WP4 – Information Management & Interoperability System

D4.9.1: Information Management & Interoperability System Public Report

Deliverable Lead: ASC

Contributing Partners: ASC, TIE, UA, UoR

Delivery Date: 2015-05

Dissemination Level: Public

Final

This deliverable provides a description of the first prototypes implementation of tasks T4.1 Assets Storage and Information Management, T4.2 Communication and Federation, T4.3 Data Characterisation Services and T4.4 Distributed Identity, Trust and Security. The deliverable identifies the descriptions of the software deliverables in WP4. This document is a living document that will be enhanced with the further delivery of the different iterations of the WP4 prototypes.

Document Status	
Deliverable Lead	ASC
Internal Reviewer 1	Jonathan Jeurissen and Koen Cooreman, TPV
Internal Reviewer 2	Barry Smith and Jennifer Walker, BDS
Type	Deliverable
Work Package	WP4 – Information Management & Interoperability System
ID	D4.9.1: Information Management & Interoperability System Public Report
Due Date	04.2015
Delivery Date	05.2015
Status	For Approval

Document History	
Versions	V0.1: ASC, Create Document Structure V0.2: ASC, TIE, UA, UoR, Provide first contributions, merge and review V0.3: ASC, TIE, UA, UoR, Updated contributions, Ready for first review V0.4: TPV, BDS, First internal review V0.5: ASC, TIE, UA, UoR, Processing review amendments, Ready for second review V0.6: ASC, TIE, UA, UoR, Processing review amendments V0.7: ASC, TIE, UA, UoR, Ready for Approval
Contributions	ASC: Norman Wessel – Document creation, contributions to all sections TIE: Fran Rodriguez – Contribution to Section 4 UA: Yoan Guitérrez – Contribution to Section 5 David Tomás – Contribution to Section 5 UoR: Marco Tiemann – Contribution to Section 6

Disclaimer

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

This public deliverable document describes the results of the developments of WP4.. It describes the prototypes status and results of the different WP4 SAM Components without exposing confidential information such as prototypes access information and source code. This kind of information can be shown in the related Programme Participants Prototypes (D4.1.1, D4.2.1, D4.3.1 and D4.4.1).

The SAM team decided to deliver in M16 an additional deliverable version of it (D4.9.0) in order to correctly describe the developments and results of the first software deliverables, especially D4.4.1 that was due at that time. Therefore, this is a 2nd version of this deliverable (D4.9.1) which contains the progress of the different prototypes until M19.

This document will be iteratively updated at the point of the delivery of the related software prototypes of the WP4 SAM Components:

- Cloud Storage (T4.1 – Assets Storage and Information Management)
- Interconnection Bus (T4.2 – Communication and Federation)
- Semantic Services (T4.3 – Data Characterisation Services)
- Identity and Security Services (T4.4 – Distributed Identity, Trust and Security)

At the moment, the deliverable contains the following software prototype information:

Task	Component	Software Prototype Deliverable	Due	Section
T4.1	Cloud Storage	D4.1.1	M16	3
T4.2	Interconnection Bus	D4.2.1	M19	4
T4.3	Semantic Services	D4.3.1	M19	5
T4.4	Identity and Security Services	D4.4.1	M19	6

Figure 1: Overview of Tasks and Software Deliverables

Each component will be developed iteratively with three prototypes, except the Cloud Storage component, where only two prototypes are planned. For each prototype iteration, each component will produce a PP software deliverable and an update of this public documentation (D4.9.x).

For each of the SAM Component prototypes, the following subsections are presented in this document:

- Scope and Relationship
- Requirements and Preparations
- Installation
- Execution and Usage
- Limitations and Further Developments
- Research Background
- Target Performance
- Summary

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

The current generation of Internet-connected devices has changed the way users interact with media. Previously, users were restricted to being passive and unidirectional consumers; now, they are proactive and interactive media users. They can comment on and rate a television show or film and search for related information regarding cast and crew, facts and trivia or even filming locations. They do this with both friends and wider social communities through the so-called “2nd Screen”.

Another related phenomenon is “Content Syndication”, which is a field of marketing where digital content is created once and delivered to consumers through various different marketing channels (devices, markets and stakeholders) simultaneously, enabling efficient content control, delivery and feedback.

However, the 2nd Screen phenomenon has grown in a disorderly manner. Tools supplied by the media provider companies (e.g. as mobile or tablet apps) limit the potential outreach and, as a result, users are not enjoying relevant contextual syndicated information. European enterprises wishing to provide services have limited methods of receiving feedback, restricting the business intelligence that can be extracted and applied in order to profit from and enrich this growing market.

SAM is reshaping the current disorganised 2nd Screen ecosystem 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 means of characterising, discovering and syndicating media assets interactively. Users will be able to consume and prosume digital assets from different syndicated sources and synchronised devices (e.g. connected televisions), creating more fulfilling experiences around the original media assets.

The SAM vision that is now becoming reality, sees the former, out-dated system of users searching for the information they desire replaced with a new approach where information reaches users on their 2nd Screen using content syndication. This is enriched through the creation of dynamic social communities related to the user and digital asset context (e.g. profiles, preferences and devices connected). These are continuously evolving social spaces 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 virtual community as well as enhance personalised knowledge and satisfaction.

1.2 Deliverable Purpose, Scope and Context

The purpose of this deliverable is to accompany the software prototypes of WP4 tasks T4.1 Assets Storage and Information Management, T4.2 Communication and Federation, T4.3 Data Characterisation Services and T4.4 Distributed Identity, Trust and Security. Each task will contribute different components to the SAM architecture. Each component is

developed iteratively in three phases as per milestones 3/4/5 at M19/25/31 (besides T4.1, which is only developed in two phases at M16/M25) and will produce a software deliverable and an update of this public documentation (D4.9.x).

As the main focus of the tasks is the development of the software itself, this accompanying document focuses on providing a short summary of the main functionalities and on serving as a user guide for the current status of the development.

1.3 Document Status and Target Audience

This document is the second iteration of the D4.9.x series and a successor of the deliverable D4.9.0, which has provided information on Section 3 only. This deliverable, D4.9.1, is listed in the DOW as public, since it presents status information related to the prototypes of the software components in WP4 tasks to the interested public.

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 at http://wiki.socialisingaroundmedia.com/index.php/Glossary.

1.5 Document Structure

This deliverable is broken down into the following sections:

- **Section 1 (Introduction):** Provides an overview of the entire document and the related pilot implementation, describing the objectives, constraints and status
- **Section 2 (WP4 Introduction):** Outlines WP4 goals and SAM Components
- **Section 3 (Assets Storage and Information Management):** Describes the latest software deliverable developed in T4.1
- **Section 4 (Communication and Federation):** Describes the latest software deliverable developed in T4.2
- **Section 5 (Data Characterisation Services):** Describes the latest software deliverable developed in T4.3
- **Section 6 (Distributed Identity, Trust and Security):** Describes the latest software deliverable developed in T4.4
- **Section 7 (Document Summary):** Briefly summarises the work presented at the deliverable, as well as the overall WP4 status
- **Annexes**
  - **Annex A:** Provides additional information on the usage of the Cloud Storage RESTful interfaces regarding different data formats
  - **Annex B:** Provides additional information on the usage of the Interconnection Bus user interface

In Sections 3 to 6 above, for each component in the SAM Architecture, the following subsections are provided:

- **Scope and Relationship:** Describes the scope of the component implementation, its purpose and the main relationships with other components implemented in SAM in the first year
- **Requirements and Preparations:** Introduces the information needed to deal with the prototype in terms of technical and non-technical requirements, software to be installed, etc.

- **Installation:** Describes the steps needed to install the software, and how to build it from source code
- **Execution and Usage**: Presents the different screens and actions implemented at the prototype itself, how to access it, and how to test the different implemented options
- **Limitations and Further Developments:** Depicts the current prototype limitations and the expected improvements
- **Research Background:** Provides papers and other scientific information considered
- **Target Performance:** Provides Key Performance Indicators (KPI) for the SAM component
- **Summary:** Describes the conclusions of the implementation of the first prototype

1.6 External Annexes and Supporting Documents

- D4.1.1 - Asset Storage and Information Management (First Prototype)
- D4.2.1 - Communication and Federation (First Prototype)
- D4.3.1 - Data Characterisation Services (First Prototype)
- D4.4.1 - Distributed Identity Trust and Security (First Prototype)

2 WP4 Introduction

WP4 defines common services and infrastructure to be used by the rest of the SAM components; this includes the information management and the interoperability system. The interoperability system is needed to support federated instances of SAM, so that users and companies can discover, purchase and consume or compose assets from different instances. The information management is needed to store and distribute assets. Distributed Identity, Trust and Security aspects will also be developed in this WP.

Specific objectives of this WP include:

- To provide the foundation and tools for data assets storage (T4.1)
- To create a SAM internal communication system for exchanging data between different components (T4.2)
- To define pluggable service interfaces for the different components (T4.2)
- To realise a management system for semantic resources, common characterisation and semantic inference (T4.3)
- To create a framework for trust, security, federated identity and multi-device association and identification (T4.4)

The result of WP4 will be a set of independent infrastructure building block components that will form a foundation of functionality for the more practical tasks of WP5-7. Each component is developed iteratively in three phases as per milestones 3/4/5 at M19/25/31 (besides T4.1, which is only developed in two phases at M16/M25) and will produce a software deliverable and an update of this public documentation (D4.9.x – this document series).

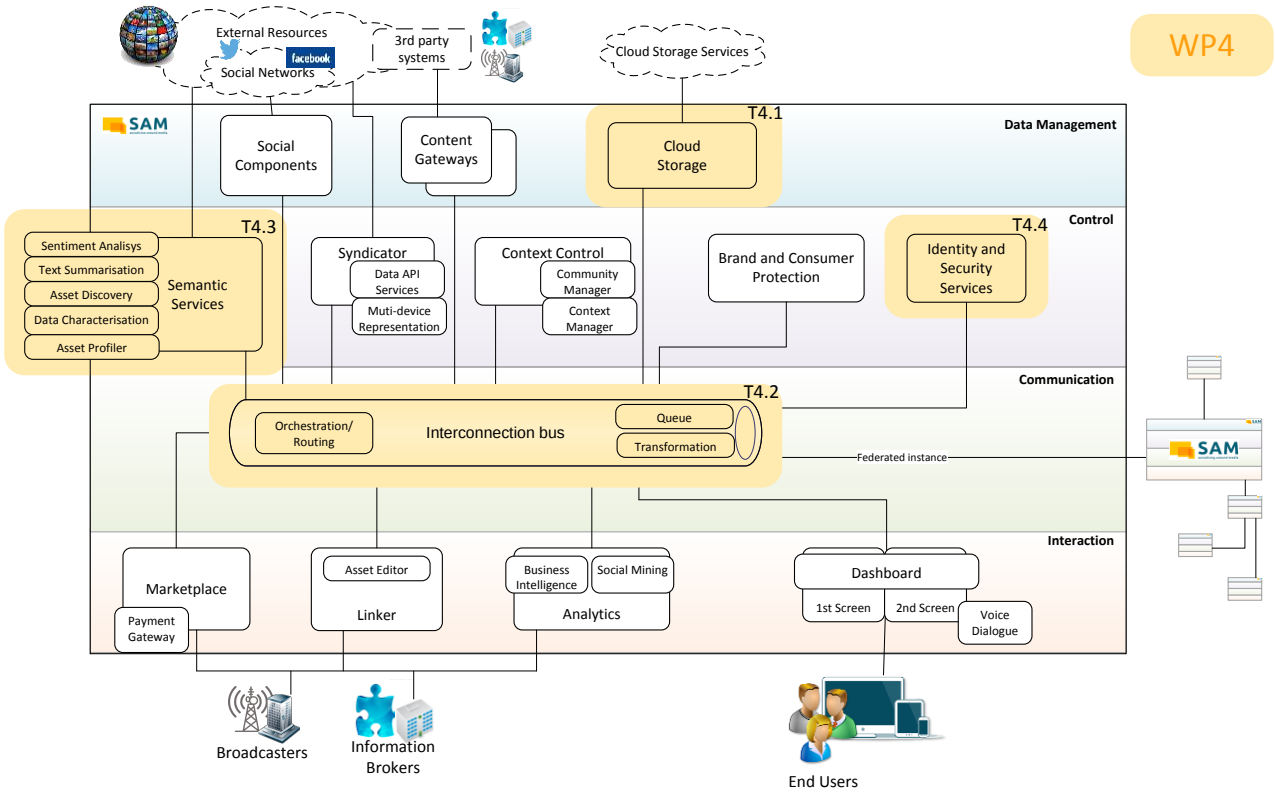

Figure 2: WP4 Components Contribution to SAM Architecture

The work in WP4 (as well as in the other development WPs) is managed by using the Agile Scrum methodology. For that purpose, a specific WP4 Scrum dashboard has been created in the SAM Jira task management system, where a representative of the WP Lead (ASC) is in charge of managing the dashboard, asking as Scrum Master.

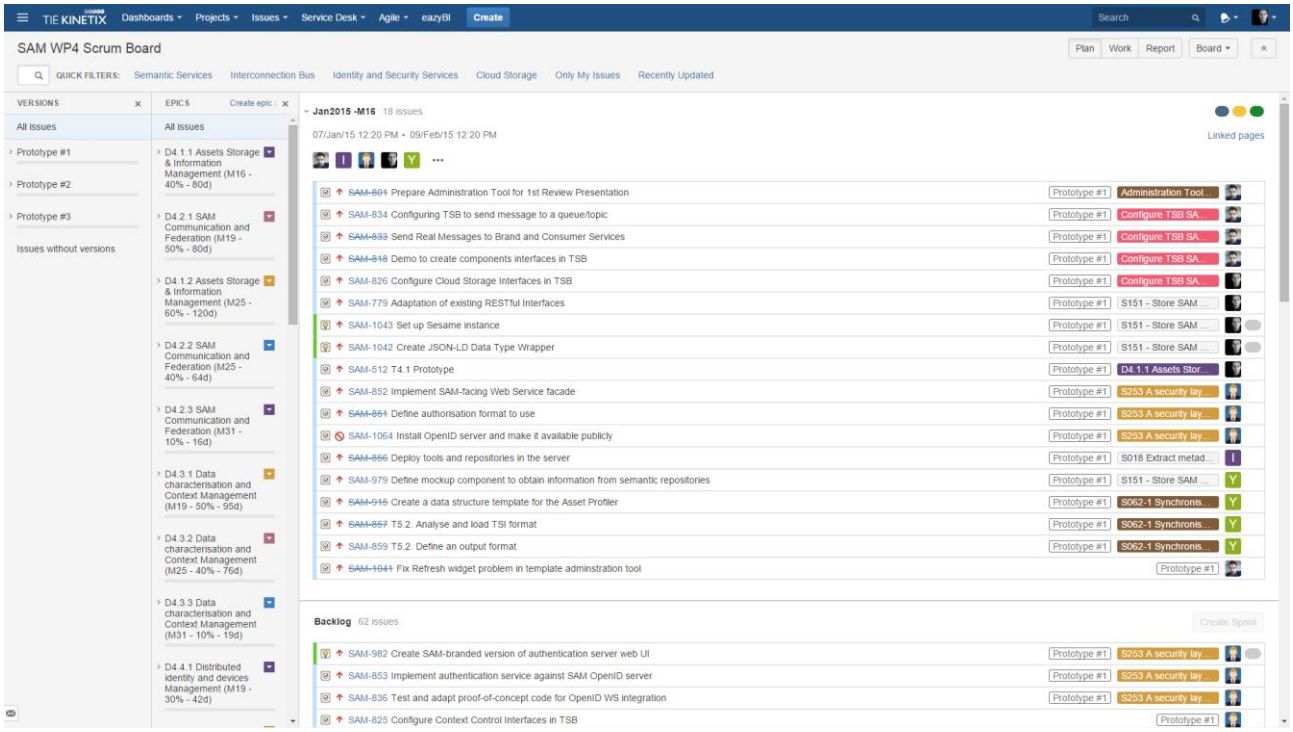

Figure 3: WP4 Scrum Dashboard

Sprints are planned monthly, and every story or task is linked to one or more specific requirement as expressed in D2.3 (User Stories and Requirements).

A planning meeting is scheduled at the beginning of each sprint in order to plan the next monthly sprint and discuss the priorities or reschedule the work not finished in the previous one. A review meeting is also scheduled at the end of each sprint in order to discuss the work done during the sprint and to find ways of improving (if necessary) the way of working.

3 Assets Storage and Information Management

This section describes the software deliverable D4.1.1, which is the first prototype release of the SAM storage and information management functionalities.

3.1 Scope and Relationship

The Cloud Storage component is responsible for storing assets and internal data used by other components of the SAM platform. Figure 4 shows the different subcomponents of the Cloud Storage, the logical connections that have been established between them, and the relationships with other components and actors in the SAM platform.

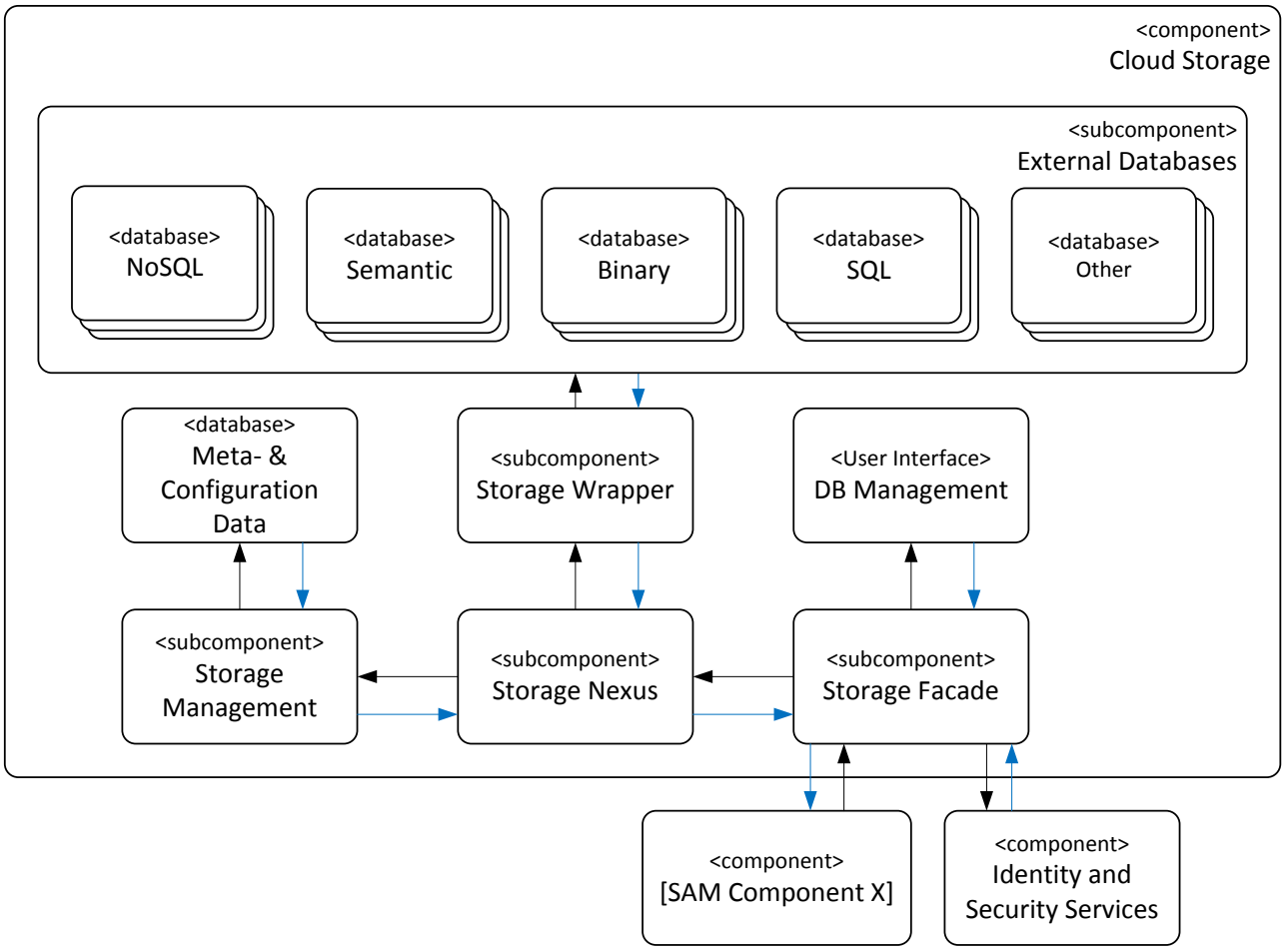

Figure 4: Overview of Cloud Storage Subcomponents

For further description of the functional and technical foundations of these subcomponents, please revisit documents D3.2.1 Section 4.7 (Architecture), D3.2.2 Section 4.8 (Functional Specification) or D3.3.1 Section 3.8 (Technical Specification).

The first prototype of T4.1 provides the basic functionalities of the Cloud Storage for semantic, semi-structured and binary data. Other components can access the Cloud Storage to perform CRUD (Create, Read, Update and Delete) operations and create/delete Buckets or modify a Bucket's access rights. The goal of this first approach is to cover 40% of the requirements of the component (see Section B 1.3.3.4 of the DoW for additional information on the effort distribution for this component in the lifespan of the project).

A summary of the tasks carried out for each subcomponent of the first version of the prototype is shown in the following table:

Subcomponent	Task
Storage Facade	Hosts RESTful interfaces for CRUD operations (MongoDB, Sesame, Amazon S3), create/delete Buckets, manage access rights for Buckets
Storage Nexus	Controls the processes for data management as well as the management of the Cloud Storage itself
Storage Wrapper	Implement Database Wrapper and Database Type Wrapper to manage the following databases/services: MongoDB, Sesame and Amazon S3
Storage Management	Encapsulate the logic for all configurations of the Cloud Storage and connected external databases
External Databases	Provide the databases instances for semantic, semi-structured and binary data

Figure 5: Tasks carried out for the First Prototype of T4.1

3.2 Requirements and Preparations

This section provides information on technical and non-technical requirements for administrators as well as software prerequisites for the installation of the Cloud Storage component.

In order to deploy the server side of the Cloud Storage, it is necessary to have an Apache Felix[1] package installed. It can be downloaded here.

For execution, the Apache Felix package has to be extracted and a Java Runtime[2] must be installed.

As the Apache Felix Runtime Environment is just a plain Java package, it does not require a special installation, but it can be configured later to run as a background service or to start automatically with the operating system. The user then needs proper start scripts and/or a Java service wrapper. As an example the Java Service Wrapper can be downloaded here. The documentation for the installation of this wrapper can be found here.

In this document, it is presumed that Apache Felix 4.4.1 is successfully extracted and can be started on the developer's computer. An installation as a service is not needed.

It's also necessary that the operating system hosting the server side must have the TCP networking port 8182 unused, as this is the port that will be used by the Cloud Storage. On Windows you can check this by using the Command Prompt and the 'netstat' command.

This prototype has been tested in Windows 7 and Ubuntu 14.04 Operating System with Oracle Java SE Development Kit 7[2], even though it is usable under all major Java platforms running in Windows or Linux.

3.3 Installation (Deployment)

This section provides guidelines on how to install and deploy the prototype of the Cloud Storage component.

[1] http://felix.apache.org
[2] http://www.java.com

Currently the installation is made by the Jenkins Continuous Integration Server[3] provided by the SAM consortium. For the Cloud Storage component, a Jenkins integration project has been created and configured to build and deploy the component in Apache Felix.

For the installation, the Cloud Storage files (provided in Java Archives (JAR) format) and their dependencies have to be copied into the bundles folder in the Felix installation. If the Apache Felix server is already running, it has to be restarted to load and start the Cloud Storage bundles.

Additionally a MongoDB instance has to be set up and configured. The MongoDB instance can be installed using a Windows installer or other scripts, depending on your operating system. After the installation using the default values the administrator has to configure the MongoDB conducting the following steps:

- Start the MongoDB instance (see Figure 6)
- Start the MongoDB configuration console (see Figure 7)
- Enter the different commands to configure the MongoDB instance (see Figure 8)

```
$MongoDB-Folder\bin\mongod.exe" --dbpath "$MongoDB-Folder/bin/data/db
```

Figure 6: Command to Start the MongoDB Instance[4]

```
$MongoDB-Folder\bin\mongo.exe
```

Figure 7: Command to Start the MongoDB Configuration Console[4]

```
use cloudstorage
db.createUser({user: "sam",pwd: "sam",roles: [{ role: "readWrite", db: "cloudstorage"
},]})
use cloudstorage-buckets
db.createUser({user: "sam",pwd: "sam",roles: [{ role: "readWrite", db: "cloudstorage"
},]})
```

Figure 8: Commands to Set Up the MongoDB Configuration

Upon the start-up, the Cloud Storage packages are loaded. To check the start of all bundles, the command in Figure 9 shows a list of all deployed bundles in the Felix environment. In Figure 10 a successful example output is shown. All Cloud Storage bundles should be in the state “Active”.

```
lb
```

Figure 9: Command to Show all Deployed Bundles in Apache Felix

[3] http://jenkins-ci.org/
[4] $MongoDB-Folder is a place holder for your local MongoDB path.

