

Figure 148: Parameter Evaluation of Technologies for the Marketplace Component

[91] https://www.playframework.com/
[92] http://www.jboss.org/
[93] http://projects.spring.io/spring-framework/

3.11.2.2 Technology Selection

The backend comparison analysed one particular combination for every important technology in the market. It shows that the technologies perform quite similar in terms of the general parameters. However, regarding the specific parameters, it shows that the Play Framework behaviour with NoSQL systems such as MongoDB is superior to the rest. This is very important since MongoDB is the selected technology in the Cloud Storage for semi-structured data storage (see Section 3.8.2.1.1).

TIE Kinetix, main implementer of this component, also has extensive experience with the Play Framework which will therefore contribute to a faster and higher quality development process.

3.11.2.3 User Interface Technology

The Marketplace will be the entry point for the user interfaces of several components in production scenario. As already mentioned in Section 3.1 the Marketplace frontend will be implemented using AngularJS to provide a common look and feel.

3.11.3 Technical Component Specification

3.11.3.1 Structure

Figure 149 provides an overview of the Marketplace, which can be found also in D3.2.1 Global Architecture Definition.

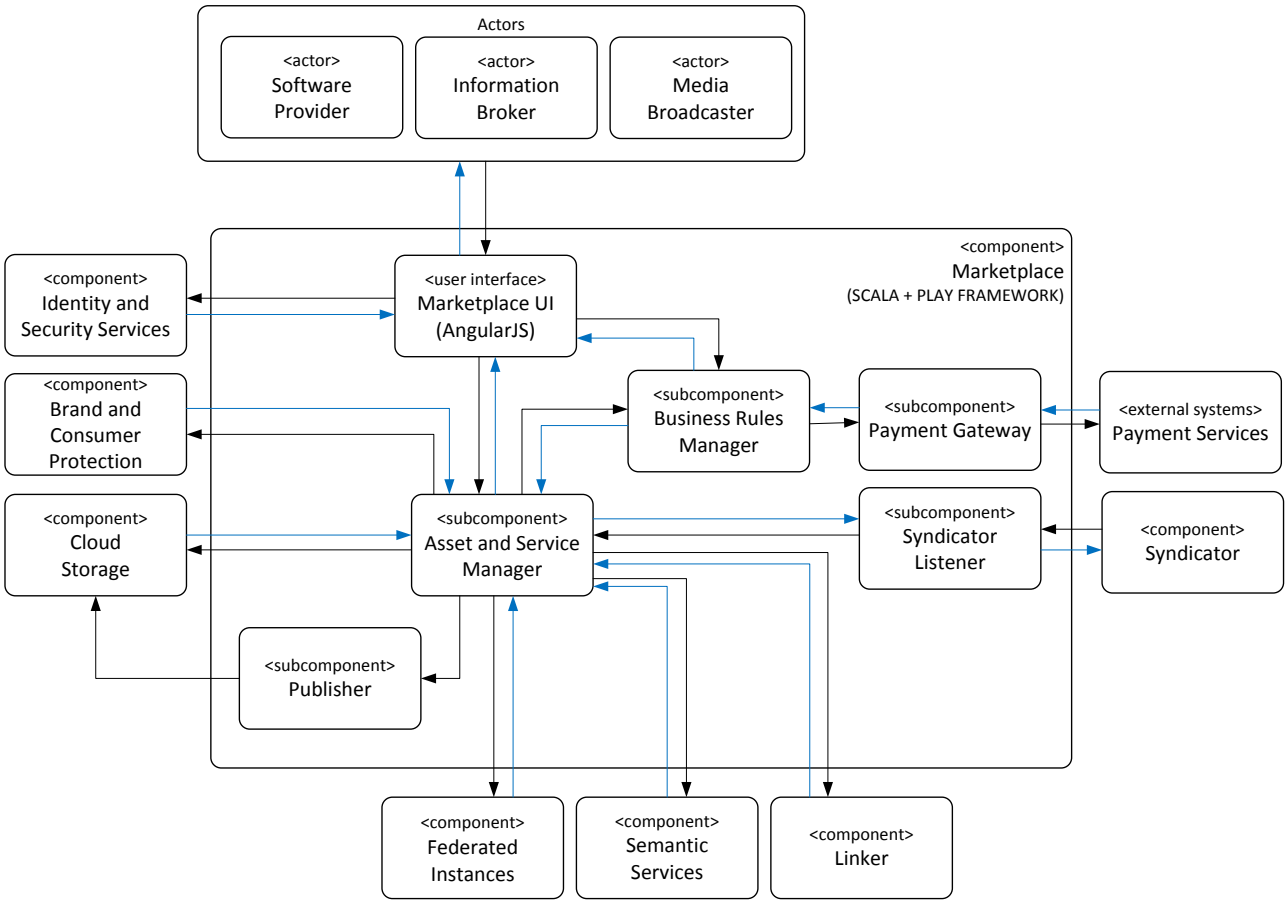

Figure 149: Overview with Selected Technologies – Marketplace

3.11.4 Specification of Interfaces, Protocols and Formats

The services of the Marketplace will be provided as RESTful interfaces (described in Section 3.11.4.1).

3.11.4.1 RESTful Interfaces

In order to describe the RESTful interfaces, each provided service is described in a separate table. Each table will show a description, URL, request parameters with an example and response type including an example.

3.11.4.1.1 Method “Get Business Rules”

The Syndicator will use the Marketplace in order to extract the business rules information to be able to syndicate the correct information. For this reason the Marketplace should define a RESTful interface called “GetBusinessRules” to provide this information to the Syndicator (Figure 150).

Get Business Rules		
Description	This interface is used to provide the business rules related to an asset or service.	
Request		
Request URL	**GET** http://example.com/api/mkt/GetBusinessRules:assetId	
Resource Parameters	Name	Description
	assetId	Asset information
Response		
HTTP Status Code	Value	Description
	200	The request was successful
	400	Bad Request
	500	Error during processing
JSON Attributes	template JSON Object	JSON object with the business rules information. Example: { "id": 1, "description": "pay with adverts”, "model": "pay-per-click”, "clicks": "500”, "price": "5”, "currency": "EUR”, "enabled": "true”,

		"paymentMethods": [{ “id”: 1,”name”: “credit card”}, { “id”: 2,”name”: “bit coins”}] }

JSON Error	JSON Object	Example:
		{
		“errors”:
		[
		{
		“errorId”: 1,
		“description”: “No rules found”
		}
		]
		}

Figure 150: RESTful Interface Description – Get Business Rules

3.11.4.1.2 **Method “Find Assets”**

In order to find assets and services to carry out the functions related with to them (Edit, Link, Sell, Publish, Acquire, etc.) the Marketplace will support the RESTful interface “FindAssets” (Figure 151) in order to facilitate the search of them.

Find Assets		
Description	This interface is used to provide assets based on search parameters.	
Request		
Request URL	**GET** http://example.com/api/mkt/FindAssets:assetFilter	
Resource Parameters	Name	Description
	assetFilter Required string	JSON object with the filter parameters. The value -1 means that the filter is not taken into account. Example 1: { “status”: “draft”, “type”: “actor”, “title”: “Mike Mikkelsen”, “description”: “007”,

		“owner”: “BDS”,
		“free”: “true”,
		“federatedInstances”: “true”,
		}
		Example 2: Assets with status “draft” including federated instances
		{
		“status”: “draft”,
		“type”: “-1”,
		“title”: “-1”,
		“description”: “-1”,
		“owner”: “-1”,
		“free”: “-1”,
		“federatedInstances”: “true”,
		}

Response		
HTTP Status Code	Value	Description
	200	The request was successful
	400	Bad Request
	500	Error during processing
JSON Attributes	template JSON Object	JSON object with the assets. Example: { “assets”: [{ “assetId”: 1, “status”: “draft”, “type”: “actor”, “title”: “Mike Mikkelsen”, “description”: “007”, “owner”: “BDS”, “free”: “true”, }, {

		“assetId”: 2, “status”: “draft”, “type”: “actor”, “title”: “Angelina Jolie”, “description”: “Tomb Raider”, “owner”: “BDS”, “free”: “true”, }] }

JSON Error	JSON Object	Example:
		{
		“errors”:
		[
		{
		“errorId”: 1,
		“description”: “No assets found”
		}
		]
		}

Figure 151: RESTful Interface Description – Find Assets

3.11.4.1.3 **Method “Find Related Assets”**

This RESTful interface “FindRelatedAssets” (Figure 152) will be used to search for assets related to other assets.

Find Related Assets		
Description	This interface look for assets related to other asset.	
Request		
Request URL	**GET** http://example.com/api/mkt/FindRelatedAssets:assetId	
Resource Parameters	Name	Description
	assetId Required string	Asset Identifier Example: *“2345“*
Response		
HTTP Status Code	Value	Description
	200	The request was successful

	400	Bad Request
	500	Error during processing

JSON Attributes	template
	JSON Object
		JSON object with the assets.
		Example:
		{
		“assets”:
		[
		{
		“assetId”: 1,
		“status”: “draft”,
		“type”: “actor”,
		“title”: “Mike Mikkelsen”,
		“description”: “007”,
		“owner”: “BDS”,
		“free”: “true”,
		},
		{
		“assetId”: 2,
		“status”: “draft”,
		“type”: “actor”,
		“title”: “Angelina Jolie”,
		“description”: “Tomb Raider”,
		“owner”: “BDS”,
		“free”: “true”,
		}
		]
		}

JSON Error	JSON Object	Example:
		{
		“errors”:
		[
		{
		“errorId”: 1,
		“description”: “No assets found”
		}

		] }

Figure 152: RESTful Interface Description – Find Related Assets

3.11.4.1.4 **Method "Get Best Selling Assets"**

This RESTful interface "GetBestSellingAssets" (Figure 153) will be used to retrieve assets sold the most.

Get Best Selling Assets		
Description	This interface looks for the most sold assets.	
Request		
Request URL	**GET** http://example.com/api/mkt/GetBestSellingAssets:assetFilter	
Resource Parameters	Name	Description

	assetFilter
	Required string
		JSON object with the filter parameters. The value -1 means that the filter is not taken into account.
		Example:
		{
		"date1": "01/01/2014",
		"date2": "10/09/2016",
		"numberOfRows": 20
		}

Response		
HTTP Status Code	Value	Description
	200	The request was successful
	400	Bad Request
	500	Error during processing
JSON Attributes	template JSON Object	JSON object with the assets. Example: { "assets": [{ "assetId": 1, "status": "draft", "type": "actor",

		“title”: “Mike mikkelsen”,
		“description”: “007”,
		“owner”: “BDS”,
		“free”: “true”,
		},
		{
		“assetId”: 2,
		“status”: “draft”,
		“type”: “actor”,
		“title”: “Angelina Jolin”,
		“description”: “Tomb Raider”,
		“owner”: “BDS”,
		“free”: “true”,
		}
		]
		}

JSON Error	JSON Object	Example:
		{
		“errors”:
		[
		{
		“errorId”: 1,
		“description”: “No assets found”
		}
		]
		}

Figure 153: RESTful Interface Description – Get Best Selling Assets

3.11.4.1.5 **Method “Get Most Popular Assets”**

This RESTful interface “GetMostPopularAssets” (Figure 154) will be used to look for the most popular assets.

Get Most Popular Assets	
Description	This interface returns the most popular assets.
Request	
Request URL	**GET** http://example.com/api/mkt/GetMostPopularAssets:assetFilter

Resource Parameters	Name	Description

	assetFilter
	Required string
		JSON object with the filter parameters. The value -1 means that the filter is not taken into account.
		Example:
		{
		“date1”: “01/01/2014”,
		“date2”: “10/09/2016”,
		“numberOfRows”: 20
		}

Response		
HTTP Status Code	Value	Description
	200	The request was successful
	400	Bad Request
	500	Error during processing
JSON Attributes	template JSON Object	JSON object with the assets. Example: { “assets”: [{ “assetId”: 1, “status”: “draft”, “type”: “actor”, “title”: “Mike Mikkelsen”, “description”: “007”, “owner”: “BDS”, “free”: “true”, }, { “assetId”: 2, “status”: “draft”, “type”: “actor”, “title”: “Angelina Jolie”, “description”: “Tomb Raider”,

		“owner”: “BDS”, “free”: “true”, }] }

JSON Error	JSON Object	Example:
		{
		“errors”:
		[
		{
		“errorId”: 1,
		“description”: “No assets found”
		}
		]
		}

Figure 154: RESTful Interface Description – Get Most Popular Assets

3.11.4.1.6 **Method “Get Services”**

In order to find assets and services to carry out the functions related to them (Edit, Link, Sell, Publish, Acquire, etc.) the Marketplace will support the interface “GetServices” (Figure 155) in order to facilitate searching for them.

Get Services		
Description	This interface is used to provide services based on search parameters.	
Request		
Request URL	**GET** http://example.com/api/mkt/GetServices:serviceFilter	
Resource Parameters	Name	Description
	serviceFilter Required string	JSON object with the filter parameters. The value -1 means that the filter is not taken into account. Example: { “status”: “draft”, “type”: “service”, “title”: “Service 1”, “description”: “007”, “owner”: “BDS”,

		“free”: “true”, “federatedInstances”: “true”, }
Response		
HTTP Status Code	Value	Description
	200	The request was successful
	400	Bad Request
	500	Error during processing

JSON Attributes	template
	JSON Object
		JSON object with the assets.
		Example:
		{
		“services”:
		[
		{
		“serviceId”: 1,
		“status”: “draft”,
		“type”: “service”,
		“title”: “Service 1”,
		“description”: “007”,
		“owner”: “BDS”,
		“free”: “true”,
		“url”: “http://services.sam.com/service1/”
		}
		]
		}

JSON Error	JSON Object	Example:
		{
		“errors”:
		[
		{
		“errorId”: 1,
		“description”: “No services found”
		}
		]
		}

Figure 155: RESTful Interface Description – Get Services

3.11.5 Summary

The Marketplace will be a web application, which will provide the entry point for the production scenario user interfaces. The Marketplace technologies have been split in frontend technologies and backend technologies. For the frontend, the Marketplace will follow the general consortium decision to use AngularJS (see Section 3.11.2.2). The backend comparison analysed one particular combination for every important technology in the market (.NET, Scala and Java) and has selected Play Framework + Scale due to the large experience of TIE Kinetix using this technology.

3.12 Linker

The Linker component (see Figure 156) will have the role of aggregating and composing SAM assets based on the preferences of the content provider. Thereby, the content provider will be able to enrich its assets with additional information coming from other assets, web or social network sources. The linking process is expected to allow the content provider to define easily but also effectively all aspects of the new, composite asset covering among others the aspects of asset time-line, audience types, delivery channels etc. This component will produce new assets, annotated and enriched with information from various sources so as to create a rich 2nd Screen experience for the SAM End Users:

- The main feature of this component is the creation and editing of asset compositions so as to define what additional content will be syndicated, when and to whom. As part of this process content providers will be able to search for related assets through specific keywords and criteria.
- The GUI of the Linker will also support “on-the-fly” preview of the asset composition which is under production and finally, the Linker will produce the description document for the asset that is produced following the asset description language defined in SAM.
- The Linker will allow content providers to load and configure modules, which enable the manipulation of the respective content types as part of their asset linking project. To this direction, the Linker component will not be limited to support specific content types, but it can be extended to additional ones through these modules or the creation of new ones.

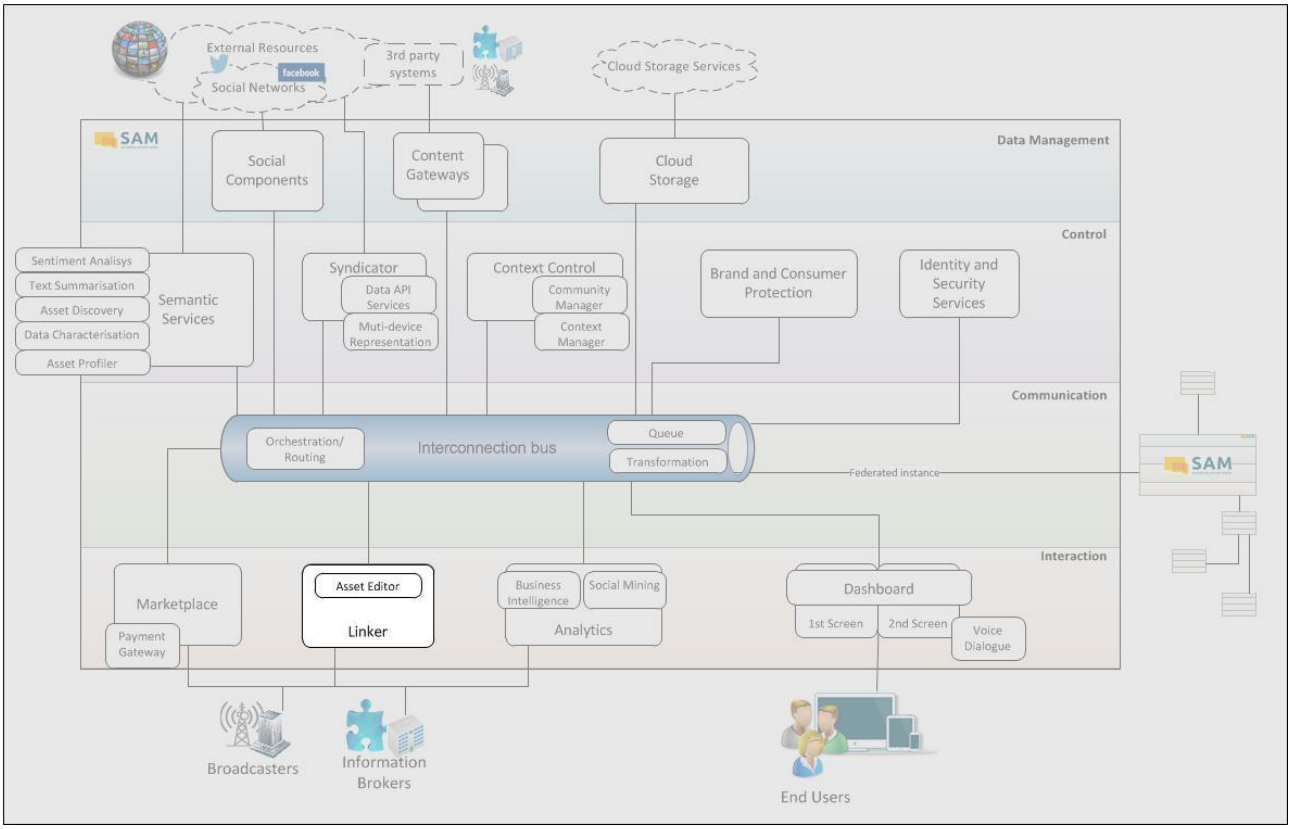

Figure 156: Architecture Overview – Linker

3.12.1 Major Design Decisions

The main challenges for the Linker component is that it should be up-to-date and user friendly since it will be used by the content editors but at the same time it should be technically robust covering all aspects of asset description and ensure compatibility with all SAM platform components.

There are three major characteristics the Linker must have as far as the technical design is concerned. First of all a decision must be made about the nature of the linker UI and more specifically whether it will be a **desktop program or a web based application**. A web-based application will eliminate the risk of OS compatibility problems and also can also be very flexible and always up to date. On the other hand a desktop application could be faster and fulfil the user-friendliness and interactivity requirements in a better way than a web-based one, but given the nature of the SAM platform it is essential for the components to interact in a homogenous environment in order to work properly without needing many adaptations.

Another reason to focus on a web-based application as a solution is that the Linker must be **integrated inside the Marketplace**. The Linker being a web application is the first step towards integration inside the Marketplace while the second step is using the appropriate technologies that will be **compatible with the marketplace** (e.g. non-conflicting JavaScript libraries).

The third characteristic the Linker should have is the distinction between the frontend UI and the backend business logic and services. This decoupling will allow the frontend application to be more agile and dynamic and the backend services to be more robust and stable.

3.12.2 Technology Comparison and Selection

In this section, technologies for frontend and backend are analysed in order to find the most suitable ones that meet the needs of the particular SAM component. Firstly there will be a short presentation of the technologies and then a comparison of these technologies will take place in order to find the one which is the most suitable.

3.12.2.1 Possible Technologies and Comparison

The technologies have been categorised into two groups: the frontend technologies and the backend technologies. As already mentioned, the frontend technologies are the ones that will be used for the user interface and the backend technologies will be used for creating the core services that will be communicating with the frontend application, the external resources and the other SAM platform components (Marketplace, Cloud Storage, etc.).

3.12.2.1.1 Frontend Technologies (User Interface)

The frontend technologies analysis and their comparison for the Linker are aligned with Section 3.1.1. The Linker will be embedded inside the Marketplace and using the same technology with the Marketplace will make the adaption easier and will also eliminate any compatibility issues.

3.12.2.1.2 Backend Technologies (Web Services)

The Web Services are responsible for the manipulation of the data (as part of the Linker backend) and the communication with the front end and the other SAM Components. The Web Services must be stable and robust to errors. **JAX-RS** is a Java specification for RESTful Web Services that provides support in creating web services according to the Representational State Transfer (REST) architectural pattern. JAX-RS uses annotations. The implementation of the JAX-RS API is supported in many frameworks. The most common and widely used frameworks are the **Apache CFX**, **Jersey** and **spring.io**.

- **Apache CXF**[94] is an open-source, fully featured Web services framework. It originated as the combination of two open-source projects: Celtix developed by IONA Technologies (acquired by Progress Software in 2008) and XFire developed by a team hosted at Codehaus. CXF helps you build and develop services using frontend programming APIs, like JAX-WS and JAX-RS. These services can speak a variety of protocols such as SOAP, XML/HTTP, RESTful HTTP, or CORBA and work over a variety of transports such as HTTP, JMS or JBI.
- **Jersey**[95] RESTful Web Services framework is open-source, production quality framework for developing RESTful Web Services in Java that provides support for JAX-RS APIs and serves as a JAX-RS (JSR 311 & JSR 339) Reference Implementation. Developing RESTful Web services that seamlessly support exposing data in a variety of representation media types and abstract away the low-level details of the client-server communication is an easy task using Jersey. Jersey extends the JAX-RS toolkit with additional features and utilities to further simplify RESTful service and client development. Jersey also exposes numerous extension SPIs so that developers may extend Jersey to best suit their needs.

[94] http://cxf.apache.org/
[95] https://jersey.java.net/

- **Spring.io**[96] is an open-source application framework and inversion of control container for the Java platform. The framework's core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE platform. Although the framework does not impose any specific programming model, it has become popular in the Java community as an alternative to or even addition to the Enterprise JavaBean (EJB) model and also includes powerful features for the implementation of Java-based Web Services. The Spring.io platform includes Foundation Layer modules and Execution Layer domain-specific runtimes (DSRs). The Foundation Layer represents the core Spring modules and associated third-party dependencies that have been harmonised to ensure a smooth development experience. The DSRs provided by the Spring IO Execution Layer dramatically simplify building production-ready, JVM-based workloads.

Parameter	Importance	JAX-RS / Jersey	JAX-RS / Apache CXF	Spring.io
Generic Parameters				
Maturity & Stability	+++	++	+	+++
Regularly Updated	+	+	+	++
Technical Up-to-Datedness / Appeal	++	+++	+++	+++
Open Source	++	+++	+++	+++
Non-Infecting	++	N/A	N/A	N/A
Code Quality	++	+	+	+++
Extensibility	+++	YES	YES	YES
Community	++	-	-	-
Performance / Scalability	++	++	++	+++
Reuse of existing developments	-/+	N/A	N/A	N/A
EU project origin	--	N/A	N/A	N/A
Platform (Portability)	++	+++	++	++
Open Standards Compliance	+++	++	+	++
Interoperability (easy integration for all platforms)	++	++	+	++
Specific Parameters				
Operating System	++	Cross-Platform	Cross-Platform	Cross-Platform
Database Compatibility	++	N/A	N/A	+++
Reliability	++	N/A	N/A	+++
Programming Language	+/-	Java	Java	Java
Documentation level	+/-	N/A	N/A	+++

Figure 157: Parameter Evaluation of Technologies for Web Services

3.12.2.2 Technology Selection

After comparing the technologies in the previous section, the selection of technology for each subcomponent is explained below.

[96] http://spring.io/

3.12.2.2.1 Frontend Technologies (User Interface)

The user interfaces Asset Editor, Asset Preview and Asset Module Configurator will be implemented using the technology AngularJS, which was chosen in Section 3.1.1.2. Also because the aforementioned user interfaces will be embedded inside the marketplace it is reasonable to choose this technology to provide a common look and feel.

3.12.2.2.2 Backend Technologies (Web Services)

The Linker prototype will use the JAX-RS technologies and more specifically the spring.io framework in order to implement the RESTful Web Services for Linker. Even though Jersey is more suitable for a simple RESTful implementation, because it is not overloaded with features and dependencies, spring.io is powerful, robust, complete in terms of features and well-maintained. This should considerably ease the implementation, deployment and testing of the Linker-backend environment.

3.12.3 Technical Component Specification

3.12.3.1 Structure

Figure 158 provides an overview of the Linker Component, which can be found also in D3.2.1 Global Architecture Definition. Additionally, this version of the overview in this chapter has been augmented with the selected technologies.

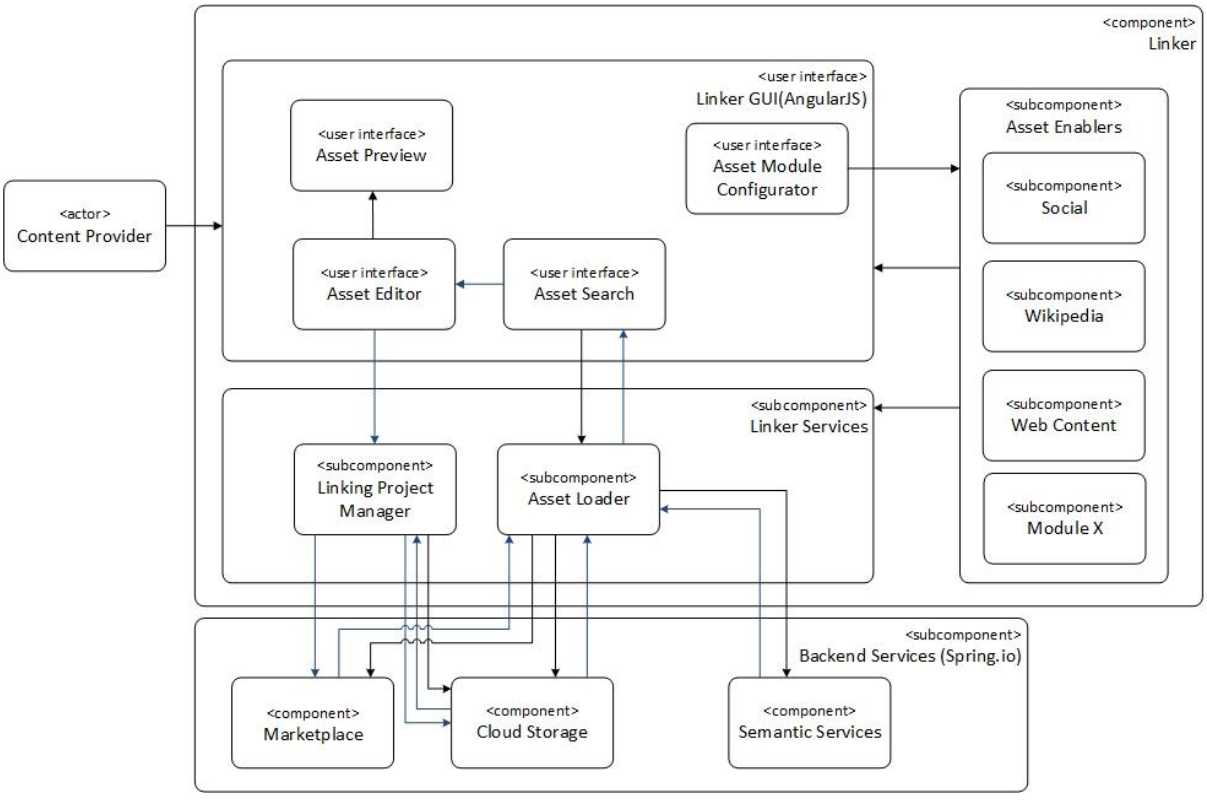

Figure 158: Overview with Selected Technologies – Linker

3.12.4 Specification of Interfaces, Protocols and Formats

The services of the Linker will be provided as RESTful interfaces (described in Section 3.12.4.1). The Linker is the responsible for the asset linking process. The communication

between the Linker component and the other SAM components is established through JSON messages as is shown in the following tables.