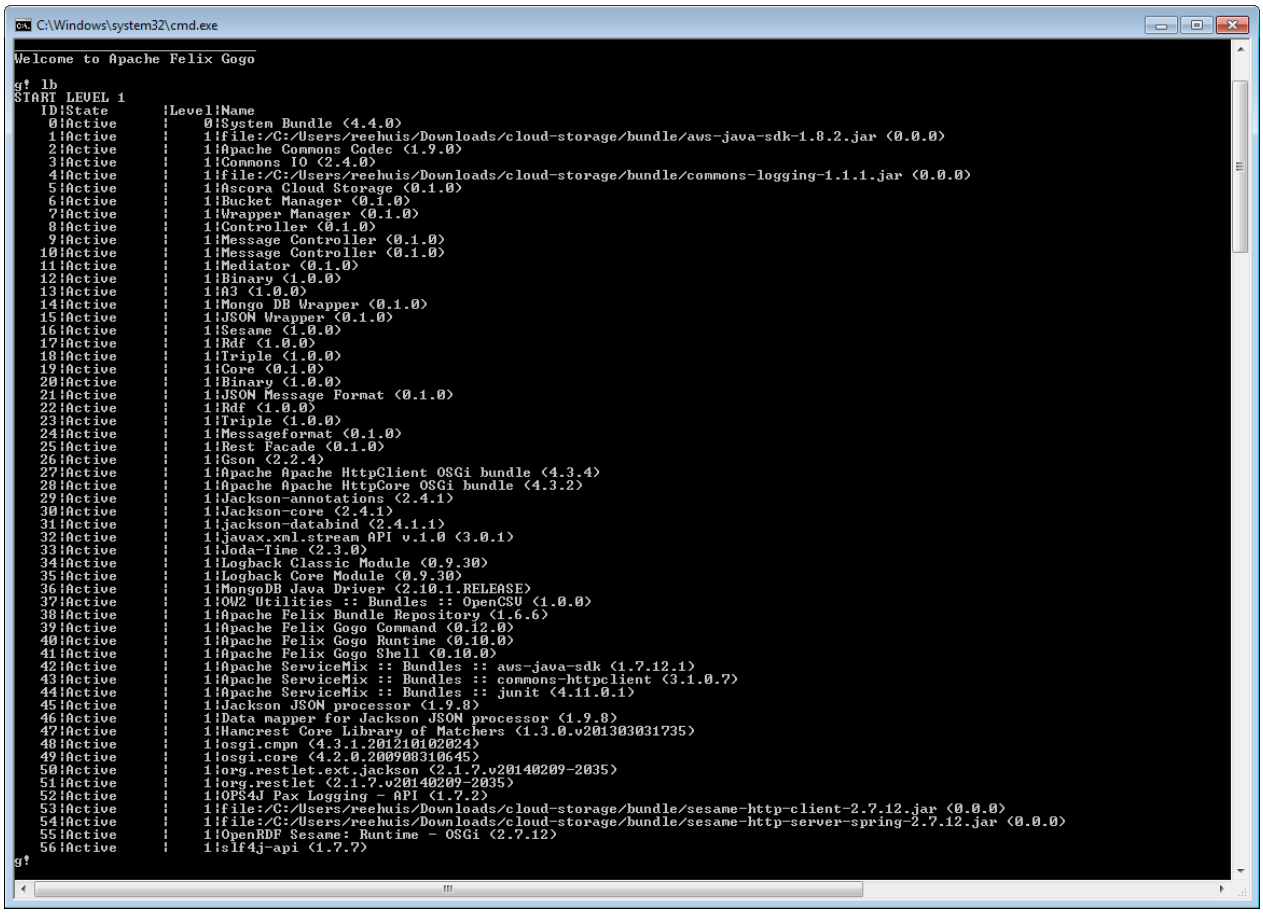

Figure 10: Output of all Deployed Bundles in the Apache Felix Server

The Cloud Storage installation does not include the configuration file providing the credentials for the binary Amazon storage, as they should not be available to the public. For tests of the binary storage of the Cloud Storage, the AwsCredentials.properties file has to be created in the Apache Felix folder. This file then has to be filled with Amazon S3 credentials like shown in Figure 11.

```
secretKey=<yourSecretKey>
accessKey=<yourAccessKey>
```

Figure 11: AwsCredentials.properties Content Template

3.4 Execution and Usage

This section describes how to use the different subcomponents of the prototype.

3.4.1 Server Side (Administrators)

In order to execute the server side of the Cloud Storage component, the Apache Felix package must be started, by executing the command in Figure 12.

```
java –jar [Apache Felix Directory]/bin/felix.jar
```

Figure 12: Apache Felix Start Command

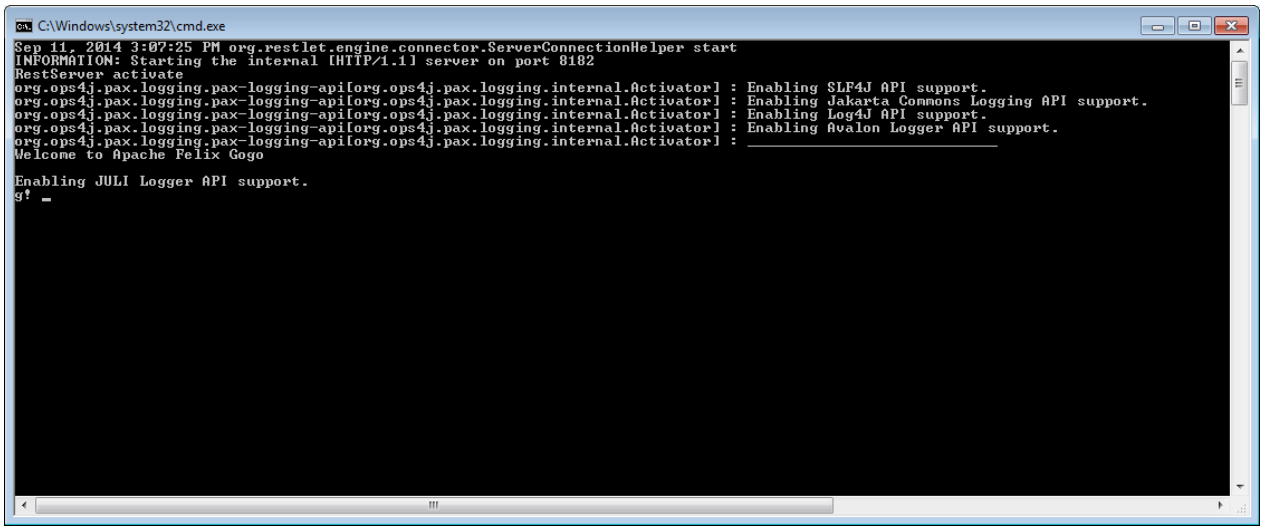

Figure 13: Apache Felix Window with Start-up Logs

Once the Apache Felix package is running, a new window appears containing all log messages during the start-up process and running the server.

The server side uses RESTful interfaces to provide the Cloud Storage functionality to developers. For all available RESTful interfaces including their HTTP Operations, expected input parameters and given output results, please refer to Section 3.4.2.1 in this document.

In order to test the GET method and the availability of the Cloud Storage, a browser can be used with the following URL:

```
http://localhost:8182/api/cs/status
```

The generated result shows a JSON response with some status information of the Cloud Storage.

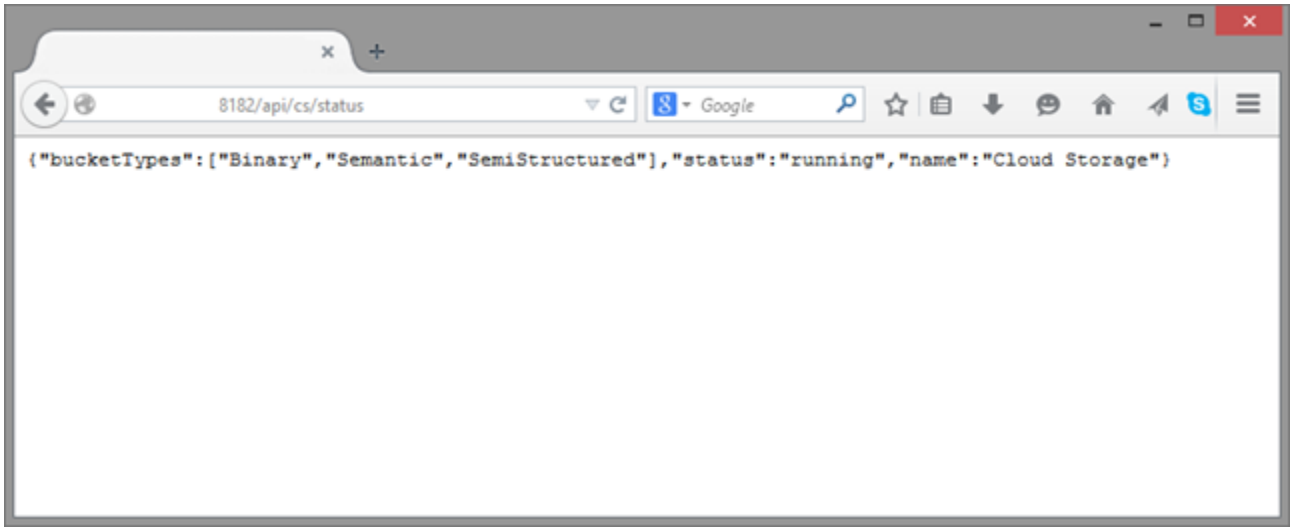

Figure 14: Response of the RESTful Status Interface

As a result of this request, a set of logs are generated displaying the activity on the server side.

The following Figure 15 shows the logs generated by the activity done by the server side, and also displays the Status component activity log, executing the content sent by the server side.

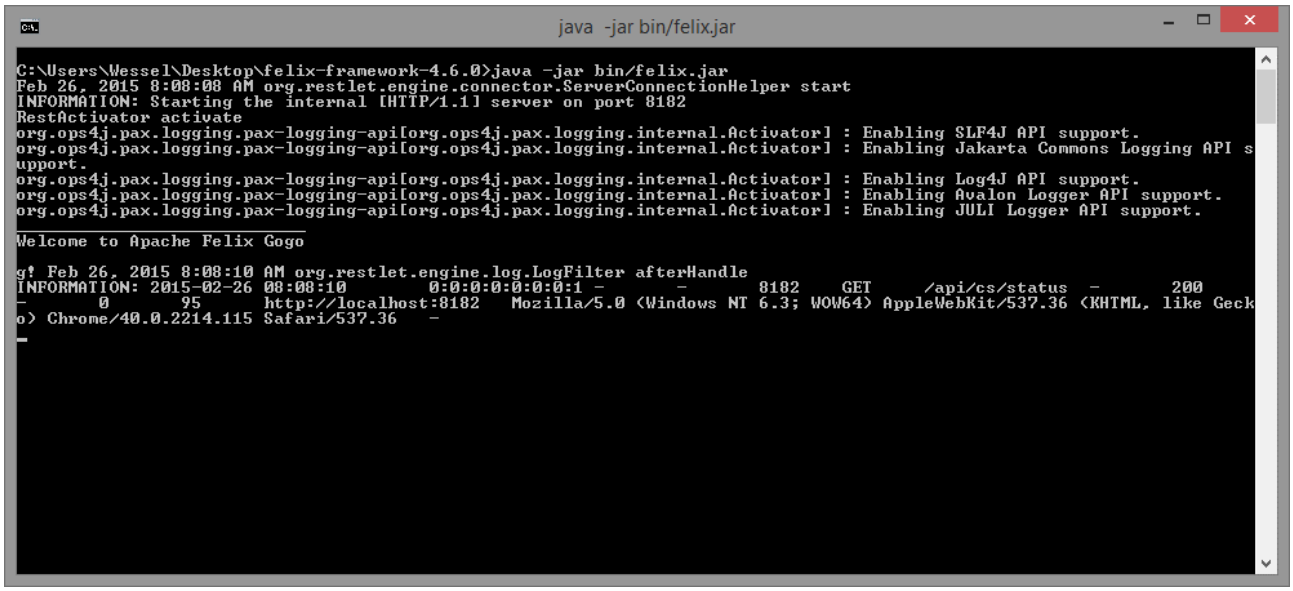

Figure 15: Server Side and Data Template Activity Log

3.4.2 Client Side (Developers)

The following sections contain information about the RESTful interfaces provided by the component. They can be used by SAM component owners to access the Cloud Storage and the stored content.

3.4.2.1 RESTful Interfaces

This section explains how to use the Cloud Storage component with the provided RESTful interfaces. In this document, it is assumed, that the general use of RESTful interfaces is known to the developer. For each interface a description and an example are given. Each description is presented as a table, which includes all necessary request and response parameters and their expected value types and values.

Section 3.4.2.2 details the expected input and output of JSON objects for each RESTful interface call.

In general, the BucketId can be any string containing letters and numbers. This BucketId ID is used to access an owner's bucket. If the developer wants to access a Bucket of another user, he/she has to append the userId of this user with an '@' to the BucketId.

```
:bucketId = myTestBucket
```

Figure 16: Example BucketId for Accessing Bucket Owned by the User

```
:bucketId = hisTestBucket@otherUser
```

Figure 17: Example BucketId for Accessing a Bucket Owned by another User

If communicating directly with the Cloud Storage component (bypassing the Interconnection Bus component) please be sure to provide the header "content-Type" with the value "application/json".

3.4.2.1.1 Bucket Interfaces

Create Bucket			
Description	Creates a new Bucket for the user		
Request			
Request URL	**PUT** http://localhost:8182/api/cs/bucket		
JSON Attributes	Name	Required	Possible Values
	bucketId	yes	anystring
	bucketType	yes	SemiStructured Binary Semantic
	accessRights	no	A list of Access Right Data Transfer Object (see Section 3.4.2.2)
Response			
HTTP Status Code	Value		Description
	200		Bucket created
	400		Bad request
	404		Bucket not found
	409		Bucket exists already
	415		Unsupported media type
	502		Database error

Figure 18: RESTful Interface Description for Creating a Bucket

```
{
   "bucketId": "testId",
   "bucketType": "SemiStructured",
   "accessRights": [
      {
         "userId": "user1",
         "right": "Read"
      }
   ]
}
```

Figure 19: Example Request for Creating 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://localhost:8182/api/cs/bucket/:bucketId		
Resource	Name	Required	Possible Values

Parameters	bucketId	yes	anystring
Response			
HTTP Status Code	Value		Description
	200		Bucket deleted
	400		Bad request
	401		Insufficient rights
	404		Bucket not found
	415		Unsupported media type

Figure 20: RESTful Interface Description for Deleting a Bucket

3.4.2.1.2 Access Right Interfaces

The following tables provide the future RESTful interfaces which enable the access right management of Buckets. They will be provided in the second prototype.

Add Access Right to Bucket			
Description	Modifies the access list of a Bucket by adding a specific access right to a specific user.		
Request			
Request URL	**PUT** http://localhost:8182/api/cs/bucket/:bucketId/acl		
Resource Parameters	Name	Required	Possible Values
	bucketId	yes	anystring
JSON Attributes	userId	yes	anystring
	right	yes	Denied Read Write Super
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	
	415	Unsupported media type	

Figure 21: RESTful Interface Description for Adding an AccessRight

```
{
   "userId": "testUser",
   "right": "Read"
}
```

Figure 22: Example Request for Adding an Access Right

Remove Access Right from Bucket			
Description	Modifies the access list 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	**POST** http://localhost:8182/api/cs/bucket/:bucketId/acl		
Resource Parameters	Name	Required	Possible Values
	bucketId	yes	anystring
JSON Attributes	userId	yes	anystring
	right	yes	Denied Read Write Super
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	
	415	Unsupported media type	

Figure 23: RESTful Interface Description for Removing an Access Right

```
{
   "userId": "testUser",
   "right": "Read"
}
```

Figure 24: Example Request for Removing an Access Right

3.4.2.1.3 Bucket Data Interfaces

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	**PUT** http://localhost:8182/api/cs/bucket/:bucketId/create		
Resource Parameters	Name	Required	Possible Values
	bucketId	yes	anystring
HTTP Parameters	object	yes	a Data Transfer

			Object in JSON
Response			
HTTP Status Code	Value	Description	
	201	Object created	
	400	Bad request	
	401	Insufficient rights	
	404	Bucket not found	
	409	Object with specific ID already exists in the Bucket	
	415	Unsupported media type	

Figure 25: RESTful Interface Description for Creating a Data Object

```
{
   "id": 0,
   "color": "white"
}
```

Figure 26: Example Request for Creating a Data Object

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 are based on the data type.		
Request			
Request URL	**POST** http://localhost:8182/api/cs/bucket/:bucketId/read		
Resource Parameters	Name	Required	Possible Values
	bucketId	yes	anystring
HTTP Parameters	query	yes	a QO in JSON (see Section 3.4.2.3)
Response			
HTTP Status Code	Value	Description	
	200	OK	
	400	Bad request	
	401	Insufficient rights	
	404	Bucket not found	
	415	Unsupported media type	
JSON Attributes	Name	Description	Return Value
	results	yes	Array of generic JSON objects

Figure 27: RESTful Interface Description for Reading a Data Object

```
{
   "id": {"$gt": -1} // Get all objects with id greater than -1
}
```

Figure 28: Example Request for Reading a Data Object

Update Data Object in Bucket			
Description	Executes an update operation on an existing Bucket. This will update all data objects found with the query object in the Bucket with the provided values in the parameter.		
Request			
Request URL	**PUT** http://localhost:8182/api/cs/bucket/:bucketId/update		
Resource Parameters	Name	Required	Possible Values
	bucketId	yes	anystring
HTTP Parameters	query	yes	a QO in JSON
	object	yes	a DTO in JSON
Response			
HTTP Status Code	Value	Description	
	200	Found objects updated	
	204	Object not found	
	400	Bad request	
	401	Insufficient rights	
	404	Bucket not found	
	415	Unsupported media type	

Figure 29: RESTful Interface Description for Updating a Data Object

```
{
   "query": {
      "id": 0
   },
   "object": {
      "id": 0,
      "color": "yellow"
   }
}
```

Figure 30: Example Request for Updating a Data Object

Deleting Data Object in 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 are based on the data type.		
Request			
Request URL	**POST** http://localhost:8182/api/cs/bucket/:bucketId/delete		
Resource Parameters	Name	Required	Possible Values
	bucketId	yes	Anystring
HTTP Parameters	query	yes	a QO in JSON
Response			
HTTP Status Code	Value	Description	
	200	Object deleted	
	204	Object not found	
	400	Bad request	
	401	Insufficient rights	
	404	Bucket not found	
	415	Unsupported media type	

Figure 31: RESTful Interface Description for Deleting Matching Data

```
{ "id": 0 }
```

Figure 32: Example Request for Deleting a Data Object

3.4.2.2 Data Transfer Object JSON Schema

Each of the following schemas are defining Data Transfer Objects (DTO), which can be used as parameters for RESTful interfaces specified in the previous section.

```
{
   "type": "object",
   "id": "http://localhost:8182/api/cs/JSON-Schema/UpdateObject",
   "properties": {
      "query": {
         "type": "object",
         "required": true
      },
      "object": {
         "type": "object",
         "required": true
      }
   }
}
```

Figure 33: JSON Schema – UpdateObject Description

```
{
   "type": "object",
   "id": " http://localhost:8182/api/cs/JSON-Schema/BinaryObject",
   "properties": {
      "binaryKey": {
         "type": "string",
         "required": true
      },
      "binaryData": {
         "type": "string",
         "required": false
      }
   }
}
```

Figure 34: JSON Schema – BinaryObject Description

```
{
   "type": "object",
   "id": "http://localhost:8182/api/cs/JSON-Schema/RDFObject",
   "properties": {
      "namespace": {
         "type": "string",
         "required": false
      },
      "rdfXmlData": {
         "type": "string",
         "required": false
      }
   }
}
```

Figure 35: JSON Schema – RDFObject Description

```
{
  "namespace": {
    "type": "string",
    "required": true
  },
  "jsonLdData": {
    "type": "string",
    "required": true
  }
}
```

Figure 36: JSON Schema – JsonLdObject Description

```
{
   "type": "object",
   "id": "http://localhost:8182/api/cs/JSON-Schema/AccessRight",
   "properties": {
      "id": {
         "type": "string",
         "required": true
      },
      "right": {
         "type": {
            "enum": [
               "Denied",
               "Read",
               "Write",
               "Super"
            ]
         },
         "required": true
      }
   }
}
```

Figure 37: JSON Schema – Access Right Object Description

```
{
   "type": "object",
   "id": "http://localhost:8182/api/cs/JSON-Schema/Bucket",
   "properties": {
      "accessRights": {
         "type": "array",
         "required": false,
         "items": {
            "type": "object",
            "id": "http://localhost:8182/api/cs/JSON-Schema/accessRight",
            "required": false,
            "properties": {
               "id": {
                  "type": "string",
                  "required": true
               },
               "right": {
                  "type": {
                     "enum": [
                        "Denied",
                        "Read",
                        "Write",
                        "Super"
                     ]
                  },
                  "required": true
               }
            }
         }
      },
      "bucketId": {
         "type": "string",
         "required": true
      },
      "bucketType": {
         "type": {
            "enum": [
               "Structured",
               "SemiStructured",
               "Semantic"
            ]
         },
         "required": true
      }
   }
```

Figure 38: JSON Schema – Bucket Object Description

3.4.2.3 Query Object JSON Schema

```
{
   "type": "object",
   "id": "http://localhost:8182/api/cs/JSON-Schema/BinaryQueryObject",
   "properties": {
      "binaryKey": {
         "type": "string",
         "required": true
      }
   }
}
```

Figure 39: JSON Schema – BinaryQueryObject Description

```
{
   "binaryKey": "file-1"
}
```

Figure 40: JSON Schema – BinaryQueryObject Example

```
{
   "type": "object",
   "id": "http://localhost:8182/api/cs/JSON-Schema/SemanticQueryObject",
   "properties": {
      "namespace": {
         "type": "string",
         "required": false
      },
      "query": {
         "type": "string",
         "required": false
      },
      "queryType": {
         "type": {
            "enum": [
               "namespace",
               "sparql"
            ]
         },
         "required": true
      }
   }
}
```

Figure 41: JSON Schema – SemanticQueryObject Description

```
{
   "query": "CONSTRUCT { ?s ?p ?o } WHERE {?s ?p ?o }"
   "queryType": "sparql"
}
```

Figure 42: JSON Schema – SemanticQueryObject Example

The **SemiStructeredQueryObject** (SQO) can be any object following the Mongo Query language.

3.5 Limitations and Further Developments

The support of relational databases (MySQL[5] as specified in the deliverable D3.3.1 Technical Specification in Section 3.8.2.2.3) and the appropriate wrapper implementation will follow in the second prototype. This functionality has been postponed due to the need of the support of JSON-LD data format, which contains semantic data, which will be saved in a Sesame[6] instance.

The RESTful interface for the update operation regarding JSON-LD has not yet been implemented.

Another limitation of the prototype is the connection with the Identity and Security Services component for data authorisation, since this component is still under development. The first prototype of T4.4 Distributed Identity, Trust and Security should be ready in M19 as specified in the DoW. For this reason, the integrated access control list feature will be used until the Identity and Security Services are available.

Also the provided interfaces still need to be integrated into the Interconnection Bus component, which was not available when the first prototype was completed.

The management web interface of the Cloud Storage will be provided in the next prototype version (see Section 3.5.1.2).

3.5.1 Prototype 2 Planned Tasks

Regarding the next steps in the Cloud Storage component, Figure 43 and the following subsections summarise the tasks planned for the second prototype (to be delivered in M25).

Subcomponent	Task
Storage Facade	Implement RESTful interfaces for CRUD operations on relational and graph data and implement missing update operation for JSON-LD data
Storage Wrapper	Implement Database Wrapper and Database Type Wrapper to support MySQL as a relational database and neo4j as a graph database
Storage Management	Support MySQL and neo4j database configurations
External Databases	Add support for MySQL and neo4j databases, providing required Data Wrapper and Data Type Wrapper

Figure 43: Tasks Planned for the Second Prototype

3.5.1.1 Additional Database Support

As specified in the deliverable D3.3.1 (Technical Specification) and in Section 3.5.1.1, the Cloud Storage component will provide the support of MySQL, a relational database management system (RDBMS[7]), in the second prototype. Due to new requirements regarding the Context Management task in Work Package 6, also the support of the graph database neo4j[8] is planned to be integrated.

[5] http://www.mysql.com/
[6] http://rdf4j.org/
[7] http://en.wikipedia.org/wiki/Relational_database_management_system
[8] http://neo4j.com/

3.5.1.2 Management Web Interface

Administrators should be able to manage the Cloud Storage component. This includes to adding, removing and configuring databases and managing existing Buckets in these databases. This will be possible by a web interface provided in the next prototype. A mock-up version of this web interface, which illustrates the approach, is already available in the Administration Tool website.