3.12.4.1 RESTful Interfaces

In order to describe the RESTful interface, each provided service is described in a separate table. This table contains an example for the JSON parameter (if applicable) as well as an example for the return value (if applicable).

3.12.4.1.1 Method “Linking Project Management”

The Linker Project Manager is the component responsible for the storage and loading of linking projects.

Linking Project Manager – Load Project		
Description	This functionality loads the project user choses to configure.	
Request		
Request URL	**GET** http://example.com/api/linker/project/:projectId	
Resource Parameters	Name	Description
	projectId Required String	ID of the project Example: *“testId”*
Response		
HTTP Status Code	Value	Description
	200	Project loaded successfully
	404	Project not found
	502	Internal Server Error

JSON Attributes	result
	Required JSON Object
		Example:
		{
		“projectId” : “id”,
		“project” : “project”
		}

Figure 159: RESTful Interface Description – Linking Project Manager – Load Project

Linking Project Manager – Save Project		
Description	This functionality saves the project.	
Request		
Request URL	**POST** http://example.com/api/linker/project/:projectId	
Resource Parameters	Name	Description
	projectId	ID of the project. If empty a new project will be saved and the ID will be returned.

	Optional String	Example**:** “*testId”*
JSON Attributes	project Required JSON Object	The linking project JSON Object.
Response		
HTTP Status Code	Value	Description
	201	Project saved
	401	Not authorised
	409	Project already exists
	500	Internal Server Error

JSON Attributes	result
	Required JSON Object
		Example:
		{
		“projectId” : “id”,
		“message” : “confirmation message”
		}

Figure 160: RESTful Interface Description – Linking Project Manager – Save Project

3.12.4.1.2 **Method “Asset Module Configuration”**

The Asset Module Configuration is responsible for loading UI elements as well as their parameters for the effective configuration of the modules.

Asset Module Configuration		
Description	This functionality loads the parameters for the specific module which is selected in the module enabler interface.	
Request		
Request URL	**GET** http://example.com/api/linker/module/:moduleId	
Resource Parameters	Name	Description
	moduleId Optional String	The ID of the module in case a specific one need to be loaded. Example**:** “*testId”*
Query Parameters	Module Parameters Object Optional key-value pairs	List of specific parameters for loading widgets Example: ?moduleType=aType&parameter=param1:value1&parameter=param2:value2 “value” : “param1value”
Response		
HTTP Status Code	Value	Description
	200	OK

	404	Module not found
	500	Internal Server Error
JSON Attributes	Module configuration Object Required JSON Object	JSON Object representing the widget configuration

Figure 161: RESTful Interface Description – Asset Module Configuration

3.12.4.1.3 **Method "Asset Search"**

The Asset Search functionality will allow locating assets based on the keywords and criteria provided by the user of the Linker component.

Asset Search		
Description	This functionality allows the search of assets stored on the SAM Cloud storage or published in the Marketplace	
Request		
Request URL	**GET** http://example.com/api/linker/asset/	
Resource Parameters	Name	Description
	None	

HTTP Parameters	Optional key-value pairs	Example:
		?assetname=aType&parameter=name1:value 1&parameter=name2:value2Parameters for querying assetsExample:
		[
		{
		"name1" : "productionYear",
		"value1" : "2004"
		},
		{
		"name2" : "country",
		"value2" : "Greece"
		}
		]

Response		
HTTP Status Code	Value	Description
	200	OK
	500	Internal Server Error
JSON	list	List of assets based on the query

Attributes	Required list	Example: *[{asset1}, {asset2}]*

Figure 162: RESTful Interface Description – Asset Search

3.12.4.1.4 **Method "Asset Editing"**

This is the core functionality of the Linker. The Linker user will be able to orchestrate the various assets that will be part of the linking projects and define in detail the various parameters for each one of the linked assets. Part of asset editing is the publication of the composed asset to the cloud storage and marketplace. This includes the validation of the linking parameters so as to be flawlessly and effectively syndicated by the SAM Platform.

Asset Editing – Load Asset		
Description	This functionality is responsible loading the descriptions of specific assets.	
Request		
Request URL	**GET** http://example.com/api/linker/asset/:assetId	
Resource Parameters	Name	Description
	assetId Required String	The ID of the asset in order to load its description. Example**:** "*testId"*
Response		
HTTP Status Code	Value	Description
	200	OK
	404	Asset not found
	500	Internal Server Error
JSON Attributes	asset Required JSON Object	The JSON representation of an asset object

Figure 163: RESTful Interface Description – Asset Editing – Load Asset

Asset Editing – Save Asset		
Description	This functionality is responsible for saving the asset composition	
Request		
Request URL	**POST** http://example.com/api/linker/asset/:assetId	
Resource Parameters	Name	Description
	assetId Required String	The ID of the asset to be saved Example**:** "*testId"*
HTTP Parameters	asset Required JSON Object	JSON representation of the asset object

Response		
HTTP Status Code	Value	Description
	200	OK
	404	Asset not found
	500	Internal Server Error

JSON Attributes	result
	Required JSON Object
		Example:
		{
		“assetId” : “id”,
		“message” : “confirmation message”
		}

Figure 164: RESTful Interface Description – Asset Editing – Save Asset

3.12.5 Summary

In this section, the Linker technical specification is described and technologies for the data management in the backend are analysed. The frontend implementation will be based on AngularJS, the same technology for the front end of the Marketplace. The Backend technology will be a RESTful application, which will communicate with the Frontend and other SAM components via JSON messages. The implementation of the Backend application is going to be created with the spring.io framework.

3.13 Analytics

Analytics is the component in charge of providing business-related reports to the different stakeholders. These reports will be based on business intelligence approaches (BI component) and advanced social mining techniques (Social Mining component) such as social graph analysis and natural language processing. The components’ place in the architecture is depicted in the following figure:

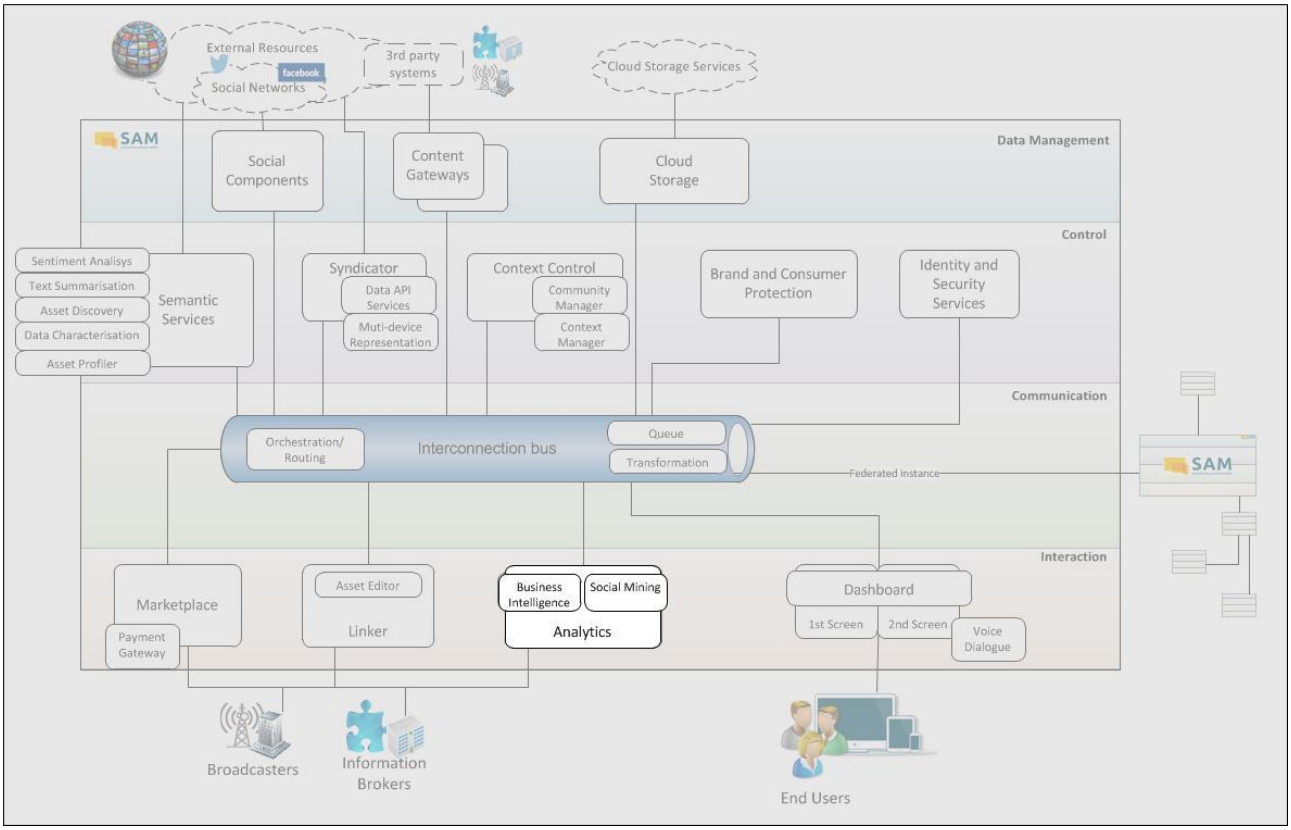

Figure 165: Architecture Overview – Analytics

3.13.1 Social Mining

Social Mining is the component that provides sentiment analysis features from User-Generated Content (UGC) for the SAM platform (see Figure 166). It also provides capabilities for the summarisation of large amounts of UGC required by the Business Intelligence subcomponent in order to create advanced reports. This component makes use of different controllers for accessing the Semantic Services technologies in order to process the UGC. The Business Intelligence component will use the above functionalities for applying Data Mining through the Social Mining web service.

- As part of the Analytics component, this subcomponent will retrieve user comments and other UGC from the Cloud Storage for further analysis. The Social Mining will be able to receive incoming queries for extracting the UGC.
- UGC will be analysed from a quantitative and qualitative point of view, using advanced data mining and semantic techniques provided by the Semantic Services component. The Social Mining component will directly interact with different components of the SAM Platform providing semantic and sentiment analysis features as well as summaries of large quantities of UGC.
- From the obtained results, the Social Mining subcomponent will provide semantic and sentiment analysis features to the Business Intelligence subcomponent according to analysed UGC as well as UGC summaries.

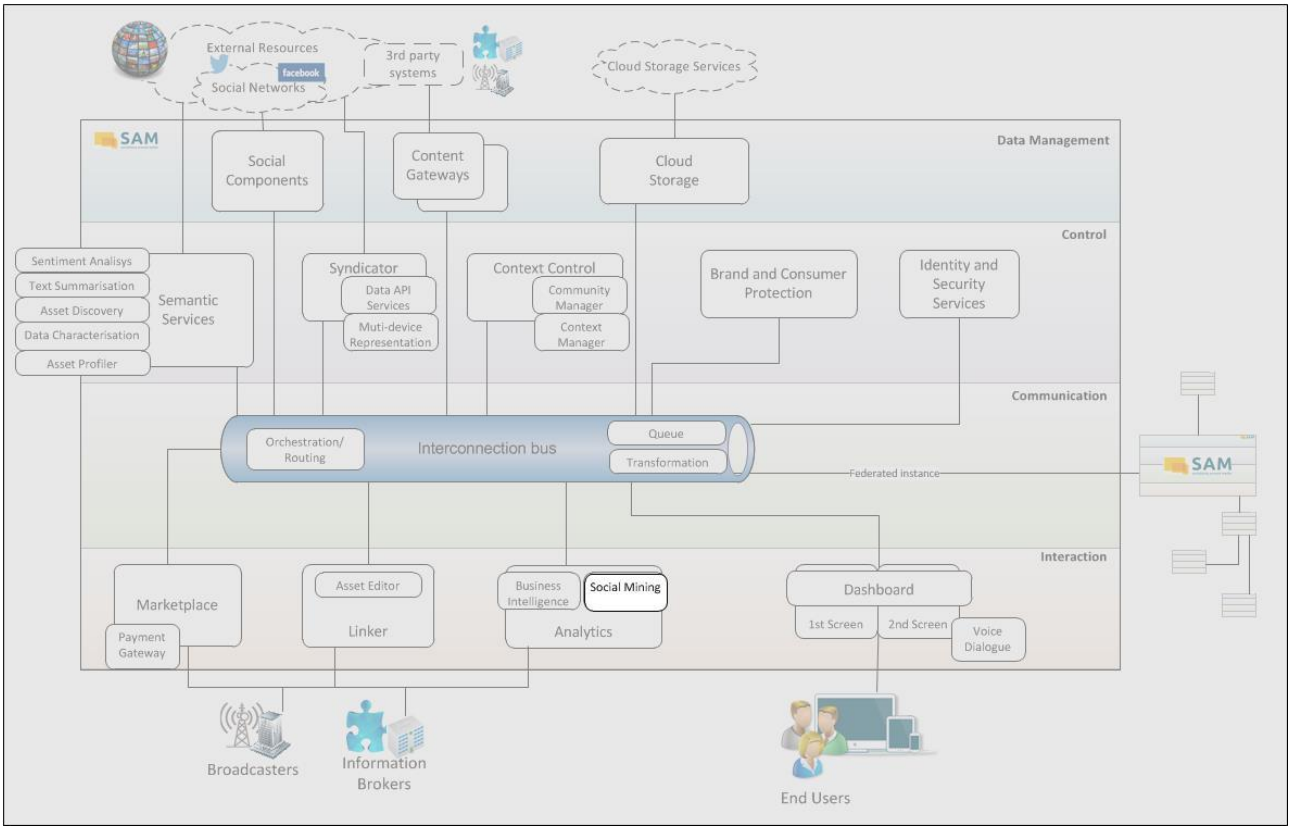

Figure 166: Architecture Overview – Social Mining

3.13.1.1 Major Design Decisions

The focus of the Social Mining is on addressing three important tasks: semantic analysis feature extraction, sentiment analysis feature extraction and the summarisation of UGC. For that, two interfaces will be created in order to take advantage of the functionalities provided by the Semantic Services component. The aim is to retrieve UGC and shape the requests to the semantic and sentiment analysis controllers.

To start the Social Mining process, the main controller will retrieve UGC (i.e. textual data gathered from user's social interaction) by querying the Cloud Storage, and will proceed to plan and manage overall sentiment analysis and semantic processes, based on the reports from the Business Intelligence subcomponent. As a result, a set of sentiment and semantic features related to the query data will be obtained. In addition, this set of features will also be organised as a single bundle to be provided to the Business Intelligence subcomponent.

In order to address the extraction of characterisation features, the Social Mining subcomponent will invoke the Characterisation interface of the Semantic Services component, which will carry out all the semantic processing functionalities required. As a result, different semantic features related to the analysed text (such as asset metadata, keywords, and ontology labels), will be extracted and sent to the Social Mining component using standard formats.

Sentiment analysis and summarisation functionalities will be addressed by the Sentiment Analysis interface from the Semantic Services component. This method will be called by the Social Mining subcomponent and will receive commands and the necessary input data to exploit the aforementioned functionalities of the Semantic Services component. As a result, sentiment polarities, intensity scores and emotion labels related with the analysed

UGC, as well as a summarised version of this UGC, will be sent to the Social Mining subcomponent using standard formats.

As can be seen, several types of consumption of functionalities are involved in each task and they are provided by the Semantic Services component.

3.13.1.2 Technology Comparison and Selection

This subsection outlines technology selection criteria and compares potential technology candidates for the implementation of the Social Mining subcomponent. Since the Social Mining component delegates all data processing in the Semantic Services, the technology comparison for addressing this task can be found in Section □ and will not be repeated here. This section will only show specific decisions taken in the context of the Social Mining component regarding technology selection.

3.13.1.2.1 Possible Technologies and Comparison

As explained in the section before, the introduction of eligible technology candidates can be found in Section □.

3.13.1.2.2 Technology Selection

The technology necessary for the development of this subcomponent does not depend on the component itself, since the semantic analysis, sentiment analysis and summarisation functionalities will be consumed through web services provided by the Semantic Services component. That is to say, the language processing tasks required in this component are carried out by the Semantic Services component and the technologies involved. A detailed description of these technologies is provided in Section 3.3.2.2.

3.13.1.2.2.1 Natural Language Processing Tools

Open NLP and Lucene were the tools selected, since both have free licenses and they cover most of the NLP tasks necessary to deal with text processing and indexing. In addition, Apache Commons Lang + Jexl libraries were selected due to the fact that they provide extra methods for dealing with text matching and logical expressions evaluation that align content inputs with the SAM knowledge base. With respect to the summarising tool, Compendium was selected since it comprises state-of-the-art summarisation techniques and it is easy to adapt to SAM specific requirements.

3.13.1.2.2.2 Semantic APIs

For supporting the specific parameters related to semantic management and ontology exploitation, Sesame client API was chosen. This tool provides a perfect match between the client software application and database, since both are part of the same framework. This API allows exploiting the semantic information stored in the SAM semantic repository.

3.13.1.2.2.3 Semantic Repositories

Different kinds of semantic repositories were chosen which will provide linguistic information to the sentiment analysis and semantic procedures. As part of these repositories, lexical databases such as WordNet and its related projects were considered. In this way, Social Mining subcomponent will make use of data from:

- WordNet, an English dictionary at sense level;

- WordNet Domains and WordNet Affect, which are domain taxonomies that categorise WordNet senses;
- And also SentiWordNet, a resource linked to the WordNet senses indicating their sentimental polarity scores.

Moreover, sentiment analysis algorithms require training instances to build the classification models. Thus, Social Mining specific examples to train the models will be gathered from datasets such as EmotiBlog and Semeval. These corpora provide annotated opinions which are useful to train sentiment analysis models.
DBpedia was selected as a knowledge base. This resource provides a semantic network that includes Wikipedia articles, offering a large amount of semantic information that is necessary for addressing the semantic characterisation.

All of these resources are under licenses that permit supporting the exploitation plan.

3.13.1.2.2.4 Machine Learning Tools

The specific parameters of the Social Mining component largely depend on machine learning techniques, since the language analysis processes carried out in the Semantic Services require machine learning algorithms in order to build more robust text analysers. MALLET was selected as the machine learning tool to be employed for the required text processing applications. This tool offers functionalities such as text classification, text clustering and topic modelling. It is licensed under Common Public License Version 1.0 (CPL), a non-viral license.

3.13.1.3 Technical Component Specification

3.13.1.3.1 Structure

Figure 167 provides an overview of the Social Mining component, also available in deliverable D3.2.1 Global Architecture Definition. The version shown here has been augmented with the selected technologies.

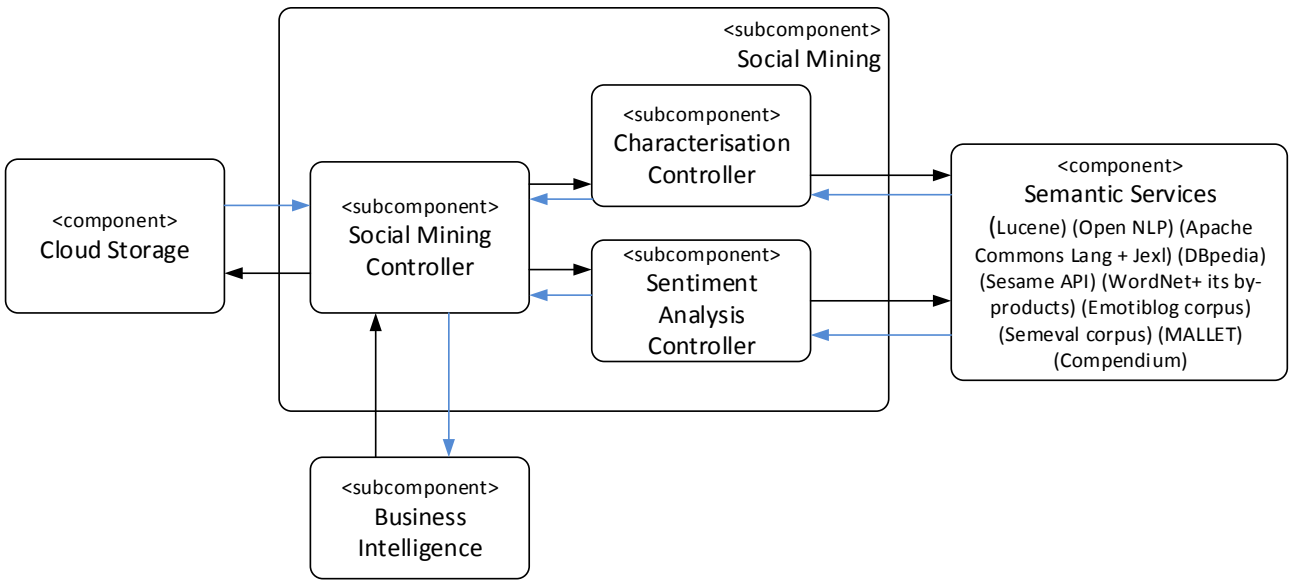

Figure 167: Overview with Selected Technologies – Social Mining

3.13.1.4 Specification of Interfaces, Protocols and Formats

The functionalities of the Social Mining subcomponent will be provided as web methods using SOAP Web Services (WS) (see Section 3.13.1.4.1). The outcomes provided by Social Mining functionalities will be JSON objects. The representation structure of these objects is described in Section 3.13.1.4.3.1.

3.13.1.4.1 Web Services Interfaces

In order to describe the different functionalities, each web method of the Social Mining WS is shown in a separate table.

3.13.1.4.1.1 Method "Process Social Data"

The web method "Process Social Data" is able to identify different semantic and sentiment analysis features involved in a given text. These semantic features or semantic units are part of the semantic repositories and the SAM Ontology (i.e. asset database) included in SAM. Given an input query for retrieving UGC related to an asset, this web method will be able to retrieve and analyse UGC from the Cloud Storage. It is worth mentioning that two parameters of this method are optional: the subjects to analyse (entities in UGC to analyse) and the depth of analysis.

Process Social Data		
Description	Consumes the characterisation functionality provided by the Semantic Service component providing semantic units related with UGCs. It also provides sentiment analysis features regarded to the UGCs once exploited the Semantic Services functionalities.	
Request		
Request URL	**POST** http://example.com/api/SocialMining?wsdl	
Web Method	ProcessSocialData	
HTTP Parameters	Name	Description
	assetIds Required string	A JSON object that represents a list of asset IDs for retrieving asset's UGCs from the Cloud Storage. Example: *{["a001", "a002"]}*
	subjectList Optional list of text	subjectList: a JSON object that represents a list of subject to evaluate with respect to asset's UGCs. Example: *{["Mads_Mikkelsen", "Ford"]}*
	analysisDepth Optional numeric value	An integer to set the depth level of analysis inside the assets. Example: {2}
	listOfTechniques Required string list	A JSON object that represents a list of strings that identifies the techniques to use. In this case Characterisation and Sentiment Analysis.

		Example:
		{
		“techniques”:
		[
		“semantic_features_extraction”,
		“sentiment_analysis_features_extraction”
		]
		}

Response		
HTTP Status Code	Value	Description
	100	Successful operation
	200	Object not found
	300	Command waiting in queue
	500	Fatal error
JSON Attributes	list Optional Social Features list	A list of JSON objects with assets. In case of the attribute "subject" of this output, an "OVERALL" value will be set for the sentiment analysis evaluation of the entire comment, and additional evaluations for every "subject". Example: [{ “userId”: “100”, “followers”: “32”, “semantic features”: [{ “comment”: “01”, "relatedLabelIds": [“ wn1”,”db10”], "relatedLabels": [“ Entertainment”, ”Actor”] "relatedLabelResources": [“WordNet”,”DBpedia”], "relationType": [“hypernym”,” hypernym”] }, { “comment”: “02”,

		"relatedLabelIds": [“ wn22”, "db45”, "db10”, "a23"],
		"relatedLabels": [“film”,” James Bond”, “star”, “Daniel Craig”],
		"relatedLabelResources": [“WordNet”,”DBpedia”, ”Asset”],
		"relationType":
		[“is_a”,”part_of”,”is_a”, “undefined”]
		}
		],
		“sentiment features”:
		[
		{
		“comment”: “01”,
		“subject" : "OVERALL",
		“positiveIntensity”: “0.75”,
		“negativeIntensity”: “0.15”,
		“emotionLabels”: [“joy”],
		“sentimentCategory”: “Positive”
		},
		{
		“comment”: “01”,
		"subject" : "Mads_Mikkelsen",
		“positiveIntensity”: “0.75”,
		“negativeIntensity”: “0.15”,
		“emotionLabels”: [“joy”],
		“sentimentCategory”: “Positive”
		},
		{
		“comment”: “02”,
		"subject" : "OVERALL",
		“positiveIntensity”: “0.75”,
		“negativeIntensity”: “0.15”,
		“emotionLabels”: [“joy”],
		“sentimentCategory”: “Positive”
		}

		] }]

Figure 168: Web Service Interface Description – Processing Social Data

3.13.1.4.1.2 Method "Summarise"

The web method interface "Summarise" is able to summarise the result of a given query (i.e., UGC from specific assets) from the Cloud Storage. In this method it is necessary to provide the target compression rate (e.g., 20% compression) that will be employed, and optionally the depth of analysis to be performed (by default the deep set value 1).

Summarise		
Description	This functionality is able to retrieve summaries form large quantity of UGC queried from the Cloud Storage.	
Request		
Request URL	POST http://example.com/api/SocialMining	
Web Method	Summarise	
HTTP Parameters	Name	Description
	assetIds Required string	A JSON object that represents a list of asset IDs for retrieving asset's UGCs from the Cloud Storage. Example: *{["a001", "a002"]}*
	analysisDepth Optional numeric value	An integer to set the depth level of analysis inside the assets. Example: *{2}*
	compressionRatio Required numeric value	This integer number indicates the reduction of the output text. Example: *{"compressionRatio": "20"}*
Response		
HTTP Status Code	Value	Description
	100	Successful operation
	200	Object not found
	300	Command waiting in queue
	500	Fatal error
JSON Attributes	Summary Optional string	It is JSON object which contains the reduced UGC. Example: {

		“Summary”: “Casino Royale is a great movie. Some of the recent James Bond themes are about not trusting anyone. You Know My Name - From James Bond - Casino Royale “ }

Figure 169: Web Service Interface Description – Summarise

3.13.1.4.2 **Java Interfaces**

The Java interfaces act as wrappers for the SOAP interfaces. They provide a convenient way to access the Social Mining APIs for developers.

3.13.1.4.2.1 Method “Process Social Data”

The Java interface “Process Social Data” needs four parameters to carry out its work. The descriptions of the parameters for each method are shown in the table below.

Process Social Data	
Method Header	public String ProcessSocialData(String assetIDs, String subjectList, Int analysisDepth, String listOfTechniques)
Parameters	assetIds a JSON object that represents a list of asset IDs for retrieving asset’s UGCs from the Cloud Storage. subjectList: a JSON object that represents a list of subject to evaluate with respect to asset‘s UGCs. analysisDepth: an integer to set the depth level of analysis inside the assets. techniques: a JSON object that represents a collection of words that indicate the techniques to apply. Those can be semanticFeaturesExtraction and/or sentimentFeaturesExtraction.
Return Value	A JSON object, socialDataFeatures, for representing semantic and sentiment analysis features extracted from the analysed UGCs.
Error Handling	In case of an error, a null-value is returned
Remarks	None

In Figure 170 the header of the method is shown.

```
/**
 * This is a sample web service operation
 * @param assetIds a JSON object that represents a list of Asset IDs for retrieving Asset's
 * UGCs from the Cloud Storage.
 * @param subjectList a JSON object that represents a list of subject to evaluate with
 * respect to Asset's UGCs.
 * @param analysisDepth an integer to set the depth level of analysis inside the assets.
 * @param listOfTechniques   a collection of words that indicate the techniques to apply.
 * Those can be semanticFeaturesExtraction and/or sentimentFeaturesExtraction.
 * @return A JSON object,  SocialDataFeatures, for representing semantic and sentiment
 * analysis features  extracted from the analysed UGCs
 */
@WebMethod(operationName = "processSocialData")
public String processSocialData(@WebParam(name = "assetIds") String assetIds,
                                (@WebParam(name = "subjectList") String subjectList,
                                (@WebParam(name = "analysisDepth") Int analysisDepth,
                                @WebParam(name="listOfTechniques") String listOfTechniques) {
      String socialDataFeatures = null;
      ...
      return socialDataFeatures;
}
```

Figure 170: Source Code Example – API Method Signature for Process Social Data

If the API call is successful a list of SocialDataFeatures objects will be returned with its semantic attribute filled, otherwise it returns null.

3.13.1.4.2.2 Method "Summarise"

For applying text summarisation, the API provides one method. It has three parameters.

Summarise	
Method Header	public String Summarise(String assetIds, Int analysisDepth, Int compressionRatio)
Parameters	assetIds: a JSON object that represents a list of asset IDs for retrieving asset's UGC from the Cloud Storage. analysisDepth: an integer to set the depth level of analysis inside the assets. compressionRatio: it is an integer that represents the percentage that the UGC will be reduced.
Return Value	The UGC summary
Error Handling	In case of an error, a null-value is returned
Remarks	None

In Figure 171 the header of the method is shown.

```
/**
 * This functionality is able to retrieve summaries form large quantity of UGC queried from
 * the Cloud Storage.
 * @param assetIds a JSON object that represents a list of Asset IDs for retrieving
 * Asset's UGCs from the Cloud Storage.
 * @param analysisDepth an integer to set the depth level of analysis inside the assets.
 * @param compressionRatio the percentage that the UGC will be reduced
 * @return  a list of UGCs reduced, in JSON format
 */
@WebMethod(operationName = "summarise")
public String summarise(@WebParam(name = "assetIds") String assetIds,
              (@WebParam(name = "analysisDepth") Int analysisDepth,
              @WebParam(name="compressionRatio") String compressionRatio) {
    String listOfUGCsReduced = null;
    ...
    return listOfUGCsReduced;
}
```

Figure 171: Source Code Example – API Method Signature for Summarise

If the API call is successful, a JSON object serialised as a string will be returned. Otherwise it returns null.

3.13.1.4.3 Content Formats

3.13.1.4.3.1 Java Data Schema (Social Mining Data Objects)

This section lists the Java data schemas which will be used by the Social Mining information infrastructure internally as well as for the communication with other components via Web Services.

All Social Mining Data Objects (SMDO) will be used for the transfer of structured data for Social Mining (internally and externally). The list of objects described in this section is not complete and subject to changes to accommodate the special needs of external components which will use the Social Mining.

We will transfer SMDO using JSON objects. The following are the different types of outputs which are coded in Java, but the outputs will be serialised and transformed into JSON objects:

- For transferring Semantic Features, the representation is shown in Figure 172
- For transferring Sentiment Analysis Features, the representation is shown in Figure 173
- For transferring Summaries, the representation is shown in Figure 175
- For transferring Social Data Features, the representation is shown in Figure 174

```
public class SemanticFeatures {

        public enum Resource { WORDNET, DBPEDIA, ASSET};
        public enum Relation {IS_A, PART_OF, HYPERONYM, HYPONYM, UNDEFINED};

        private List<String> relatedLabelIds;
        private List<Resource> relatedLabelResources;
        private List<Relation> relationType;
}
```

Figure 172: Java Data Schema – Semantic Features

```
 public class SentimentAnalysisFeatures {
    private enum Sentiment {POSITIVE, NEGATIVE}

    private double positiveIntensity;
    private double negativeIntensity;
    private List<String> emotionLabels;
    private Sentiment sentiment;
}
```

Figure 173: Java Data Schema – Sentiment Analysis Features

```
public class SocialFeatures {

    private String userID;
    private int followers;
    private SemanticFeatures semanticFeatures;
    private SentimentFeatures sentimentFeautures;
}
```

Figure 174: Java Data Schema – Social Data Features

```
public class Summary {
    private String summary;
}
```

Figure 175: Java Data Schema – Summary

3.13.1.5 Summary

In this section, the Social Mining subcomponent has been technically specified. During the technology selection (Section 3.13.1.2.2), it was mentioned that this component will be strongly related to the Semantic Services component. That means that this component will be using and consuming the Semantic Services functionalities. Therefore, the Social Mining subcomponent will rely on the text analysis functionalities of the Semantic Services to process Social Media data.

In addition, the service interfaces of the Social Mining subcomponent have been defined in this section. This service will use the functionalities provided by the Semantic Services component to obtain semantic and sentiment analysis features, as well as summaries from UGC. The outcome of Social Mining will be employed by the Business Intelligence subcomponent, which will receive JSON objects where all necessary data will be included in a simple structure.

3.13.2 Business Intelligence

In this section, the functionalities and interfaces to carry out the quantitative and qualitative analysis needed to provide business related reports to the Media Broadcaster and Information Broker actors are defined. The Business Intelligence (BI) subcomponent is in charge of providing BI reports to the different stakeholders, so that they can make decisions based on the reports provided. Figure 176 shows the location of the BI subcomponent within the SAM platform. The main functionalities of this subcomponent are:

- Providing a graphical interface to help the actors define BI reports and visualise them.

- Implementing the mechanism to manage and analyse the information in the Cloud Storage from a BI context in order to build these reports
- Interacting with Social Mining component in order to provide sentiment analysis information using advanced data mining and semantic techniques

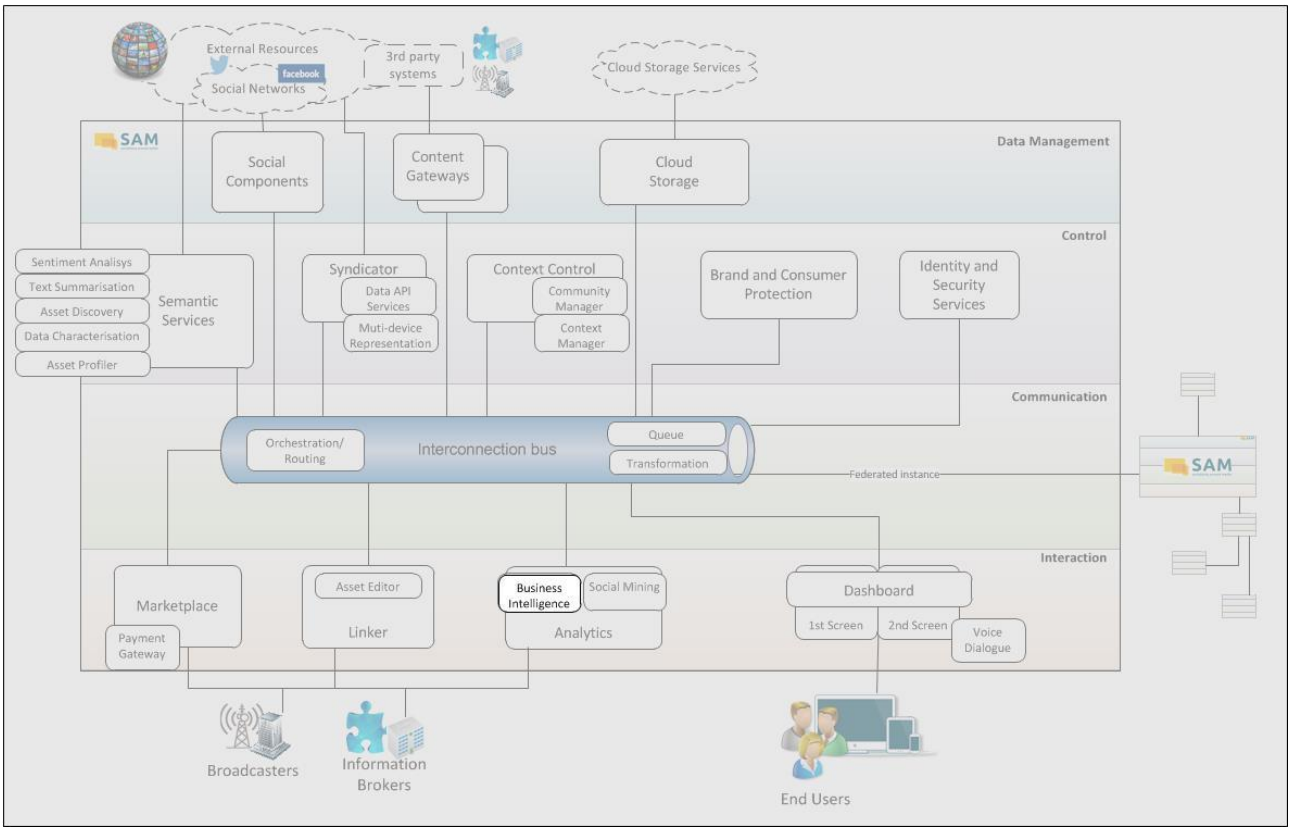

Figure 176: Architecture Overview – Business Intelligence

3.13.2.1 Major Design Decisions

The Business Intelligence subcomponent provides the reports and analysis in the SAM platform. Therefore it is of maximum importance that the selected technology provides the following features:

- **An Intuitive Web User Interface:** Content Providers will need to define and generate reports in an easy and fast way in order to make decisions based on these reports around the clock. For this reason, it is crucial to provide a user-friendly web user interface that provides access anytime, anywhere.
- **Ad-Hoc Reporting and Analysis:** SAM's BI solution must offer simple ad-hoc reporting capabilities make it easy for any worker level, including high-level staff members, to quickly build and run their own reports any time they need them.
- **Flexible Formatting Options:** SAM Content providers should be able to present their information in different ways. Therefore the selected BI technology must allow users to output their reports as Excel spreadsheets, Word documents, web pages, Adobe PDF files, or other common formats.
- **Dynamic Information Distribution:** The BI tool should allow SAM Content Providers to schedule reports to automatically run at pre-set days and times. This helps ensuring that the information being used to support decision making is updated at regular intervals, so it is up-to-date and accurate.

- **Modular Design:** Decoupling the data integration and data warehouse implementation from the user interface with reporting and analytics functionality will allow the maximising of capabilities and a flexible design.

3.13.2.2 Technology Comparison and Selection

The implementation is separated into two sub-systems, the backend system consisting of the data integration and Data warehouse components, and the frontend system responsible for the user interface with reporting and analytics functionality. The following sections outline the technology selection criteria and compares existing technologies for the two the two separate subsystems – potential candidates for the realisation of BI Services. Within the subsections the areas that need to be implemented or improved to fully meet SAM specific requirements are also identified and presented. The selected technologies will be used as a base for the development phase in the realisation of the technical design of this component: The specific selection criteria to be used are specified in the deliverable D3.2.2 Functional Specification, Section 4.13.2.4 and will be used in the next sections.

3.13.2.2.1 Possible Technologies and Comparison

For a more modular design which can take advantage of the unique qualities of different technologies, the implementation is separated into two subsystems. The implementation is composed of the backend subsystem consisting of the data integration and data warehouse components, and the frontend subsystem responsible for the user interface with reporting and analytics functionality. The following sections describe to the technology selection and comparison of these subsystems.

3.13.2.2.1.1 Backend Subsystem

Revolution eXtreme is a data integration and data warehouse solution provided by TIE Kinetix. It provides a very unique solution that automatically generates the underlying data warehouse and data integration artefacts based on a simple high level configuration which results in very fast and easy integration and deployment of the BI backend system. No other product currently on the market provides such an automated data warehouse generation solution. Since building and maintaining a quality data warehouse is arguably the hardest and most crucial part of any quality BI deployment, Revolution eXtreme is uniquely positioned within the BI market.

The generated data warehouse is based on the Microsoft industrial BI standards: Microsoft SQL Server as a database platform, SQL Server Integration Services as an ETL platform and SQL Server Analysis Services as online analytical processing platform. The Microsoft BI stack supports and is supported by a multitude of data sources and frontend systems.

Revolution eXtreme provides an intuitive graphical configuration user interface rather than coding. It also provides an automatic data warehouse documentation feature; the full data warehouse description is stored in the metadata repository and can be easy accessed or exported into different formats. It also supports enhanced traceability of data.

Revolution eXtreme is therefore a unique product that is very suitable for implementing the SAM BI backend subsystem. Since it is based on the generic Microsoft's BI suite, it can also be easily replaced in future without risk of vendor lock-in.

A full comparison with other backend systems is therefore not required since no other product provides such an automated solution resulting in very efficient deployment times while being based on generic technologies.

3.13.2.2.1.2 Frontend Subsystem

In order to implement the BI frontend subsystem, several options have to be considered as base technologies. The following selection will compare the most promising frontend technologies:

- **Logi Analytics (formerly Logi XML)**[97] focuses on the user-facing aspect of BI offering a very easy-to-use and embeddable platform that includes reporting, analysis and dashboards for both IT and business users, plus data integration. Licencing is based on number of cores or CPUs and not number of users which is a perfect model for the SAM use case. It supports a wide variety of data sources, both cloud and on-premises. It provides ad-hoc analysis and OLAP as part of its analysis features. For reporting it provides ad-hoc reporting and customisable features and dashboards.
- **Good Data**[98] is a cloud BI and analytics specialist delivering a complete BI solution as Software as a Service. It provides a range of front-end BI capabilities and packaged analytic applications that complement its comprehensive cloud and on-premises source data integration and cloud-based data warehouse platform. It also focuses on embedded deployments. It provides an OLAP-integrated user interface for analytics functionality. For reporting, it provides ad-hoc reporting, customisable features, dashboards, and performance measurements. It is highly rated by Gartner for ease of use for both end users and developers.
- **Birst**[99] BI platform is primarily a cloud-based offering. It includes a broad range of components, such as data integration, federation and modelling, a data warehouse with a semantic layer, reporting, dashboards, mobile BI and a recently announced interactive visualisation tool. The BI solutions also work with many existing systems, such as Salesforce, SAP, or Microsoft Analysis Services. Total cost of ownership is its biggest selling point, with functionality and implementation cost and effort also rated high as determined by Gartner's Magic Quarter Report of 2014[100]. Shortcomings are limited reporting features and since it is Adobe Flash-based and iOS users are not supported.
- **Jaspersoft** [101] sells an end-to-end, open-source BI and data integration platform featuring a low-cost-of-ownership value proposition. It is often used to build embedded BI applications. Jaspersoft has a scalable, modular, standards-based design that allows the flexibility needed for a wide variety of deployments from on-premises to cloud. It supports many dig-data data-sources and connects to Cassandra, MongoDB, and Hadoop. It provides ad-hoc easily shared reporting, customisable dashboards, and performance measurements as reporting features. Its analysis features include ad-hoc and predictive analysis, OLPA, trend indicators, and a user-friendly interface.
- **Microsoft**[102] offers a competitive and expanding set of BI and analytics capabilities, packaging and pricing. Its reporting and analytics features are primarily based on Excel with the Power View add-on while the data integration is handled by Power Pivot and

[97] http://www.logianalytics.com/
[98] http://www.gooddata.com/
[99] http://www.birst.com/
[100] https://www.gartner.com/doc/2668318/magic-quadrant-business-intelligence-analytics
[101] https://www.jaspersoft.com/
[102] http://www.microsoft.com/en-us/server-cloud/solutions/business-intelligence/

Power Query with the data warehouse implemented on SQL Server. On-premises as well as cloud deployments are supported, both via SharePoint portal. Being based on standard Microsoft Office products, a Microsoft BI solution is very attractive as the components are often already deployed in many companies; it is a complete integrated solution with ease of accessibility, as well as the availability of skilled resources.

Parameter	Importance	Logi Analytics	Good Data	Birst	Jaspersoft	Microsoft
General Parameters						
Maturity & Stability	+++	++	+++	++	++	+++
Regularly Updated	+	+++	+++	++	+++	+++
Technical Up-to-Datedness / Appeal	++	+++	+++	++	++	+++
Open Source	+	NO	NO	NO	YES	NO
Non-Infecting	++	YES	YES	YES	YES	YES
Code Quality	+	N/A	N/A	N/A	++	N/A
Extensibility	+++	NO	NO	NO	++	NO
Community	+	+	+	+	++	+
Performance / Scalability	++	++	++	++	+++	++
Reuse of existing developments	++	NO	NO	NO	NO	NO
EU project origin	--	NO	NO	NO	NO	NO
Platform (Portability)	+++	++	+	++	+++	+
Open Standards Compliance	+	N/A	N/A	N/A	N/A	N/A
Interoperability (easy integration for all platforms)	+++	+++	+++	++	++	+
Specific Parameters						
Intuitive User Interface	+++	+++	+++	++	++	++
Graphical Analytics	+++	+++	+++	++	+++	+++
Advanced Report Support	+++	+++	+++	+	+++	+++
Multiple Data Formats Export	+++	+++	+++	++	+++	++
Geographical Data Support	+	+++	+++	++	++	+++
Ad hoc reporting	+++	++	+++	++	++	++
Ad hoc data analyse	+++	++	+++	++	++	+

Figure 177: Parameter Evaluation of Technologies for Business Intelligence

3.13.2.2.2 **Technology Selection**

After comparing the above-mentioned front end technologies, Logi Analytics was selected for the user interface with reporting and analytics functionality. Logi Analytics was selected for the following reasons:

- It is primarily focused on the frontend areas of BI
- It provides flexible deployment options which can be tailored for any use case, stand-alone or embedded within on-premises, cloud, or any hybrid thereof
- Powerful intuitive user friendly interface focussed on enabling business users to quickly and easily create reports and dashboard as well as in-depth discovery and analysis of data

- The licencing of core processes only, without any end-user number restrictions, makes it specifically suitable for the SAM use case

As stated in Section 3.13.2.2.1.1, Revolution eXtreme was select for the backend subsystem for the following reasons:

- Its unique approach of automated data warehouse generation significantly speeds up and simplifies the data integration process, which is arguably the hardest part of any high quality BI solution. This results in increased end-user acceptance, shorter development cycles and higher trust in the data.
- As it seamlessly integrates will all Microsoft BI components, it can be connected to a wide variety of data sources as well as a wide range of front end systems. It is therefore independent of front end and data sources.

Since Revolution is a product of TIE Kinetix, one of the SAM partners, the appropriate expertise and resources are available to provide fast and efficient integration and extension of this product within the SAM platform.

3.13.2.2.3 User Interface Technology

As described in Section 3.13.2.2.2 the Logi Analytics product was selected witch already provides a graphical user interface. As a result, no specific user interface technology comparison has to be done.

3.13.2.3 Technical Component Specification

3.13.2.3.1 Structure

Figure 178 provides an overview of the Business Intelligence, which can be found also in D3.2.1 Global Architecture Definition, Section 4.12.2. The diagram has remained unchanged as no changes to the Business Intelligence component have been made since the specification in deliverable D3.2.1 Global Architecture Definition.

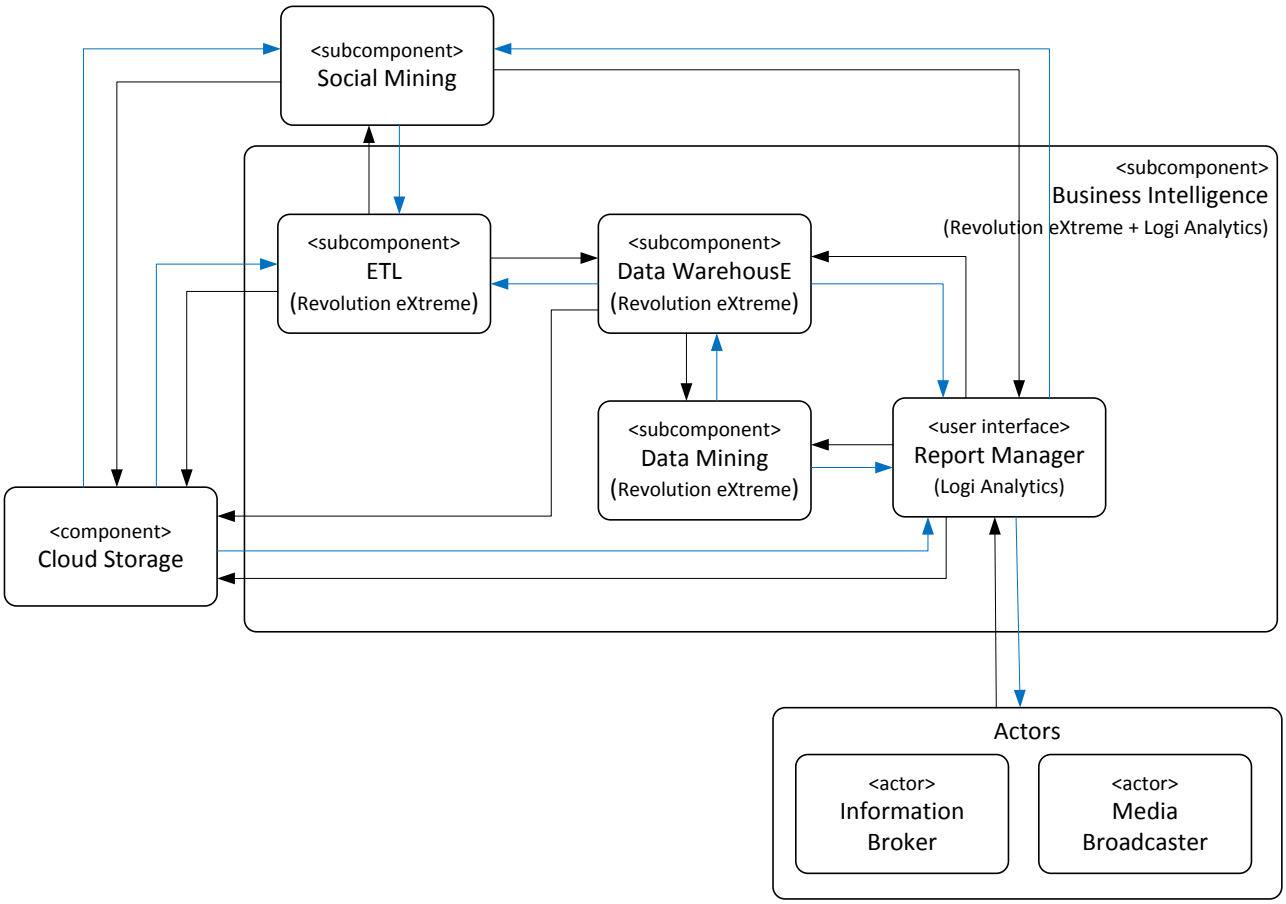

Figure 178: Overview with Selected Technologies – Business Intelligence

3.13.2.4 Specification of Interfaces, Protocols and Formats

The BI component does not provide any internal interfaces to any other SAM component. The only interface is the graphical user interface provided for the Actors to compose reports and analyse data. Since the BI component does not expose any interfaces, no formats or protocols are required or specified.

3.13.2.5 Summary

In this section the Business Intelligence component was technically specified. In Section 3.13.2.2.1.1, Revolution eXtreme was selected for backend subsystem with ETL and Data Warehousing functionality, while in Section 3.13.2.2.2, Logi Analytics was selected as frontend subsystem to provide the user interface with advanced reporting and analysis functionality for the Information Broker and Media Broadcaster actors to provide companies with the appropriate information to support further business decisions.

3.14 Dashboard

The Dashboard provides the End User with a Graphical User Interface for the interaction with the SAM platform. It is present on both 1st and 2nd Screen and consists of different widgets, having a few subcomponents in common (like widget representation and management, user authentication, maintaining device information, linking to the Syndication, etc.). Those common subcomponents were joined together under the Generic Dashboard. The Generic Dashboard will be present on both 1st Screen and 2nd Screen devices, on top of which each device will add its specific subcomponents.

Section 3.14.1 describes the technical specification of Generic Dashboard, i.e. the common components of both devices. Section 3.14.2 describes the 1st Screen specific components; Section 3.14.3 describes the 2nd Screen specific components; Section 3.14.4 describes the Inter-Device-Communication mechanism and finally, Section 3.14.5 focuses on the Voice Dialogue component.

3.14.1 Generic Dashboard

The Generic Dashboard application contains common subcomponents for 1st and 2nd Screen as shown in Figure 179. It will be able to play videos as well as display configurable widgets. It interacts with Syndicator and Social components to obtain relevant information.

- The Generic Dashboard holds a collection of common elements that the 1st and 2nd Screen components will use. Everything defined in Generic Dashboard can be used on either of the screens
- Through the Dashboard Control and the Widget Manager, metadata is distributed to the appropriate widgets and across the 1st and 2nd Screen Dashboard
- The Generic Dashboard interacts with the Social Components, the Syndicator, the Identity and Security Services and an external video source

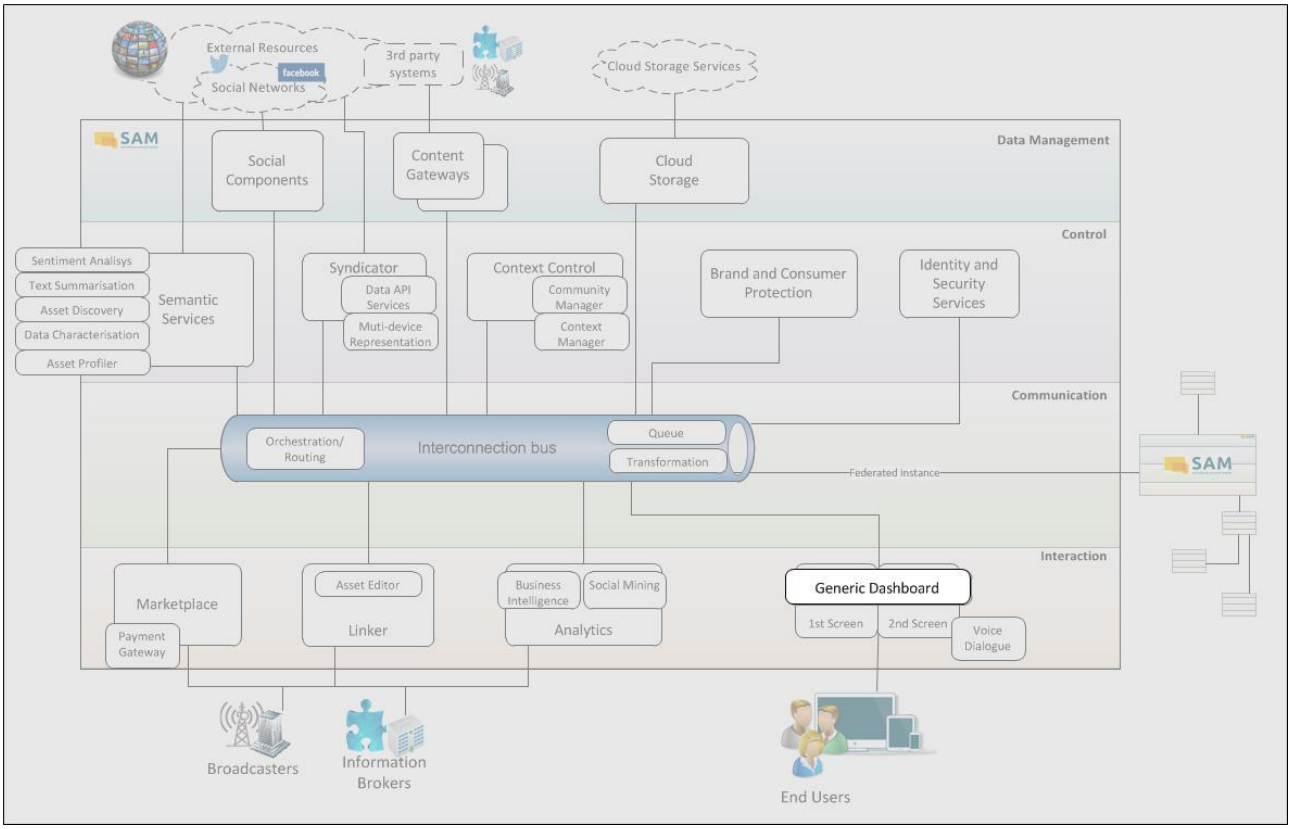

Figure 179: Architecture Overview – Generic Dashboard

3.14.1.1 Major Design Decisions

The Generic Dashboard has to provide a common user interface for the SAM End Users regardless of the device being used. In order to reuse generic elements across devices (i.e. across different platforms), it needs to be embedded and displayed in a device-specific Generic Dashboard Viewer. Different target platforms may include TVs, tablets and smartphones. The integration effort should be minimal.