3.5.1.3 Availability

The development for the second prototype will focus on the aspect of availability, which is the ability of this component/service to stay in a workable state, even if a problem occurs, which means in the context of the Cloud Storage that if the communication with a database fails, a backup database is available which contains the same data.

3.5.1.4 Scalability

The second prototype will also focus on the scalability aspect. The Cloud Storage should be capable of interacting with all connected components regardless of the workload. One approach to providing sufficient performance is to implement a so-called load balancer linked with multiple instances of the Cloud Storage, which manages incoming queries for a better utilisation through distribution.

3.6 Research Background

For the implementation and overall approach of the current prototype the following papers have been researched and taken into consideration:

Source	Subcomponent	Description
L. Qian et al., “Cloud Computing: An Overview” in Cloud Computing, Berlin Heidelberg, Germany, Springer, 2009, bp 3, pp 626-631.	General	This paper has been considered in an overall context of the component.
Q. Zhang; L. Cheng; R. Boutaba, “Cloud computing: State-of-the-Art and Research Challenges” in Journal of Internet Services and Applications, London, UK, Springer, 2010, vol. 1, pp 7-18.	General	This paper has been considered in an overall context of the component.
M. Al Morsy; J. Grundy; I Müller, “An Analysis of the Cloud Computing Security Problem” in Proceedings of APSEC 2010 Cloud Workshop, Sydney, Australia, 2010.	Storage Nexus	This paper has been considered in the context of authorisation and authentication implementation.
Wu, Jiyi, et al. "Cloud Storage As the Infrastructure of Cloud Computing." Intelligent Computing and Cognitive Informatics (ICICCI), 2010 International Conference on. IEEE, 2010.	General	This paper has been considered in an overall context of the component.

Figure 44: Research Background Cloud Storage

3.7 Target Performance

For the implementation and the overall approach of the current prototype, the following papers have been researched and taken into consideration:

Topic	Description	Target KPI
Response time	The time spent between requesting data from Restful interface to the time the interface client has received the data.	Data request roundtrip response time of <= 3 seconds for 90% of single operation requests.
Availability	As was described in Section 3.5.1.3, the Cloud Storage component should provide a high availability. For example, the availability of the connected Amazon S3 storage shall be 99.99%. Due to the fact that this is a prototype version and this component is one of many, SAM does not strive for such a high availability.	After the provision of the second prototype version the availability shall be 80% or higher.
Scalability	As was described in Section 3.5.1.4, the Cloud Storage component should be able to provide sufficient performance regardless of the workload. The approach will be to implement a load balancer to enable multiple instances of the Cloud Storage.	This component should provide a load balancer subcomponent which enables the access of multiple instances (>2) of this component at the same time.

Figure 45: Target Performance Cloud Storage

3.8 Summary

This section provides a description of the first prototype of the Cloud Storage component developed in task T4.1 Assets Storage and Information Management. The main outcome of this task is the software of the Cloud Storage component. This prototype is the first of the two iterations planned for this component.

It has presented the requirements necessary for both users and developers in order to manage the Cloud Storage component, including installation and deployment instructions. Additionally an updated version of the interface specifications has been provided.

The last section was dedicated to describe the limitations of the current prototype, also describing the next steps considered for the second version of the component, which should be delivered in M25.

4 Communication and Federation

This section describes the software deliverable D4.2.1, which is the first prototype release of task T4.2, Communication and Federation task.

4.1 Scope and Relationship

The Interconnection Bus plays a central role in the SAM architecture allowing reliable transmission of information between the different SAM components and federated instances. It directly interacts with the different components of the Platform and is able to communicate with auxiliary external systems through the different pluggable Communication Adapters mechanism. It also provides advanced queuing mechanisms (allowing event-driven approaches), services orchestration, message routing and transformations.

The Interconnection Bus component is split in several subcomponents and the logical connections that need to be established between them are represented in Figure 44.

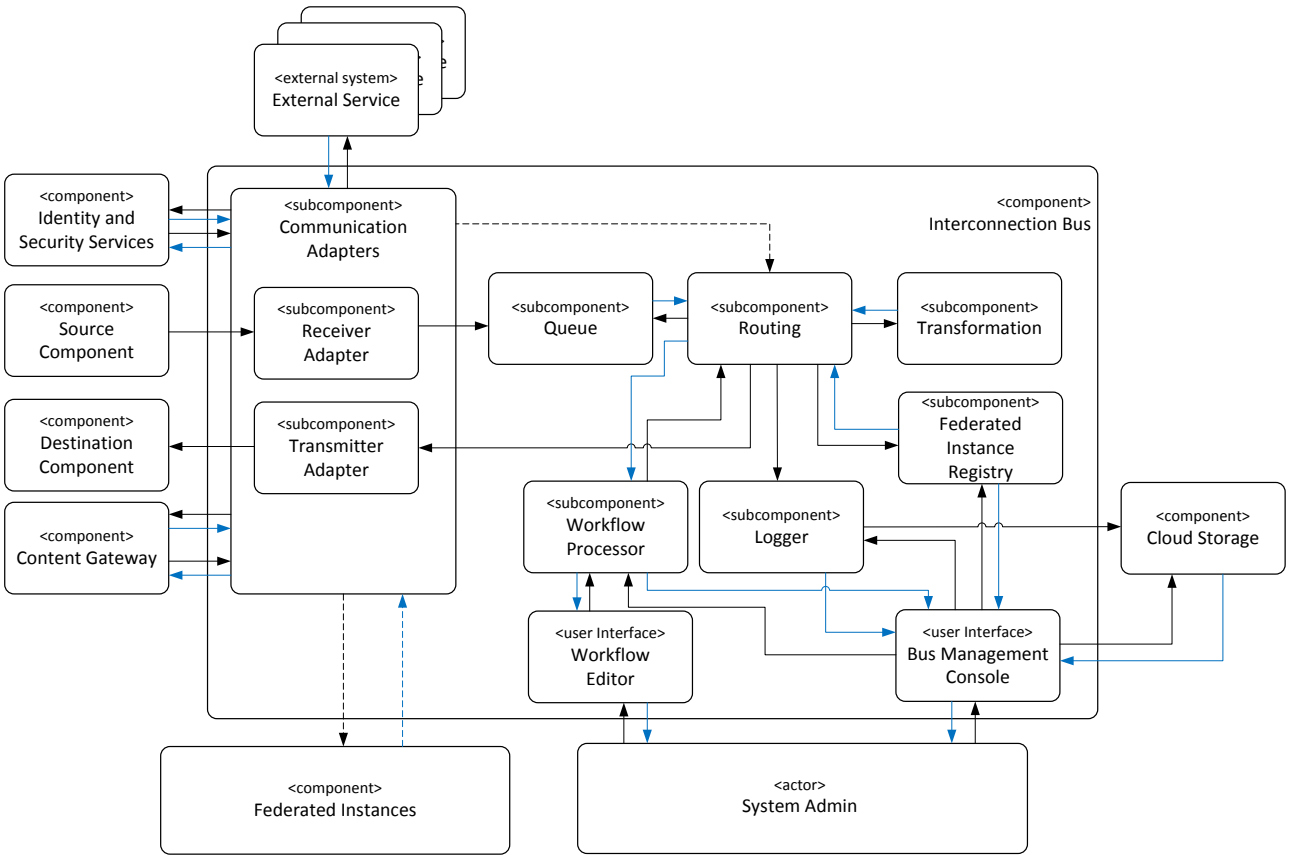

Figure 46: Overview of Interconnection Bus Subcomponents

For further description of the functional and technical foundations of these subcomponents, please revisit documents D3.2.1 Section 4.1 (Architecture), D3.2.2 Section 4.2 (Functional Specification) or D3.3.1 Section 3.2 (Technical Specification).

As it was decided in the technical analysis (see deliverable D3.3.1 Technical Specification), the component has been implemented using an Enterprise Service Bus (ESB) approach based on a TIE Kinetix technology called TIE Smart Bridge (TSB). The

first prototype of T4.2 aims at describing how to configure the different SAM components and services in TSB and how to send messages between them.

In order to carry out these main objectives, a new TSB instance (SAM TSB) has been adapted and deployed in an internal TIE Kinetix server. Furthermore, as part of the Communication Adapter subcomponent, a web interface called SAM Communication Module (SAM CL) has been implemented in order to configure the different components and services settings. The SAM CL has been linked in the SAM Administration Tool in order to allow the access for all SAM administrators.

A summary of the tasks carried out for each subcomponent of the first version of the prototype is shown in the following table:

Subcomponent	Task
Communication Adapter	Implementation of SAM CL and improvements in the mechanism that allow the implementation and publishing of different Communication Adapters so that they can be used by the different Source and Destination SAM components. In that way, the different components can define which their favourite protocol and data formats are, and the SAM TSB is able to receive, transform and deliver the messages in the correct format and protocol.
Queue	Improvements in the engine to control the internal queues, where the messages are temporally stored. The Queue subcomponent is in charge of ensuring that messages are not lost. In the next prototypes the engine will be finalised to implement topic-based publish-subscribe mechanisms.
Routing	The engine has been adapted to process the messages and send them to their destination.
Transformation	Improvements in the engine to process the messages and send them to their destination based on the latest concurrency features.
Logger	Improvements in the Logger engine to register the information of the messages flowing inside the SAM TSB instance based on a new exception strategy. In the next prototype, part of this information will be stored in the Cloud Storage in order to show appropriate information in the Administration Tool.
Bus Management Console	Configuration of SAM TSB instance user interface in order to show the logs and status of the components. It was also implemented of the SAM CL web interface in order to configure the different components and services settings.

Figure 47: Tasks carried out for the First Prototype of T4.2

4.2 Requirements and Preparations

This section provides information on technical and non-technical requirements for users and developers, as well as software prerequisites for the installation of the Interconnection Bus component.

4.2.1 For Users

Registered users in the Administration Tool can use the SAM CL user interface in order to create, edit and delete components and services. A screenshot of the current layout of this interface is shown in Figure 48.

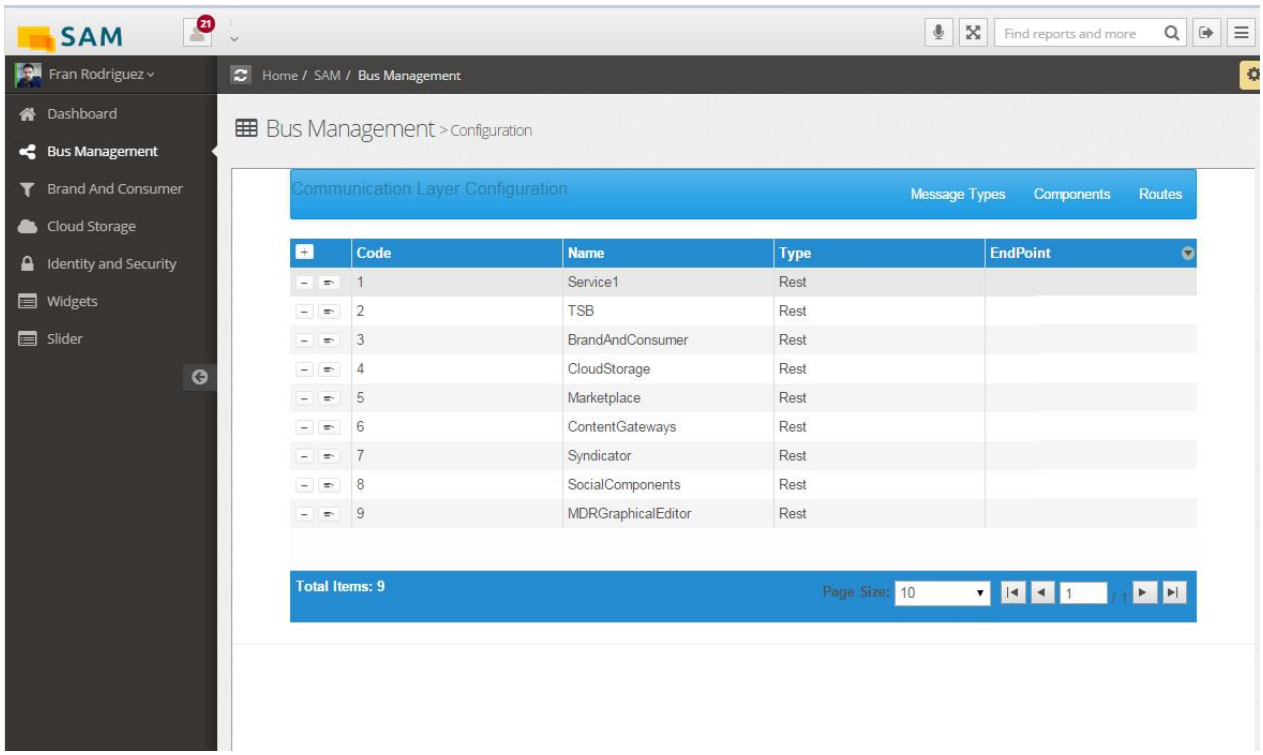

Figure 48: SAM Communication Module Interface

4.2.2 For Developers

The SAM CL is software based on .NET technologies, thus the following requirements are necessary to deploy the software:

- Windows Server 2012 (or superior)
- .NET Framework 4.5 (or superior)
- Internet Information Server 7.0 (or superior)
- Administrator privileges

4.3 Installation (Deployment)

The SAM TSB instance has been installed in a TIE Kinetix server since it contains sensitive information about the official TIE Kinetix Smart Bridge product. Therefore, the installation details cannot be provided. However, the SAM TSB instance has a public URL accessible from the Administration Tool (see D4.2.1) and additionally it is accessible for the SAM partners via Remote Desktop.

Due to the fact that the SAM Communication Module is a plug-in of TSB, it has also been installed in TIE Kinetix servers, thus some of the installation details cannot be shown. However, the following steps should be followed for the correct deployment:

1. Store SAM CL source code in a physical path
2. Open IIS Manager
3. In the Connections pane, right-click the Sites node in the tree, and then click Add Web Site
4. In the Add Web Site dialog box, type “SAMCL” in the Web site name box
5. Click Select if you want to select a different application pool than the one listed in the Application Pool box. In the Select Application Pool dialog box, select an application pool from the Application Pool list and then click OK

6. In the Physical path box, type the physical path of the Web site's folder, or click the browse button (...) to navigate the file system to find the folder
7. Select “HTTP” protocol for the Web site from the Type list
8. Keep the default value in the IP address box as All Unassigned
9. Type a port number in the Port text box
10. Select the Start Web Site Immediately check box
11. Click OK

4.4 Execution and Usage

This section explains how to execute the SAM TSB instance in order to use the features adapted for this 1st Prototype and the SAM CL in order to manage the messages types, components, services and routes.

4.4.1 SAM TSB instance

As it was mentioned before, for this first prototype, it has been set up a new TSB instance (SAM TSB) and adapted for the SAM requirements. Furthermore, several performance improvements have been carried out. In the following subsections is described the current available functionalities for the SAM Administrators.

4.4.1.1 Login

Through this interface different SAM Administrators can enter their credentials. Once validated the SAM Administrators are logged into the system. The instructions to obtain credentials are described in deliverable D4.2.1.

Figure 49: SAM TSB Instance - Login

4.4.1.2 Dashboard

The dashboard contains different widgets in order to provide quick views about the following topics:

- **Errors in the System:** This widget shows the number of undeliverable and unrecognised documents
- **System Health:** This widget shows the information about the status of the different parts of the system such as CPUs (TIEKinetix Server has 2 CPUs), memory, disk space, services and archiving (external storing service for messages). The green button means that the health of component is good and red button means that something wrong is happening. In order to see what is exactly happening, the user can put the cursor over the circle and a tooltip message will show the detailed information.
- **Last 6 Documents:** This widget will provide a quick view of the last 6 documents processed in the SAM TSB instance
- **Document Statistics:** This widget shows different stats about the transmission of the messages such as number of received messages, sent or in process

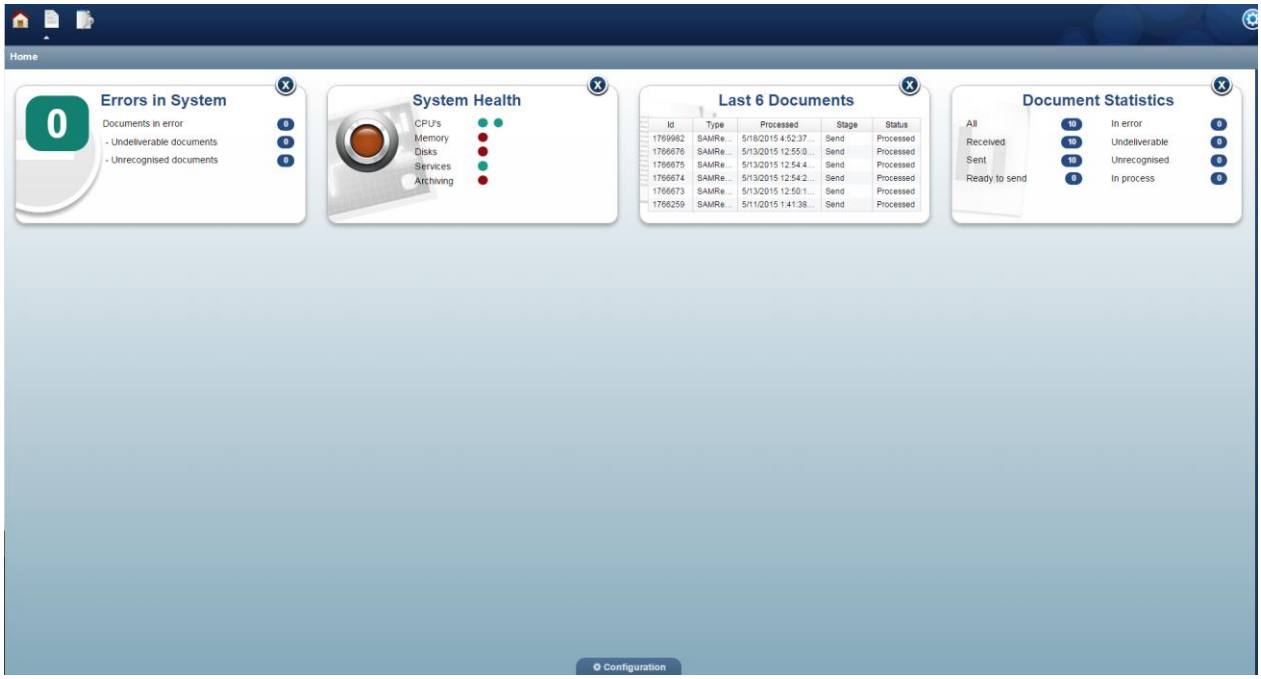

Figure 50: SAM TSB Instance - Dashboard

4.4.1.3 Documents

In SAM TSB, all the messages are contained in documents and the documents are received, analysed and send to the destination. For this reason the document can have several statuses (received, ready to send, sent, undeliverable, in error, unrecognized, in process or archived). By default, it is defined several folders which contains the messages for each of the several statuses. Clinking on the different folders, it is possible to see the list of messages in each status. This list can be filtered to see the documents processed in the same day, 7 days before or 30 days before.

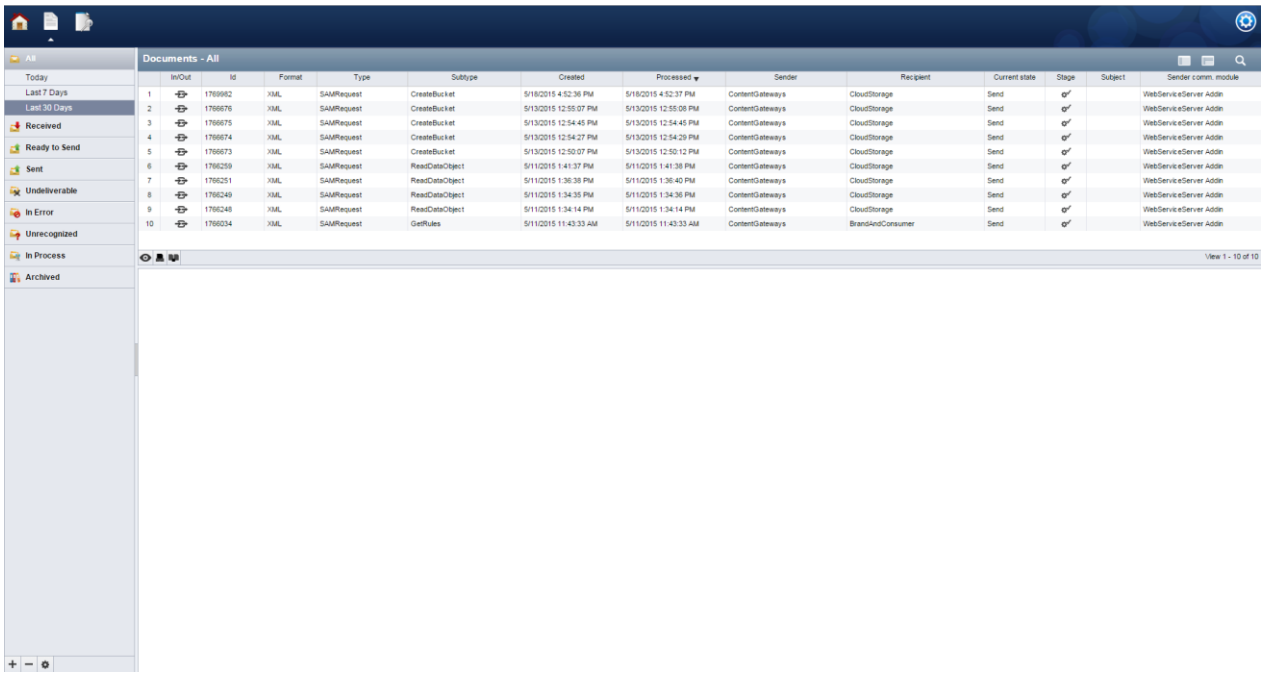

Figure 51: SAM TSB Instance - Documents

In case that the SAM Administrator wants to see the details of the message, he/she have to click on the message. After that, the details will be shown in the bottom part. It is possible to see the information for each one of the transmission steps (Received, Analysed and Sent). The information of the document is split in two sections: “General” section and “Document Contents”. The “General” section provides the following information:

- **Id:** This is a unique document identifier
- **Created:** Date when the message was created
- **Processed:** Date when the message was processed
- **State Type:** State of the Message
- **Stage:** If the message is processed correctly or not
- **Format:** Document Format (XML, JSON, etc.)
- **Type:** Type of the message, which value should always be “SAMRequest”
- **Subtype:** Name of the service, e.g. “CreateBucket”
- **Sender:** Component which sends the message, e.g. “ContentGateways”
- **Recipient:** Component which receives the message, e.g. “CloudStorage”
- **Extra Information:** Some extra information, normally avoid

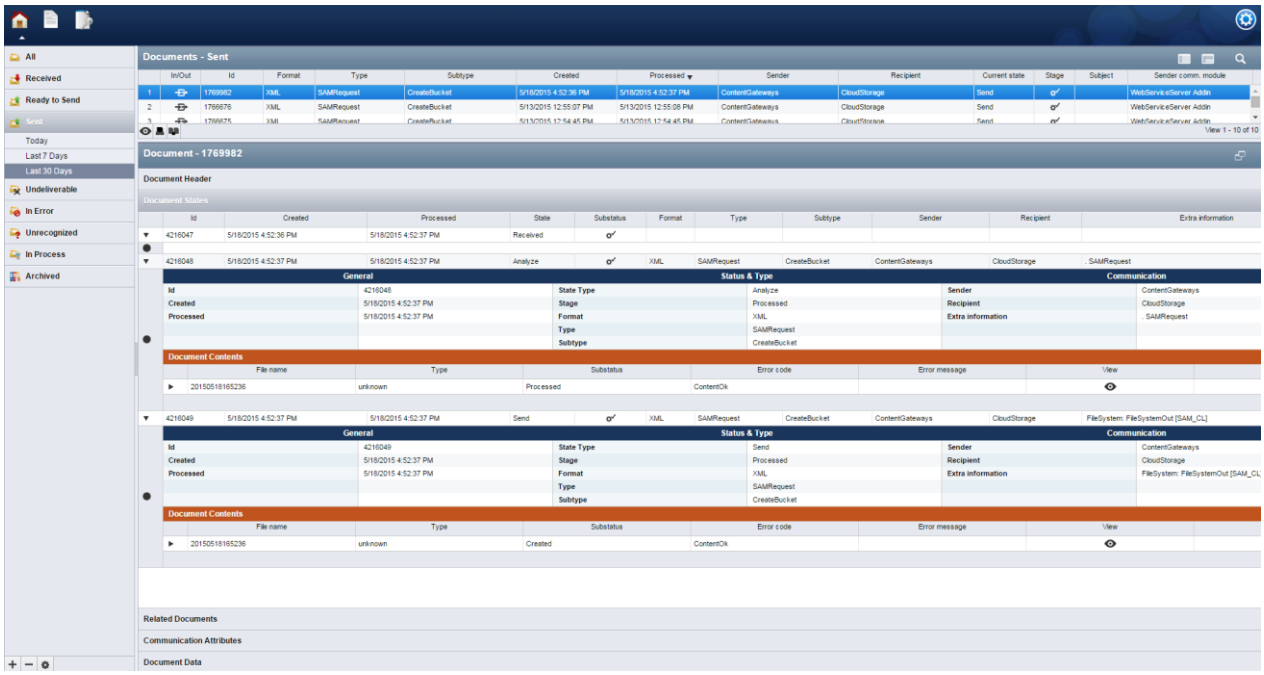

Figure 52: SAM TSB Instance – Documents Detail

The “Document Contents” section shows possible error codes and the associated message. It is also possible to see the original message clicking on the eye icon.