The Generic Dashboard also needs to provide clear interfaces between widgets and Widget Manager for development of SAM and 3rd party widgets. Those interfaces should be easy to understand by the general developer community, stimulating them to create and deploy their own 3rd party widgets with the least effort needed.

It also uses interfaces from the Social Components, Syndicator, Identity and Security Services as well as the external video source.

The Graphical User Interface of the Generic Dashboard is the window to the user and determines how the End User perceives the SAM platform. This imposes several requirements. The Generic Dashboard should be highly configurable so that the user can adjust each and every widget (its size, presence on the screen, etc.). The design should adhere to the latest and widely spread user interface design principles, so that the user can instantly and intuitively use the interface.

3.14.1.2 Technology Comparison and Selection

In order to realise the Generic Dashboard component several options have to be considered as base technology. The assessment of these technologies is presented in the following sections.

3.14.1.2.1 Main Technology Selection

The graphical user interface of the Generic Dashboard will be built using one of the main technologies together with supporting frameworks reducing the development work. The main technology selection can be found in this section, whereas in the next Section 3.14.1.2.2 the choice of supporting technologies is described.

3.14.1.2.1.1 Possible Technologies and Comparison

The Generic Dashboard must be embedded in Dashboard Viewer of each of the End User devices. The Dashboard Viewer subcomponent is part of the 1st Screen and 2nd Screen components and a device-specific wrapper around the Generic Dashboard component, which in turn must operate device-independent across different platforms to minimize integration effort. The following runtimes can be selected to do so:

- **Adobe AIR**[103] is a cross-platform runtime for the development of rich applications. It utilises Adobe Flash, Apache Flex, HTML, JavaScript and XML. It supports a broad range of devices including desktop computers, netbooks, tablets, smartphones, and TVs, all of which require a pre-installed or included application runtime. Supported software platforms include Android, iOS, BlackBerry, Microsoft Windows, OS X. Flash applications are run within a contained Flash Player instance, whereas HTML/JavaScript/Ajax web applications are run within the included WebKit rendering engine. A single AIR instance can run multiple browsers at the same time. JavaScript content is however executed with certain security limitations. Adobe AIR has reasonably broad reach in the developer community as Flash/ActionScript is considered to be mature and popular. Adobe AIR may not be a good candidate for a long term strategy for mobile development, especially taking into account the rapid decline of Flash-related technologies and lack of its support on iOS devices.
- **HTML5**[104] is a mark-up language seen as core technology in presenting content over the Internet. The fifth major version of HTML adds new syntactic features (i.e.

[103] http://www.adobe.com/products/air
[104] http://www.w3.org/TR/html5/

simplifying handling of the multimedia and graphical content) as well as new APIs for complex web applications development. It is also seen as a cross-platform mobile application development language, due to the presence of web browsers on almost every End User device. The Responsive Web Design (RWD) approach aims at creating Graphical User Interfaces with optimal viewing experience across different screen formats. A web application written in HTML5 can be embedded in a platform-specific Dashboard Viewer by the means of WebView components or using several cross-platform tools such as Apache Cordova. Knowledge of HTML, JavaScript and CSS is widespread among developers. Usage of additional RWD JavaScript frameworks will greatly speed up the development and establish a well-structured basis for future development.

- **Qt**[105] is a cross-platform application and UI framework that targets a number of embedded, desktop and mobile platforms. It uses QML (seen as a CSS- and JavaScript-like language) together with an extensive set of C++ libraries (containing intuitive APIs for things like threading, networking, animations, etc.) and GUI components written in C++ to build applications. Qt is available as a free/open-source version from Qt Project and a commercial one offered by Digia. Qt's integrated development environment, consisting of Qt Creator and Qt Designer, provide a complete tool chain to build applications. However, using the proprietary QML proprietary language, specific to Qt's stack, leads to platform lock-in.

Parameter	Importance	Adobe AIR	HTML5	Qt
Generic Parameters				
Maturity & Stability	+	+++	++	+++
Regularly Updated	+	++	+++	+
Technical Up-to-Datedness / Appeal	++	++	+++	+
Open Source	+	NO	N/A	YES
Non-Infecting	-	NO	N/A	YES
Code Quality	++	++	++	++
Extensibility	++	+	++	+
Community	+	++	+++	+
Performance / Scalability	+	++	++	++
Reuse of existing developments	++	-	+++	+
EU project origin	-	-	-	-
Platform (Portability)	+++	++	+++	++
Open Standards Compliance	+	+	+	+
Interoperability (easy integration for all platforms)	++	+	+++	++
Specific Parameters				
Timeliness of related content	++	++	++	++
Configurability	+++	++	++	+
Low development threshold	+	+	+++	+
Network load	++	++	++	++
Reusability	+	++	+++	+

[105] http://www.qt.io/

Figure 180: Parameter Evaluation of Technologies for Generic Dashboard

3.14.1.2.1.2 Technology Selection

For developing the Generic Dashboard, HTML5 will be used, as it is a well-established and comprehensive language for user interfaces. Together with related technologies such as JavaScript and CSS, it makes for a complete interactive solution for building interactive End User interfaces. The <video> element will provide required handling of video source, whereas the RWD HTML5/JavaScript framework will provide unified widget building blocks. The comparison and selection of the RWD HTML5/JavaScript framework is described in the following chapter.

3.14.1.2.2 Supporting Technology Selection

In the previous section, HTML5 was selected as main technology for the Generic Dashboard. Adding one of many existing HTML5/JavaScript frameworks will ensure high extensibility and reusability with minimum development threshold. The selection of this framework is conducted in this section.

3.14.1.2.2.1 Possible Technologies and Comparison

The End User must be provided with a highly configurable graphical user interface that is responsive, providing optimal viewing experience and is designed according to well-established UX paradigms. This interface will be implemented as single page application, providing a more fluid experience with widgets. For future widget developers it is essential to specify a well-known and structured base to build their own widgets. This can be achieved with the help of the following frameworks:

- **Sencha Touch**[106] – is a mature framework that comes with several pre-defined layouts, an advanced JavaScript class system, a graphical user interface and command line interface tools to develop the necessary application faster. The capability for an Model-Store-View-Controller (MSVC) pattern, a large variety of readily available widgets and an integrated complete DOM manipulation interface are also the strong sides of this framework. Nonetheless layouts cannot be considered full RWD and they need adaptation when it comes to desktop and big screen (such as TV) environments.
- **jQuery Mobile**[107] **+ Backbone**[108] – jQuery Mobile brings responsive web design, whereas Backbone provides the JavaScript architecture with MV* patterns, supporting prototype-based inheritance and object extensions, communicates well with RESTful back ends, and uses jQuery DOM manipulation – performing relatively faster compared to other JS frameworks. jQuery mobile was intended for mobile usage first but it can also be used on desktops and big screens without any modifications needed.
- **AngularJS**[109] **+ Ionic**[110] – Ionic is a quite young framework, built on top of AngularJS from Google, provides application structure with a lot of focus on the user interface (that is almost Google-like in style and behaviour), offers separation of business logic with MV* patterns, and leverages RWD principles to optimise experience on mobile devices. It uses a subset of the included jQuery library to handle DOM manipulations.

[106] http://www.sencha.com/products/touch
[107] http://jquerymobile.com/
[108] http://backbonejs.org/
[109] https://angularjs.org/
[110] http://ionicframework.com/

However the desktop and big screen support is lacking. Neither Angular JS nor Ionic use a class system.

- **AngularJS + Bootstrap**[111] – AngularJS extends applications with MV* patterns providing developers with directives steering HTML tags allowing for fast web development. Together with Bootstrap providing mobile first RWD as well as support for desktop and big screen browsers they create good development starting point with lots of re-usable components.

Parameter	Importance	Sensa Touch	jQueryMobile Backbone	AngularJS Ionic	AngularJS Bootstrap
Generic Parameters					
Maturity & Stability	+	+++	++	+	++
Regularly Updated	+	+++	++	+	+++
Technical Up-to-Datedness / Appeal	++	++	+++	+++	+++
Open Source	+	NO	YES	YES	YES
Non-Infecting	-	NO	NO	NO	NO
Code Quality	++	++	++	+	+++
Extensibility	++	+	+++	++	++
Community	+	+	+++	++	+++
Performance / Scalability	+	+++	++	++	+++
Reuse of existing developments	++	-	+	-	-
EU project origin	-	-	-	-	-
Platform (Portability)	+++	++	+++	++	+++
Open Standards Compliance	+	+	+	+	+
Interoperability (easy integration for all platforms)	++	++	++	++	++
Specific Parameters					
Timeliness of related content	++	+++	+++	++	+++
Configurability	+++	++	+++	++	+++
Low development threshold	+	+	++	++	+++
Network load	++	++	++	++	++
Reusability	+	+	+++	++	+++

Figure 181: Parameter Evaluation of Technologies for Generic Dashboard

3.14.1.2.2.2 Technology Selection

As an HTML5 supporting framework, AngularJS + Bootstrap was chosen. Due to its popularity and high degree of testability, it will greatly help the development of highly configurable widgets with the required quality. With AngularJS DOM processing speed, the timeliness of related content can be easily achieved. Together with Bootstrap controls and RWD grids it allows developing out-of-the box GUI covering a wide range of displays.

[111] http://getbootstrap.com/

3.14.1.3 Technical Component Specification

3.14.1.3.1 Structure

Figure 182 provides an overview of the Generic Dashboard which can be found also in D3.2.1 Global Architecture Definition. Additionally, this version has been completed with the selected technologies. The Video Viewer contains a Video List object, which can contain videos, as specified in Section 3.14.1.4.2.1.1. The Widget Manager can contain widgets. Each widget has access to the configuration and device data, as specified in Section 3.14.1.4.1.2.

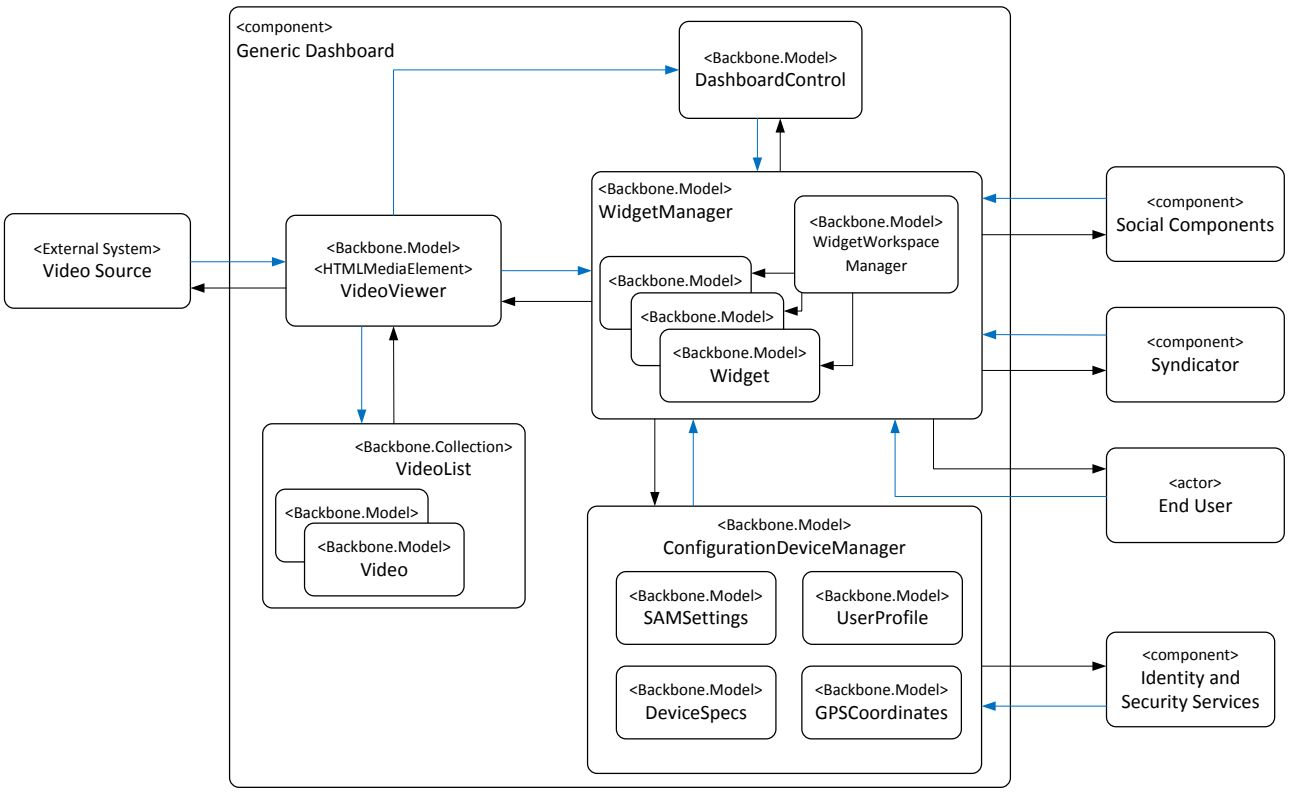

Figure 182: Overview with Selected Technologies – Generic Dashboard

3.14.1.4 Specification of Interfaces, Protocols and Formats

The Generic Dashboard interfaces will be provided as AngularJS modules, described in the following sections.

3.14.1.4.1 Dashboard Interfaces

This section provides the description of JavaScript interfaces used in components of Generic Dashboard, such as Video Viewer, Widget, Widget Manager, Dashboard Control, Configuration and Device Manager.

3.14.1.4.1.1 Video Viewer

The Video Viewer will provide each widget with the video list from current source and the possibility to select and control video stream. To interact with this list, the following methods will be provided:

- selectVideo(id)

- startVideo()
- pauseVideo()
- stopVideo()
- fastForward()
- jumpTo(videoTime)
- volumeUp()
- volumeDown()
- mute()

Select Video	
Description	Using this function, a specified video stream can be selected on the source.
Method Header	selectVideo(id)
Parameters	id – video ID to be selected. The ID must be matching one of video IDs from the list obtained using getVideoList()
Return Value	true on success
Error Handling	In case of an error, false is returned
Remarks	None

Figure 183: Select Video Method Signature

Start Video	
Description	Using this function, the currently selected video can be played.
Method Header	startVideo()
Parameters	none
Return Value	true on success
Error Handling	In case of an error, false is returned
Remarks	None

Figure 184: Start Video Method Signature

Pause Video	
Description	Using this function, the currently selected video can be paused (or resumed if already paused).
Method Header	pauseVideo()
Parameters	none
Return Value	true on success

Error Handling	In case of an error, false is returned
Remarks	The video can be resumed by using either play() or pause() functions

Figure 185: Pause Video Method Signature

Stop Video	
Description	Using this function, the currently selected video is stopped.
Method Header	stopVideo()
Parameters	none
Return Value	true on success
Error Handling	In case of an error, false is returned
Remarks	Time position of the video is reset to the beginning of the stream.

Figure 186: Stop Video Method Signature

Fast Forward	
Description	Using this function, the currently selected video can be fast forwarded.
Method Header	fastForward()
Parameters	none
Return Value	true on success
Error Handling	In case of an error, false is returned
Remarks	Video is forwarded by increasing its playback rate by 4

Figure 187: Fast Forward Method Signature

Jump To	
Description	Using this function, the currently selected video can be skipped to the specified position.
Method Header	jumpTo(videoTime)
Parameters	videoTime – position in seconds since the beginning of the stream
Return Value	true on success
Error Handling	In case of an error, false is returned
Remarks	None

Figure 188: Jump To Method Signature

Volume Up	
Description	Using this function, the currently played volume can be increased.
Method Header	volumeUp()
Parameters	None
Return Value	true on success
Error Handling	In case of an error, false is returned
Remarks	None

Figure 189: Volume Up Method Signature

Volume Down	
Description	Using this function, the currently played volume can be decreased.
Method Header	volumeDown()
Parameters	None
Return Value	true on success
Error Handling	In case of an error, false is returned
Remarks	None

Figure 190: Volume Down Method Signature

Mute	
Description	Using this function, the currently played volume can be muted.
Method Header	mute()
Parameters	None
Return Value	true on success
Error Handling	In case of an error, false is returned
Remarks	None

Figure 191: Mute Method Signature

3.14.1.4.1.2 Widget

A widget is a basic entity, written in HTML5/JavaScript, displayed on the Generic Dashboard that can interact with the End User. It can be either displayed on the 1st or the 2nd Screen. It communicates with other widgets via messages, and also subscribes to receive additional content notifications about currently played video. For each widget

which needs to interact with Social Components, an URI is set allowing to do so. A widget must implement the following interface:

- setSocialComponentsURI(uri)
- updateNotification(contentType, content)
- receiveMessage(messageType, message)

Set Social Components URI	
Description	Using this function, the Widget Manager sets social component URI if the widget has subscribed for social components notification.
Method Header	setSocialComponentsURI(uri)
Parameters	uri – universal resource identifier of specific social component to be used by widget.
Return Value	true on success
Error Handling	In case of an error, a false-value is returned.
Remarks	This function is only called for the widgets that had previously subscribed for content notifications with contentType = social.

Figure 192: Set Social Components URI Method Signature

Update Notification	
Description	Using this function, Widget Manager will send the content updates from different external components (such as Syndicator or Social Components) to the widget to be processed and displayed.
Method Header	updateNotification(contentType, content)
Parameters	contentType – type of delivered content, as specified in subscribeForContent function content – delivered content by external component, formatted as specified by the external component interface
Return Value	true on success
Error Handling	In case of an error, a false-value is returned
Remarks	None

Figure 193: Update Notification Method Signature

Receive Message	
Description	Using this function, Widget Manager will provide to the widget received message to be processed if the widget subscribed for this message type events.
Method Header	receiveMessage(messageType, message)
Parameters	messageType – type of the message that was delivered, specified by

	source widget message – contents of received message, as specified by source widget
Return Value	true on success
Error Handling	In case of an error, a false-value is returned
Remarks	This function is only called for the widgets that had previously subscribed for receiving messages of this type

Figure 194: Receive Message Method Signature

3.14.1.4.1.3 Widget Manager

The Widget Manager is responsible for communication between widgets and other SAM components (such as Social Components, Syndicator) using the Interconnection Bus component. It sends the time updates to those components, receiving in response any relevant additional content. The content is unpacked and pre-processed here and is then sent to the proper widgets. It is a widget's responsibility to subscribe to the Widget Manager to receive content notifications. Only content updates with the type specified during subscription will be delivered to the widget. Widgets can subscribe for multiple of content types by repeating the subscription process. An overview of the interfaces that the Widget Manager provides is presented below.

```
(function() {
var app = angular.module('widgetManager', ['widget'])
.controller('widgetManagerCtrl', function() {
        this.subscribeForContent = function(Widget, contentType, frequency) {
                ...
        };
});
});
```

Figure 195: Overview of the Widget Manager Interface

Subscribe for Content	
Description	Using this function, Widgets can subscribe to receive content updates of a certain type with specified frequency.
Method Header	subscribeForContent(Widget, contentType, frequency)
Parameters	Widget – the Widget Model object requesting the content updates contentType – one of the following available content types: "social", "syndicator" frequency – specifies interval in seconds between each update, 0 meaning immediate updates
Return Value	true on success
Error Handling	In case of an error, a false-value is returned

Remarks	None

Figure 196: Subscribe for Content Method Signature

3.14.1.4.1.4 Dashboard Control

The Dashboard Control is responsible for Inter-Widget Communication as well as for the communication between multiple screens. It is a Widget's responsibility to subscribe to the Dashboard Control to receive message notifications. Only messages with the type specified during subscription will be delivered to the Widget. Widgets can subscribe to multiple message types by repeating the subscription process. An overview of the interfaces that the Dashboard control provides is presented below.

```
(function() {
        var app = angular.module('dashboardControl', ['widget'])
        .controller('dashboardControlCtrl', function() {
                this.subscribeForMessage = function(Widget, messageType) {
                        ...
                };
                this.sendMessage = function(messageType, message) {
                        ...
                };
        });
});
```

Figure 197: Overview of the Dashboard Control Interface

Subscribe to Message	
Description	Using this function, widgets can subscribe to receive messages of certain type.
Method Header	subscribeForMessage(Widget, messageType)
Parameters	Widget – the Widget Model object requesting the message updates messageType – type of the message, defined by source widget
Return Value	true on success
Error Handling	In case of an error, a false-value is returned
Remarks	None

Figure 198: Subscribe for Message Method Signature

Send Message	
Description	Using this function, widgets can send messages.
Method Header	sendMessage(messageType, message)
Parameters	messageType – type of the message message – content of the message

Return Value	true on success
Error Handling	In case of an error, a false-value is returned
Remarks	A Widget that is executing that function defines the type as well as the message content. All the widgets that are subscribed to that message type across multiple screens will receive the message. It is up to the widgets to agree on message format.

Figure 199: Send Message Method Signature

3.14.1.4.1.5 Configuration and Device Manager

The Configuration and Device Manager is responsible for delivering information about the SAM platform, End User, device used and other user-specific data (e.g. geolocation). It is also the communication point towards the Identity and Security Services providing means for user authentication. Widgets can utilise this info to present the End User with personalised content well-suited to the current display properties, location and preferences. An overview of the interfaces that the Configuration and Device Manager provides is presented below.

```
(function() {
	var app = angular.module('configurationDeviceManager', [])
	.controller('configurationDeviceManagerCtrl', function() {
		this.samSettings = {
			eCommerce	: false,
			advertisement	: false,
			statistics	: false,
			geolocation	: false
		};
		this.userProfile = {
			id			: 0,
			name		: "Some Name",
			login		: "someLogin",
			password	: "somePassword",
			email		: "some@email.com"
		};
		this.deviceSpecs = {
			displayHeight	: "640",
			displayWidth	: "480",
			orientation	: "portrait",
			GPScapable	: false,
			firstScreen	: false
		};
		this.GPSCoordinates = {
			latitude	: "37.422005",
			longitude	: "-122.084095",
			accuracy	: "3.0f",
			altitude	: "",
		};
		this.authenticateUser = function(login, password) {
```

```
                ...
            };
            this.isFirstScreen =  function() {
                ...
            };
        });
});
```

Figure 200: Overview of the Configuration and Device Manager Interface

Authenticate User	
Description	Using this function, the Dashboard Configuration widget can authenticate a SAM user using the provided login and password.
Method Header	authenticateUser(login, password)
Parameters	login – user provided login password – user provided passwords
Return Value	true on success
Error Handling	In case of an error, a false-value is returned
Remarks	None

Figure 201: Authenticate User Method Signature

Is First Screen	
Description	Using this function, widgets can stay informed about the role of the current screen.
Method Header	isFirstScreen()
Parameters	None
Return Value	true if current device has 1st Screen
Error Handling	In case of an error or if device has 2nd Screen, a false-value is returned
Remarks	None

Figure 202: Is First Screen Method Signature

3.14.1.4.2 Content Formats

3.14.1.4.2.1 JavaScript Formats

This section lists the JSON schemas which will be used by the Generic Dashboard for the communication with other components.

3.14.1.4.2.1.1 Video List

Each widget will have the possibility to obtain additional information from the Video Viewer present on the 1st Screen. This information also contains the video list from the currently connected video source. For each video in the video list the following details will be provided:

- **id** – unique identifier of the video stream in the currently selected source
- **title** – title of the video stream
- **duration** – length of the video stream, in seconds

```
video = {id:0, title: "some title", duration: 8563};
videoList = [video1, video2, ...];
```

Figure 203: JavaScript Model Representation of the Video List

3.14.1.4.2.1.2 Configuration and Device Manager

Each widget will have the possibility to obtain (and modify) the configuration and device data, including End User profile data, geolocation information, SAM settings for the current End User, and Device specification. This information is structured as follows:

- **SAMSettings** – this structure holds SAM-specific settings that user can modify via User Interface
  - **eCommerce** – whether the e-commerce is enabled
  - **advertisement** – whether the advertisements are enabled
  - **statistics** – whether the usage statistics are enabled
  - **geolocation** – whether the geolocation is permitted
- **UserProfile** – structure providing information about the End User. This structure can also be used to register a new End User
  - **id** – unique identifier of the user
  - **name** – user display name
  - **login** – user login
  - **password** – user password
  - **email** – user email address
- **DeviceSpecs** – collection of attributes holding info about the device on which current dashboard is displayed. It is set by Dashboard viewer and accessible to Generic Dashboard and widgets contained within in order to properly render the user interface
  - **displayHeight** – provides the display height in pixels
  - **displayWidth** – provides the display width in pixels
  - **orientation** – information about device orientation (portrait / landscape)
  - **GPScapable** – whether the device is equipped with GPS receiver
  - **firstScreen** – whether the current device is 1st Screen device
- **GPSCoordinates** – collection of GPS location information fields, set by Dashboard viewer as the result of GPS coordinates request
  - **latitude** – latitude value
  - **longitude** – longitude value
  - **accuracy** – location accuracy
  - **altitude** – altitude value, if present

```
samSettings = {
        eCommerce       : false,
        advertisement   : false,
        statistics      : false,
```

```
        geolocation     : false
};
userProfile = {
        id              : 0,
        name            : "Some Name",
        login           : "someLogin",
        password        : "somePassword",
        email           : "some@email.com"
};
deviceSpecs = {
        displayHeight   : "640",
        displayWidth    : "480",
        orientation     : "portrait",
        GPScapable      : false,
        firstScreen     : false
};
GPSCoordinates = {
        latitude        : "37.422005",
        longitude       : "-122.084095",
        accuracy        : "3.0f",
        altitude        : "",
};
```

Figure 204: JavaScript Model Representation of the Configuration and Device Manager Data

3.14.1.5 Summary

In this section, the Generic Dashboard was technically specified. During the technology selection (Section 3.14.1.2), HTML5 together with AngularJS + Bootstrap were selected as a framework to develop a Generic Dashboard that can later be integrated in 1st and 2nd Screen. A Generic Widget was specified with interfaces that it must provide within the SAM platform. Through the Configuration and Device Manager interface it is possible to query and update the SAM configuration and device specification parameters. The Widget Manager interface provides widgets with the possibility of content updates, whereas the Dashboard Control provides the message-passing interface. Together with the Video Viewer interface which provides video specific commands (e.g. selecting, playing), this allows for the full required SAM platform functionality.

According to this specification, the following components have to be implemented:

- AngularJS JavaScript Model containing all logic of Generic Dashboard subcomponents
- HTML5 frontend using Bootstrap to present visually attractive Graphical User Interface to the SAM End User

The Technical Specification of the Generic Dashboard is focused on interoperability; as such the chosen HTML5 technology allows for an easy cross-platform integration into multiple device-specific Dashboard Viewer components, present on both 1st Screen and 2nd Screen devices.

3.14.2 1st Screen

The Generic Dashboard component is displayed in the Dashboard Viewer on the 1st Screen device. It is extended with the 1st Screen specific components which need to be added to achieve the required functionality.1st Screen provides End Users with the Graphical User Interface of the SAM Platform (see Figure 205).

- The 1st Screen consists of a component to represent the main video. This component is made visible through the Dashboard Viewer, which will show the video by using the Video Viewer subcomponent embedded in the Generic Dashboard. On the 1st Screen, the Video Viewer will be enabled.
- It will be able to discover the 2nd Screen device and trigger a launch component to launch the Dashboard Viewer, which is present on the 2nd Screen device.
- The 1st Screen also includes ways to communicate with the 2nd Screen device, through the Inter-Device-Communication component. Optionally, the 1st Screen includes ways to synchronise content by means of external systems, such as Automatic Content Recognition (ACR).

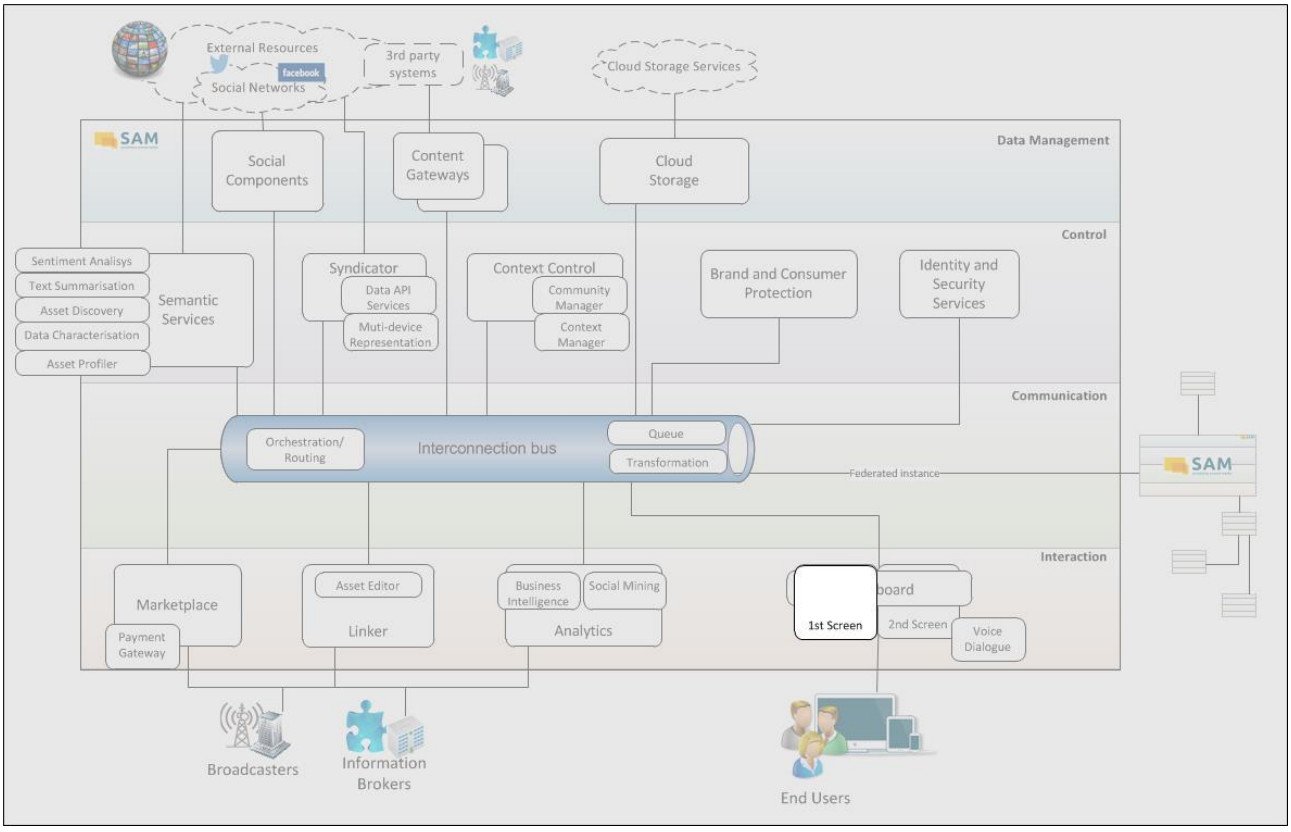

Figure 205: Architecture Overview – 1st Screen

3.14.2.1 Major Design Decisions

The 1st Screen is responsible for displaying the Generic Dashboard component, written in HTML5 (see Section 3.14.1), and its content on a 1st Screen device. This imposes that the Dashboard Viewer subcomponent has to be able to properly display HTML5 elements.

The 1st Screen device will contain the main video widget as well as some of the widgets providing related information, accompanying the video. Widgets have to be able to access device-specific information (i.e. video list).

It is assumed that the 1st Screen can be shared amongst the users (i.e. there is only one 1st Screen) which implies bigger display sizes - preferably TVs. The Dashboard Viewer subcomponent has to appropriately handle displaying the Generic Dashboard component on such devices. Also, the 1st Screen Interaction subcomponent will need to properly interface between End User, using remote or some other pointing device, and the Generic Dashboard component.

Currently on the market there are many TV applications developer frameworks, each bound to a single manufacturer. Each one provides its own API for application developers. 1st Screen application should be usable on a wide range of devices, so it is essential to select a framework that has high device coverage.

3.14.2.2 Technology Comparison and Selection

This subsection outlines technology selection criteria and compares existing technologies – potential candidates for the realisation of a 1st Screen. Within the subsections the areas that need to be implemented or improved to fully meet SAM specific requirements are also identified and presented. The selected technologies will be used as a basis for the development phase in the realisation of the technical design of this component: The specific selection criteria to be used are specified in the document D3.2.2 Functional Specification and will be used in the next sections.

3.14.2.2.1 Possible Technologies and Comparison

In order to provide all of the required features, the 1st Screen has to have access to the native APIs of the platform it is installed on. This will influence how the display is built up (i.e. resizing according to the display size), how it interacts with user interface (i.e. gesture control), the possibility to launch and communicate with 2nd Screens, and also how to obtain a video stream. This can be achieved by using one of the approaches mentioned in the following sections.

3.14.2.2.1.1 1st Screen Application

Native platform application:

Developments in native language for a specific platform achieves better integration by the provided device API. The application's performance is considered to improve, using a native well-known device UI. Native platform applications can use all the device features, device notification system and they can also work offline. They provide also WebView elements for embedding HTML subcomponents. A couple of the most popular examples are:

- **Java for Android**[112] – developed in Java, targets devices running Google Android, such as smartphones, phablets, tablets, laptops and nowadays also TVs, from different manufacturers.
- **Objective-C for iOS**[113] – developed in Objective-C, targets Apple iOS devices, such as iPhones, iPods and iPads, with biggest currently available screen size is 9.7inch. A modified iOS version is also currently running on an Apple TV digital media player that, combined with a TV set, can play HD content.

Hybrid application:

[112] http://developer.android.com/index.html
[113] https://developer.apple.com/

Applications developed as hybrid are mainly HTML5/JavaScript based, with a thin native client for each target platform, making full usage of WebView elements to run and interact with the End User. They are generally slower and the client wrapper does not always provide access to all native platform features. A couple of most popular examples are:

- **Apache Cordova**[114] – where applications written using HTML/JavaScript/CSS run inside a WebView of a native thin client on the device itself. With multiple API plugins available, it is easy to access the device's native API. However it requires attention to performance from developers.
- **Appcelerator Titanium**[115] – is an open-source framework for creating mobile apps on a variety of platforms using a single JavaScript codebase. All the source code is deployed to the device where it is interpreted using a JavaScript engine (e.g. Mozilla Rhino, Apple JavaScriptCore). High development speed and a native UI are the positive sides of using this framework, whereas issues are found in behaviour of the cross-platform API, stability and memory management.

Mobile web application:

Mobile web applications consist of multiple websites written in HTML5 that are run in a web browser environment. They emulate a device's look and feel by using 3rd party libraries. They have very limited access to the native API of a device (i.e. GPS, camera, etc.), which they try to overcome (e.g. offline storage is mimicked through browser caching). Additionally, the majority of features are inaccessible (such as multi-touch, notifications, accelerometer, etc.). The performance is comparable with hybrid applications and is thus worse than native apps.

Parameter	Importance	Java for Android	ObjectiveC for iOS	Apache Cordova	Appcelerator Titanium	Mobile Web App
Generic Parameters						
Maturity & Stability	+++	++	+++	+	+	++
Regularly Updated	+/-	+++	+++	++	+	++
Technical Up-to-Datedness / Appeal	++	+++	+++	++	++	++
Open Source	+/-	N/A	N/A	YES	YES	N/A
Non-Infecting	+	N/A	N/A	NO	NO	N/A
Code Quality	+++	+++	+++	++	++	++
Extensibility	++	+++	+++	++	++	+
Community	-	+++	+++	+	+	++
Performance / Scalability	++	+++	+++	++	++	++
Reuse of existing developments	++	++	-	+	-	++
EU project origin	+/-	-	-	-	-	-
Platform (Portability)	+	+	-	++	++	+++
Open Standards Compliance	-					
Interoperability (easy integration for all platforms)	++	+	-	++	++	+++
Specific Parameters						
Easy navigation	++	+++	+++	+++	+++	+++
Network load / Direct Communication	++	+++	+++	++	++	++

[114] http://cordova.apache.org/
[115] http://www.appcelerator.com/

Responsiveness	+	+++	+++	++	++	++
Connection success rate	+++	+++	+++	++	++	++
Connection setup time	+	+++	+++	++	++	++

Figure 206: Parameter Evaluation of Technologies for 1st Screen

3.14.2.2.1.2 Inter-Device-Communication

Discovery technologies:

- **Universal Plug and Play (UPnP)**[116] – UPnP is a set of networking protocols that permits networked devices, such as personal computers, printers, Internet gateways, Wi-Fi access points and mobile devices to seamlessly discover each other's presence on the network and establish functional network services for data sharing, communications, and entertainment. The UPnP technology is promoted by the UPnP Forum.
- **Discovery And Launch (DIAL)**[117] – DIAL is a mechanism for discovering and launching applications. The protocol works without requiring a pairing between devices. DIAL allows communication with or between 2nd Screen devices, such as tablets, mobile phones and 1st Screen devices, such as Smart TVs.
- **Simple Service Discovery Protocol (SSDP)**[118] – SSDP is a network protocol for advertisement and discovery of network services and presence information. It accomplishes this without assistance of server-based configuration mechanisms, such as DHCP or DNS.
- **Multicast DNS (mDNS)**[119]**/DNS Service Discovery (DNS-SD)**[120] – MDNS in combination with DNS-SD allows clients to discover a named list of services by type in a specified domain using standard DNS queries. MDNS/DNS-SD is a zero configuration service, utilizing essentially the same programming interfaces, packet formats and operating semantics as the unicast DNS. While it is designed to be stand-alone capable, it can work in concert with unicast DNS servers.

Parameter	Importance	UPnP	DIAL	SSDP	uDNS/DNS-SD
Generic Parameters					
Maturity & Stability	+++	+++	++	+++	++
Regularly Updated	+/-	-	++	++	++
Technical Up-to-Datedness / Appeal	++	+/-	+++	++	+++
Open Source	+/-	+	++	++	++
Non-Infecting	+	++	++	++	++
Code Quality	+++	+	++	++	++
Extensibility	++	---	++	++	++
Community	-	--	++	+	+
Performance / Scalability	++	--	++	++	+/-
Reuse of existing developments	++	-	++	++	+
EU project origin	+/-	-	-	-	-
Platform	+	-	++	+	+

[116] http://en.wikipedia.org/wiki/Universal_Plug_and_Play
[117] http://en.wikipedia.org/wiki/DIscovery_And_Launch
[118] http://en.wikipedia.org/wiki/Simple_Service_Discovery_Protocol
[119] http://en.wikipedia.org/wiki/Multicast_DNS
[120] http://en.wikipedia.org/wiki/DNS-SD#DNS-based_service_discovery

(Portability)					
Open Standards Compliance	-	++	++	++	++
Interoperability (easy integration for all platforms)	++	+	+/-	+/-	+/-
Specific Parameters					
Easy navigation	++	n/a	n/a	n/a	n/a
Network load / Direct Communication	++	-	+	++	++
Responsiveness	+	n/a	n/a	n/a	n/a
Connection success rate	+++	++	++	++	++
Connection setup time	+	+	++	++	++

Figure 207: Parameter Evaluation of Technologies for Inter-Device-Communication

Communication technologies:

- **Simple Object Access Protocol (SOAP)**[121] – SOAP is a protocol for exchanging structured information in computer networks, mainly in the implementation of web services. It relies on XML for its message format, and usually relies on other application layer protocols, such as HTTP for message negotiation and transmission.
- **XML-RPC**[122] – XML-RPC is a remote procedure call protocol which uses XML to encode its calls and HTTP as a transport mechanism. XML-RPC is simpler to use and understand than SOAP, because it allows only one method of method serialisation, has a simpler security model and does not require the creation of WSDL service descriptions.
- **JSON-RPC**[123] – JSON-RPC is a remote procedure call protocol encoded in JSON. It is a very simple protocol and similar to XML-RPC. JSON-RPC allows for notifications (data sent to the server that does not require a response) and for multiple calls to be sent to the server which may be answered out of order.
- **REpresentational State Transfer (REST)**[124] – REST is an architectural style usually applied to the development of web services. One can characterise web services as "RESTful" if they conform to a set of constraints applied to universal resource indicators (URI's) and operations. REST is simpler to use than SOAP, because it does not require writing or using a provided server program (to serve data) and a client program (to request data).
- **WebSocket**[125] – WebSocket is a protocol providing full-duplex communications channels over a single TCP connection. WebSocket is designed to be implemented by web browsers and web servers, but it can be used by any client or server application. More interaction between a browser and a web site is made possible by providing a standardised way for the server to send content to the browser without being solicited by the client, and allowing for messages to be passed back and forth while keeping the connection open.

[121] http://en.wikipedia.org/wiki/SOAP
[122] http://en.wikipedia.org/wiki/XML-RPC
[123] http://en.wikipedia.org/wiki/JSON-RPC
[124] http://en.wikipedia.org/wiki/Representational_state_transfer
[125] http://en.wikipedia.org/wiki/WebSocket

Parameter	Importance	SOAP	XML-RPC	JSON-RPC	REST	WebSocket
Generic Parameters						
Maturity & Stability	+++	++	++	++	++	++
Regularly Updated	+/-	+	-	+	++	+++
Technical Up-to-Datedness / Appeal	++	--	-	+/-	++	++
Open Source	+/-	-	+/-	+	+++	++
Non-Infecting	+	++	++	++	++	++
Code Quality	+++	+	++	++	+++	++
Extensibility	++	-	+	++	+++	+++
Community	-	--	-	+/-	++	++
Performance / Scalability	++	-	+/-	++	-	+++
Reuse of existing developments	++	--	--	--	+	+
EU project origin	+/-	-	-	-	-	-
Platform (Portability)	+	-	-	-	++	++
Open Standards Compliance	-	++	++	++	++	++
Interoperability (easy integration for all platforms)	++	--	-	-	++	++
Specific Parameters						
Easy navigation	++	n/a	n/a	n/a	n/a	n/a
Network load / Direct Communication	++	-	+/-	+	-	++
Responsiveness	+	n/a	n/a	n/a	n/a	n/a
Connection success rate	+++	++	++	++	++	++
Connection setup time	+	+	+	+	+	+

Figure 208: Parameter Evaluation of Technologies for Inter-Device Communication

Integrated Solutions:

- **JointSpace**[126] – JointSpace is the open API used on Philips TV's to discover and interact with the TV from a 2nd Screen. JointSpace relies on mDNS/DNS-SD for discovery and REST for communication.
- **HbbTV**[127] – The upcoming HbbTV v2.0 specification includes features for communication with 2nd Screen applications. HbbTV is supported by all major TV brands. DIAL is the technology selected for discovery and WebSocket for communication.

Parameter	Importance	Custom Selection	JointSpace	HbbTV
Generic Parameters				
Maturity & Stability	+++	+/-	+	+++
Regularly Updated	+/-	+	-	++
Technical Up-to-Datedness / Appeal	++	++	+	+++
Open Source	+/-	++	--	++
Non-Infecting	+	+	+	+
Code Quality	+++	+/-	+/-	+++
Extensibility	++	++	+	+

[126] http://jointspace.sourceforge.net/
[127] http://www.hbbtv.org/

Community	-	+/-	--	+
Performance / Scalability	++	++	+	++
Reuse of existing developments	++	+	+	++
EU project origin	+/-	-	-	-
Platform (Portability)	+	++	+/-	++
Open Standards Compliance	-	++	-	+++
Interoperability (easy integration for all platforms)	++	+	+	+++
Specific Parameters				
Easy navigation	++	n/a	n/a	n/a
Network load / Direct Communication	++	++	+/-	++
Responsiveness	+	n/a	n/a	n/a
Connection success rate	+++	++	+	++
Connection setup time	+	+	+	+

Figure 209: Parameter Evaluation of Technologies for Inter-Device Communication

3.14.2.2.2 Technology Selection

3.14.2.2.2.1 1st Screen

As the 1st Screen will be targeting a big display (such as a TV) shared between multiple of End Users, it will be implemented using Java for Android SDK. The operating system developed by Google is seen to be expanding also towards that area, thus creating an application using these tools will allow the application to be used on a multitude of devices, not bound to a specific manufacturer. It also allows for better integration into the device and increased performance compared to other solutions.

3.14.2.2.2.2 Inter-Device-Communication

HbbTV was selected for Inter-Device-Communication and discovery, because it combines a number of state-of-the-art technologies which individually score very well in the comparison with their counterparts.

3.14.2.3 Technical Component Specification

3.14.2.3.1 Structure

Below, Figure 210 provides an overview of the 1st Screen, which can be found also in D3.2.1 Global Architecture Definition. Additionally, this version of the architectural overview has been added with the selected technologies.

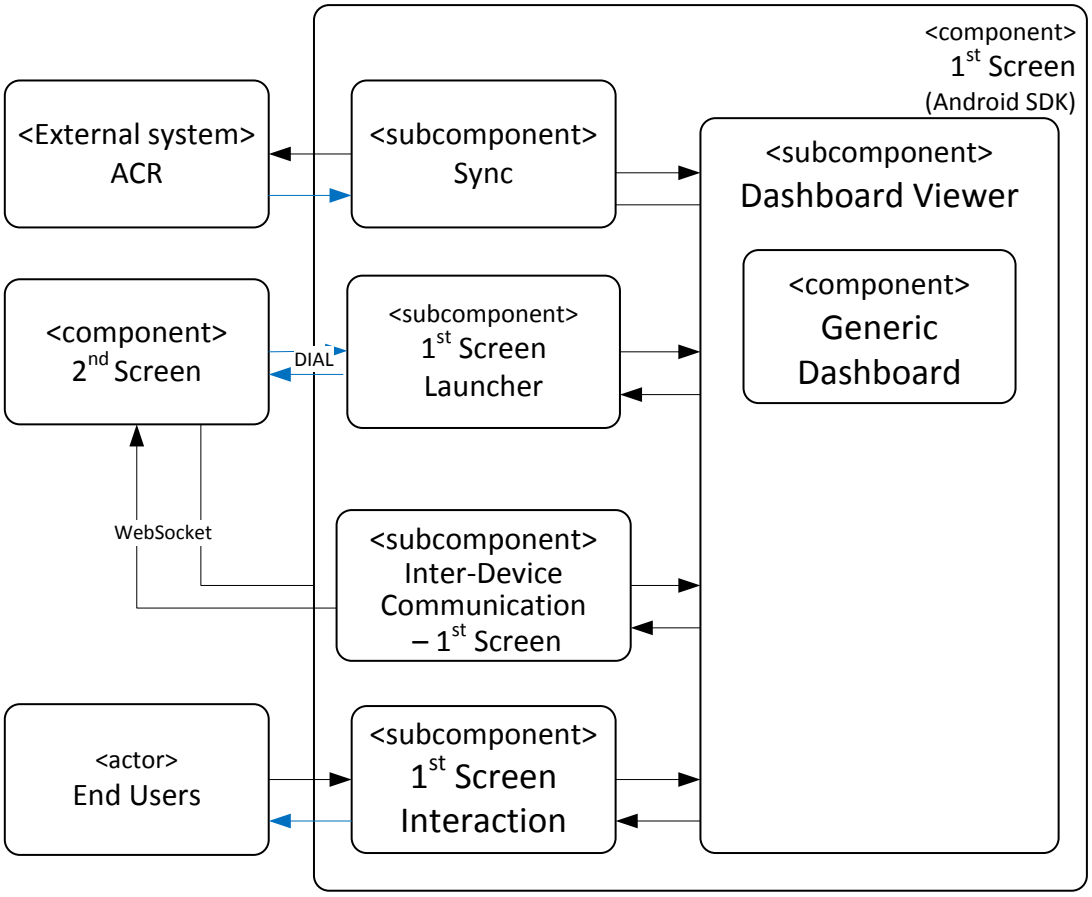

Figure 210: Overview with Selected Technologies – 1st Screen

3.14.2.3.2 Internal Sequence Diagrams

The main internal sequence diagrams for 1st Screen component are described in document D3.2.2 Functional Specification in Section 4.14.2. In this section additional interaction diagrams are for the Inter-Device-Communication subcomponent are described. The HbbTV specification for interaction with 2nd Screens provides solutions for 3 main use cases:

- Discovery of the 1st Screen from a 2nd Screen application and launching of an application on the 1st Screen
- Discovery of the 2nd Screen from a 1st Screen application and launching of an application on the 2nd Screen
- Communication between 1st and 2nd Screen applications

The diagrams below give an overview of the interactions for these use cases.

3.14.2.3.2.1 Discovery and Launch of the 1st Screen from the 2nd Screen

Figure 211 illustrates the sequence to discover and launch a 1st Screen application from the 2nd Screen.

1. The 2nd Screen application discovers the 1st Screen device and its launcher service via DIAL
2. The 2nd Screen applications sends a HTTP POST request to the launcher service of the 1st Screen

3. The 1st Screen launcher service launches the application and returns the result to the 2nd Screen application in the HTTP response

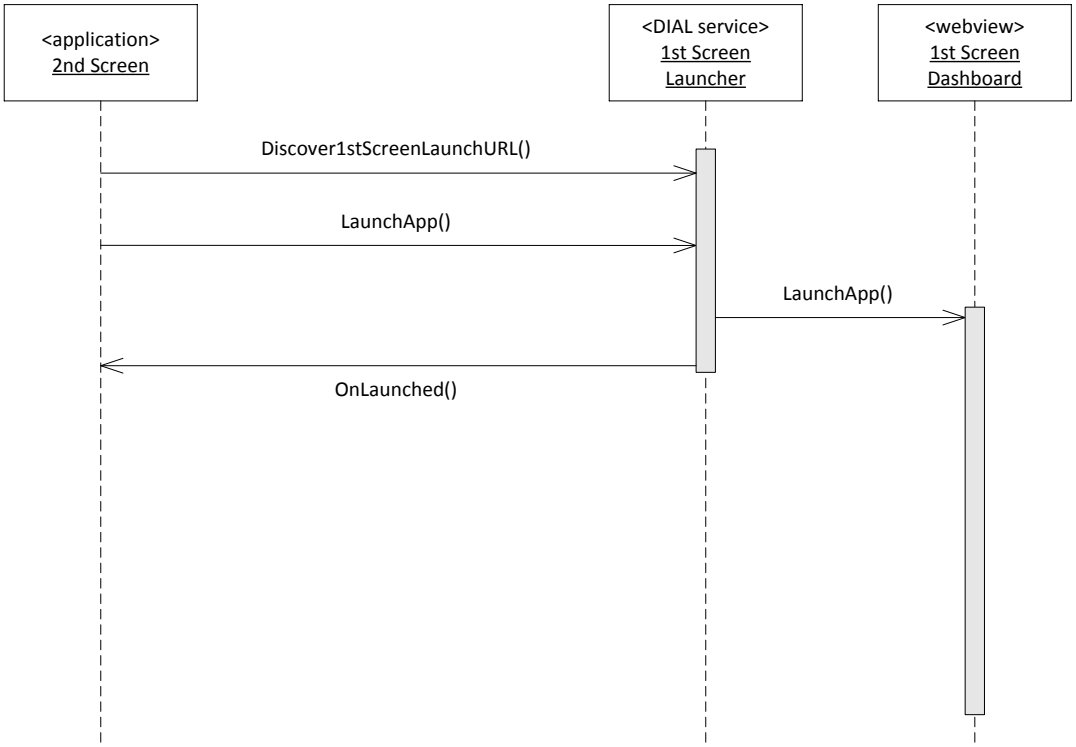

Figure 211: Discovering the 1st Screen and Launching an Application

3.14.2.3.2.2 Discovery and Launch of the 2nd Screen from the 1st Screen

The 1st Screen application uses the HbbTVCSManager object (see Section 3.14.2.4.1) to discover and launch 2nd Screen and launch 2nd Screen applications.

Figure 212 illustrates the sequence to discover and launch a 2nd Screen application from the 1st Screen.

1. The 1st Screen application requests a list of 2nd Screen devices with device information
2. The 1st Screen application constructs a JSON object containing a launch request and sends it to the 2nd Screen application
3. The launcher application on the 2nd Screen device interprets the launch request, attempts to launch the application and returns the result

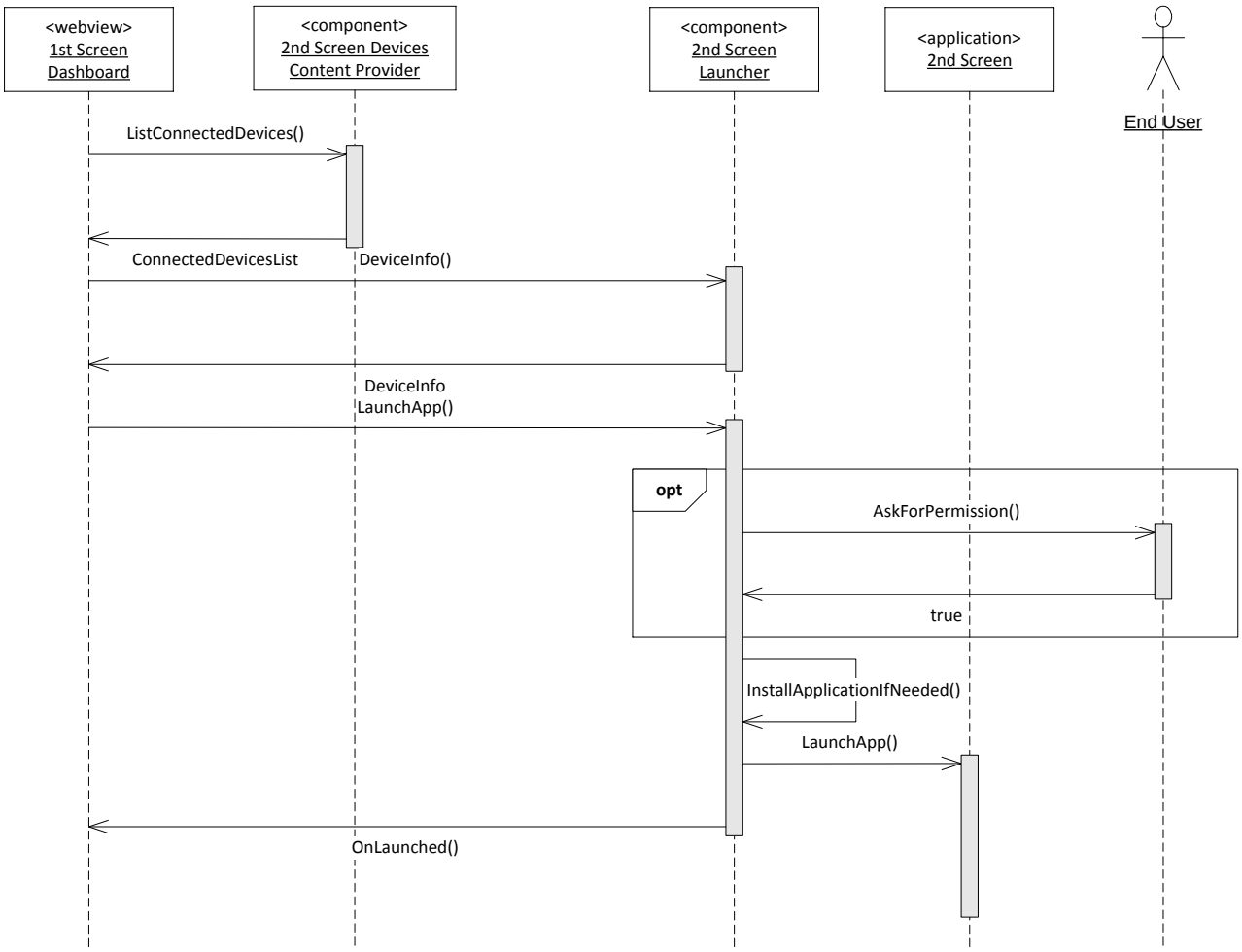

Figure 212: Discovering the 2nd Screen and Launching an Application

3.14.2.3.2.3 Communication between 1st Screen and 2nd Screen Applications

For Inter-Device-Communication, a WebSocket server implemented on the 1st Screen device is used.