Figure 53: SAM TSB Instance – Message View

4.4.1.4 Search Document

By clicking on the loupe icon, it is possible to introduce multiple filters to search for a specific set of messages. The filters are based on the document information and are subdivided in several groups:

- **General:** General information filters
- **Partners:** Senders and Recipients filters
- **Date & Time:** Time filters
- **Advanced:** Filters about state, stage, format, type or subtype

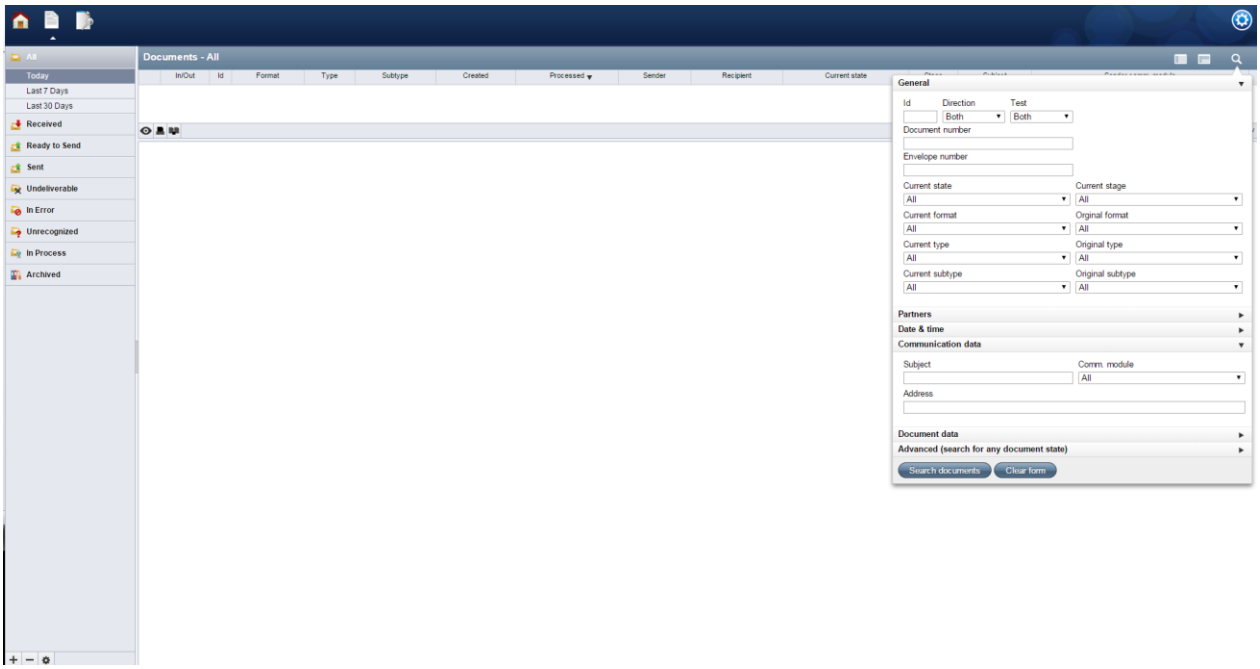

Figure 54: SAM TSB Instance – Search Document

4.4.1.5 Smart Folder

Clicking on the “+” icon in the bottom of the screen, it is possible to create Smart Folder. Based on the search fields, it is possible to create smart folders to have a quick view of the document that meet with the search specifications.

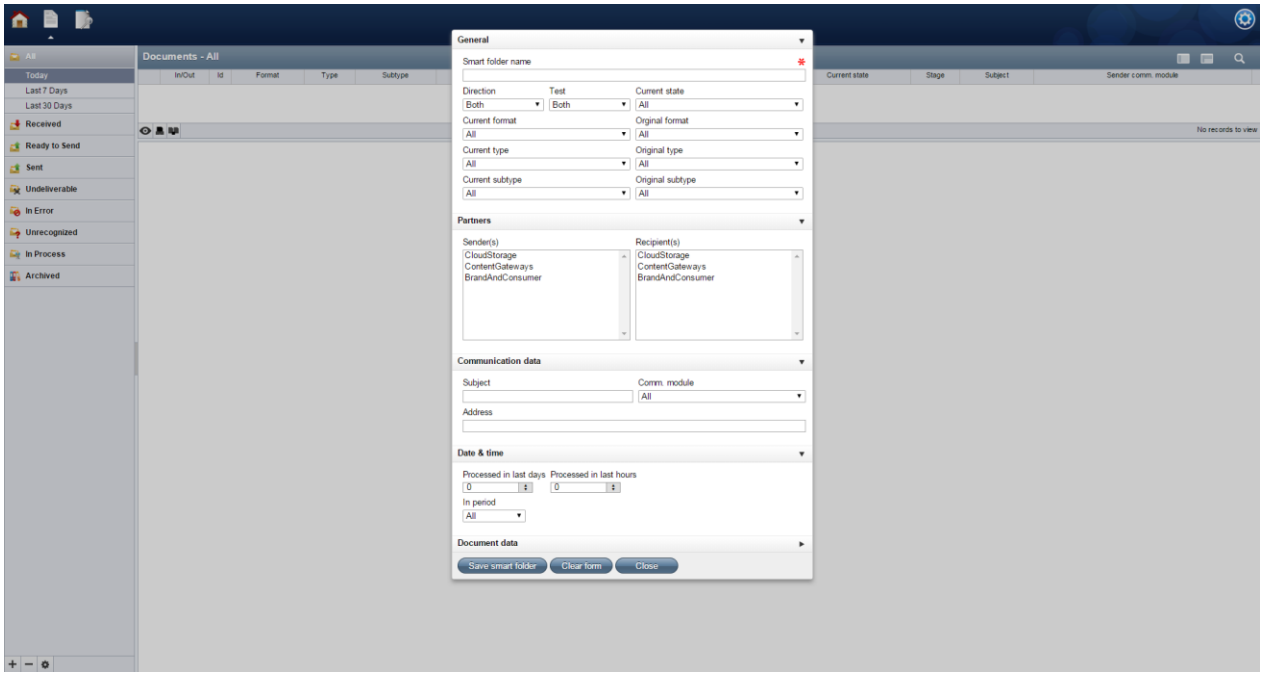

Figure 55: SAM TSB Instance – Smart Folder

4.4.2 SAM CL

As part of the Communication Adapter subcomponent, a user interface has been implemented, which provides functionalities to create message types, components, services and routes.

The general idea is that the different SAM components communicate with the Interconnection Bus by using a specific XML format (the SAM Component Message Format) and the Interconnection Bus will be then in charge of forwarding these messages to the predefined destinations, keeping control of the events that occur during the transmission, message transformation, reception and acknowledgement (if necessary) of the information.

The following sections explain the SAM Component Message Format and how to use the SAM CL to register new components and add new methods.

4.4.2.1 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 basic message information and body (<Payload>) in case extra embedded information is required.
- **<Source>:** This label will identify the source component. This name should be exactly the same as the component in the communication layer (see Section 4.4.2.2)
- **<Destination>:** This label will identify the destination component. This name should be exactly the same as the component in the communication layer (see Section 4.4.2.2
- **<Service>:** This label will identify the service. This name should be exactly the same as the service in the communication layer (see Section 4.4.2.3)
- **<Topic>:** Not used yet
- **<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>
      <Source>ContentGateways</Source>
      <Destination>BrandAndConsumer</Destination>
      <Service>GetRules</Service>
      <Topic> </Topic>
      <Parameters>
         <Parameter>
            <Name>param1</Name>
            <Value>BDS</Value>
         </Parameter>
      </Parameters>
   </Header>
   <Payload>
      {
         "description": "Fran Test from TSB",
         "owner": "BDS",
         "active": true,
         "criteria":
         [{
            "field": {
               "iname": "User Age",
               "jname": "user.age",
               "jtype": "int",
               "toString": false
            },
            "condition": "GREATERTHANOREQUALS",
            "value": "16"
         }]
      }
   </Payload>
</SAMRequest>
```

Figure 56: SAM Component Message Example

4.4.2.2 Register a New Component

The first step is to create an entry for each SAM component. Once the component is created it is possible to add new service methods. The following list contains the necessary steps:

1. Click on “Component” tab
2. Click on “+” icon in the first cell in the first column in the table.
3. Fill in the form
   - **Name:** This name should be without spaces and representative of the component. For example, Cloud Storage can be “CloudStorage”.
   - **Type:** Rest since all interfaces will be RESTful interfaces
   - **Endpoint:** The main endpoint for the component. For instance, http://localhost:8080/api/cs or http://localhost:8080/BCP/api/v1/

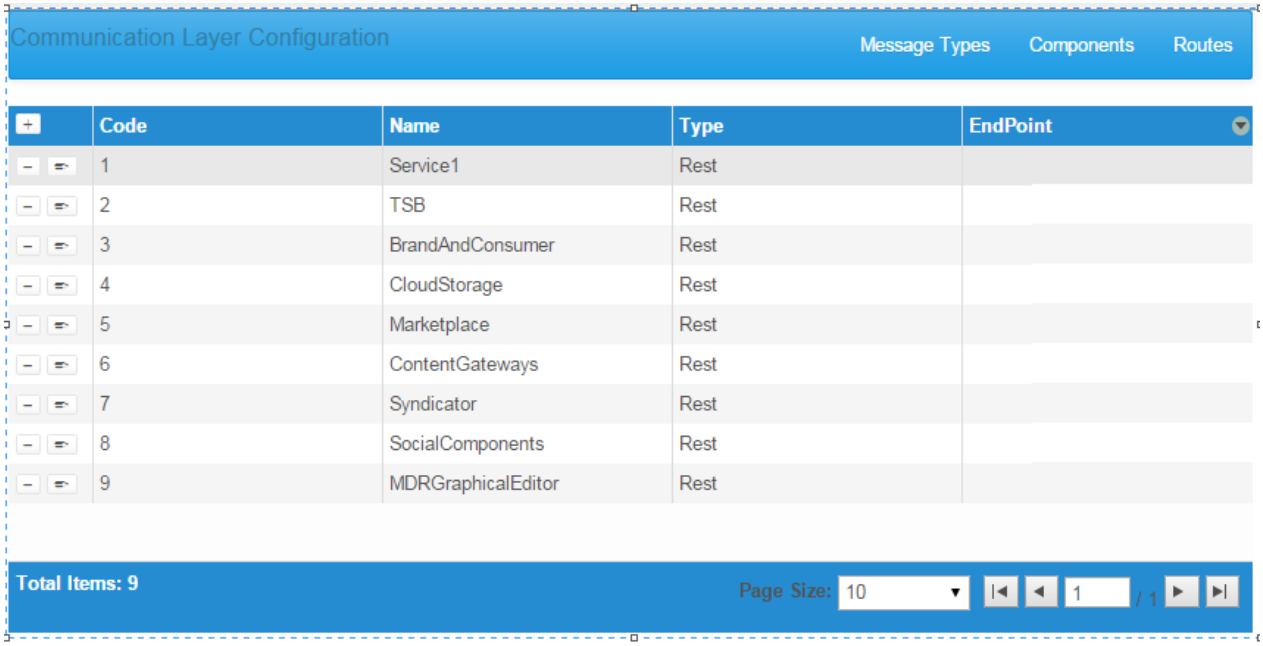

Figure 57: Component Main Screen

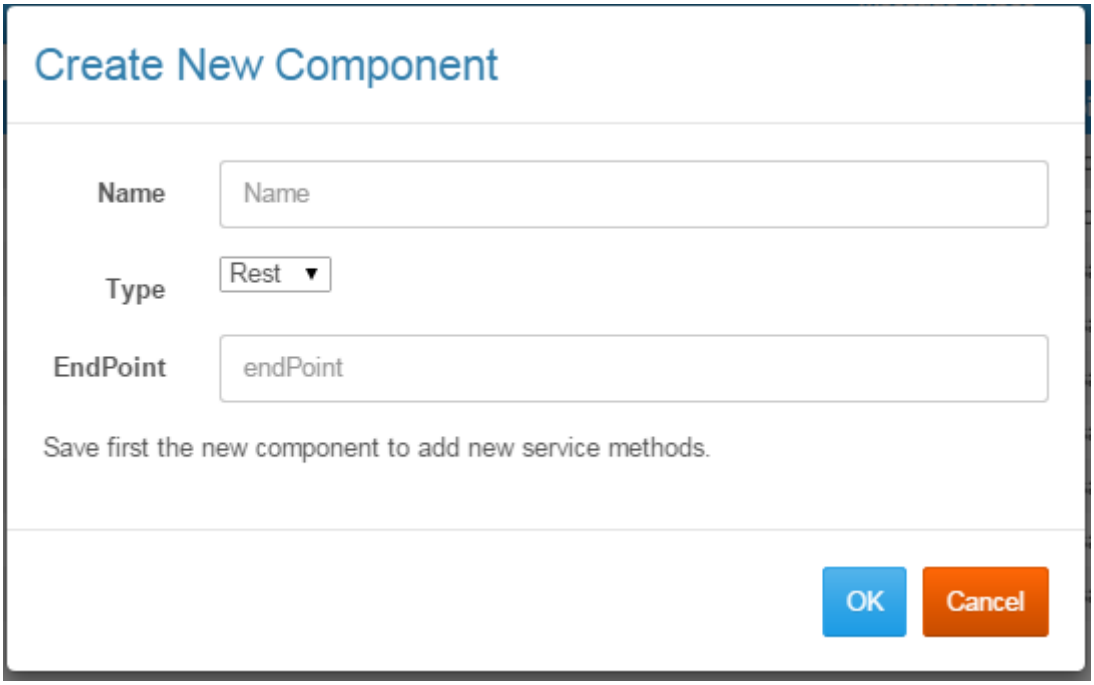

Figure 58: Create New Component

4.4.2.3 Add a New Method

The methods or services are part of the component, thus in order to create a new service it is necessary to edit the component and click on the “+” icon in the first cell of the first column in the table. In this screen it is also possible to update the general information of the component.

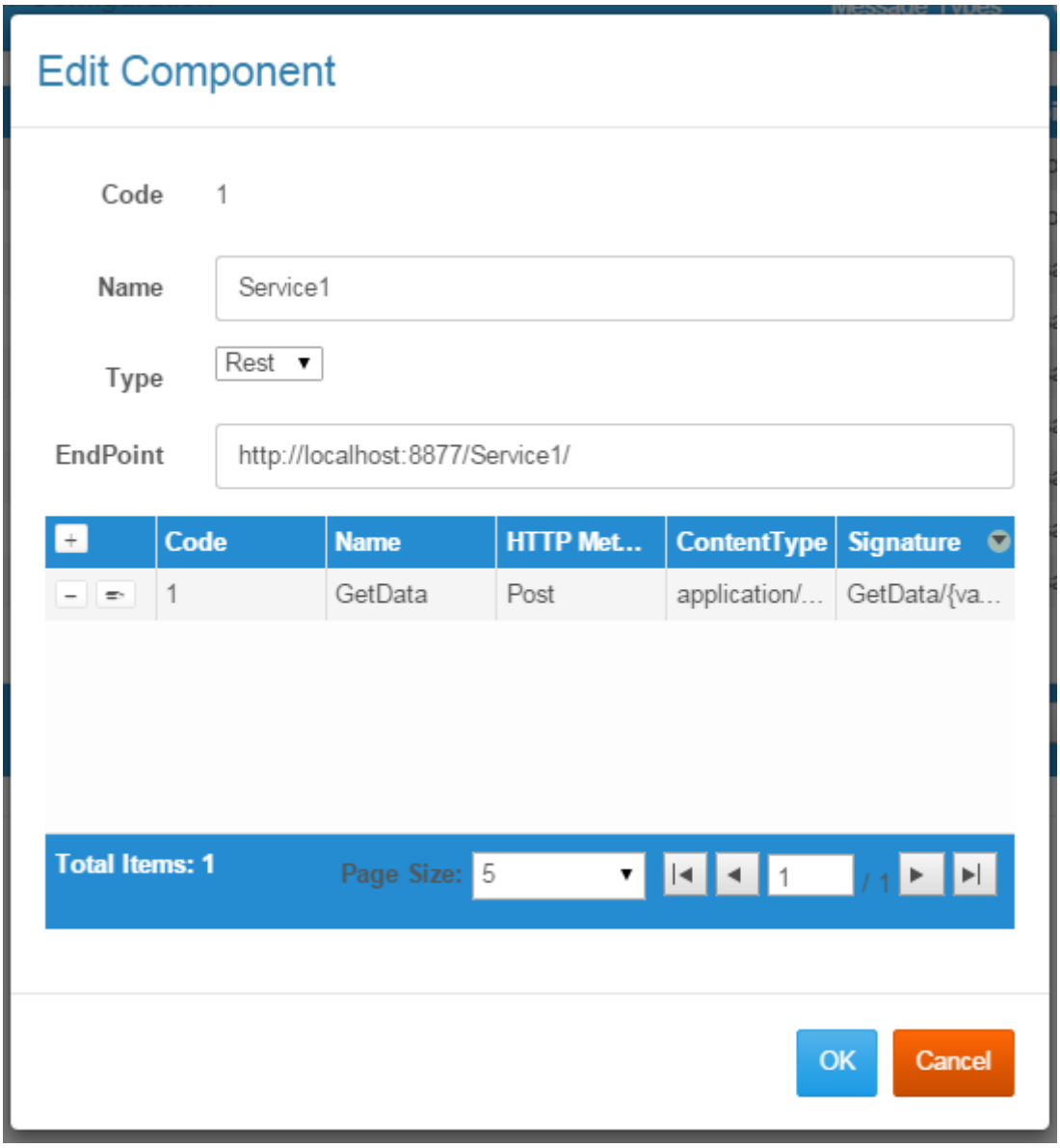

Figure 59: Edit Component

In order to create a new method it is necessary to provide the following information:

Field	Description
Name	Name of the service without spaces and representative. For instance, “CreateBucket”
Http Method	PUT, GET, POST, DELETE
Signature	Name of the service. For instance, “Bucket” Or “rules?owner={sowner}”
ContentType	Type of the content. For instance “application/json”
Parameters	• If the information is in the SAM Component Message payload “RequestBody.application/json=/SAMRequest/Payload” • If the parameter is included in the signature. "UrlSegment:sowner=/SAMRequest/Header/Parameters/Parameter[Name = 'param1']/Value” • Different parameters can be introduced separately with a line-break

Figure 60: Add a New Method – Required Information

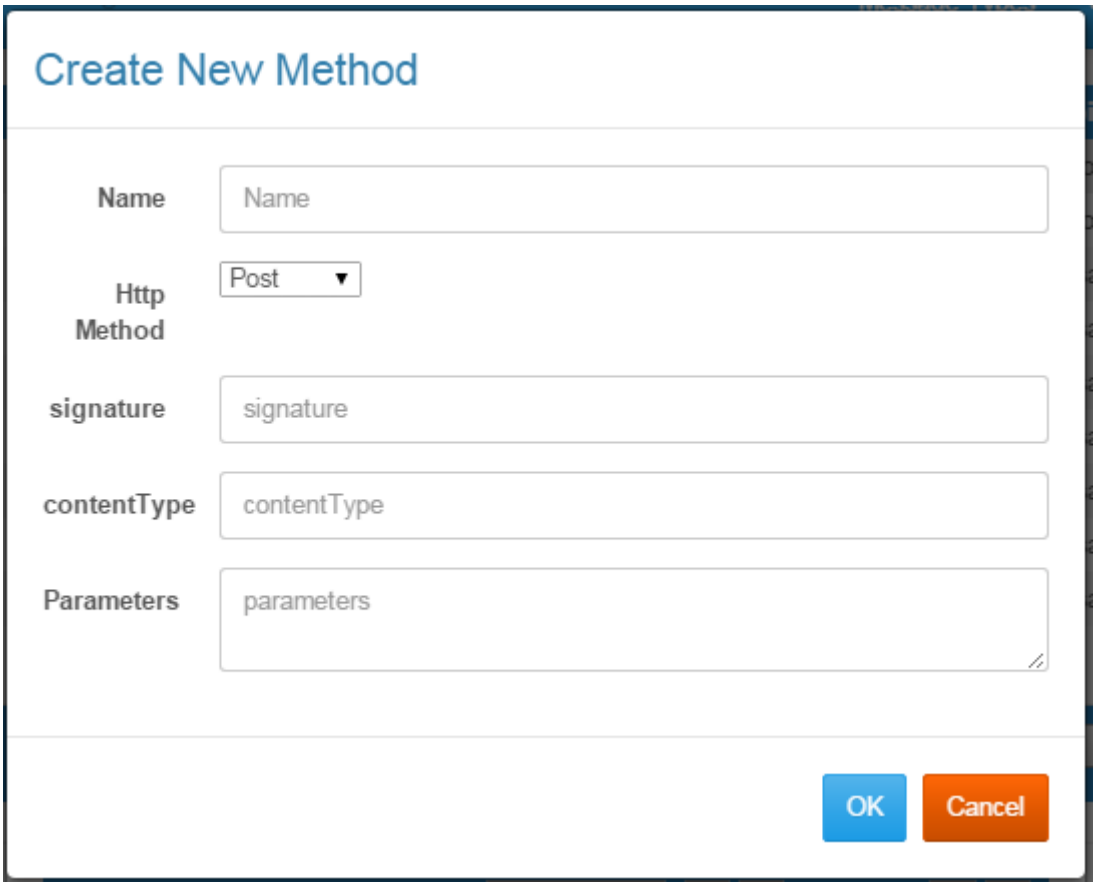

Figure 61: Create a New Method

4.5 Limitations and Further Developments

One of the limitations of the prototype is the connection with the Identity and Security Services component for data authorisation, since this component is under development. The first prototype of T4.4 Distributed Identity, Trust and Security should be ready in M19 as specified in the DoW. For this reason, the integrated access control list feature will be used until the Identity and Security Services are available.

4.5.1 Prototype 2 Planned Tasks

Regarding the next steps in the Interconnection Bus component, Figure 43 and the following subsections summarise the tasks planned for the second prototype (to be delivered in M25).

Subcomponent	Task
Queue	Finalise the engine to implement topic based publish-subscribe mechanisms
Workflows Processor/Editor	Implement the engine to control and orchestrate the different extra actions that can be added in the messages processing - e.g. before delivering a message, for instance, specific transformations, access to SAM Component services, digital signature services, etc.
Federated Instances	Implement the system to allow the different SAM Instances to interact and exchange information
Bus Management Console	Integrate the SAM CL with Identity and Security Services in order to provide the security system
Logger	Adapt the Logger to store the logs in the Cloud Storage component and retrieve them from there.
Communication Adapter	The necessary service will be implemented in order to provide Content Gateways with the appropriate interoperability and data gathering features
Transformation	This subcomponent will be used by the Routing subcomponent in order to execute the transformations between different messages and data formats

Figure 62: Tasks Planned for the Second Prototype

4.6 Research Background

For the implementation and the overall approach of the current prototype, the following papers have been researched and taken into consideration:

Source	Subcomponent	Description
Turning your legacy systems into the future profit: innovation in production management	Interconnection Bus	In this article, authors present the results of the ARUM[9] (Adaptive pRodUction Management) project, funded from the European Union's Seventh Framework Programme (FP7), which started with applying a traditional enterprise Service-Oriented Architecture (SOA) paradigm to solving an integration problem in the domain of production ramp-up of highly customised products such as aircrafts, ships, etc.
Enterprise Service Bus: A Performance Evaluation[10]	Interconnection Bus	In this study, three open source Enterprise Service Buses (ESBs) are evaluated and compared, both qualitatively and quantitatively. The empirical results were statistically tested to determine the statistical significance of the results.
Enterprise Service Bus Monitoring Framework for SOA Systems[11]	Interconnection Bus	The paper presents a Monitoring Framework for the integration layer of SOA systems realised by an ESB. It introduces a generic ESB Metamodel (EMM) and defines mechanisms which gather monitoring data related to the model entities
Evaluating Caching Strategies for Cloud Data Access Using an Enterprise Service Bus[12]	Interconnection Bus	In this work, an approach for transparently accessing data migrated to the Storage Cloud using a multi-tenant open source ESB as the basis is proposed and carried out. Furthermore, the ESB has been enhanced with a Quality of Service feature (QoS) awareness by integrating it with an open source caching solution.

Figure 63: Research Background Interconnection Bus

4.7 Target Performance

For this component the following key performance indicators (KPI) have been defined:

[9] http://arum-project.eu/

[10] http://www.scirp.org/journal/paperinformation.aspx?paperid=6768#.VSZY5PmUe3w

[11] http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5928481&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D5928481

[12] http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6903485&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6903485

Topic	Description	Target KPI
Reliable Messaging / Receipt Acknowledgement	Delivering messages between the different SAM Platform actors reliably. Reliability is an important parameter for the main communication foundation of SAM, so it is of maximum importance.	Since the most important aspect is to be sure that all messages are flowing correctly this component should achieve a **98%** reliability.
Communication Performance	Communication Performance is measured in terms of the number of messages per second.	100 messages per second across the whole platform in debug mode
Translations Performance	Translations Performance is measured in terms of the number of translations per second	10 translations per second in debug mode

Figure 64: Target Performance Interconnection Bus

4.8 Summary

This section provides a description of the first prototype of the Interconnection Bus component developed in task T4.2 Communication and Federation. The main outcome of this task is the software of the Interconnection Bus component. This prototype is the first of the three iterations planned for this component.

It has presented the requirements necessary for both users and developers in order to manage the SAM Communication Layer, including the instructions to create/edit/delete components and services. Additionally a version of the SAM Component Message has been provided.

The last section was dedicated to describe the limitations of the current prototype, also describing the next steps considered for the second version of the component, which should be delivered in M25.

5 Data Characterisation Services

This section describes the deliverable D4.3.1 based on the first prototype of the Semantic Services component. Note this component is being developed as a central component to deal with human language processing. This component involves functionalities such as Sentiment Analysis and Text Summarisation, which will be exploited by the Social Mining component described in D6.4.1.

5.1 Scope and Relationship

Semantic Services is the component in charge of providing natural language processing functionalities to the platform, offering other components in SAM technologies to perform data characterisation, ontology exploitation and the aforementioned Sentiment Analysis and Text Summarisation. Figure 65 shows the different subcomponents of the Semantic Services, the logical connections established between them and the relationships with other components and actors in SAM.

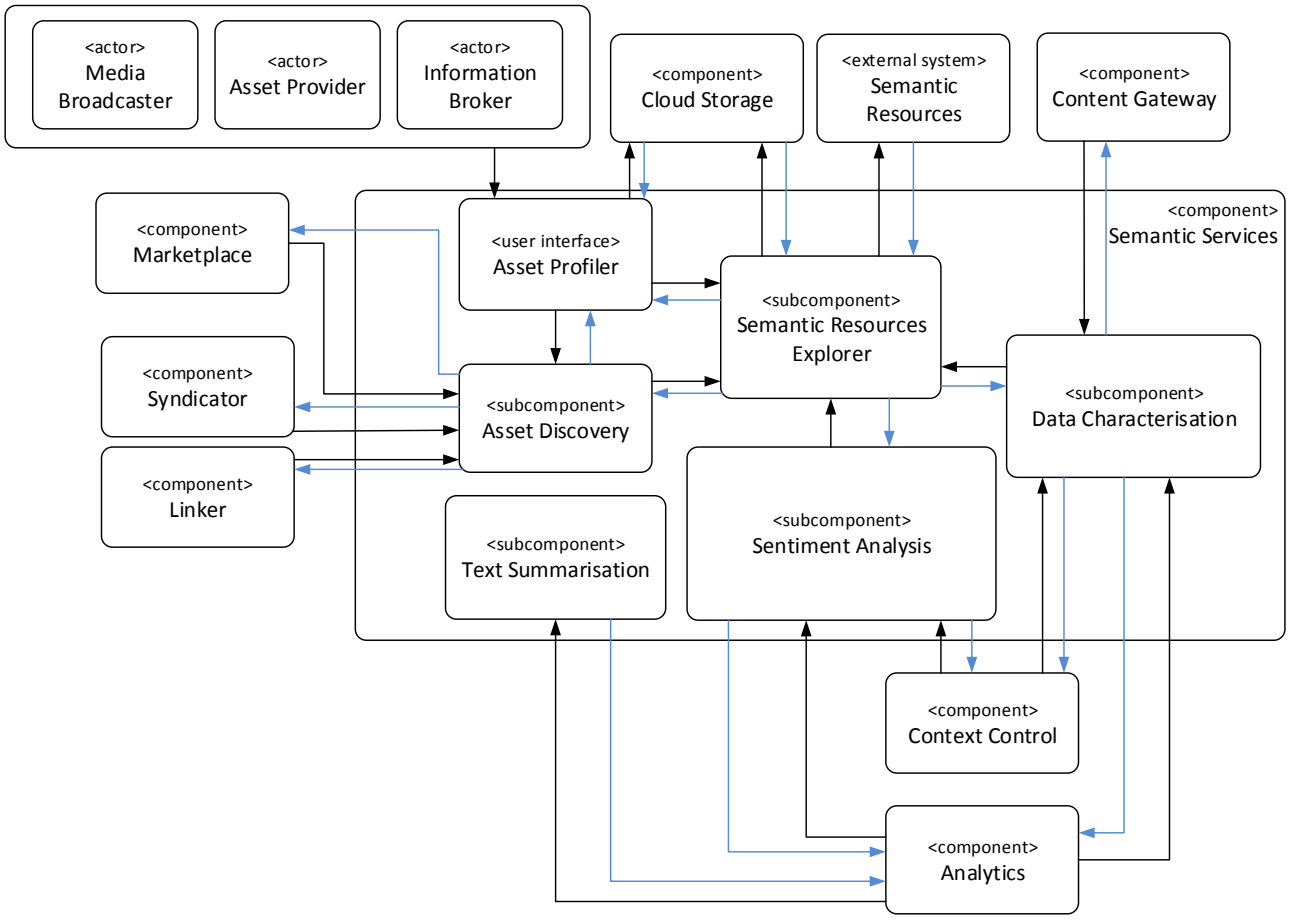

Figure 65: Overview of Semantic Services Subcomponents

For further description of the functional and technical foundations of these subcomponents, please revisit documents D3.2.1 Section 4.2 (Architecture), D3.2.2 Section 4.3 (Functional Specification) or D3.3.1 Section 3.3 (Technical Specification).

This component includes five accessible subcomponents and one user interface which in this first prototype address the tasks described in Figure 66. The tasks labelled as [**mock-up**] identify those functionalities that have not been developed in this prototype, although

mock-up interfaces have been created and documented, including the definition of input and output data schemas for interoperability.

Subcomponent	Task
Data Characterisation	The following functionalities have been implemented: • When external assets are imported into the SAM platform by the Content Gateways component (see D5.2.1 for more information), the Data Characterisation subcomponent performs a mapping between the structure of the imported data and the Asset structure (ontology) defined in SAM. • Given an input text (usually obtained from the Asset's content), this subcomponent identifies occurring mentions to Wikipedia entities (a process known as entity linking[13]). • This subcomponent also identifies mentions to existing Assets in SAM, both in Asset's content and user generated comments [**mock-up**]
Asset Discovery	The following functionalities have been implemented supported by the methods developed by internal Semantic Services subcomponents: • Receives a query (in the form of a set of keywords) and provides a list of Assets related to that query [**mock-up**] • Asset CRUD (create, read, update and delete) operations
Sentiment Analysis	Performs sentiment analysing on User Generated Content (UGC)
Text Summarisation	Summarises large amounts of UGC, keeping only the most representative comments provided by the users regarding specific Assets [**mock-up**]
User Interface	**Task**
Asset Profiler	User interface that provides functionalities to edit, create and delete Assets [**mock-up**]

Figure 66: Tasks carried out for the First Prototype of T4.3

Note the Semantic Resources Explorer is a subcomponent that supports internal functionalities of the Semantic Services (as shown in Figure 65) regarding the accessibility to the semantic data of SAM, i.e. Assets stored into the Cloud Storage and external data extracted from DBpedia. Therefore the current report of progress will be associated to those subcomponents that use its functionalities. The Semantic Resources Explorer involves the following functionalities (the last one being a mock-up):

- Asset CRUDs operations: it allows storing and querying Assets useful to other Semantic Services subcomponents
- Exploration of DBpedia entities: it allows querying DBpedia
- Exploration of Sentiment Analysis resources: it allows querying Sentiment Analysis lexicons for sentiment evidences and lexical units [**mock-up**]

5.2 Requirements and Preparations

This section provides information on technical and non-technical requirements for developers, as well as software prerequisites for the installation of the Semantic Services component.

[13] http://en.wikipedia.org/wiki/Entity_linking

5.2.1 For Developers

This component offers public web interfaces to test all the functionalities. These interfaces were created as documentation of the services and no special identification is required for testing. All the functionalities documented include input and output data schemas as well as an example of use.

For developers who want to create their own Semantic Services instance, both Oracle Java 1.7[14] and Apache Tomcat 8.0[15] are required. This component includes four RESTful services that need to be deployed on the web application server: Sentiment Analysis, Asset Discovery, Text Summarisation and Data Characterisation. All the code developed and configuration files created for the Semantic Services is stored in the Git distributed version control repository maintained by the SAM consortium. See deliverable D4.3.1 for more information on how to access this repository.

5.3 Installation (Deployment)

Currently the installation can be made using the Jenkins Continuous Integration Server provided by the SAM consortium. For the Semantic Services component, a Jenkins integration project has been created and configured to build and deploy each subcomponent in Apache Tomcat 8.0. The steps to set up a running instance are the following:

1. Make sure to have Apache Tomcat 8.0, Oracle Java 1.7 and Ant 1.7.1 installed on the device.
2. Download the sources from the SAM Git repository.
3. Edit the file ApplicationConfig.java and modify the following attribute: SWAGGER_BASE_URL = `"http://[serverURL:portNumber]/[projectName]/socialmining"`, where serverURL:portNumber corresponds to the URL of the Tomcat web and its port number, and projectName refers to the name of the project.
4. Execute the following command-line `[root_directory_path] ant`, where `root_directory_path` refers the path of the project's root directory.
5. Step 4 will produce a new file, a file named `[projectName].war` inside the `dist` directory, which will be assigned the same name of the project. This file will constitute the Web application ARchive (WAR) of the current web project.
6. Finally, to deploy the WAR file in the web server application, it is necessary to go through the administration console of the Tomcat server, select the option `deploy application` and then select the file `[projectName].war` created in the previous step.

In this first prototype, the interfaces of the Semantic Services have already been integrated as part of the Semantic Services component.

5.4 Execution and Usage

This section explains how to access and manage the Semantic Services component with the provided RESTful interfaces. The following explanation assumes that developers are familiar with the general use of RESTful interfaces. As mentioned before, a description and

[14] https://www.java.com
[15] http://tomcat.apache.org/tomcat-8.0-doc/

an example are provided for each interface. Descriptions contain a screenshot including all the required data requests, response parameters and expected types and values.

5.4.1.1 RESTful Interfaces

In this first prototype, some of the RESTful interfaces provide fully functional components, whereas others are just mimicking the expected behaviour of the finished components (mock-ups), as described in Figure 66. The following sections describe specific details for accessing these interfaces. Figure 67 shows a screenshot of the web interface created to test the RESTful Services available for the Semantic Services component.

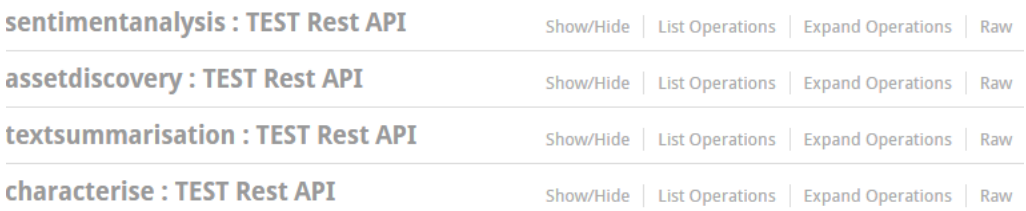

Figure 67: RESTful Services for the Semantic Services Component

5.4.1.1.1 Data Characterisation

The Data Characterisation subcomponent provides ontology mapping and semantic characterisation of content. In order to address these functionalities, two interfaces are described. The first one provides a mapping procedure between incoming data structures and the SAM ontology, whereas the second one is dedicated to characterise textual content by identifying Wikipedia entities in text.

5.4.1.1.1.1 Ontology Mapping

This first prototype provides a RESTful interface for suggesting alignments between the SAM ontology and the structure of the contents that will be imported into the platform by the Content Gateways. The documentation demo page for this service can be accessed in the following URL:

http://localhost:8080/SemanticServices/doc/#!/characterise/mapping

A screenshot of this page is shown on Figure 68. In the `Parameters` section of this demo page, the `body` textbox can be filled with a JSON example. The `Try it out!` button runs the service.