Figure 213 illustrates the sequence to set up a WebSocket communication channel between an application on the 1st Screen and an application on the 2nd Screen.

1. The 1st Screen application retrieves the WebSocket server URL from the HbbTVCSManager object
2. The 2nd Screen application retrieves the WebSocket server URL from the DIAL service
3. Both 1st Screen and 2nd Screen applications connect as clients to the WebSocket server specifying a common application name
4. The WebSocket server matches and pairs client connections based on the application name and relays communication data between the paired connections

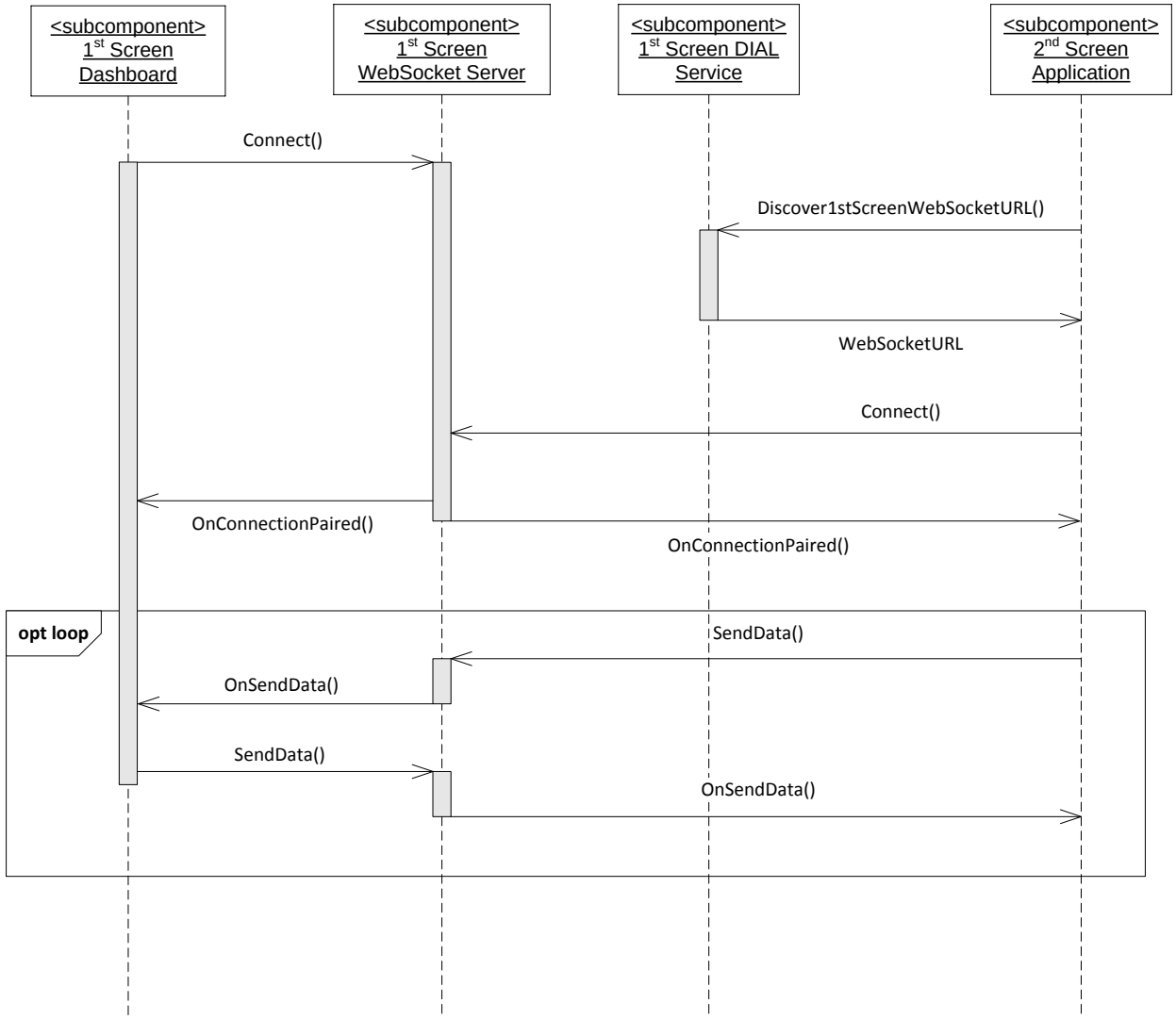

Figure 213: Inter-Device-Communication Using a WebSocket Server on the 1st Screen

3.14.2.4 Specification of Interfaces, Protocols and Formats

Three interfaces are presented in this section:

- Through a JavaScript interface, an embedded object HbbTVCSManager is available to 1st Screen applications
- A DIAL interface facilitates the discovery and launching of 1st Screen applications
- A WebSocket server interfaces is provided by the 1st Screen device allowing 1st Screen applications to communicate with 2nd Screen applications on the home network.

3.14.2.4.1 JavaScript Interface

An embedded object HbbTVCSManager is available to 1st Screen applications. This object is available on devices that implement the future HbbTV 2.0 standard. "CS" is an acronym for Companion Screen, the term used by HbbTV to refer to 2nd Screen devices. It enables applications to:

- Discover 2nd Screens with a running launcher application
- Launch or install an application on a 2nd Screen device

- Discover the base URLs of the local and remote end points for application to application communication

3.14.2.4.1.1 Method "Discover 2nd Screens"

Before a 2nd Screen application can be launched / installed via the launcher application, connected launcher applications have to be discovered by the HbbTV application. This can be achieved by using the "discoverCSLaunchers" API:

Discover 2nd Screens	
Method Header	bool discoverCSLaunchers(function onCSDiscovery)
Parameters	onCSDiscovery: A callback function. See Figure 215 for the details.
Return Value	This returns true to indicate that the function has completed with no errors (and that callbacks are expected), false otherwise.
Remarks	Finds running 2nd Screen launcher applications on the home network, along with their enumeration ID, a friendly name, and their OS information.

Figure 214: Discover CS Launchers Method Signature

Discovery Callback	
Method Header	callback onCSDiscovery(int enumId, String friendlyName, String CsOsId)
Parameters	enumId: A unique ID for a 2nd Screen friendlyName: A friendly name for the 2nd Screen. It is optional that this parameter is returned. If it is not returned, it shall be set to the empty string "". CsOsId: The device information string. See further section on formats for details
Return Value	n/a
Remarks	This callback is called for each 2nd Screen device that is connected. 2nd Screen devices that are subsequently connected (i.e. after the call to discoverCSLaunchers) to the 1st Screen shall not cause the onCSDiscovery callback to fire.

Figure 215: Discovery Callback Method Signature

3.14.2.4.1.2 Method "Launch 2nd Screen Application"

To launch or install a 2nd Screen application the "launchCSApp" method needs to be called. The action that the launcher application on the 2nd Screen device needs to undertake is described by the payload string. The semantics of the instruction and the payload format are described further in Section 3.14.2.4.4.2.

Launch Application on 2nd Screen	
Method Header	Boolean launchCSApp(Integer enumId, String payload, function onCSLaunch)

Parameters	enumId: The enumeration ID of the 2nd Screen device to which the payload parameter string shall be sent. payload: See formats section below for the definition of format of the payload parameter string. onCSLaunch: A callback function. See below for the details.
Return Value	This returns true to indicate that the function has completed with no errors (and that callbacks are expected), false otherwise.
Remarks	Sends a payload string to the launcher application which contains an instruction set for the launcher application to execute. The result of the Launch operation is communicated to the caller application via the onCSLaunch callback.

Figure 216: Launch CS Application Method Signature

Launch Callback	
Method Header	callback onCSLaunch(int enumId, int errorCode)
Parameters	enumId: A unique ID for a launcher application errorCode: See below for error codes
Return Value	n/a
Remarks	Since there are certain actions that the Launcher application may undertake before responding (and thus the terminal invoking the onCSLaunch callback), there may be a long delay. Applications will therefore be responsible for timing out.

Figure 217: Launch Callback Method Signature

Error Code	Numeric value	Error Description
opRejected	0	The request was rejected automatically by the launcher application.
opDenied	1	The request was blocked by the user.
opNotGuaranteed	2	The request was completed successfully.
invalidId	3	The enumId is no longer available (i.e. it has become unavailable since discovery occurred).
generalError	4	A general error has occurred.

Figure 218: Launch Callback Error Codes

3.14.2.4.1.3 Method “Retrieving Base URL for Discovery and Communication”

If a 2nd Screen application needs to launch a 1st Screen application or use a WebSocket connection for communication with a 1st Screen application, then it needs to either

- Pass these parameters upon launch (as a URL query parameters on the application launch URL), or
- Use the DIAL discovery methods

For the first method to be possible, the calling 1st Screen application needs to be able to determine the locations of the WebSocket server URL so it can construct the launch URL before initiating the launch.

The methods below enable a 1st Screen application to determine the locations of these service endpoints.

Application Launch URL	
Method Header	String getAppLaunchUrl()
Parameters	None
Return Value	Returns the URL of the application launch service 1st Screen.
Remarks	None

Figure 219: Get App Launch Url Method Signature

WebSocket Server 1st Screen URL	
Method Header	String getApp2AppLocalBaseUrl()
Parameters	None
Return Value	Returns the base URL of the WebSocket server for use by the 1st Screen application.
Remarks	None

Figure 220: Get App2App Local Base Url Method Signature

WebSocket Server 2nd Screen URL	
Method Header	String getApp2AppRemoteBaseUrl()
Parameters	None
Return Value	Returns the base URL of the WebSocket server for use by the 2nd Screen application.
Remarks	None

Figure 221: Get App2App Remote Base Url Method Signature

3.14.2.4.2 DIAL Interface

The discovery and launching of 1st Screen applications will be provided through a DIAL interface.

3.14.2.4.2.1 Method "Discovering 1st Screen using DIAL"

HbbTV is a DIAL application registered at the DIAL registry as "HbbTV". Before the different URLs for launch and communication can be determined, the DIAL REST Service and the DIAL Application Resource URL need to be found. This is achieved using the mechanisms described in the DIAL specification. This consists of an SSDP M-SEARCH

request and response, followed by an HTTP GET to the URL obtained from the LOCATION: header in the M-SEARCH response.

This HTTP response contains an Application-URL header and a body. The response body is a UPnP device description as required by the DIAL specification. The DIAL Application Resource URL for HbbTV is the DIAL REST Service URL followed by a single slash character (”/”) and the application name (e.g.”HbbTV”, see example on Figure 222).

```
http://192.168.1.11:11111/apps/HbbTV
```

Figure 222: Example DIAL Resource URL

A device on a home network initiates device discovery by performing an M-SEARCH from the SSDP protocol with the Search Target header (ST) as defined by DIAL (see Figure 223). The terminal responses with HTTP/1.1 OK and LOCATION header, together with DIAL ST (see Figure 224). The home network device requests the device description file by an HTTP GET request to the LOCATION URL (see Figure 225). The terminal responds with HTTP/1.1 OK header containing the Application-URL as defined in DIAL (see Figure 226).

```
M-SEARCH * HTTP/1.1
HOST: 239.255.255.250:1900
MAN: "ssdp:discover"
MX: <seconds to delay response>
ST: urn:dial-multiscreen-org:service:dial:1
```

Figure 223: DIAL Device Discovery Request

```
HTTP/1.1 200 OK
CACHE-CONTROL: max-age = <seconds until advertisement expires>
EXT:
LOCATION: <URL for UPnP description for root device>
SERVER: <OS/version UPnP/1.0 product/version>
ST: urn:dial-multiscreen-org:service:dial:1
Figure 224: DIAL Device Discovery ResponseGET <path component of the LOCATION
URL> HTTP/1.1
Origin: http://sam.services. com/
```

Figure 225: Device Description Request

```
HTTP/1.1 200 OK
CONTENT-LANGUAGE: <language used in description>
CONTENT-LENGTH: <bytes in body>
CONTENT-TYPE: text/xml; charset="utf-8"
Application-URL: http://192.168.1.11.11111/apps
Access-Control-Allow-Origin:*
```

Figure 226: Device Description Response

3.14.2.4.2.2 Method “Launching 1st Screen Application using DIAL”

An application can be launched by a 2nd Screen by sending an XML AIT to the launch URL of the 1st Screen. It is done by sending an HTTP POST request to the Application Resource URL. The Application Resource URL is obtained from the discovery phase using DIAL Service Discovery (previous paragraph). The BODY data of the request contains an XML AIT describing the application to be launched (see [HBBTV] clause 7.2.3.2).

On receiving the HTTP POST request, the 1st Screen attempts to launch the application and returns the status using HTTP response codes:

Response Code	Description

201 CREATED	The application was launched successfully.
403 FORBIDDEN	The application could not be launched because the operation is rejected by the terminal or the user.
404 NOT FOUND	The application could not be launched because it could not be retrieved successfully, e.g. due to invalid application URL or application server unavailable.
503 SERVICE UNAVAILABLE	The application could not be launched because of the 1st Screen's current state.

Figure 227: Launch HTTP Response Codes

3.14.2.4.2.3 Method "Discovering the URL for Inter-Device-Communication using DIAL"

Discovery of the URL for Inter-Device-Communication is performed using an HTTP GET to the DIAL Application Resource. The response includes an XML document in the body, as described in the DIAL specification. It includes mandatory elements and one <additionalData> element, specifically for HbbTV, containing a <X_HbbTV_App2AppURL> element that provides the absolute URL of a Web Socket Server URL. A HTTP GET message is sent to 192.168.1.11, port 11111 (Figure 228) which produces an appropriate response (Figure 229).

```
GET /apps/HbbTV HTTP/1.1
Origin: http://sam.services.com/
```

Figure 228: Application Information Request

```
HTTP/1.1 200 OK
Origin: http://sam.services.com/

<?xml version="1.0" encoding="UTF-8"?>
<service xmlns="urn:dial-multiscreen-org:schemas:dial" dialVer="1.7">
    <name>HbbTV</name>
    <options allowStop="false"/>
    <state>running</state>
    <additionalData>
        <X_HbbTV_App2AppURL>URL of App2App communication service endpoint
        </X_HbbTV_App2AppURL>
    </additionalData>
</service>
```

Figure 229: Application Information Response

3.14.2.4.3 WebSocket Interface

A 1st Screen device provides a WebSocket server which allows 1st Screen applications to communicate with 2nd Screen applications on the home network.1st Screen applications determine the local URL of the WebSocket server using the JavaScript APIs from the HbbTVCSManager object (see previous section). 2nd Screen applications determine the URL of the WebSocket server via the DIAL service or can optionally receive this URL from the 1st Screen applications if they are launched by the 1st Screen application.

Both 1st Screen and 2nd Screen applications connect to the URL on the 1st Screen device using the WebSocket protocol in the role of a client of the WebSocket protocol. The server implementation on the 1st Screen device applies pairing rules to determine whether to pair connections from two clients. It then relays messages between the two client connections that are paired. The WebSocket server provides two URL's implementing the server-side of the WebSocket protocol specification [WSOCK]:

- Local URL is used for connections made by 1st Screen applications
- Remote URL is used for connections made by applications on other devices on the home network, including remote 2nd Screen applications

The resource-name to be used in the initial protocol request by the client has a format depicted in Figure 230, where:

- **base-url-resource-name** is the resource-name derived from the WebSocket URL
- **app-endpoint** is application specific. This is used in the process of matching this connection from a client to a corresponding client with which messages over the WebSocket protocol will be relayed. This will be chosen by developers to avoid collisions with other developers' applications.

```
resource-name = base-url-resource-name app-endpoint
```

Figure 230: Resource Name Format

When a WebSocket connection from a client comes in, the WebSocket server shall receive the client request handshake, but it shall not respond immediately with a handshake response. Instead it shall wait until the connection has been paired, or until the client disconnects. A connection in this state constitutes a waiting connection. If the resource-name used in the GET line of the request handshake from the client does not match the rules defined in the previous paragraph, the WebSocket sever responds with a "404 NOT FOUND" response and closes the WebSocket connection.

When a new waiting connection is established by a client, the WebSocket server will immediately try to determine whether there is a matching waiting connection. The terminal pairs only one connection to the local service end-point with exactly one other connection to the remote service end-point. If an application wants to establish multiple concurrent communication channels it must open a new client connection to the WebSocket server after its previous connection is paired.

When connections from two clients enter into a state of being paired to each other, the WebSocket server immediately completes the WebSocket protocol handshake for both client connections, to establish open WebSocket protocol connections to both of them. The connections are now both considered to be open, and the clients to be connected and paired.

Once paired and connections to both clients are open, the WebSocket server acts as a relay to pass messages between them, providing, in effect, a full-duplex bi-directional communication stream. When either client sends a WebSocket protocol frame, the terminal, upon receipt of that frame, immediately relays its contents to the other client via the corresponding WebSocket connection and maintains the same payload type. If one of the paired clients sends a Close frame or disconnects without sending a Close frame, then the WebSocket server additionally commences the process of disconnecting the other client by sending a corresponding Close.

3.14.2.4.4 **Formats**

3.14.2.4.4.1 CsOsId String

The CsOsId string, passed as parameter in the "onCSDiscovery" callback, is a string containing the following information about a 2nd Screen device:

- The identity of the Launcher application
- List of app store(s) that the Launcher application is aware of

- Existing User Agent string representative of a browser engine on the 2nd Screen device

The acronym "CS" stands for Companion Screen, a term used in the future HbbTV 2.0 standard to refer to 2nd Screen devices. The BNF syntax is presented on Figure 231.

BNF Syntax of the Companion Screen OS Identifier String	
csoid	launcher WS user_agent_string
launcher	launcher_product [launcher_comment]
launcher_product	launcher_name "/" launcher_version
launcher_comment	WS "(" comment_body ")"
comment_body	comment 0*(";" WS comment)
comment	store_info | manufacturer_specific_comment
store_info	"appstore" "/" app_store_id
WS	1*" "

Figure 231: BNF Syntax of the Companion Screen OS Identifier String

The comment optionally lists available app stores on the 2nd Screen device or any developer specific additional information. The User Agent string is derived directly from the platform/operating-system on which the Launcher Application is running.

3.14.2.4.4.2 Payload

The payload string, passed as parameter in the “launchCSApp” call, may carry two operation instructions:

- Install Application
- Launch Application

The payload data for the install and launch operations are carried as Strings which contain JSON formatted data.

The install operation may contain multiple sources for an installation to be sourced from, which are tried in the order specified. The install URL shall be appropriate to the associated store. An example is as follows:

```
{
  "install" : [
    {
      "installUrl" : "<url_to_install_from_store_1>",
      "appStoreId"  : "<name string for store_1>"
    },
    {
      "installUrl" : "<url_to_install_from_store_2>",
      "appStoreId"  : "<name string for store_2>"
    },
    ...,
    {
      "installUrl" : "<url_to_install_from_store_N>"
```

```
    }
  ]
}
```

Figure 232: Example Install Operation Payload

The launch operation contains either a single launch entry or two launch entries. The type of the launch entries is either “native” or “html”. Examples for a single launch entry and for two launch entries are as follows:

```
{
  "launch" : [
    {"launchUrl" : "<url to launch from>", "appType" : "html"}
  ]
}

{
  "launch" : [
    {"launchUrl" : "<url to launch from>", "appType" : "native"},
    {"launchUrl" : "<url to launch from>", "appType" : "html"}
  ]
}
```

Figure 233: Example Launch Operation Payload

The JSON schema for the install and launch operations payload of the “launchCSapp” function is defined as follows:

```
{
  "type": "object",
  "oneOf": [
    {
      "type": "object",
      "required": [
        "install"
      ],
      "install": {
        "type": "array",
        "items": {
          "type": "object",
          "properties": {
            "installUrl": {
              "type": "string",
              "format": "uri"
            },
            "appStoreId": {
              "type": "string"
            }
          },
          "required": [
            "installUrl"
          ],
          "additionalProperties": "false"
        },
        "minItems": 1
      }
    },
    {
      "type": "object",
      "required": [
        "launch"
      ],
      "launch": {
```

```
        "type": "array",
        "items": {
          "type": "object",
          "properties": {
            "launchUrl": {
              "type": "string",
              "format": "uri"
            },
            "appType": {
              "enum": [
                "native",
                "html"
              ]
            }
          },
          "required": [
            "launchUrl",
            "appType"
          ],
          "additionalProperties": "false"
        },
        "minItems": 1,
        "maxItems": 2
      }
    }
  ]
}
```

Figure 234: Operations Payload JSON Schema

3.14.2.5 Summary

In this section, the 1st Screen component was technically specified. During the technology selection (Section 3.14.2.2.2), Java for Android was selected as programming language used to develop the 1st Screen application. This will allows it to run on a multitude of Android-enabled devices, across different manufacturers, and on TV screens. HbbTV was selected for Inter-Device-Communication component, combining a number of state-of-the-art technologies widely supported on 1st Screen devices.

According to this specification, the following components have to be implemented:

- Java application for 1st Screen component
- WebView element embedding the Generic Dashboard component
- DIAL service allowing for remote discovery and launch
- WebSocket server for Inter-Device-Communication

3.14.3 2nd Screen

The 2nd Screen component (see Figure 235) will enable the prosumption of content corresponding to the current video which the End User is viewing. The End Users can consume suitable content, using voice commands to interact with the 2nd Screen application and interact with social networks by providing additional content (user-generated content) to friends and other SAM users.

- The 2nd Screen will act as a container for the Generic Dashboard (see Section 3.14.1)
- The 2nd Screen will also contain the Inter-Device-Communication (see Section 3.14.4), which will enable the communication between widgets and the 1st Screen component

and this component and the Voice Dialogue ((see Section 3.14.5), which will enable voice interaction.

- The 2nd Screen will enable the detection of the 1st Screen component and notifications

These functionalities will be implemented as an Android application. The Generic Dashboard component will contain the additional content which will be displayed by the 2nd Screen application using a WebView element (A WebView is an integrated browser in the mobile application, which can be used to deliver full-fledged HTML, CSS and JS applications or pages).

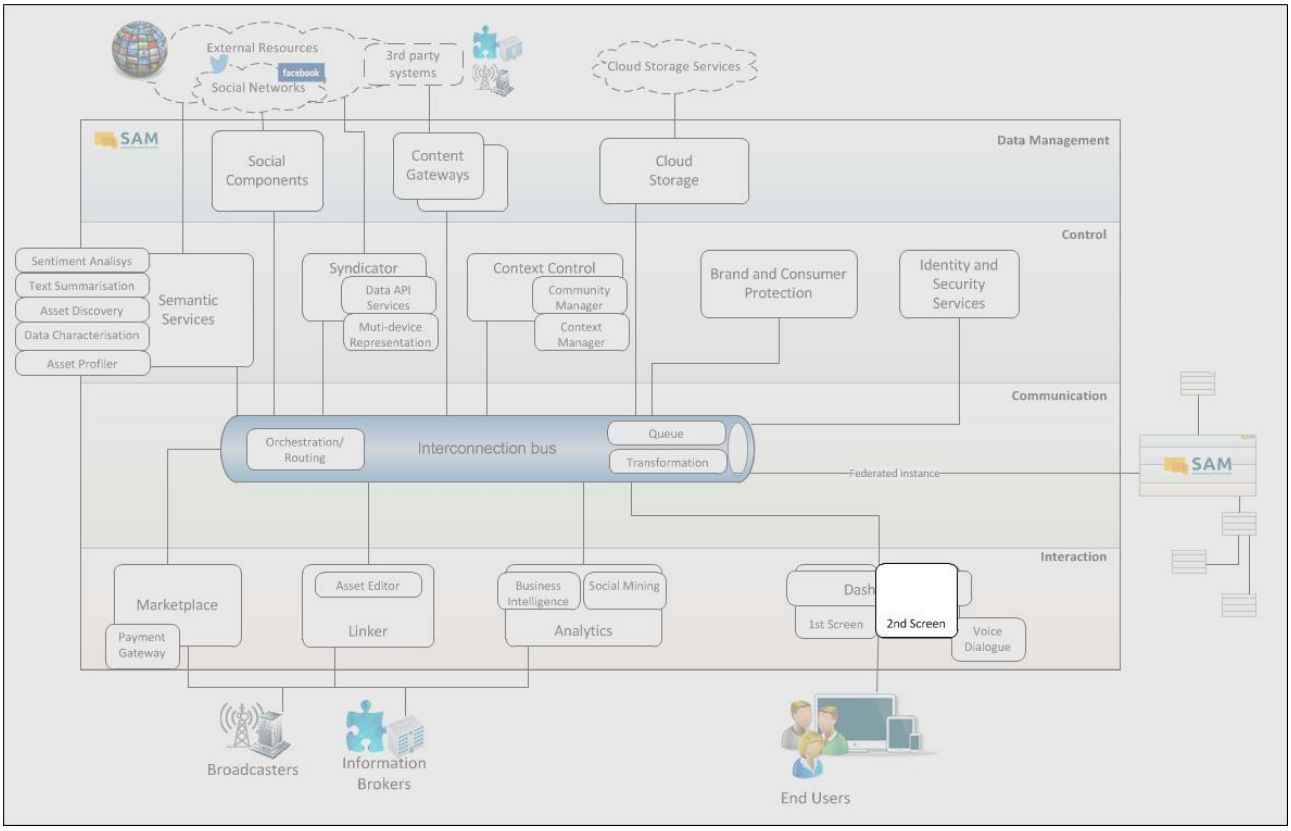

Figure 235: Architecture Overview – 2nd Screen

3.14.3.1 Major Design Decisions

The 2nd Screen should be able to show additional information to the video watched by the End User using his/her 1st Screen (see Section 3.14.1). This additional information will be displayed on the Generic Dashboard which will contain widgets with different kinds of information corresponding to the consumed video and will be integrated using a WebView container element (see Section 3.14.1).

The 2nd Screen also needs to discover available 1st Screen devices and the corresponding interfaces in the local network so that a connection between both components can be established. The approach for the necessary connection has been described in the 1st Screen component section (see Section 3.14.2.2.2.2). The 2nd Screen should run on current tablets and smartphones and support hardware access which will be needed for adaptation of content to be displayed. Of course, this will affect the choice of the operating system. In the next sections, this decision will be explained in greater detail.

3.14.3.2 Technology Comparison and Selection

This subsection outlines technology selection criteria and compares existing technologies – potential candidates for the realisation of the 2nd Screen. Within the subsection the areas that need to be implemented or improved to fully meet SAM specific requirements are also identified and presented. The selected technologies will be used as a basis for the development phase in the realisation of the technical design of this component. The target platform has to be open for hardware access to make use of internal sensors such as the GPS module of the device and outputs like the audio device. The internal sensors will be used by other SAM components for context improvements and user interaction.

3.14.3.2.1 Possible Technologies

In order to realise the 2nd Screen component, several options have to be considered for implementing the base technology. The assessment of these technologies is presented in the following paragraphs.

- **Android SDK**[128] is a set of APIs that allows developers to quickly and easily create apps for Android devices. It contains tools for designing user interfaces and a documentation for the API. Essentially an Android app consists of activities (programs that the user interacts with), services (programs that run in the background or provide some function to other apps), and broadcast receivers (programs that catch information important to your app).
- **Xamarin.Mobile**[129] is a library that exposes a single set of APIs for accessing common mobile device functionality across the iOS, Android, and Windows Phone platforms. This increases the amount of code developers can share across mobile platforms, making mobile app development more productive. Xamarin.Mobile currently abstracts the contacts, camera, and geo-location APIs across iOS, Android and Windows Phone platforms. Future plans include notifications and accelerometer services. It is based on Mono95 and uses C#, so developers can reuse existing .NET Framework libraries that have been successfully ported to the Mono Framework.
- **Titanium Framework**[130] makes use of HTML5 for platform-independent mobile development. It provides developers with its own Integrated Development Environment (IDE). The Titanium Framework combines many mobile devices and platforms inside one central development platform to provide fast access to as many users as possible. Due to the focus on web technologies, it only provides very basic and high-level APIs to access the hardware of mobile devices.
- **Cocoa Touch**[131] is a UI framework that allows developers to quickly and easily create apps for iOS devices. Similar to the Android SDK this framework enables the developer to create new user interfaces. It also comes with an API documentation to ease the development process. Cocoa uses Objective-C as code language and allows the usage of C and C++ as coding languages.

Parameter	Importance	Android SDK	Xamarin.Mobile	Titanium Framework	Cocoa Touch
Generic Parameters					
Maturity & Stability	+++	+++	++	+	+++
Regularly Updated	+/-	+++	+++	+++	+++

[128] http://developer.android.com/index.html
[129] http://xamarin.com/mobileapi
[130] http://www.appcelerator.com/titanium/
[131] https://developer.apple.com/technologies/ios/cocoa-touch.html

Technical Up-to-Datedness / Appeal	++	+++	++	+	+++
Open Source	+	YES	NO	NO	NO
Non-Infecting	+	YES	YES	YES	YES
Code Quality	++	N/A	N/A	N/A	N/A
Extensibility	+++	+++	++	+/-	+++
Community	-	+++	++	+	++
Performance / Scalability	+++	+++	++	+	+++
Reuse of existing developments	+/-	+	+/-	-	-
EU project origin	---	NO	NO	NO	NO
Platform (Portability)	+	---	++	+++	---
Open Standards Compliance	+	+++	+	+	+/-
Interoperability (easy integration for all platforms)	++	N/A	N/A	N/A	N/A
Specific Parameters					
Usability	++	+++	++	+/-	+++
Access to device specifications and sensors	+++	+++	++	+/-	+++
Mobile support	+++	+++	+++	+++	+++
Mobile OS APIs	++	+++	+++	+/-	+++
Web browser component support	+++	YES	YES	YES	YES

Figure 236: Parameter Evaluation of Technologies for Mobile Applications

3.14.3.2.2 **Technology Selection**

The operating system is one of the most important choices regarding the 2nd Screen since there are many requirements to the runtime itself. The chosen operating system of the device should be widely available to reach the highest possible End User range. Figure 237 shows the market share on the worldwide smartphone market to ease the choice for the operating system.

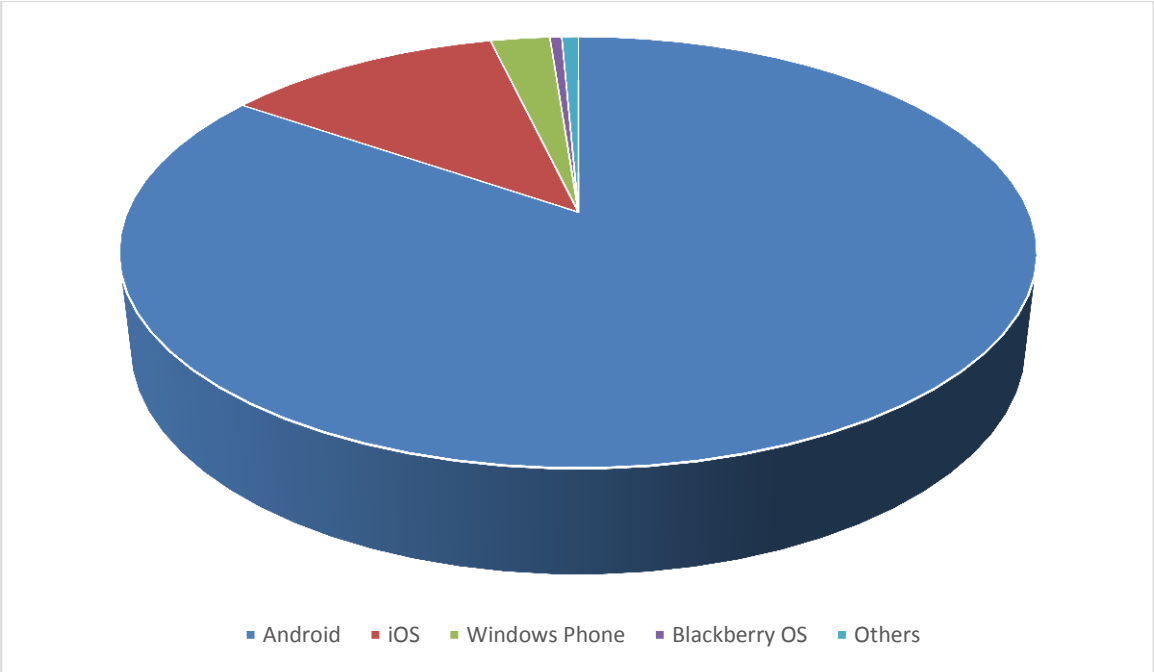

Figure 237: Mobile Operating System Market Share - Q2 2014

According to the statistic provided by [IDC14] the highest market share of mobile operating systems is taken by the Android platform, which is a reason to choose Android. Another reason is the more commonly known programming language Java instead of the more specialised programming language Objective-C, which is mandatory for the Android contender, the iOS platform. Other platforms are not considered in this section since their market share is too low to reach the targeted audience, and as they also did not meet the criteria in the previous sections.

Another possibility would be to choose a hybrid application framework like Xamarin. The consortium has already tested these frameworks in a variety of other projects, but always found the additional work that needs to be invested in the framework to be higher than the benefits of publishing to multiple platforms. Especially as the Android market presents an extremely high portion of the market and is still growing, selecting Android seems to be the most effective way to use development resources to create a high quality product and if necessary provide an iOS application after the project time if desirable for commercial reasons.

3.14.3.3 Technical Component Specification

3.14.3.3.1 Structure

Figure 238 provides an overview of the 2nd Screen which can be found also in D3.2.1 Global Architecture Definition. Additionally, this version of the overview has been augmented with the selected technologies. As seen in Figure 238 it consists of the following subcomponents:

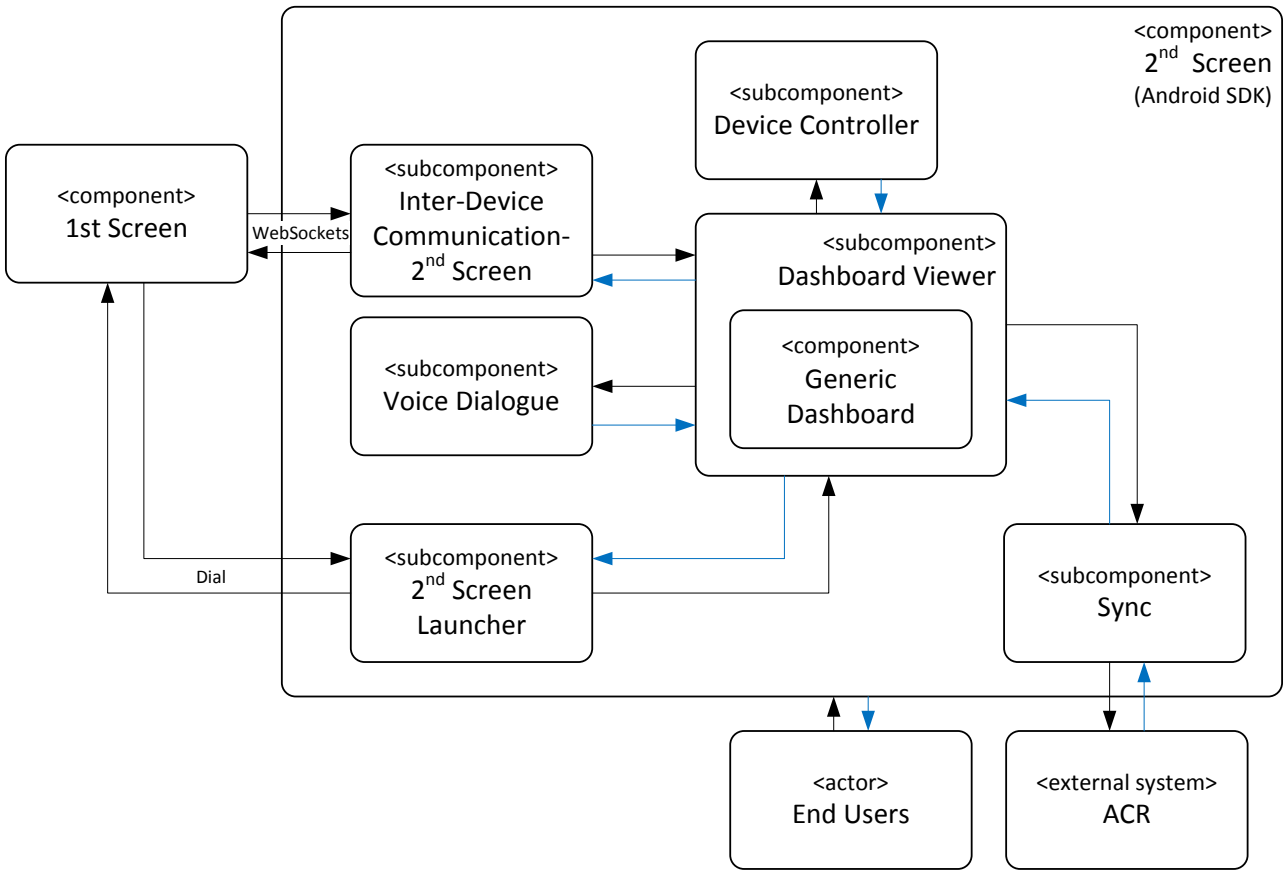

Figure 238: Overview with Selected Technologies – 2nd Screen

3.14.3.4 Specification of Interfaces, Protocols and Formats

The 2nd Screen component will provide JavaScript interfaces which will be used to communicate with the Generic Dashboard component.

3.14.3.4.1 JavaScript Interfaces

The JavaScript interfaces will be used by the Generic Dashboard to enable the interaction with the 2nd Screen and other components within the 2nd Screen. They will be added to the WebView element in the Android application.

3.14.3.4.1.1 Method “Get GPS Coordinates”

For providing the GPS coordinates, the API provides the method “getGPSCoordinates”. The method has no parameters.

Get GPS Coordinates	
Method Header	api.prototype.getGPSCoordinates()
Parameters	None
Return Value	GPSCoordinates object
Error Handling	In case of an error, null is returned
Remarks	Only available on GPS-enabled devices.

Figure 239: JavaScript Interface Description – Get GPS Coordinates

In Figure 240, the signature of the method for “getGPSCoordinates” is shown as well as an example for the usage of this method.

```
/**
 * @returns {GPSCoordinates} or null
 */
api.prototype.getGPSCoordinates = function() { ... }

var gps = api().getGPSCoordinates();
```

Figure 240: Source Code Example – API Method Signature for providing current GPS coordinates.

The example usage of the method at the bottom of Figure 240 will provide the stored GPS coordinates. If the API call is successful, a GPSCoordinates (see Figure 241) object will be returned, otherwise null.

```
/**
 * GPSCoordinates object
 *
 * @param lat Latitude
 * @param long Longitude
 */
function GPSCoordinates(lat, long) {
      this.lat = lat;
      this.long = long;
}
```

Figure 241: Source Code Example – GPS Coordinates Object

3.14.3.4.1.2 Method “Get Device Specification”

For providing the specification of the current device, the API provides the method “getDeviceSpecification”. The method has no parameter.

Get Device Specification	
Method Header	api.prototype.getDeviceSpecification()
Parameters	None
Return Value	DeviceSpecification object
Error Handling	In case of an error, null is returned
Remarks	None

Figure 242: JavaScript Interface Description – Get Device Specification

In Figure 250, the signature of the method for “getDeviceSpecification” is shown as well as an example for the usage of this method.

```
/**
 * @returns {DeviceSpecification}
 */
api.prototype.getDeviceSpecification = function () { ... }

var specs = api().getDeviceSpecification();
```

Figure 243: Source Code Example – API Method Signature for providing the specification of the current device.

The example usage of the method at the bottom of Figure 250 will provide the stored GPS coordinates. If the API call is successful, a DeviceSpecification (see Figure 251) object will be returned, otherwise null.

```
/**
 * DeviceSpecification object
 *
 * @param width Screen width
 * @param height Screen Height
 * @param
 *
 */
function DeviceSpecification(width, height) {
      this.ScreenWidth = width;
      this.ScreenHeight = height;
}
```

Figure 244: Source Code Example – DeviceSpecification Object

3.14.3.4.1.3 Method “Start Voice Interaction”

For listening to voice commands from the End User, the API provides the method “startVoiceInteraction”. The method has no parameter.

Start Voice Interaction	
Method Header	api.prototype.startVoiceInteraction()
Parameters	None
Return Value	True on success
Error Handling	In case of an error, false is returned.
Remarks	This method will execute the start method in the Voice Dialogue (see Section 3.14.5).

Figure 245: JavaScript Interface Description – Initiate Voice Interaction

In Figure 246, the signature of the method for “startVoiceInteraction“ is shown as well as an example for the usage of this method.

```
/**
 * @returns {Boolean}
 */
api.prototype.startVoiceInteraction = function () { ... }

var result = api().startVoiceInteraction();
```

Figure 246: Source Code Example – API Method Signature for starting Voice Interaction.

The example usage of the method at the bottom Figure 246 will start the voice interaction process, where the Voice Dialogue will listen to the End User and processes this voice input. If the API call is successful, true will be returned, otherwise false.

3.14.3.4.1.4 Method "Send Message to 1st Screen"

For sending a message to the 1st Screen, the API provides the method "sendMessageToFirstScreen". The method has one parameter, which will contain the action to be performed.

Send Message to 1st Screen	
Method Header	api.prototype.sendMessageToFirstScreen()
Parameters	message: a message object which should be transferred.
Return Value	True on success
Error Handling	In case of an error, false is returned.
Remarks	None

Figure 247: JavaScript Interface Description – Send Message to 1st Screen

In Figure 248, the signature of the method for "sendMessageToFirstScreen" is shown as well as an example for the usage of this method.

```
/**
 * @returns {Boolean}
 */
api.prototype.sendMessageToFirstScreen = function (message) {...}

var result = api().sendMessageToFirstScreen(message);
```

Figure 248: Source Code Example – API Method Signature for sending a message to the 1st Screen.

The example usage of the method at the bottom of Figure 248 will send a message to the 1st Screen using the Inter-Device-Communication subcomponent. If the API call is successful, true will be returned, otherwise false.

3.14.3.5 Summary

The 2nd Screen will be responsible for displaying the Generic Dashboard component and its content on a 2nd Screen device. It will also support the communication with a 1st Screen component by using the Inter-Device-Communication component. Additionally, voice interaction will be supported using the Voice Dialogue component.

The 2nd Screen component will be a mobile application for the Android platform. It will display the Generic Dashboard in a WebView element and communicate by using JavaScript.

3.14.4 Inter-Device-Communication

The Inter-Device-Communication (see Figure 249) will provide communication between the 1st Screen and 2nd Screen components regarding the Generic Dashboard, which will be contained by the Dashboard Viewer subcomponents in each screen.

- The Inter-Device-Communication will enable the communication between a 1st Screen and the Dashboard Viewer subcomponent contained by its 2nd Screen component
- The Inter-Device-Communication will enable the communication between one 1st Screen and multiple 2nd Screen components (1-to-n)

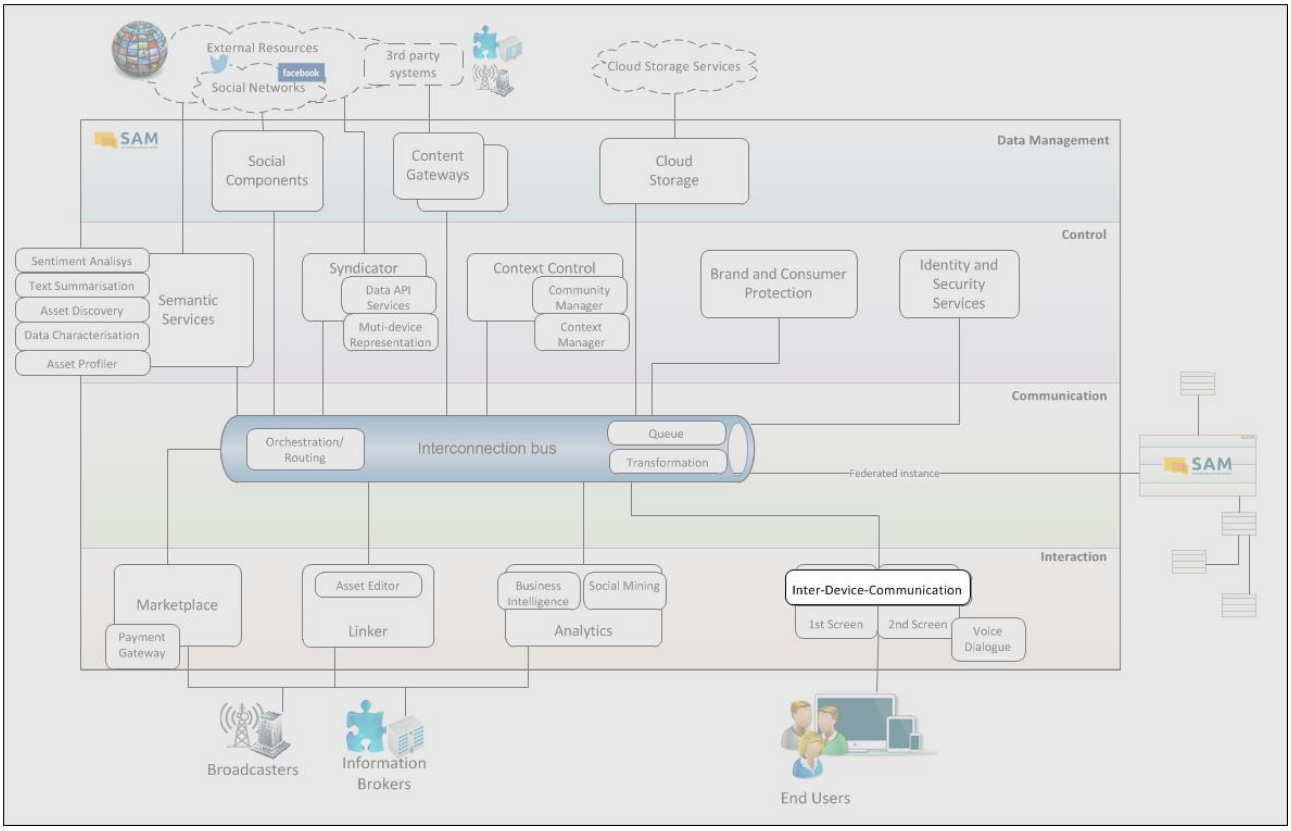

Figure 249: Architecture Overview – Inter-Device-Communication

3.14.4.1 Major Design Decisions

This component will be located in 2nd Screen component as a communication adapter for the 1st Screen component. The 1st Screen component will send information about the current displayed video element to multiple 2nd Screen components. It is vital that the connection is stable and this information will be transmitted with low latency. This component also has to provide secure communication and the chosen technology should not generate much overhead in configuration and transmission context.

3.14.4.1.1 Sockets vs. HTTP

Sockets and HTTP are both communication protocols running on web servers and web browsers. They are both using the TCP port 80 which has the benefit of not colliding with firewalls which block non-web Internet connections. Both sockets and HTTP need a handshake to establish a connection using a HTTP header, but once it is established many differences come up:

HTTP is a request-response protocol which means that the client has to send a request and will receive a response. Both messages contain an HTTP header which generates an overhead comparing to sockets. After such a transmission it also needs a new connection establishment for the next data transfer (message). Additionally, HTTP is unidirectional, which means connections are only able to transfer payload data in one direction. As a conclusion, two connections have to be established to communicate. As an alternative, the client could poll the server after a certain time (interval) which would create unnecessary traffic.

WebSockets are a special subtype of sockets. After establishing the connection using one HTTP request and response (handshake), the following messages can be received using the already established connection. Because WebSockets support full-duplex communication, messages can be send and received by using one connection per client, enabling more send/received message in the same time as HTTP. Because of this advantage and the abolition of HTTP overhead it is clear that WebSockets are the better choice.

3.14.4.1.2 Lightweight vs. Heavyweight

Heavyweight in this context would be an extra message bus running on a separate server, which would require a lot of steps to finally have the system running (like e.g. familiarisation, seting up server and dependencies, adaptation to the current context). Finally, it would provide the functionality needed but with a big overhead.

Lightweight would be a point-to-point connection between the 1st Screen, on which the server runs, and the 2nd Screen. Additional and unused functionality will make the system more complex and error-prone.

3.14.4.1.3 Local vs. Remote Solution

Due to the fact that 1st Screen and 2nd Screen devices mostly will be located in the same place, a local connection between both can be assumed. A local connection has advantages such as low latency, less interference by the surroundings and the network topology and less private data using the Internet.

3.14.4.2 Technology Comparison and Selection

This subsection outlines the technology selection criteria and compares existing technologies – potential candidates for the realisation of Inter-Device-Communication. Within the subsection the areas that need to be implemented or improved to fully meet SAM specific requirements are also identified and presented. The selected technologies will be used as a basis for the development of the technical design of this component.

3.14.4.2.1 Possible Technologies and Comparison

In order to implement the communication to be provided by the Inter-Device-Communication component, several options have to be considered as base technology. The assessment of these technologies is presented in the following paragraphs.

- Using basic **REST or WSDL-based Web services** is a widely accepted and standards-conformant pattern for implementing all kinds of APIs all over the Web. Within SAM this will also be supported by the Interconnection Bus and other components. For the Inter-Device-Communication these standards does not represent a complete solution as the low latency requirement and high quantity of exchanged

messages demands for a constant connection and makes the long request times of HTTP a showstopper.

- **Apache ActiveMQ**[132] is a messaging bus based on a similar socket connection as XMPP made for high throughput and with many client libraries available for different languages and platforms. It also has support for REST, SOAP, JMS, XMPP and other technologies, and integrates Apache Camel to use Enterprise Integration Patterns. Additionally, it supports SSL and JAAS (Java Authentication and Authorisation Service) and is used in many Apache projects. For SAM, ActiveMQ provides too broad a set of configurable functionality and the overhead of too many possibilities and configurations is perceived as an impeding factor for SAM. The projects references all lie within the Apache community. Real-world problems like using socket connections through a router Network Address Translation or a server firewall can be a problem. Additionally, ActiveMQ and Camel include a lot of different technology concepts that would need to be understood by a broad range of implementers.
- **ServiceMix**[133] is an Enterprise Service Bus and consists of ActiveMQ, Apache ODE, which is a complete BPEL engine, and a dozen other Apache projects, configured with Apache Maven and bundled via OSGi and running on integrated servers. While ServiceMix supports a very high number of features, it is so complex that only few users can be reported, and those that use ServiceMix have decided to use the commercial product Fuse-ESB which is a supported version of ServiceMix - which means they can rely on a support team to help with issues, which also adds additional cost. Basically, ServiceMix is struggling with the same problems as ActiveMQ. ServiceMix was already chosen in a national project in Germany where it was abandoned by the team after three month because of too high complexity.
- **WebSockets** is a full-duplex protocol using TCP. It uses HTTP for the handshake but further communication is based on message streams using TCP without the need of further usage of HTTP. Messages can be sent without having the recipient asked for permission. Also, the recipient will not have to confirm the reception of a message, which argues for WebSockets as a lightweight protocol. Also the usage of port 80 enables the protocol to prevent most collisions with firewalls.
- **Plain Sockets**, or Raw Sockets, can be used for the data transfer between network devices. Mostly, Raw Sockets are used in network-near software because of the configuration potential. So, in contrast to for instance WebSockets, you can configure your own TCP header.

Parameter	Importance	REST/ SOAP	ActiveMQ	Service-Mix	Web-Sockets	Plain Sockets
Generic Parameters						
Maturity & Stability	+	+++	++	++	++	+++
Regularly Updated	+/-	+	++	++	+/-	-
Technical Up-to-Dateness / Appeal	++	+++	++	++	+++	-
Open Source	+/-	YES	YES	YES	YES	YES
Non-Infecting	+	YES	YES	YES	YES	YES
Code Quality	++	N/A	N/A	N/A	N/A	N/A
Extensibility	+/-	N/A	+++	+++	N/A	N/A

[132] http://activemq.apache.org/
[133] http://servicemix.apache.org/

Community	+/-	+++	+++	+++	++	+/-
Performance / Scalability	+++	+	+++	+++	+++	+++
Reuse of existing developments	+	++	+	+	+	-
EU project origin	---	NO	NO	NO	NO	NO
Platform (Portability)	++	+++	+++	+++	+++	+++
Open Standards Compliance	+	+++	+++	+++	+++	+++
Interoperability (easy integration for all platforms)	++	+++	+++	+++	++	+
Specific Parameters						
Low Latency	+++	+	+++	+++	+++	+++
Data Throughput	-	+/-	+++	+++	+	+++
Multiple connections	+	+++	+++	+++	+++	+++
Duplex Connections	++	-	+++	+++	+++	+++
Support of common web technologies	+	+++	+	+	+++	---
Little framework & configuration overhead	+++	+++	+/-	-	+++	+++

Figure 250: Parameter Evaluation of Technologies for Communication

3.14.4.2.2 **Technology Selection**

In order to implement the communication part of the IDC component, WebSocket has been chosen. HTTP based communication, i.e. SOAP or REST, was not selected, mainly because of the long request times of a single request. Streaming data and calls through a socket connection requires only a single-time connection negotiation, and the remaining calls and data transfers happen instantly. Also, the two-way communication is a critical advantage, as only the 2nd Screen needs to connect to the 1st Screen to enable two-way communication. Using HTTP based technologies, the 2nd Screen would also have to have a server that the 1st Screen would have to discover and call in order to send calls or data. Another advantage is that the overhead of the HTTP protocol (the HTTP headers) would add up when a lot of communication takes place, and this can be ignored using a socket connection.

ServiceMix was not selected because of two main reasons. Firstly, it is a completely general framework that tries to solve too many problems at once and for this includes a high number of different technologies, frameworks, concepts and dependencies. This is much too generic for the SAM problem space and the consortium had experienced problems with the breadth of functionality of ServiceMix in other projects before. This breadth of functionality directly translated into overhead in setup work, configuration work and learning, necessary for using the different concepts, languages and technologies that need to be understood and applied with the ServiceMix framework. Secondly, there is no notable real-world application that has been built using ServiceMix, which is an indicator for an unproductive framework.

The decision against Apache ActiveMQ was not as easy, as ActiveMQ is much more focused than ServiceMix. Still, the features, dependencies and configuration necessary are seen as too severe to choose ActiveMQ over a standard like WebSockets.

When deciding between pure Socket connections and WebSockets, WebSockets satisfied the requirements better, because WebSockets is a standard made for two-way communication that should be easily established and just has the minimum set of features needed to be lean and effective tool for the job of communicating locally between two or more devices.