```
The mapping of incoming structures (represented by the data inserted into the body
textbox) with the SAM ontology is based on semantic structural algorithms [SE13]. These
algorithms are based on the Levenshtein distance [LEV66] in order to establish a
measure for getting the most similar terms to align between two ontologies, and
consequently searching the most similar structures by exploring adjacent semantic
relations. In this way, the most coincident structures are calculated. As a result, a
confidence score of the most similar matching terms of both ontologies can be obtained
(see
```

Figure 70).

Continuing with the demo description, the section `Response Messages` provides information of possible responses of the mapping service when the execution is performed (`Successful operation`, `Object not found`, `Fatal error`, etc.).

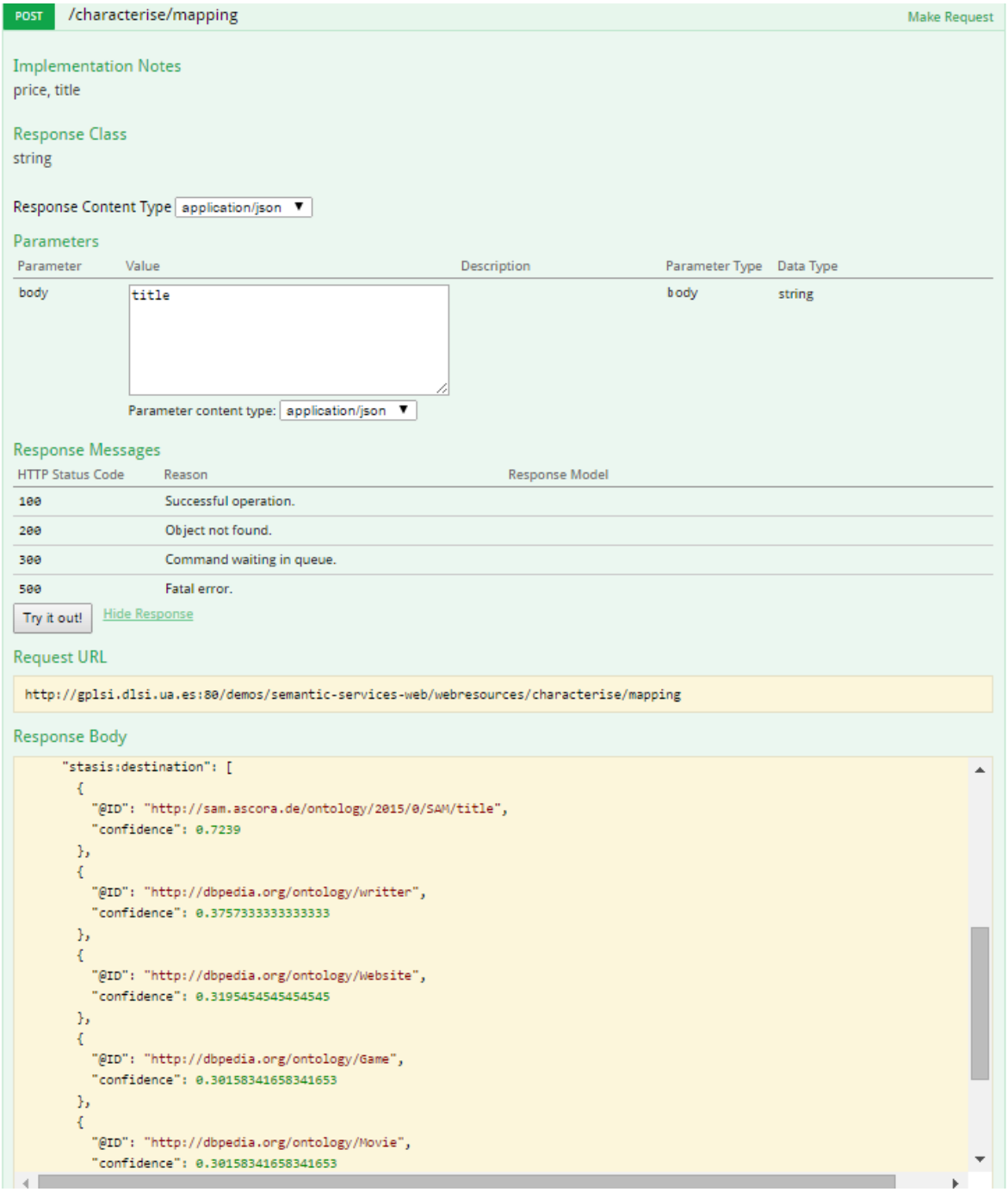

Figure 68: Documentation Demo Page for the Ontology Mapping RESTful Service

The input format of the information to be included in the `body` textbox has been simplified for the sake of demonstration. In this first prototype, the goal established in the project is to map input data provided by SAM user partner BDS to the SAM ontology. To this end, the mapping service assumes that the structure of the incoming data is that resulting of the import process carried out by the Content Gateways on BDS data.

Figure 69 shows and example of this structure.

```
{ “incomingContent”:
“<rdf:RDF
    xmlns:cdm_xsd="http://stasisproject.net/cdm_xsd#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:stasis="http://stasisproject.net/model#"
    xmlns:cdm="http://stasisproject.net/cdm#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:cdm_edi="http://stasisproject.net/cdm_edi#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xml:base="http://www.stasis-project.net/">
  <stasis:SLS rdf:about="http://stasisproject.net/model#TLSImpl_1416474881479">
    <stasis:source>
      <cdm:SimpleNode rdf:about="schema/...">
        <cdm:SchemaReferencePath>/catalog/book/genre</cdm:SchemaReferencePath>
        <cdm:orderInsideCompartment>0</cdm:orderInsideCompartment>
        <cdm:positionLength>0</cdm:positionLength>
        <stasis:ID>http://www.stasis-project.net/schema/...</stasis:ID>
        <cdm:SemanticReferencePath>/catalog/book/genre</cdm:SemanticReferencePath>
        <cdm:positionFrom>0</cdm:positionFrom>
        <stasis:shortName>genre</stasis:shortName>
        <cdm:mandatory>false</cdm:mandatory>
        <cdm:hasLabel>genre</cdm:hasLabel>
        <stasis:Name>genre</stasis:Name>
      </cdm:SimpleNode>
    </stasis:source>
  </stasis:SLS>
  <stasis:SLS rdf:about="http://stasisproject.net/model#TLSImpl_1416474881539">
    <stasis:source>
      <cdm:SimpleNode rdf:about="schema/...">
        <cdm:SchemaReferencePath>/catalog/book/title</cdm:SchemaReferencePath>
        <stasis:ID>http://www.stasis-project.net/schema/...</stasis:ID>
        <cdm:mandatory>false</cdm:mandatory>
        <cdm:SemanticReferencePath>/catalog/book/title</cdm:SemanticReferencePath>
        <cdm:orderInsideCompartment>0</cdm:orderInsideCompartment>
        <cdm:positionLength>0</cdm:positionLength>
        <cdm:hasLabel>title</cdm:hasLabel>
        <stasis:Name>title</stasis:Name>
        <cdm:positionFrom>0</cdm:positionFrom>
        <stasis:shortName>title</stasis:shortName>
      </cdm:SimpleNode>
    </stasis:source>
  </stasis:SLS>
  <stasis:SLS rdf:about="http://stasisproject.net/model#TLSImpl_1416474881499">
    <stasis:source>
      <cdm:SimpleNode rdf:about="schema/...">
        <cdm:positionLength>0</cdm:positionLength>
        <stasis:shortName>price</stasis:shortName>
        <cdm:SchemaReferencePath>/catalog/book/price</cdm:SchemaReferencePath>
        <cdm:hasLabel>price</cdm:hasLabel>
        <cdm:positionFrom>0</cdm:positionFrom>
        <cdm:mandatory>false</cdm:mandatory>
        <stasis:Name>price</stasis:Name>
        <stasis:ID>http://www.stasis-project.net/schema/...</stasis:ID>
        <cdm:SemanticReferencePath>/catalog/book/price</cdm:SemanticReferencePath>
        <cdm:orderInsideCompartment>0</cdm:orderInsideCompartment>
      </cdm:SimpleNode>
    </stasis:source>
  </stasis:SLS>
```

```
</rdf:RDF>”
}
```

Figure 69: Example of Structure Generated by the Content Gateways in OWL Format and Included in a JSON

The documentation demo page only requires the user to enter the name of the labels that the user wants to map to the SAM ontology (such as `price` or `title` in the example given in Figure 69) in the `body` textbox. The response of the service is a JSON providing a list of alignment suggestions between the label introduced and the concepts of the SAM ontology.

Figure 70 shows an example of suggestions for the label `price`.

```
{
  "@context": {
    "SAM": "http://sam.ascora.de/ontology/2015/0/SAM#",
    "cdm": "http://stasisproject.net/cdm#",
    "cdm_edi": "http://stasisproject.net/cdm_edi#",
    "cdm_xsd": "http://stasisproject.net/cdm_xsd#",
    "owl": "http://www.w3.org/2002/07/owl#",
    "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
    "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
    "stasis": "http://stasisproject.net/model#",
    "xsd": "http://www.w3.org/2001/XMLSchema#",
    "confidence": {
      "@id": "xsd:float",
      "@type": "xsd:float" }
  },
  "stasis:SLS": {
    "@id": "stasis:TLSImpl_1416475846635",
    "stasis:destination": [
      { "@ID": "http://sam.ascora.de/ontology/2015/0/SAM/price",
        "confidence": 0.7757333333333333 },
      { "@ID": "http://dbpedia.org/ontology/Person",
        "confidence": 0.4316666666666667},
      { "@ID": "http://dbpedia.org/ontology/writter",
        "confidence": 0.3757333333333333 },
      { "@ID": "http://dbpedia.org/ontology/Website",
        "confidence": 0.3195454545454545 },
      { "@ID": "http://sam.ascora.de/ontology/2015/0/SAM/series",
        "confidence": 0.30906666666666666 }
    ],
    "stasis:source": {
      "@id": "http://www.stasis-project.net/schema/...
      "stasis:Name": "price" }
  }
}
```

Figure 70: Mapping Result in JSON Format for Label `price`

5.4.1.1.1.2 Content Characterisation

In the first prototype, the main contribution to this task is the development of an entity linking method that identifies mentions to Wikipedia entities in an input text. A demonstration web page has been developed to test this functionality and will be available under:

```
http://localhost:8080/SemanticServices/
```

Figure 71 shows a screenshot of the web interface. For an input text, the service retrieves a list of suggestions for every Wikipedia entity detected in its content. The list of suggestions is ordered by a confidence score between 0 and 1, indicating the level of trust in the prediction given. In the current approach, this confidence score is based on two key features. The first one is the number of incoming links to the Wikipedia article - more links implies more relevance; the second one is the similarity between the context (list of words adjacent to the target entity) of the entity found in text and the description of that entity in Wikipedia. For this purpose, the Lesk disambiguation algorithm has been employed [LES86].

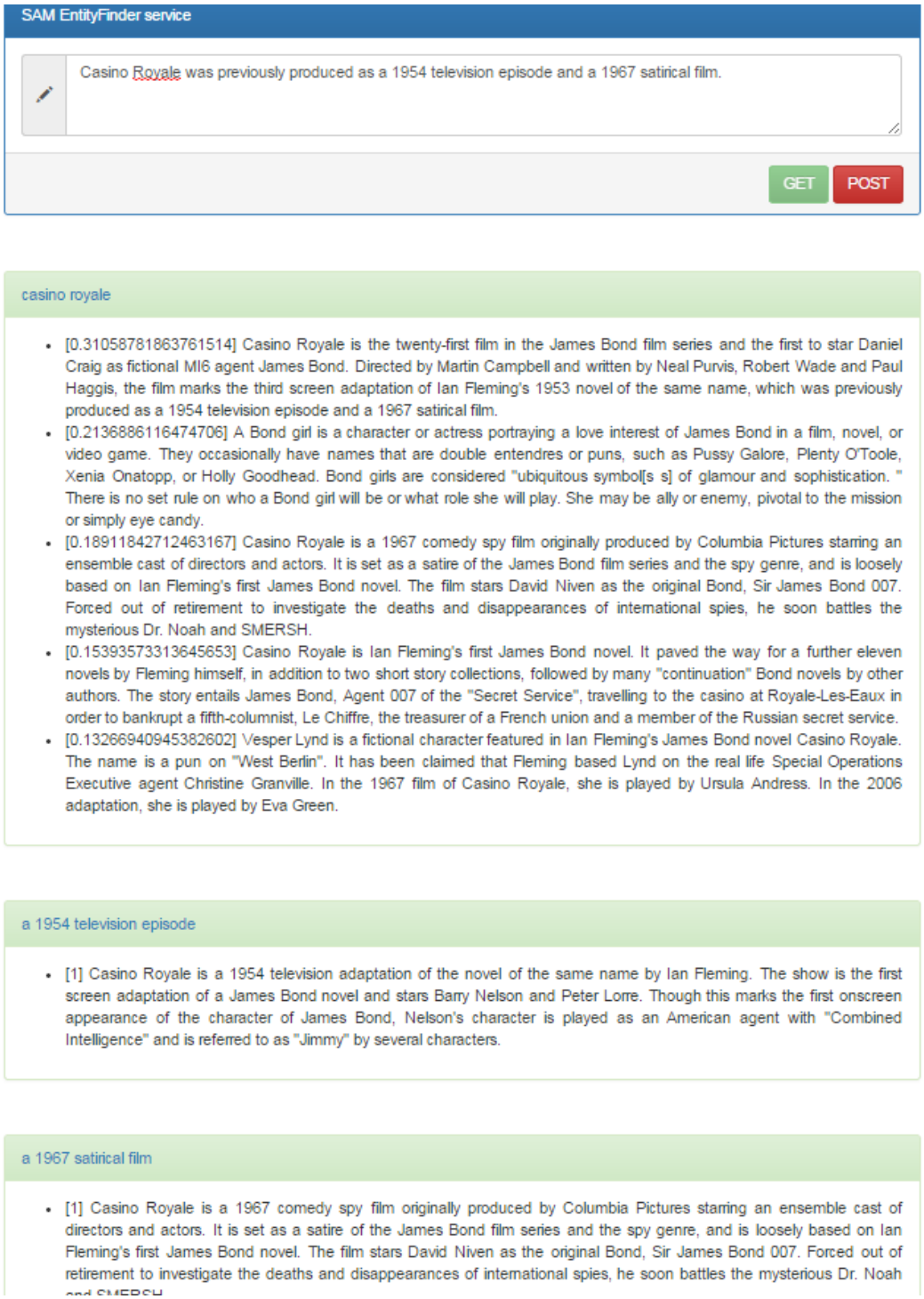

Figure 71: Identification of Entities in Wikipedia Occurring in an Input Text

As mentioned in Figure 66, another task to be carried out by the Content Characterisation subcomponent is the identification of Asset mentions in text (both Asset's content and UGCs). In the current prototype, the only available version of these functionalities is a mock-up. The documentation demo page for the identification of Asset mentions in text obtained from Asset content can be accessed and tested in the following URL:

```
http://localhost:8080/SemanticServices/doc/#!/characterise/asset
```

The input for this service is a JSON object which includes an Asset object and a list of Asset's attributes named `path`. This value indicates what attribute's content should be

analysed by the characterisation process. As a result of this operation, a list of Asset and Wikipedia entities will be obtained.

Similarly, the demo page for the identification of Asset mentions in UGC can be accessed and tested in this URL:

```
http://localhost:8080/SemanticServices/doc/#!/characterise/ugc
```

In this case, the input for this service is a JSON object including both the `ugc` to analyse and the `context (`a list of keywords that identify those Assets that are being used at the same context[16]) where the `ugc` has been generated. As a result of this operation, a list of Assets and Wikipedia entities will be obtained.

As already mentioned in Figure 68, in the `Parameters` section the `body` textbox can be filled with a JSON example, checking the result using the `Try it out!` button. The section `Response Messages` provide information on the possible responses when the process is carried out.

Future work on these two functionalities is described on Section 5.5.

5.4.1.1.2 Asset Discovery

This section describes the two main operations of this subcomponent. The first one is to provide Asset suggestions based on the functionalities provided by internal subcomponents. This functionality is not available in this first prototype. Further information on the development of this service can be found in Section 5.5. Although the functionality is not available, a mock-up documentation demo page has been developed to provide input and output information of the service. This mock-up can be accessed in the following URL:

```
http://localhost:8080/SemanticServices/doc/#!/assetdiscovery/asset
```

This interface (see Figure 72) requires as input a JSON object containing a set of keywords and context information related to the Asset requested, together with information on the user profile to customise the search. As result, a list of Assets will be retrieved which are coded as JSON-DL (see Figure 73).

[16] This context refers those Assets that are being consumed by an end user in the Dashboard.

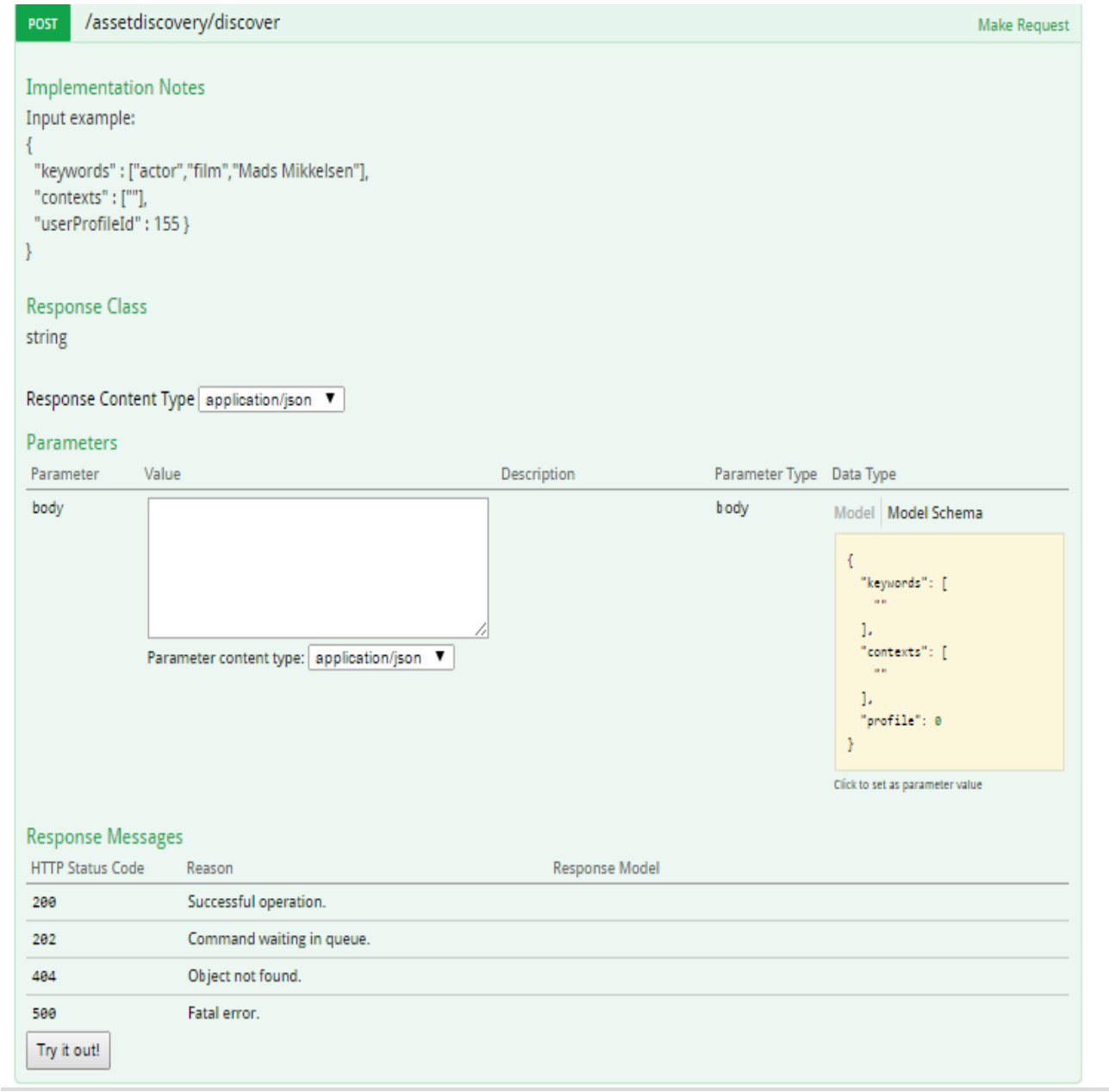

Figure 72 Asset Discovery Mock-up RESTful Interface

```
{
  "list":[  {
     "@context": {
       "sam": "http://sam.ascora.de/sam/SAM_Ontology.owl"
     },
     "@type": "sam:Asset",
     "@id": "sam:Mads_Mikkelsen",
     "id": "150",
     "name": "Mads Mikkelsen",
     "owner": "sam:BDS",
     "assetLanguage": "EN",
     "assetSourceFormat": "avi",
     "URL": https://www.youtube.com/watch?v=uv1cDiF9dUk,
     …
  }]
}
```

Figure 73: JSON List of Assets in JSON-LD Format

The second functionality provided by this subcomponent is the Asset exploration and management based on the CRUD operations exposed by internal Semantic Services subcomponents (i.e. Semantic Resource Explorer). The version available of this subcomponent for the first prototype already includes all these CRUD operations.

Each operation involves a RESTful interface and a documentation demo page, where information on how to proceed with the CRUDs and Asset attributes exploration functions is given. In the `Parameters` section of each operation, the `body` textbox can be filled with a JSON example, checking the result using the `Try it out!` button. The section `Response Messages` provides information on the possible responses of the service when the execution is carried out.

The RESTful CRUDs interfaces provide the following functionalities: `load`, `delete`, `create` and `update`. The `load` method selects Assets instances by using filters and indicating what Assets' attributes have to be included into the queried Assets (see Figure 74). The documentation web page can be accessed in the following URL:

```
http://localhost:8080/SemanticServices/doc/#!/assetdiscovery/load
```

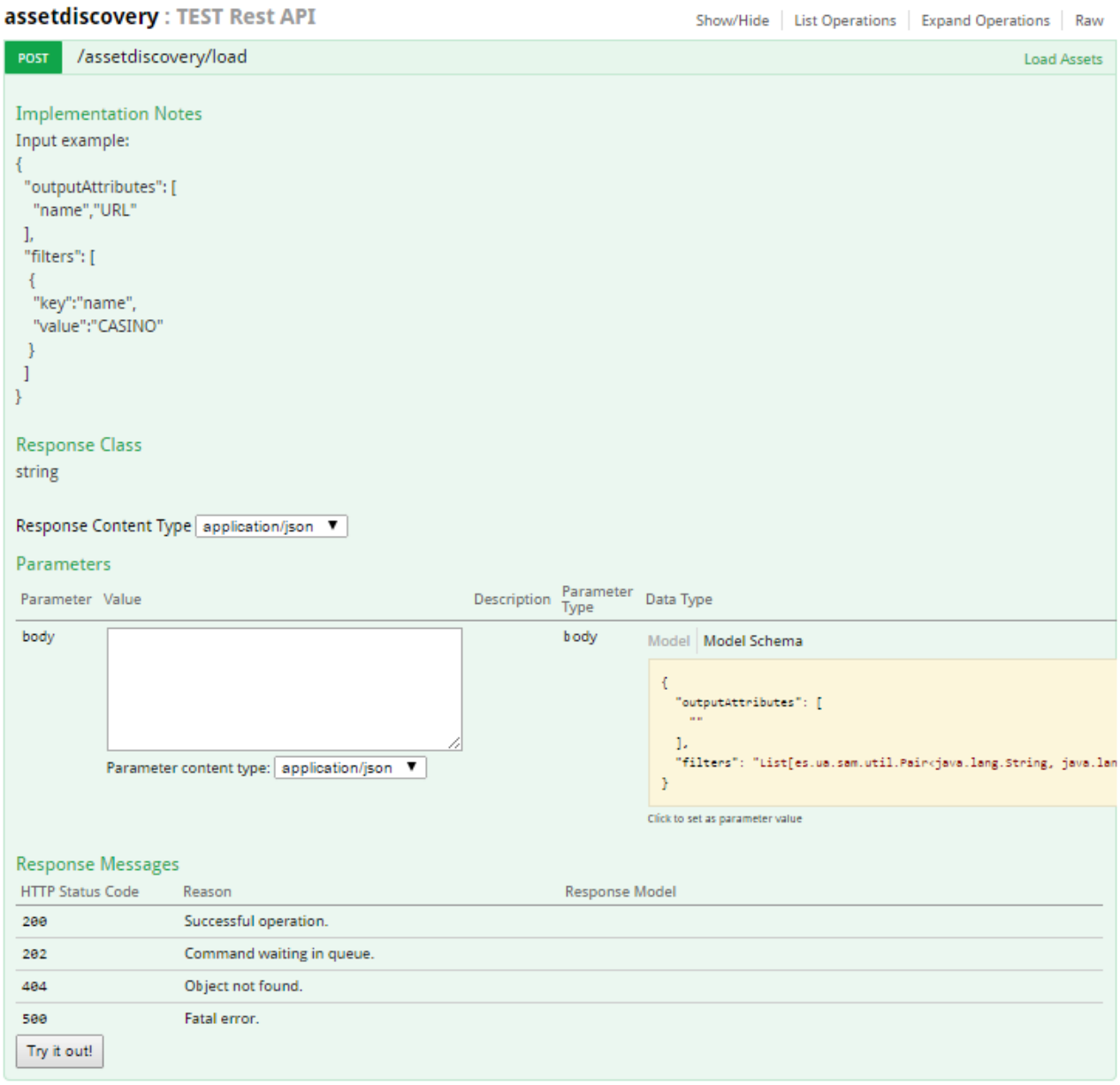

Figure 74 Load Assets RESTful Interface

This interface provides an example of the JSON input and output formats. The result of this method is a collection of Assets which attributes match the value specified in the `outputAttributes` field (see Figure 75). Using this method, the Assets obtained will just include the specified attributes (`name` and `URL` shown in Figure 76), disregarding attributes such as `id`, `owner`, `assetLanguage`, `assetSourceFormat`, and others (see the differences between attributes in Figure 73 and Figure 76).

```
{
  "outputAttributes": ["name", "URL"];
  "filters": [{ "key": "name"; "value": "Casino"}]
}
```

Figure 75: Load Data Input Example in JSON Format

```
{
  "list":[{
    "@context": {
      "sam": "http://sam.ascora.de/sam/SAM_Ontology.owl"
    },
    "@type": "sam:Asset",
    "@id": "sam:Casino_Royale",
    "name": "Casino Royale",
    "URL": "https://www.youtube.com/watch?v=uv1cDiF9dUk"
  }]
}
```

Figure 76: JSON List of Assets in JSON-LD Format only including Selected Attributes

The operation `delete` removes an Asset instance by indicating its unique identifier (see Figure 77). The documentation web page of this service can be found in the following URL:

```
http://localhost:8080/SemanticServices/doc/#!/assetdiscovery/delete
```

This interface provides an example of the JSON input (an Asset identifier) and output (a Boolean value indicating if the operation was successful) formats.

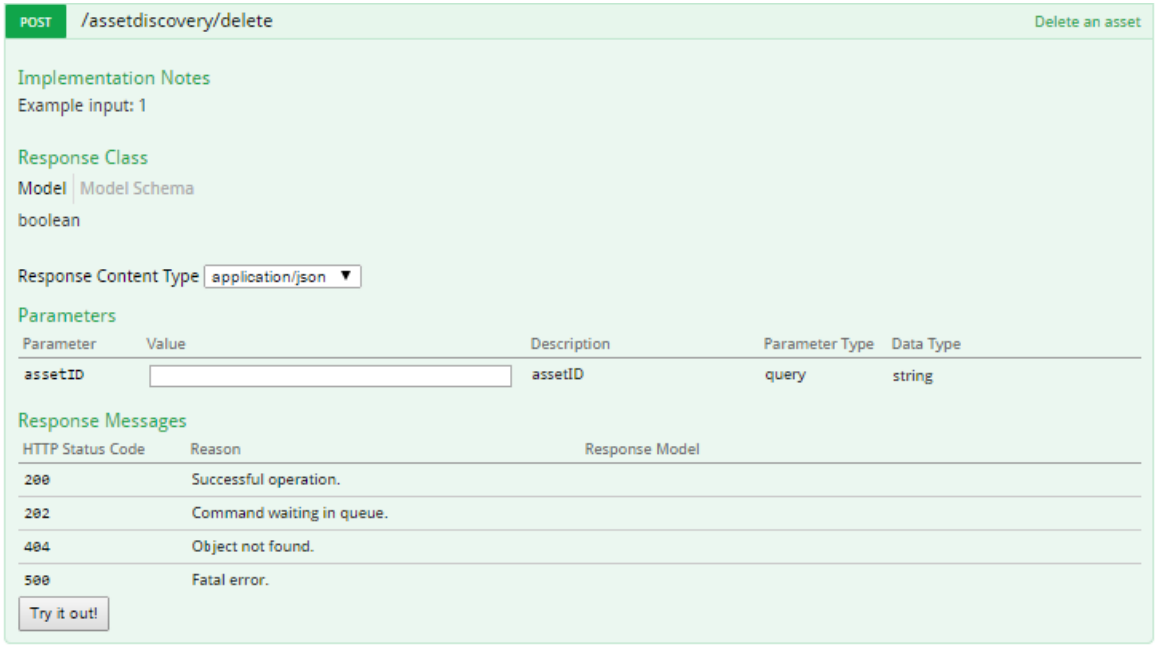

Figure 77: Delete Asset RESTful Interface

The operation `create` stores a specific Asset into the Cloud Storage (see Figure 78). The documentation web page of this service can be accessed in the following URL:

```
http://localhost:8080/SemanticServices/doc/#!/assetdiscovery/create
```

This interface provides an example of the JSON input (an Asset in JSON-LD format) and output (a unique identifier assigned to the new Asset stored which has string format).

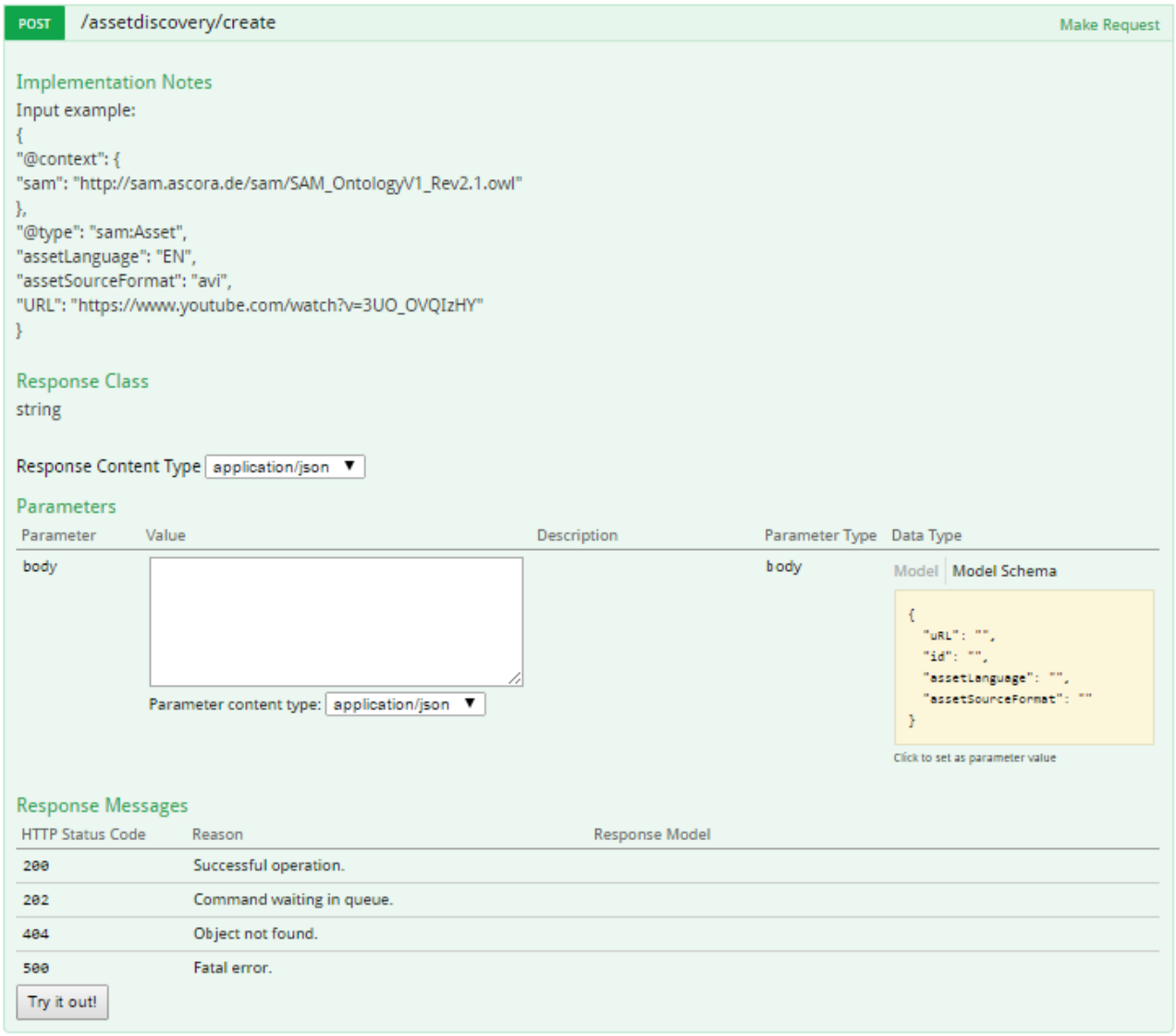

Figure 78: Create Asset RESTful Interface

Finally, the operation `update` allows the changing of content of an existing Asset by submitting a modified Asset instance (see Figure 79). The documentation web page of this service is available in the following URL:

```
http://localhost:8080/SemanticServices/doc/#!/assetdiscovery/update
```

This interface provides an example of the JSON input (an Asset in JSON-LD format) and output (a Boolean value indicating if the operation was successful).

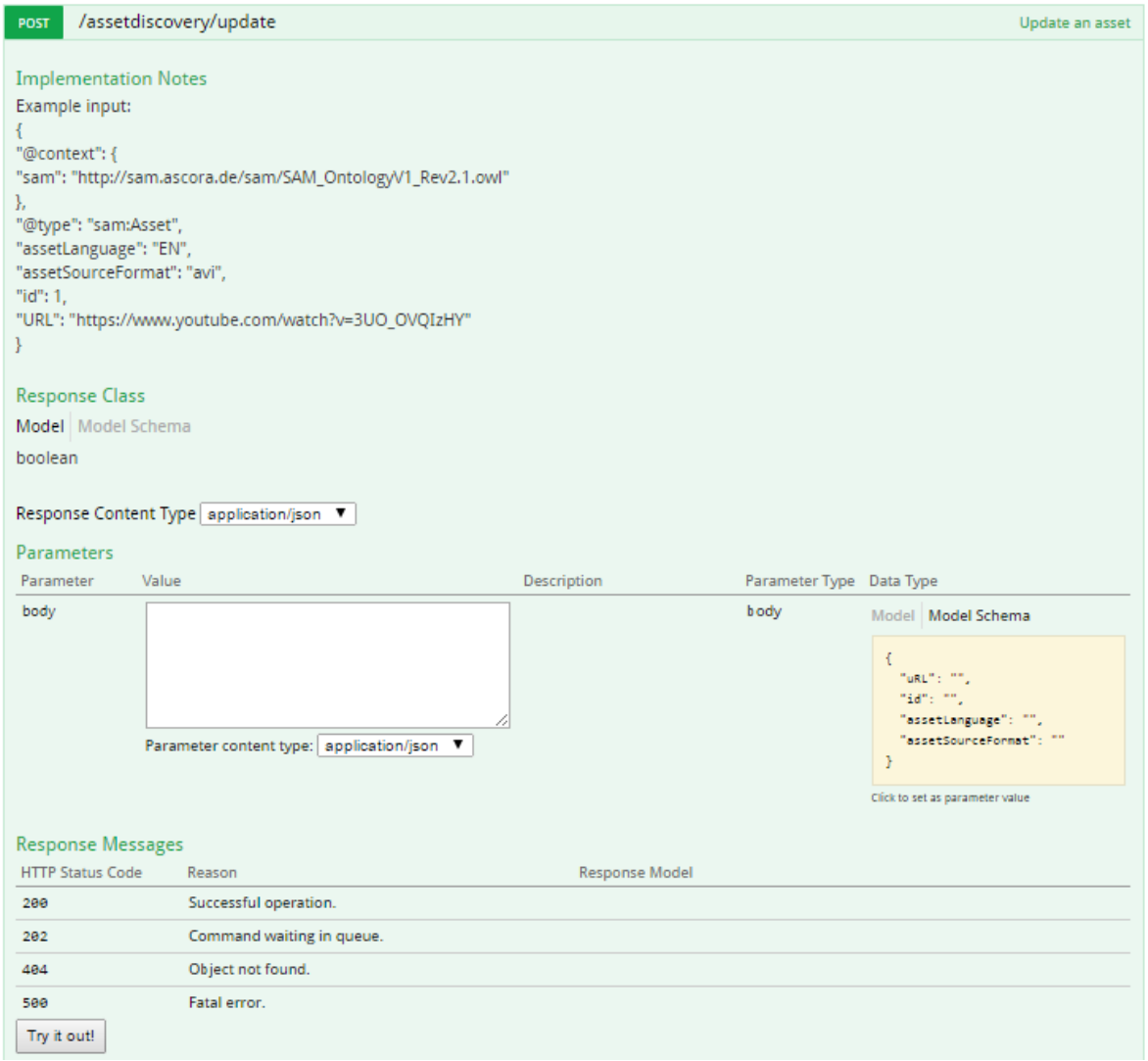

Figure 79: Update Asset RESTful Interface

5.4.1.1.3 Text Summarisation

This RESTful interface provides functionalities for summarising large amounts of UGC, keeping only the most representative comments provided by the users regarding specific Assets. In the first prototype this functionality has been implemented as a mock-up. The interface for this mock-up can be accessed in the following URL:

```
http://localhost:8080/SemanticServices/doc/#!/textsummarisation/summarise
```

As shown in Figure 80, the input of this operation consists on a JSON object with the following attributes:

- `ugc`: It refers UGC, comments provided by the users regarding specific Assets
- `compressionRatio`: Numeric value to indicate the reduction percentage of the `ugc` to keep only the most representative comments in the output

As a result, this interface provides a JSON object with an attribute named `summary`, where the most representative comments included into the `ugc` are represented (see Figure 81).