3.14.4.3 Technical Component Specification

3.14.4.3.1 Structure

Figure 251 provides an overview of the Inter-Device-Communication which can be found also in D3.2.1 Global Architecture Definition. Additionally this version of the overview in this chapter has been augmented with the selected technologies. The Inter-Device-Communication will realise the communication between the Dashboard Viewer and 1st Screen devices based on WebSockets. As seen in Figure 251, the IDC component will consists of the following subcomponents:

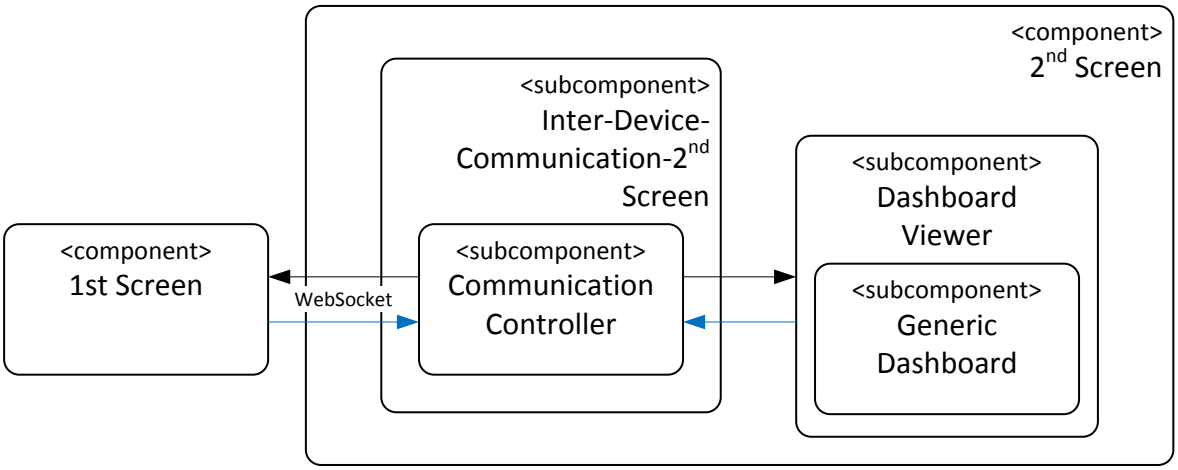

Figure 251: Overview with Selected Technologies – Inter-Device-Communication

3.14.4.4 Specification of Interfaces, Protocols and Formats

The communication between 1st Screen and 2nd Screen devices will be based on the interfaces in the following section.

3.14.4.4.1 Java Interfaces

The Java interfaces will provide a convenient way to use the Inter-Device-Communication for Android and Java developers.

3.14.4.4.1.1 Method “Provide Connection Details”

For providing the connection details, the API provides the method “provideConnectionDetails”. The method has three parameters of which the last two are optional. The URI object, which points to the WebSocket server and the username and password needed.

Provide Connection Details

Method Header	public boolean provideConnectionDetails(URI uri, String username, String password)
Parameters	• uri: The URI which points to the server • username: The username needed to establish a connection • password: The password needed, to establish a connection
Return Value	True on success
Error Handling	In case of an error, false is returned
Remarks	The connection details provided will be checked on their format but not if a connection can be established.

Figure 252: Java Interface – Provide Connection Details

In Figure 253, the signature of the method for “provideConnectionDetails” is shown as well as an example for the usage of this method.

```
/**
* @param uri, the uri which points to the server
* @param username, the username needed to establish a connection
* @param password, the password needed to establish a connection
* @return on success true, otherwise false
*/
public boolean provideConnectionDetails(URI uri, String username, String
password)

boolean success = api.provideConnectionDetails(new
URI("http://example.com/"), "username", "password");
```

Figure 253: Source Code Example – API Method Signature for Provide Connection Details

The example usage of the method at the bottom of Figure 253 will provide connection details. If the API call is successful true will be returned, otherwise false.

3.14.4.4.1.2 Method “Connect”

For the creation of a connection between the WebSocket server at the 1st Screen and a 2nd Screen component, the API provides the method “Connect”. The method has no parameter.

Connect	
Method Header	public boolean connect()
Parameters	None
Return Value	True on success
Error Handling	In case of an error, false is returned
Remarks	None

Figure 254: Java Interface – Connect

In Figure 255, the signature of the method for “Connect” is shown as well as an example for the usage of this method.

```
/**
 * @return on success true, otherwise false
 */
public boolean connect()

boolean success = api.connect();
```

Figure 255: Source Code Example – API Method Signature for Connect to a 1st Screen

The example usage of the method at the bottom of Figure 253 will create a connection using the provided connection details of Figure 255. If the API call is successful, true will be returned, otherwise false.

3.14.4.4.1.3 Method “Disconnect”

For closing an existing connection between the WebSocket server at the 1st Screen and a 2nd Screen component, the API provides the method “Disconnect”. The method has no parameters.

Disconnect	
Method Header	public boolean disconnect()
Parameters	None
Return Value	True on success
Error Handling	In case of an error, false is returned
Remarks	None

Figure 256: Java Interface – Disconnect

In Figure 257, the signature of the method for “Disconnect” is shown as well as an example for the usage of this method.

```
/**
 * @return on success true, otherwise false
 */
public boolean disconnect()

boolean success = api.disconnect();
```

Figure 257: Source Code Example – API Method Signature for Disconnect from a 1st Screen

The example usage of the method at the bottom of Figure 257 will close an existing connection to a WebSocket server at a 1st Screen. If the API call is successful true will be returned, otherwise false.

3.14.4.4.1.4 Method “Send WebSocket Message”

For sending data/messages to the WebSocket server at the 1st Screen component, the API provides the method “SendWSMessage”. The method has one parameter, which will contain the object to be sent.

Send WS Message	
Method Header	public void sendWSMessage(Object message)
Parameters	message: The object containing the data to be sent to the 1st Screen component
Return Value	None
Error Handling	None
Remarks	Due to the chosen communication protocol there will be no confirmation of receipt when sending messages using WebSocket.

Figure 258: Java Interface – Send WebSocket Message

In Figure 259, the signature of the method for sending a messing using a WebSocket connection is shown as well as an example for the usage of this method.

```
/**
 * @param message, the message object to be send
 * @return on success true, otherwise false
 */
public void sendWSMessage(Object message)

api.sendWSMessage(dvMessage);
```

Figure 259: Source Code Example – API Method Signature for Send a WebSocket Message

The example usage of the method at the bottom of Figure 259 will send a message to the WebSocket server at a 1st Screen.

3.14.4.5 Summary

The Inter-Device-Communication is responsible for a connection between the 1st Screen and 2nd Screen component. It will distribute information from and to the Dashboard Viewer subcomponent, which will be provided by the 2nd Screen component.

The communication with the 1st Screen component will be realised using a WebSocket connection. Due to the fact this subcomponent will be part of the 2nd Screen device, it will be implemented using the programming language Java.

3.14.5 Voice Dialogue

The Voice Dialogue subcomponent (VD) will enable voice interaction support in the 2nd Screen. It will communicate through the Dashboard Viewer with the 2nd Screen so that it can remain synchronised with it. Automatic Speech Recognition (ASR) and Text-To-Speech (TTS) engines will also be connected to Voice Dialogue as external services. Figure 260 shows how the component is placed within the overall SAM Platform environment.

Thus, in general, VD will:

- Exchange information with input/output sources for both graphical and spoken interaction
- Keep track of the information gathered by the user during the current dialogue session

- Decide how the dialogue should progress next, based on this information

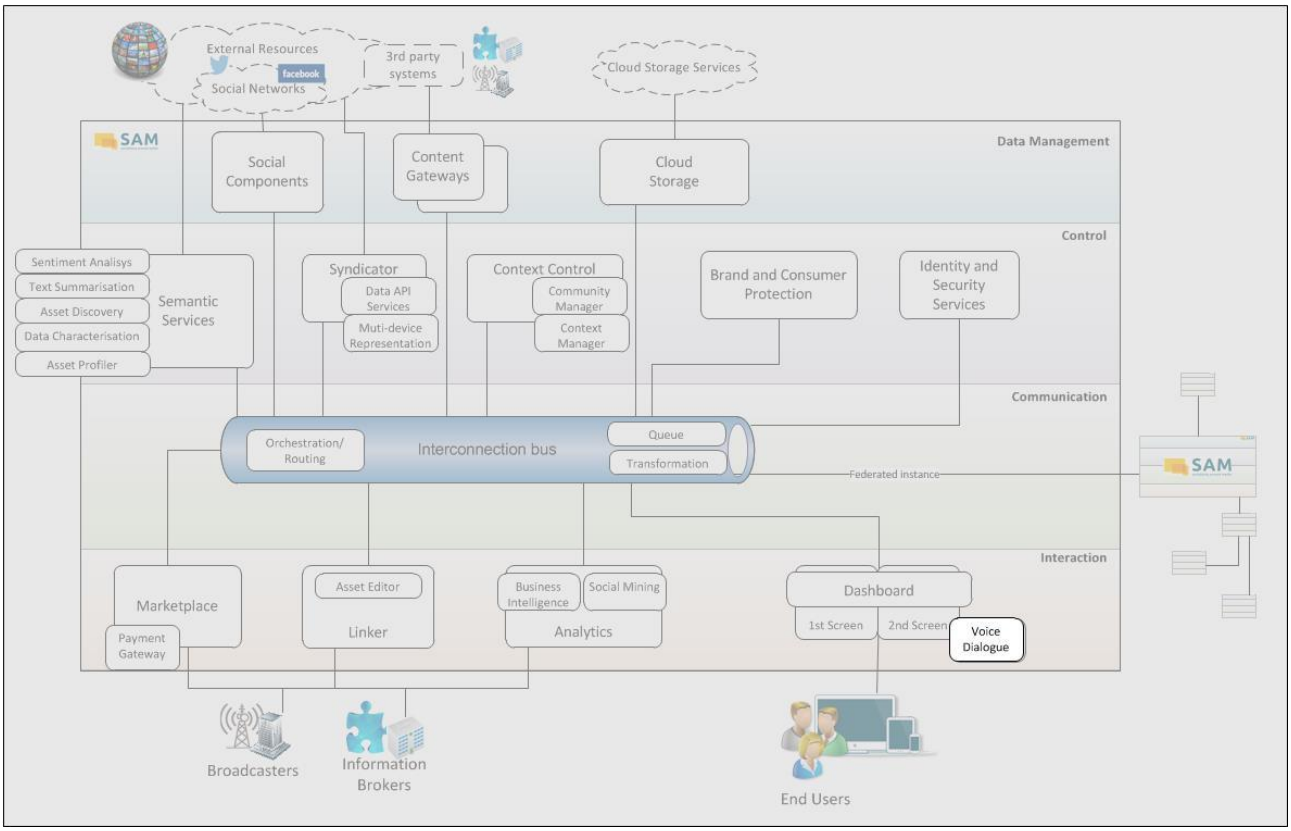

Figure 260: Architecture Overview – Voice Dialogue

3.14.5.1 Major Design Decisions

It has been decided that the VD component will be implemented as a cloud-based service. This architecture has the advantages of being independent of the hardware's capabilities and better protecting the source code. It also enables the developer to effortlessly update the core mechanics of the application without need for the user to manually update her thin client.

3.14.5.2 Technology Comparison and Selection

3.14.5.2.1 Possible Technologies and Comparison

3.14.5.2.1.1 Dialogue Manager

Since VD will be developed by Talkamatic, which specialises in building dialogue systems, choosing an internally built system for this task can be considered as the de facto best option. Moreover, Talkamatic Dialogue Manager (TDM) is a dialogue managing system that hosts a series of advantages compared to other solutions.

Two of the most popular voice-based dialogue products are Apple's Siri and Google's Google Now. Both of them sport various features with one important omission (especially in the context of SAM): they lack an API that can be used to integrate 3rd party applications into them.

Setting this factor aside, it can be said that both systems lack features that TDM supports.

First, they do not support full multi-modality. Interface units such as lists require the user to look at the screen and eventually select items in them via the GUI. This can be quite limiting in use contexts where the End User needs to be as little distracted as possible (as in SAM).

Additionally, specifically Siri prefers to assume about missing pieces of information in a conversation instead of asking about them. The End User has to provide all related information in one utterance, a requirement that leads to less flexibility. Siri also does not support returning to a topic that has been temporarily interrupted by another task. For example, if a user asks for the weather while booking tickets, upon returning to the booking menu, all submitted information will be lost.

There are two other dialogue products that, in contrast with Google's and Apple's solutions, offer an API: Microsoft's Cortana and Nuance's Nina Mobile. Cortana's API is limited to defining a set of voice commands that 3rd party apps can dynamically fill in with a number of parameters. These voice commands are parsed as bare strings leaving no possibility for syntactic parsing or natural language processing in general.

Although less known than Siri, Nina Mobile is a Virtual Assistant for mobile Customer Service applications and brings the ease of voice control to functionalities such as user authentication, menu navigation and financial transactions. However, Nina is restricted to handle the domains that Nuance supplies. For example, in order to build a music playing Nina, Nuance supplies all the needed bits and pieces for such an application. Thus, developers are locked within a finite set of possible application types and can hardly mix domains in order to create a cross-domain application.

3.14.5.2.1.2 Automatic Speech Recognition

Automatic Speech Recognition (ASR) is the external service that will be used by VD in order to get input from the End-User. Relatively few technologies exist in the market and even fewer are widely available, easy to implement and reliable:

- The **Nuance Vocon**[134] recogniser is fast and runs directly on the device. It can cope with relatively large grammars with reasonable speed, and can handle several hundred thousand utterances. The Software Development Kit (SDK) costs €1000 + €1000 per language. It also requires a small runtime licence fee for every unit.
- **Nuance NDEV**[135] is a cloud-based recogniser from Nuance. It has a general language model, but there are also specialised language models for specific domains such as music and TV. It offers different service levels, including a free option allowing 5000 transactions per day. Since the recogniser is cloud-based, a good network connection is required when using speech input. Latency can also be higher than for an embedded recogniser.
- **Nuance Vocon Hybrid**[136] provides a combination of Vocon and NDEV. An embedded recogniser (Vocon) runs on the device, providing fast recognition for a smaller grammar, while the software is also connected to a cloud based recogniser, providing slower, network dependent, but wider-coverage recognitions.
- The ASR from **Google**[137] is free to use within the Android platform. It is an open recogniser, and Google currently provides two different language models: web search

[134] http://www.nuance.com/industries/automotive/products/VoCon-3200.asp
[135] http://dragonmobile.nuancemobiledeveloper.com/public/index.php?task=home
[136] http://www.nuance.com/for-business/speech-recognition-solutions/vocon-hybrid/index.htm
[137] http://developer.android.com/reference/android/speech/SpeechRecognizer.html

and free form. The web-search language model is adapted for search queries (isolated words, keywords, no complete sentences), while the free-form model supports a wider range of utterances. Like NDEV, Google ASR suffers from the limitations of cloud based recognisers (network requirement and latency).

- **PocketSphinx**[138] is an open-source recogniser available under GPL and supports open dictation recognition as well as grammar based recognition. Its main disadvantage is its slow performance.

Parameter	Importance	Nuance Vocon	Nuance NDEV	Nuance Vocon Hybrid	Google	PocketSphinx
Generic Parameters						
Maturity & Stability	++	+++	+++	+++	+++	+++
Regularly Updated	+/-	++	++	++	++	++
Technical Up-to-Datedness / Appeal	+/-	++	++	++	++	++
Open Source	+/-	NO	NO	NO	NO	YES
Non-Infecting	+	NO	NO	NO	NO	YES
Code Quality	++	+++	+++	+++	+++	++
Extensibility	+/-	+	+	+	+	+++
Community	+/-	-	-	-	+++	++
Performance / Scalability	++	+++	+++	+++	+++	+
Reuse of existing developments	++	-	-	-	-	-
EU project origin	---	NO	NO	NO	NO	NO
Platform (Portability)	++	+++	+++	+++	+++	+
Open Standards Compliance	+	+	+	+	+	++
Interoperability (easy integration for all platforms)	++	++	++	++	++	+++
Specific Parameters						
Natural Language Extensibility (Multilingual Support)	+++	+++	+++	++	++	++
Quality	+++	+++	+++	+++	+++	+
Realtime/Speed (ASR / TTS)	+++	+++	-	++	+	--
Large Coverage (ASR)	++	++	+++	+++	+++	--
Cost (ASR / TTS)	++	-	-	-	+++	+++
N-best Results (ASR)	+	+++	+++	+++	++	+++
Partial Results (ASR)	+++	---	---	---	+++	+++

Figure 261: Parameter Evaluation of Technologies for Automatic Speech Recognition

3.14.5.2.1.3 Text To Speech

As for automatic speech recognizers, the market for text-to-speech (TTS) engines for mobile use is dominated by Nuance and Google. Another similarity is the availability of

[138] http://cmusphinx.sourceforge.net/

both embedded and cloud-based solutions. In contrast to ASRs, the issue of coverage has no significance, as all TTS products have full coverage within the respective languages.

- The TTS feature of **NDEV** from **Nuance** has similar characteristics and pricing options as the ASR feature described in Section 3.14.5.2.2.2.
- The TTS engine from **Google** is free to use within the Android platform.[139] Both embedded and networked voices are supported.
- **Acapela**[140] is a commercial TTS which can be downloaded and installed as a voice on the Android device. Once a voice has been installed, it can be selected for use with the built-in Android API and therefore uses the same technical solution as the Google TTS. The price is currently 3.99 € per language and device.
- **Acapela VAAS** offers Voice as a Service (VAAS)[141], a commercial cloud-based solution. In contrast to the device-installable voices, VAAS does not support the built-in Android API. Instead, transactions are managed over HTTP.
- **eSpeak**[142] is a free and open-source TTS which has been ported to Android. It is still in an early development phase. The built-in Android API is used when accessing the eSpeak voices.

Parameter	Importance	Nuance NDEV	Acapela	Google	Acapela VAAS	eSpeak
Generic Parameters						
Maturity & Stability	++	+++	+++	+++	+++	-
Regularly Updated	+/-	+	+	+	+	+
Technical Up-to-Datedness / Appeal	+/-	+	+	+	+	+
Open Source	+/-	NO	NO	NO	NO	YES
Non-Infecting	+	NO	NO	NO	NO	YES
Code Quality	++	++	++	+	++	+/-
Extensibility	+/-	+	+	+	+	+++
Community	+/-	-	++	+++	++	+
Performance / Scalability	++	+++	+++	++	+	++
Reuse of existing developments	++	-	-	-	-	-
EU project origin	---	NO	NO	NO	NO	NO
Platform (Portability)	++	+++	+++	+++	++	++
Open Standards Compliance	+	+	+	+	+	++
Interoperability (easy integration for all platforms)	++	++	++	++	+	++
Specific Parameters						
Natural Language Extensibility (Multilingual Support)	+++	+++	+++	+++	+++	+++
Quality	+++	+++	+++	++	+++	+
Realtime/Speed (ASR / TTS)	+++	++	++	++	+	+++
Cost	++	+++	+++	+++	++	++

[139] http://developer.android.com/reference/android/speech/SpeechRecognizer.html
[140] http://www.acapela-group.com/products/
[141] http://www.acapela-vaas.com/
[142] http://espeak.sourceforge.net/

(ASR / TTS)						

Figure 262: Parameter Evaluation of Technologies for Text To Speech

3.14.5.2.2 Technology Selection

3.14.5.2.2.1 Dialogue Manager

TDM is a Dialogue Manager that emphasises natural dialogue, topic switching, goal accommodation and topic grounding.

These features will all be extensively used within SAM. The End Users will have to be able to complete different tasks without needing to switch menus following their hierarchical organisation. Also, they will need occasionally to know that the system understood their input.

TDM also allows easy integration with graphical interfaces something that will be implemented in SAM.

3.14.5.2.2.2 Automatic Speech Recognition

As it can be seen in Figure 261, Google's native Android ASR is ostensibly more fitting for the purposes of the SAM project. It combines comparable performance with other high-end software (such as Nuance's products) with the advantage of being offered at no cost and being native to the mobile platform that will be used for SAM.

3.14.5.2.2.3 Text To Speech

As can be seen in Figure 262, Google's native Android TTS solution is ostensibly more fitting for the purposes of the SAM project along with Acapela's locally running software. This was expected since both solutions use Android's TTS engine. They combine comparable performance to other high-end software (such as Nuance's product) with the advantage of being offered free of charge. However, Google's offering has the advantage of working out of the box and having a greater community to seek help from, so it will be preferred for the purposes of SAM. Additionally, it can be used to evaluate the potential quality gain of using Acapela in the future.

3.14.5.3 Technical Component Specification

3.14.5.3.1 Structure

Figure 263 provides an overview of Voice Dialogue which can be found also in D3.2.1 Global Architecture Definition. Additionally, the version of the overview in this chapter has been augmented with the selected technologies for each subcomponent.

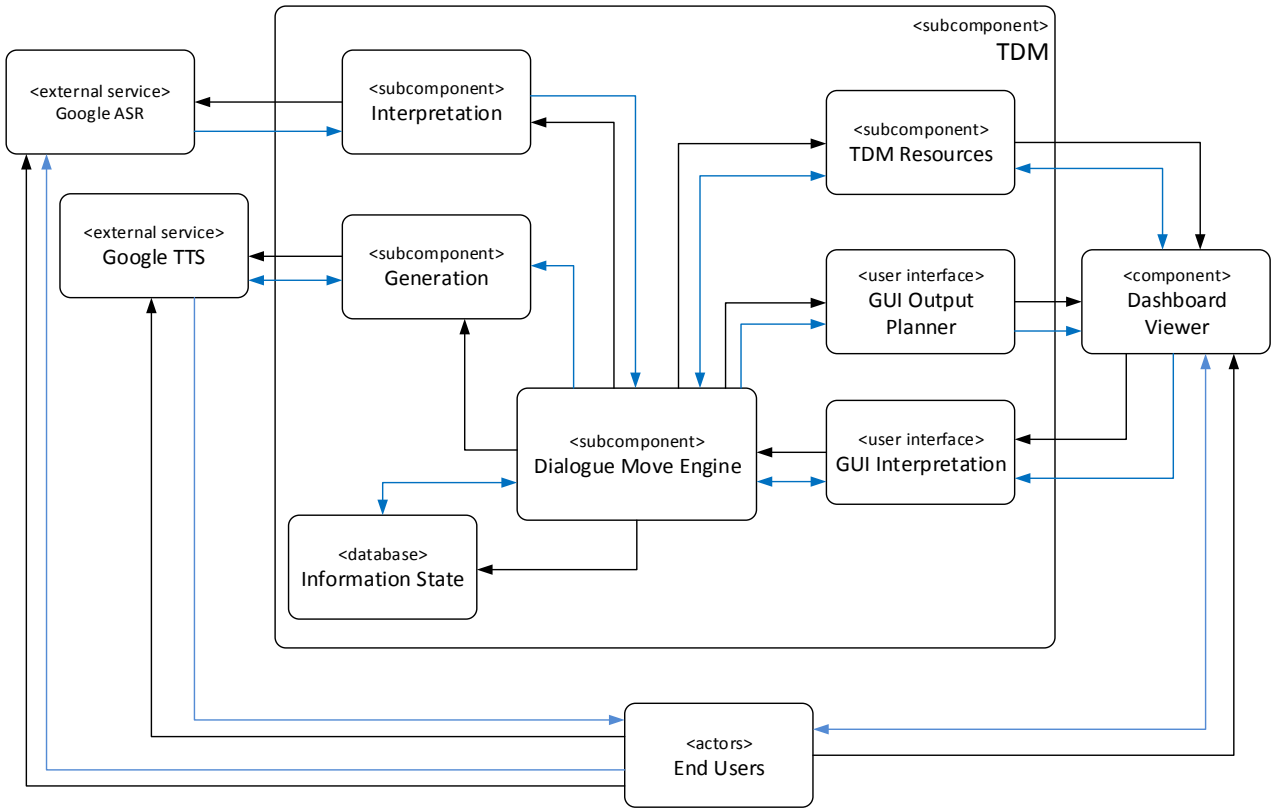

Figure 263: Overview with Selected Technologies – Voice Dialogue

3.14.5.4 Specification of Interfaces, Protocols and Formats

The services of Voice Dialogue will be provided through a Java interface.

3.14.5.4.1 Java Interfaces

In order for TDM to be able to communicate with the Dashboard Viewer, a Java API needs to function as a connector between the two communicating parts. The Dashboard Viewer will call the right method from the API and the API will trigger an event which will be handled by a WebSocket server which connects TDM and the API.

3.14.5.4.1.1 Method “PTT Activated”

This method is invoked when the push-to-talk (PTT) icon in the GUI is clicked. The notification will then be forwarded to the ASR so that the system starts listening to the microphone.

PTT Activated	
Method Header	public void pttActivated()
Parameters	None
Return Value	The method has no return value. In the case of an error, an exception is raised.
Error Handling	If an error is encountered, e.g., the system is already in a listening state, the method throws an exception describing the nature of the error.

Remarks	None

Figure 264: Java Interface – PTT Activated

```
/**
* @throws PTTException
*/

public void pttActivated() throws VDException;

…

//Notifies TDM that the PTT has been activated
api.pttActivated();
```

Figure 265: Source Code Example – API Method Signature and Usage of PTT Activated

3.14.5.4.1.2 Method “Notify About Menu Transition”

In case that the user needs to interact via the GUI with something that the Voice Dialogue needs to know of, this method is called. This method will then forward the message to the backend, allowing the dialogue manager to update its state and respond appropriately.

Notify About Menu Transition	
Method Header	public void notifyAboutMenuTransition(Map parameters)
Parameters	parameters: Contains the transition target’s ID along with one or more parameter values
Return Value	The method has no return value. In the case of an error, an exception is raised.
Error Handling	If an error is encountered, e.g. in the case of an unexpected value, the method throws an exception describing the nature of the error.
Remarks	None

Figure 266: Java Interface – Get Message from GUI Method Specification

```
/**
* @throws TransitionException
*/

public void notifyAboutMenuTransition() throws TransitionException;

…

Map parameters = new HashMap();
parameters.put(“id”, “PostOption”);
parameters.put(“message”, “I really like this movie”);

//Notifies TDM about a menu Tranisition
api.notifyAboutMenuTransition ();
```

Figure 267: Source Code Example – API Method Signature and Usage of Get Message from GUI

3.14.5.5 Summary

In this section, Voice Dialogue was technically specified. During the technology selection (Section 3.14.5.2), TDM was selected as a Dialogue Manager and Android's native ASR and TTS engines as external services.

TDM was selected due to its unique offerings in dialogue management and its internal development by one of SAM's partners, Talkamatic. Android's ASR and TTS were chosen based on their out-of-the-box functionality and cost effectiveness.

There is one abstraction needed for the VD component which is a Java API for connection to TDM's backend.

4 Conclusions

This deliverable provides the Technical Specification of the SAM platform, which follows the Functional Specification deliverable (D3.2.2). It is fully aligned with the related project deliverables, specifically, the User Stories and Requirements Analysis deliverable (D2.3), the Global Architecture Definition deliverable (D3.2.1) and the Functional Specification deliverable (D3.2.2). This deliverable defines the detailed technical design of each of the 19 SAM components.

Each of the components is elaborated with respect to its major design decisions, technology comparison and selection, technical specification and specification of interfaces, protocols and formats. During the technology comparison process, generic and specific parameters were considered in order to select the suitable technology for the specific component. During this selection process, associated licensing issues for the individual components were also considered.

As defined in the Global Architecture Definition, for each component an overall technical characterisation has been provided, explaining its functionalities and implementation. The required interfaces and communication protocols between the components are detailed with respect to component specifications and internal sequence diagrams. The related API specification of the individual components is technically analysed and the data exchange between the components is defined with necessary content formats and protocols.

The provided technical specifications will act as a stable and reliable foundation for the upcoming software development tasks within SAM and its work packages WP4 – WP7. Nevertheless, naturally, during the course of the project, certain aspects of the software architecture may be changed for technical reasons and due to unforeseen issues.

References

[BBM09] E. Boldrini, A. Balahur, P. Martínez-Barco, and A. Montoyo, “EmotiBlog: an annotation scheme for emotion detection and analysis in non-traditional textual genres,” in Proceedings of the 5th International Conference on Data Mining (DMIN 2009), 2009.

[DCT11] K. Dentler, R. Cornet, A. Ten Teije, and N. De Keizer, “Comparison of reasoners for large ontologies in the OWL 2 EL profile,” Semantic Web, vol. 2. pp. 71–87, 2011.

[IDC14] IDC Corporate USA(2014). *Smartphone OS Market Share, Q2 2014* [Online]. Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp

[WSOCK] IETF RFC 6455. *The WebSocket protocol* [Online]. Available: https://tools.ietf.org/html/rfc6455