```
{
  "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...",
  "compressionRatio" : "20"
}
```

Figure 80: Summarisation Input in JSON format

```
{
  "summary": "Casino Royale is a great movie."
}
```

Figure 81: Summarisation Output in JSON format

5.4.1.1.4 Sentiment Analysis

As in previous subcomponents, there is a documentation demo page exposing the Sentiment Analysis functionalities:

```
http://localhost:8080/SemanticServices/doc/#!/sentimentanalysis/analyse
```

Figure 82 shows a screenshot of this interface. In the `Parameters` section the `body` textbox can be filled with a JSON example, checking the result by clicking on the `Try it out!` button. The Sentiment Analysis system developed is based on a combination of different ranking algorithms with skip-gram patterns extraction [FGG13].

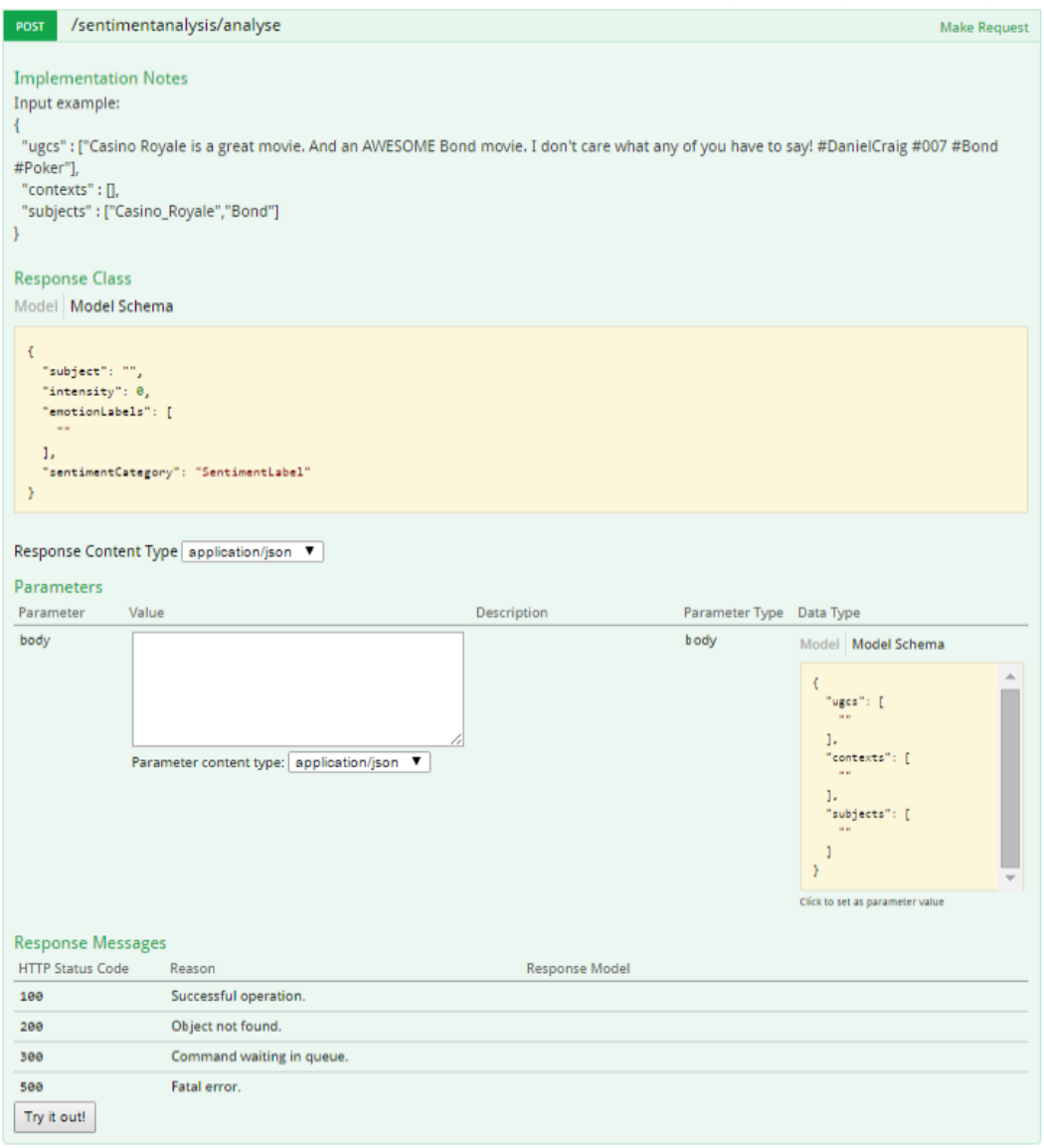

Figure 82: Sentiment Analysis RESTful interface

Figure 83 shows an example of the input for this service. The input JSON object for this service includes the following attributes:

- `ugc`: It represents the user comments to be analysed
- `context`: A list of keywords that identifies the context where the user comments were carried out
- `subject`: A list of target entities to perform aspect-based Sentiment Analysis (i.e., analyse the opinion about specific features of an Asset, such as the performance of an actor in a film)

```
{
  "ugc" : ["Casino Royale is a great movie. But a bad Bond movie. #DanielCraig #007
#Bond #Poker"],
  "contexts" : [],
  "subjects" : ["Casino_Royale","Bond"]
}
```

Figure 83: Data Input Example in JSON Format

A JSON object is obtained as an output (see Figure 84) providing both overall (e.g. "subject": "OVERALL") and specific subject (e.g. "subject": "Casino Royale") intensity (e.g. "intensity": 0.8310321) and polarity (e.g. "sentimentCategory": "positive") scores. There is also information regarding emotion labels (emotionLables), although this functionality is not available in this first prototype.


```
{
  "semanticAttributes": [
    [
      { "subject": "OVERALL",
        "intensity": 0.647352,
        "emotionLabels": null,
        "sentimentCategory": "positive"
      },
      { "subject": "Casino_Royale",
        "intensity": 0.8310321,
        "emotionLabels": null,
        "sentimentCategory": "positive"
      },
      { "subject": "Bond",
        "intensity": 0.52452224,
        "emotionLabels": null,
        "sentimentCategory": "negative"
      }
    ]
  ]
}
```


Figure 84: Data Output Example in JSON Format

Finally, the section Response Messages of the documentation page provides information on the possible responses of the service when the execution is performed.

5.4.1.2 Asset Profiler

The Semantic Services component includes a web user interface, the Asset Profiler, to facilitate Asset creation and edition. The final prototype will include three tabs for profiling Assets:

- main tab: It allows setting main characteristics of the Assets, such as title, owner and language (See Figure 85)
- semantic tab: It allows automatic characterisation of Assets in order to connect them with other Assets and external sources, such as Wikipedia entries (See Figure 86)
- social tab: It allows linking Assets with social media channels like Facebook and Twitter

In this first prototype the main and semantic tabs have been developed as mock-ups. As part of this subcomponent, the Asset Profiler backend is being developed in parallel by providing data views to the user interface.

As a result of the edition process, an Asset can be created, updated, or deleted from the SAM platform.

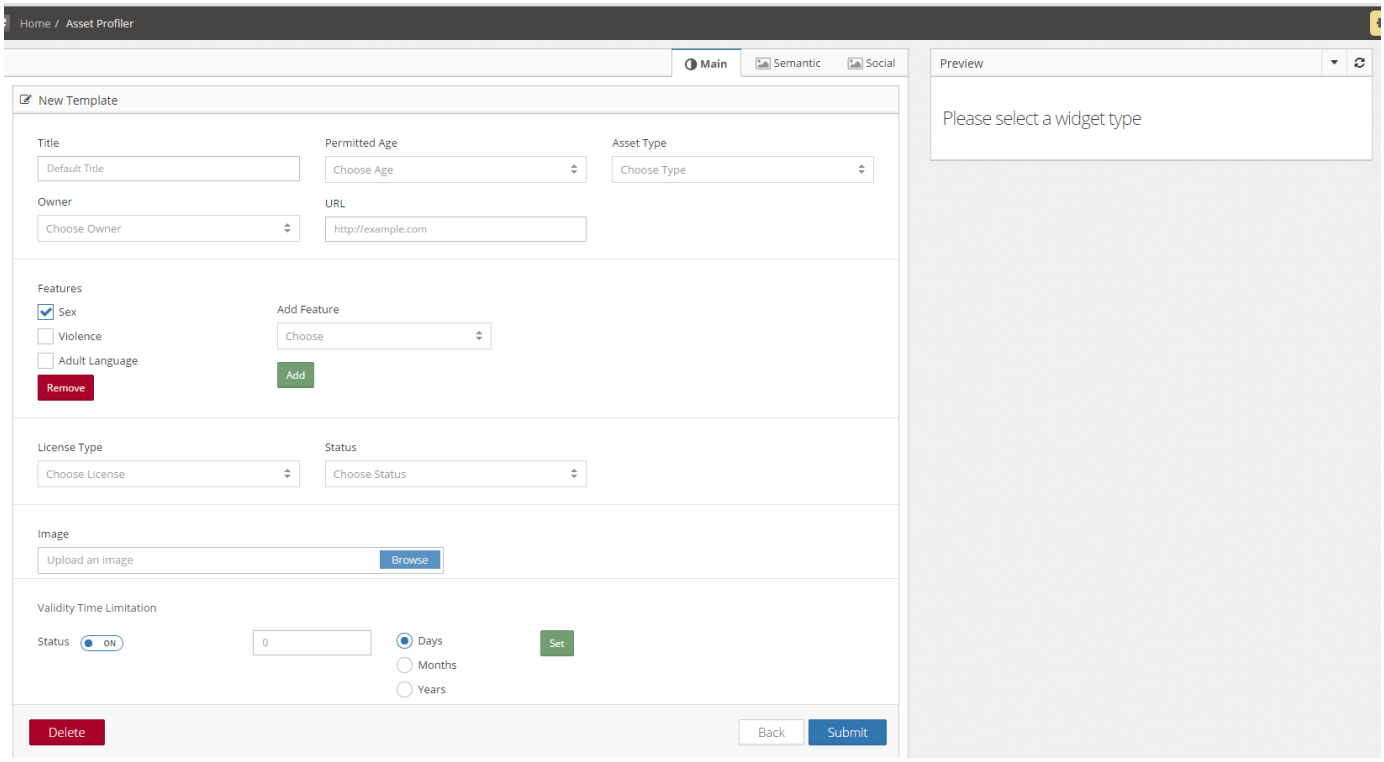

Figure 85: Main Tab of the Asset Profiler Mock-up

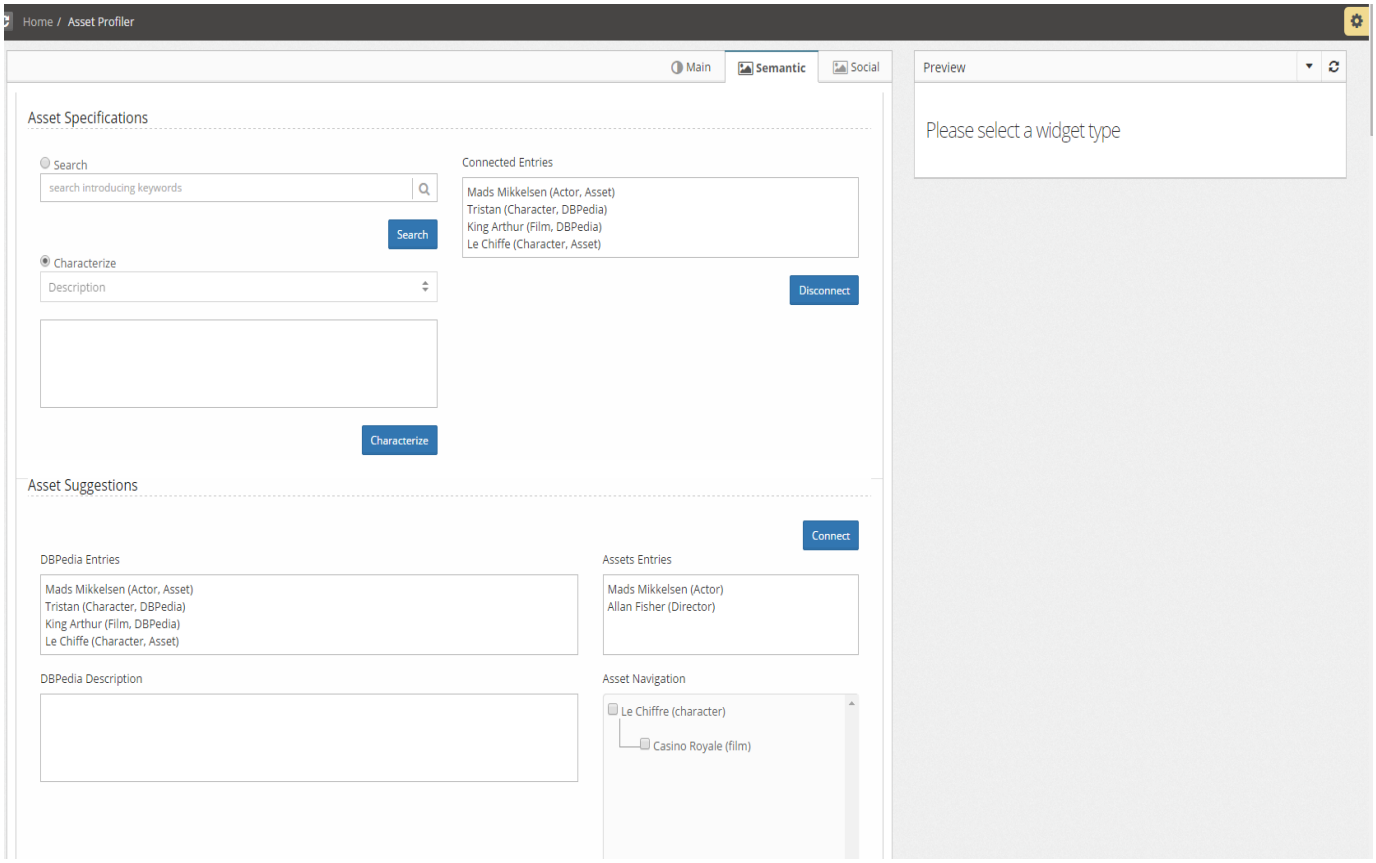

Figure 86: Semantic Tab of the Asset Profiler Mock-up

5.5 Limitations and Further Developments

The list of limitations presented below is conditioned by the current status of the prototype, i.e., they should be overcome in the next version (as described in Section 5.5.1):

- **Asset Discovery:** For the current prototype version this subcomponent will not provide functionalities for discovering Assets, since it requires fully operative CRUD operations on Assets to start its development.
- **Ontology Mapping:** The mapping service is working only on partner BDS datasets. The approach should be applied and tested on different sample sets.
- **Content Characterisation:** In the current prototype, the identification of Asset mentions in text is not yet implemented. A mock-up is provided for the Content Characterisation service that always offers the same response when queried.
- **Asset Profiler:** The frontend of this interface currently provides two out of three tabs. The development of the Social tab is not included in this first prototype. The backend is not available yet, since it depends on the progress of the internal subcomponents (i.e. Semantic Resources Explorer) and the CRUD operations on Assets in the Cloud Storage.
- **Sentiment Analysis:** The current service does not provide emotion detection on UGC.
- **Text Summarisation:** This functionality is not available in this first prototype.

5.5.1 Prototype 2 Planned Tasks

Figure 87 summarise the tasks planned for the second prototype of the Semantic Services (to be delivered in M25).

Subcomponent	Task
Asset Discovery	Implement the retrieval of specific Assets from the Cloud Storage.
Data Characterisation	Identification of mentions to Assets stored in SAM in text obtained from both Assets' content and UGC. Extend ontology mapping to a variety of input data structures.
Sentiment Analysis	Identify emotions on UGC.
Text Summarisation	Provide a first approach to Text Summarisation functionality on UGC.
Asset Discovery	Implement a first approach to infer Assets from given keywords. Explore the SAM ontology.
Asset Profiler	Create the social tab and connect the frontend with the backend.

Figure 87: Tasks Planned for the Second Prototype

5.6 Research Background

For the current prototype implementation and the overall approach the following related research work was taken into consideration:

Source	Subcomponent	Description
[SE13] Shvaiko, P., & Euzenat, J. Ontology matching: state of the art and future challenges. Knowledge and Data Engineering, IEEE Transactions on, vol 25 (1), pp 158-176, 2013.	Data Characterisation	This paper has been considered to implement the Semantic Analysis in the ontology mapping process.
[LES86] Lesk, M. Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In SIGDOC '86: Proceedings of the 5th annual international conference on Systems documentation, pp 24-26, New York, NY, USA. ACM, 1986.	Data Characterisation	This paper has been considered to apply word sense disambiguation algorithms for the entity linking functionality.
[GVM11] Gutiérrez, Y.; Vázquez, S.; Montoyo, A. Improving WSD using ISR-WN with Relevant Semantic Trees and SemCor Senses Frequency. In proceedings of Recent Advances in Natural Language Processing, pp 233–239, Hissar, Bulgaria, 2011.	Data Characterisation	This paper has been considered to apply disambiguation algorithms for the Data Characterisation functionality.
[FGG13] Fernández, J.; Gutiérrez, Y.; Gómez, JM.; Martínez-Barco, P.; Montoyo, A.; Muñoz, R. Sentiment Analysis of Spanish Tweets Using a Ranking Algorithm and Skipgrams. Journal Sociedad Espanola de Procesamiento de Lenguaje Natural, pp 133-142, 2013.	Sentiment Analysis	This paper has been considered in order to adapt Sentiment Analysis algorithms to the SAM scenario.
[LEV66] Levenshtein, V. I. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. In Soviet Physics Doklady vol. 10, pp. 707, 1966	Data Characterisation	This paper has been considered in order to implement the ontology mapping process and measuring the lexical distance between terms.

Figure 88: Research Background for the Semantic Services

5.7 Target Performance

For this component the following key performance indicators (KPI) have been defined:

Topic	Description	Target KPI
Ontology Mapping Performance	The structure of imported data in SAM should be mapped to the concepts of the ontology defined in the project. The Data Characterisation component has to provide a list of suggestions for this mapping. Content providers receive these suggestions and manually decide the correct mapping.	The number of suggestions for every label will be **less or equal to 5**, thus limiting the number of mappings that the content provider has to review. The system will be able to provide the correct label for mapping in this set of five elements with a **precision of 75%**.
Entity Linking Performance	Given a text, the Data Characterisation subcomponent is able to identify mentions to Wikipedia entities in its content. The system provides a list of suggestions of entities to link, and the contents providers have to manually decide which ones they want to link to their content.	The number of suggestions for every Wikipedia entity detected will be **less or equal to 5**, thus limiting the number of suggestions that the content provider has to review. The system will be able to correctly provide a valid entity in this set of five elements with a **precision of 60%**. The **recall** of the system (the ability to effectively detect these entities) will be set to **70**%.
Sentiment Analysis Performance	The Sentiment Analysis subcomponent is able to identify intensity and polarity in user comments. It also provides aspect-based analysis, i.e., identifies opinions made by users on specific features (i.e. the user could submit a comment with a positive opinion on a film, but a negative opinion on a particular actor performing on that film).	The system will be able to effectively detect the **global polarity** of a comment with a **precision of 50%**. Relative **intensity** (whether an opinion is stronger than other) will be detected with a **precision of 60%**. Aspect-based analysis will be performed with a precision of 40%.

Figure 89: Target Performance Semantic Services

5.8 Summary

This section provided a description of the first prototype of the Semantic Services component developed in task T4.3 (Data Characterisation and Context Management). The main outcomes of this task are the different subcomponents that provide natural language processing functionalities to other components in the platform. These subcomponents are: Data Characterisation, Asset Discovery, Sentiment Analysis, Text Summarisation and Asset Profiler.

The development of most of these subcomponents has already started. These developments include a documentation demo web page in order to test their current functionalities and status. This section also presented instructions about installation and deployment of the Semantic Services.

The last subsections describe the limitations of the current prototype, the next steps to carry out for the second version of the component (which should be delivered in M25), and the performance indicators for the components created.

6 Distributed Identity, Trust and Security

This section describes the software deliverable D4.4.1, which is the first prototype release of the SAM Identity, Trust and Security functionalities.

6.1 Scope and Relationship

The Identity, Trust and Security functionalities for SAM are implemented in the SAM Identity and Security Services (ISS) component. Figure 90 shows the subcomponent of the complete Identity and Security Services component as envisioned for a post-prototype system. Subcomponent boxes drawn with dotted lines indicate subcomponents that are not in the focus of development within the SAM project. These subcomponents will not be implemented during the project but will be required for post-prototype commercialisation efforts.

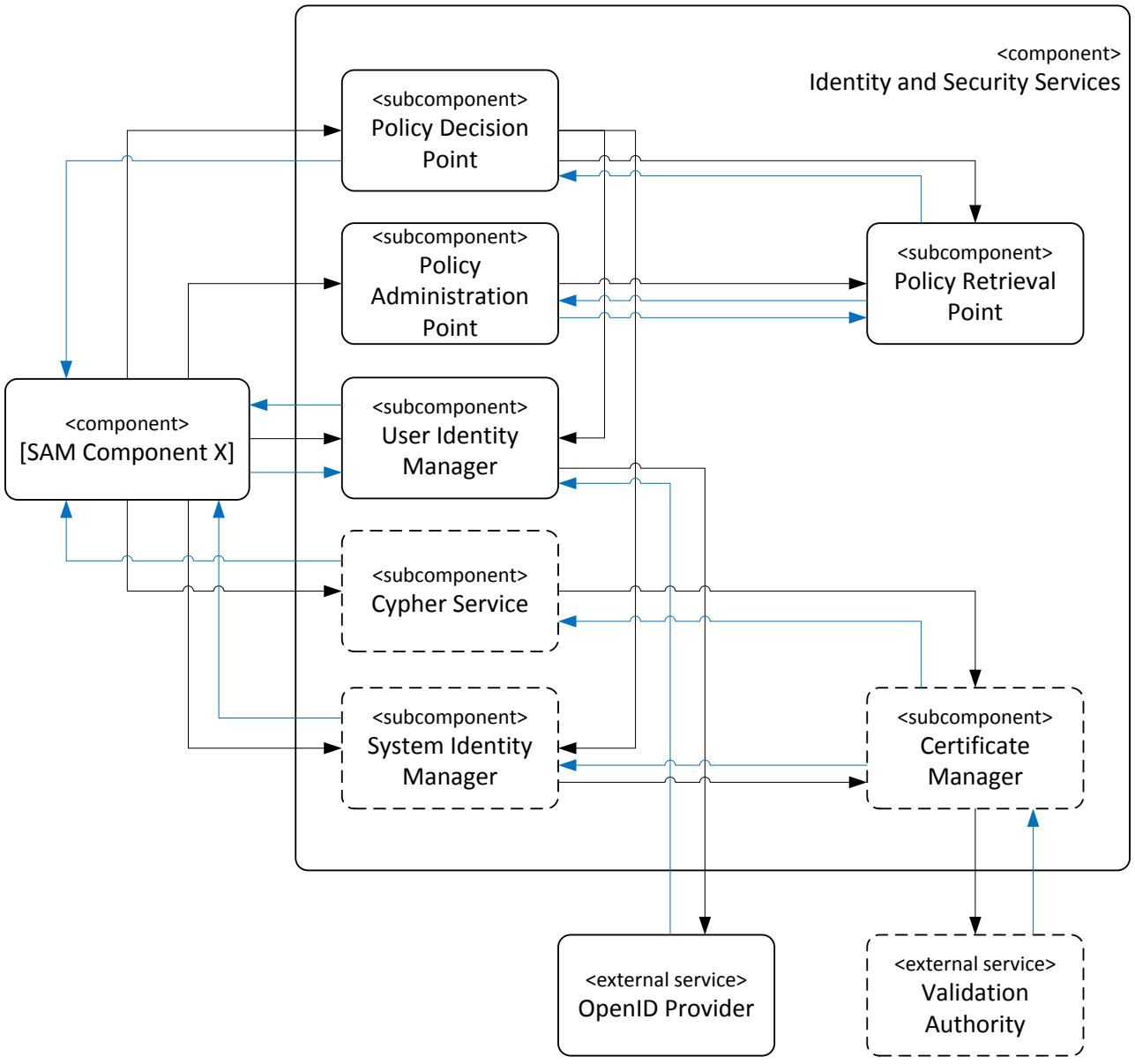

Figure 90: Overview of Identity and Security Services Subcomponents

For further description of the functional and technical foundations of these subcomponents, please revisit documents D3.2.1 Section 4.9 (Architecture), D3.2.2 Section 4.10 (Functional Specification) or D3.3.1 Section 3.10 (Technical Specification).

The first prototype of Task T4.4 is concerned with the implementation of the core functionalities for identity management; hence the focus of the first prototype has been placed on the supply of a functioning OpenID provider and the implementation of the core user identity management functionalities.

A summary of the tasks carried out for the first version of the prototype is shown in the table below:

Subcomponent	Task
User Identity Manager	Host defined RESTful Web Services for user identity management; process authentication requests through OpenID provider
OpenID Provider	Select, install and configure suitable OpenID server for user identity management; test installed OpenID provider; enable use of OpenID provider with SAM user interfaces

Figure 91: Tasks Completed for the First Prototype of T4.4

6.2 Requirements and Preparations

This section provides information on the technical and non-technical (e.g. organisational or administrative) requirements for personnel involved in the deployment of an instance of the ISS component.

Two main scenarios can be differentiated in terms of deployment: installation of the ISS component using an existing installation of a SAM Gluu server instance and installation of the ISS component together with a new installation of a SAM Gluu server instance.

6.2.1 SAM Gluu Server Instance

When hosting a new installation of a SAM Gluu server instance, the following system configuration is recommended for deployment of the Gluu server instance:

- Server hardware with at least two CPU cores assigned to running the Gluu server, 4 GB RAM memory and 8 GB disk space
- Ubuntu Linux 14.04 “Trusty” (exact version required)
- Oracle Java JDK or OpenJDK version 1.7 or later

6.2.2 ISS Component

The ISS component has been implemented as a Java Servlet. The component requires a suitable release of the Apache Tomcat Servlet engine to be installed and configured to run as a server instance. The ISS component has been developed for and tested with Apache Tomcat 8.0 on Ubuntu Linux. It is recommended to use Apache Tomcat 8.0 for deployments of the ISS component. While compatibility with Apache Tomcat 8.0 on other operating systems (in particular Microsoft Windows) is not explicitly tested during development, the component should generally be readily deployable to Apache Tomcat 8.0 on other operating systems than Ubuntu Linux if required.

The component is developed using the Oracle Java Development Kit 1.7. This or a compatible release of a certified Java development kit is required to run both the Apache Tomcat 8.0 server and the ISS component.

The port on which the ISS component operates can be configured in the Servlet container configuration if needed. Any existing firewalls should be configured to allow external access to the relevant HTTP/HTTPS ports specified for the Apache Tomcat server that will host the ISS component.

6.3 Installation (Deployment)

This section provides guidelines on how to install and deploy the prototype ISS component installation. It also provides references concerning the installation and post-installation steps required for the installation of a SAM Gluu server instance.

6.3.1 SAM Gluu Server Instance

In order to deploy a SAM Gluu server instance, the prerequisites described in Section 6.2.1 need to be addressed. Describing the complete installation sequence required for a Gluu server instance is beyond the scope of this document. References for further documentation from the Gluu project website are given where needed to provide the relevant detailed information. The following major steps are required:

1. Installation of the Gluu server package for Ubuntu Linux as described at http://www.gluu.org/docs/admin-guide/installation/ubuntu/
2. Generation and signing of the security certificates that are required for the Gluu server; this is an administrative process that needs to be carried out in cooperation with your Internet domain services provider. Please note that a valid certification that is signed by a generally recognised certification authority is required for the operation of a Gluu server instance.
3. Configuration of the Gluu server package with the generated and signed security certificate as described in http://www.gluu.org/docs/admin-guide/certificates/

6.3.2 ISS Component

The SAM ISS component is implemented as a Java servlet. For the first prototype version, several files need to be configured within the SAM ISS component Web Application Archive (WAR) if setup options other than the assumed default options are to be used. The following files may need to be modified:

- Servlet deployment descriptor in “WEB-INF/web.xml” if the servlet mapping needs to be modified (e.g. in order to avoid clashes in naming)
- ISS servlet configuration file in “WEB-INF/classes/iss.conf” if the component connection parameters to the SAM Gluu server instance needs to be changed.

It is recommended to edit configuration files within the WAR archive file as opposed to modifying files after the deployment of the servlet to a servlet container.

Once it has been configured as required, the ISS Java WAR archive can be installed via the usual installation mechanisms provided by servlet containers, including deployment to folders or manual installation via servlet container user interface components.

The ISS component built through the project build environment is configured to support the project-internal server installations of Gluu and the SAM TSB. If these are to be used, then the WAR file configuration does not have to be modified.

6.4 Execution and Usage

This section describes how to use the first prototype of this component.

6.4.1 Server Side (Administrators)

If a custom installation of the Gluu server is to be run, the following actions need to be taken:

1. Execute the service start script for Gluu through the command

```
$Gluu-Folder\bin\service gluu-server start
```

Figure 92: Command to Start the MongoDB Configuration Console

2. Observe console output to identify whether any error messages are shown for the Gluu server instance (in particular whether the server can bind to the relevant server ports).

The ISS component, once installed on a suitable servlet container, will automatically be started together with the servlet engine. For Apache Tomcat 8.0, the following start-up script should be executed to start the server instance (for the *NIX version of the server). The Tomcat servlet engine should be configured to start automatically on system start-up in a production environment (see http://tomcat.apache.org/tomcat-8.0-doc/setup.html).

The Gluu server administration interface provides functionalities to configure components as valid counterparties for communicating with the server. For SAM identity management, it will only be necessary to configure the SAM dashboard used to log in end users, the SAM administration user interface and the ISS component itself. The following user interface dialogue is used to configure additional components as clients with which users can interact:

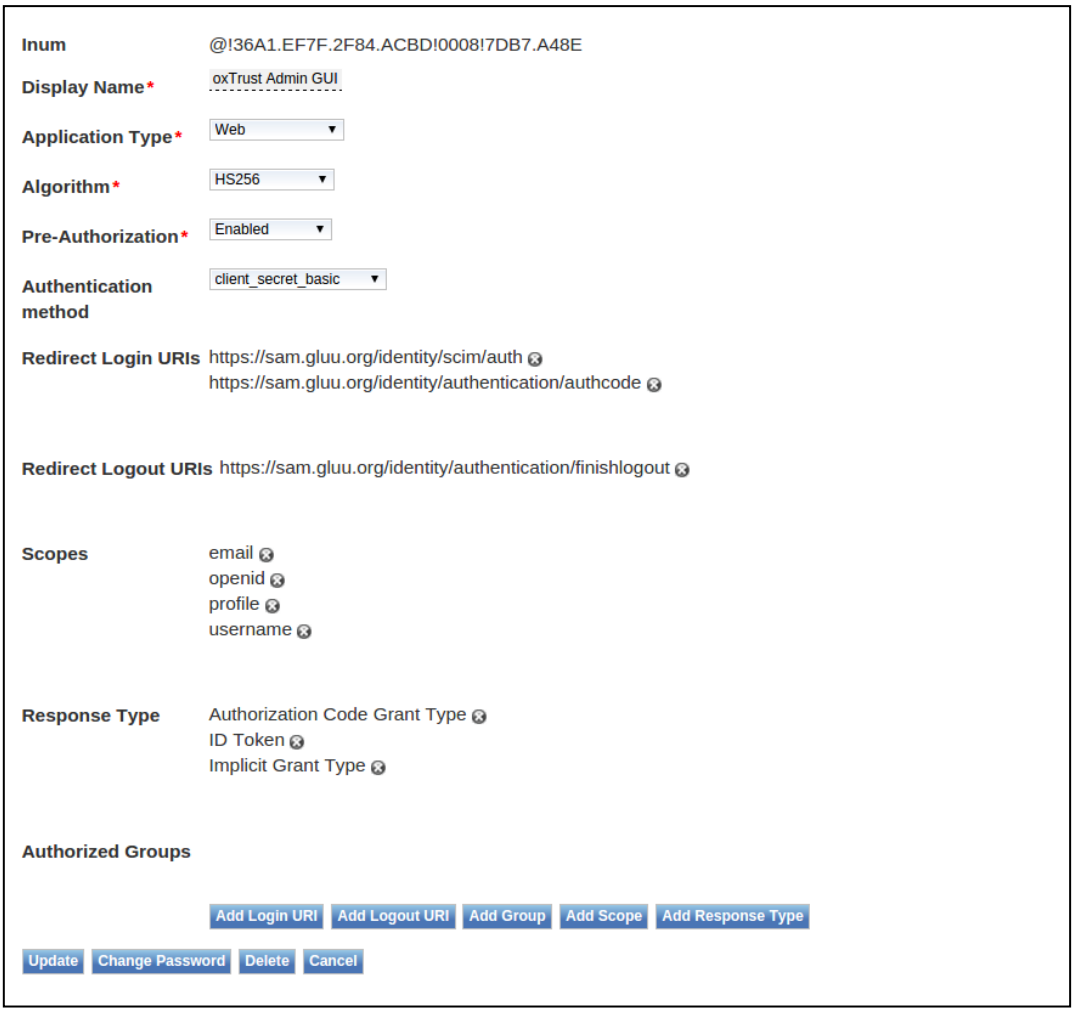

Figure 93: Gluu Client Addition User Interface

The Gluu server user interface furthermore allows the management of user accounts and user groups, which can be configured as shown in the user interface screenshots Figure 94 and Figure 95 below.

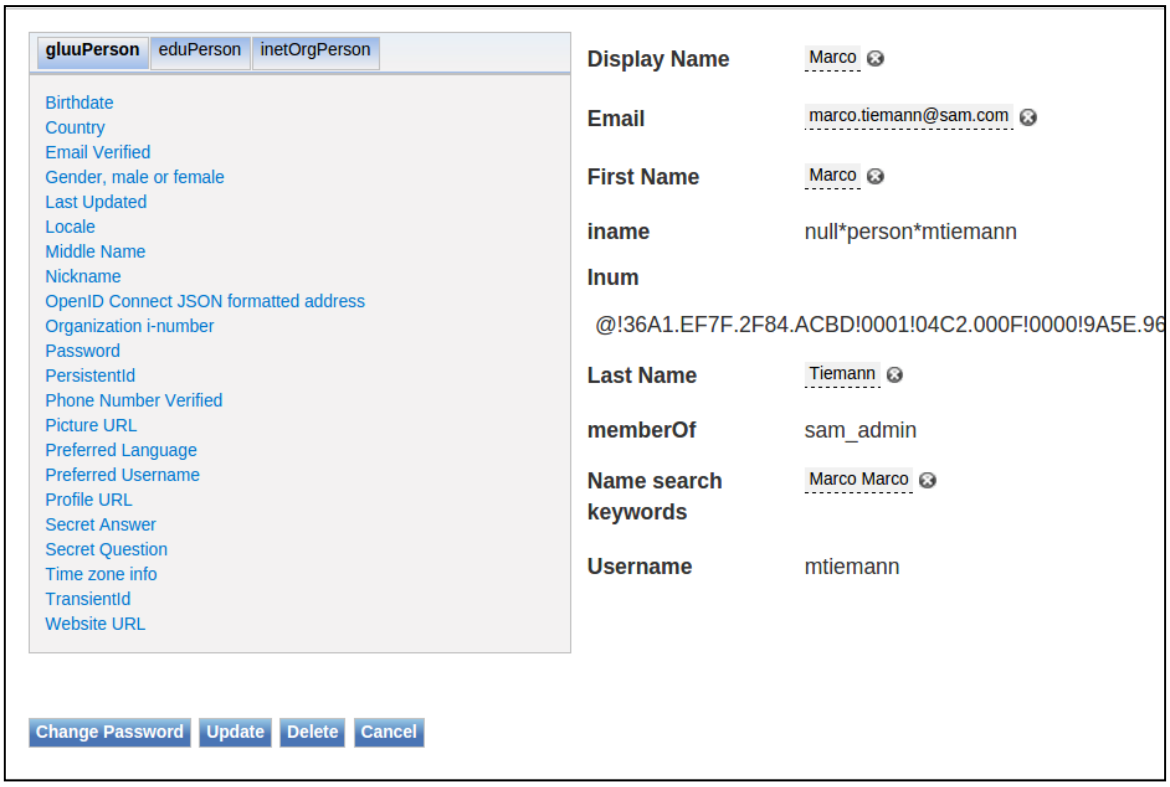

Figure 94: Gluu User Addition Panel with Mandatory and Optional User Properties

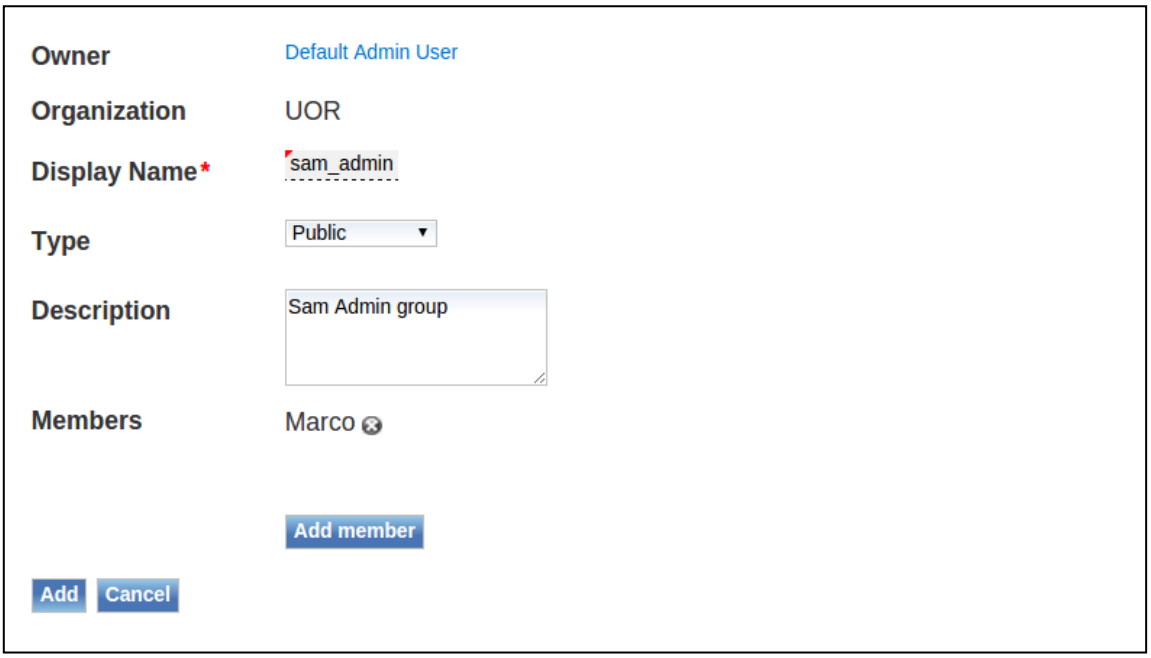

Figure 95: Gluu Group Management Panel for the Definition of Groups for Access Management Purposes

6.4.2 Client Side (Developers)

Interactions with the ISS component are handled through RESTful interfaces. As already mentioned earlier (see Section 3.4.2.1) it is assumed that the reader is familiar with the general usage of RESTful Web Service interfaces.

6.4.2.1 Method “Authenticate”

The ISS component provides a Web Method call for authenticating credentials. However, this method should not be used in order to authenticate users of the SAM platform – users should authenticate directly against the Gluu server.

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 96: RESTful Interface Description – Authenticate

6.4.2.2 Method “Verify Authentication Token”

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	ID of the user’s authentication token

	Required string	Example: *"abcdef"*
Response		
HTTP Status Code	Value	Description
	200	Verification successful
	401	Authentication failure
	500	Error during processing

Figure 97: RESTful Interface Description – Verify Authentication Token

6.5 Limitations and Further Developments

Currently, the management of the Identity and Security Service components including the third party Gluu server is managed through configuration files and through a separate Gluu server management web user interface. This should be migrated to the overall SAM administration web user interface in prototype 3.

Identity management federation support has not yet been tested for this system prototype. However, the features enabling federated identity management are available in the release. Testing will be undertaken as part of the next prototype development.

Prototype 1 is concerned purely with the integration of identity management functionalities and not with authorisation support. Authorisation will be integrated as part of prototype 2.

Figure 98 summarises the tasks planned for prototype 2 of the Identity and Security Service components to be delivered in M25.

Subcomponent	Task
User Identity Manager	Integrate User Identity Manager with other relevant SAM components
OpenID Provider	Finalise the system configuration, add user configurations for system development, testing and evaluation
Policy Decision Point	Implement first Policy Decision Point prototype
Policy Administration Point	Implement first Policy Administration Point prototype
Policy Retrieval Point	Implement first Policy Retrieval Point prototype

Figure 98: Tasks Planned for Prototype 2

6.6 Research Background

For the implementation and the overall approach of the current prototype, the following papers have been researched and taken into consideration:

Source	Subcomponent	Description
David Recordon and Drummond Reed, „OpenID 2.0: a platform for user-centric identity management“, In: Proceedings of the Second ACM Workshop on Digital Identity Management DIM'06, 2006, pp. 11-16.	OpenID Provider	Description of OpenID 2.0
Hyun-Kyung Oh and Seung-Hun Jin, „The Security Limitations of SSO in OpenID“, In: Proceedings of the 10th International Conference on Advanced Communication Theory ICACT 2008, 2998, pp. 16098-1611.	OpenID Provider	Description of issues to be considered when using OpenID for federation or single sign-on solutions.
Bart van Delft and Martihn Oostdijk, „A Security Analysis of OpenID“, In: Proceedings of the Second IFIP WG 11.6 Working Conference, IDMAN 2010, Oslo, Norway, November 18-19, 2010, pp. 73-84.	User Identity Manager	Description of security issues to consider when using OpenID for authentication
Eghbak Ghazizadeh, Jamalul-Iail Ab Manan, Mazdak Zamani and Abolghasem Pashang, „A Survey on Security Issues of Federated Identity in the Cloud Computing“, In: Proceedings of the 4th International IEEE Conference on Cloud Computing Technology and Science, 2012, pp. 562-565.	User Identity Manager	Comparison of federated identity management approaches including OpenID and the possible alternative SAML

Figure 99: Research Background for Identity and Security Services

6.7 Target Performance

The overall performance of the backend Gluu server system has been documented on the website of the system at http://www.gluu.org/docs/articles/benchmarking/. It is not the goal of this project to repeat benchmarking of the Gluu server platform; instead, the defined targets apply to end-to-end request processing over the real-world infrastructure that is used throughout the project, including distribution of the TSB server system, the Gluu server system and the ISS component operating environment across separate sites in three countries. This is made explicit here as it is expected that a significant proportion of the real-world time spent with ISS component functionalities will be attributable to network traffic time.

The following target performance indicators have been defined for the Identity and Security Services component concerning identity management and user authentication:

Topic	Description	Target KPI
Authentication time	The time spent between submission of authentication data from a SAM user interface to the time the user interface client has received the authentication token or „deny“ response	Authentication roundtrip response time of <= 5 seconds for 90% of requests regardless of under „no load“ conditions of the authentication server
Token verification time	The time spent between receipt of an authentication token at the SAM TSB and the receipt of the authentication token verification response at the SAM TSB	Verification roundtrip response time of <= 2 seconds for 90% of requests regardless of under „no load“ conditions of the authentication server
Token verification throughput	The number of token verification requests successfully handled between SAM TSB and ISS component as described for Token verification above within a 60 second time frame	Average verification throughput of >= 250 verification requests per minute on prototype hardware

Figure 100: Target Performance Identity and Security Services

Target values will be reviewed when the network conditions between the sites involved have been tested with the relevant target systems. Deliverable D4.9.2 will provide evaluation data concerning the achieved performance levels for authentication and will define the performance targets for authorisation functionalities.

6.8 Summary

This section provides a description of the first prototype of the ISS component developed in Task 4.4 “Distributed Identity, Trust and Security”. The main outcome of this task is the installation, configuration and integration of identity management for user authentication in SAM. This prototype is the first of three iterations planned for this component.

The description outlines the requirements for installing and operating both installation of the Gluu server system used in SAM and of the ISS component to be deployed as part of the SAM infrastructure. It should be noted that it is not expected that other project partners will need to install separate instances of the Gluu server system.

Upcoming work will focus on the implementation of authorisation functionalities and on the integration of Identity and Security Services with other SAM components as specified in the SAM deliverable D3.2.2.

7 Document Summary

This document is the second version of the deliverable series D4.9.x and provides information about all tasks and software prototypes of WP4 in contrast to the predecessor deliverable D4.9.0 which has been created to provide early information about the task T4.1 Assets Storage and Information Management (see Section 3).

This document provides information about running tasks in work package 4:

- T4.1 Assets Storage and Information Management (Section 3),
- T4.2 Communication and Federation (Section 4),
- T4.3 Data Characterisation Services and (Section 5) and
- T4.4 Distributed Identity, Trust and Security (Section 6)

The information for each task provided contains:

- Scope of the pilot implementation, its purpose and the main relationships with other modules implemented in SAM in the first year
- Information needed to deal with the pilot in terms of technical and non-technical requirements, software to be installed, etc.
- Steps needed to install the pilot software and how to build it from source code
- Different screens and actions implemented at the pilot itself, how to access it, and how to test the different implemented options
- Current pilot limitations and the expected improvements
- Papers and other scientific information considered
- Key Performance Indicators (KPIs) for the SAM component
- Conclusion of the implementation of the first prototype

References

[FGG13] Fernández, J.; Gutiérrez, Y.; Gómez, JM.; Martínez-Barco, P.; Montoyo, A.; Muñoz, R. Sentiment Analysis of Spanish Tweets Using a Ranking Algorithm and Skipgrams. Journal Sociedad Espanola de Procesamiento de Lenguaje Natural, pp 133-142, 2013.

[GVM11] Gutiérrez, Y.; Vázquez, S.; Montoyo, A. Improving WSD using ISR-WN with Relevant Semantic Trees and SemCor Senses Frequency. In proceedings of Recent Advances in Natural Language Processing, pp 233–239, Hissar, Bulgaria, 2011.

[LES86] Lesk, M. Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In SIGDOC '86: Proceedings of the 5th annual international conference on Systems documentation, pp 24-26, New York, NY, USA. ACM, 1986.

[LEV66] Levenshtein, V. I. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. In Soviet Physics Doklady vol. 10, pp. 707, 1966.

[SE13] Shvaiko, P., & Euzenat, J. Ontology matching: state of the art and future challenges. Knowledge and Data Engineering, IEEE Transactions on, vol 25 (1), pp 158-176, 2013.

Annex A: Cloud Storage Usage Examples

This annex provides additional information regarding the Cloud Storage and its RESTful interfaces. To be more precise it provides example data for CRUD operations for the supported data types, e.g. JSON for MongoDB or JSON-LD for Sesame. RESTful interfaces like “Create Bucket” are similar for each data type and do not need to be listed here (see Section 3.4.2.1.1 for more information).

Example 1: JSON in MongoDB (Semi-Structured)

The following subsections show examples on how to perform CRUD operations on a semi-structured database using JSON data.

Create Data Object

The following example shows how to add an object to the Bucket. This object will be stored as one object in an array. You can add an additional object to the Bucket and the array will be extended with that object.

```
{
   "id": 0,
   "color": "white"
}
```

Figure 101: Cloud Storage Example: Create Data Object (Semi-Structured)

Read Data Object

The following example shows how to retrieve all objects from the Bucket, which id attribute is greater than -1. Find more query definitions at the MongoDB manual.

```
{
   "id": {"$gt": -1}
}
```

Figure 102: Cloud Storage Example: Read Data Object (Semi-Structured)

Update Data Object

The following example shows how to update a specific data object. By providing a query object, the specific object will be found and replaced/updated by the object which has been provided too.

```
{
   "query": {
      "id": 0
   },
   "object": {
      "id": 0,
      "color": "yellow"
   }
}
```

Figure 103: Cloud Storage Example: Update Data Object (Semi-Structured)

Delete Data Object

The following example shows how to delete a data object. This is the same as the query object in the previous example but without the “Query-Object” structure. Note: It deletes the first occurrence only.

```
{ "id": 0 }
```

Figure 104: Cloud Storage Example: Delete Data Object (Semi-Structured)

Example 2: JSON-LD in Sesame (Semantic)

The following subsections show examples on how to perform CRUD operations on a semantic database using JSON-LD data.

Create Data Object

The following example shows how to add an object to the Bucket. This object will be stored as one object in an array. You can add an additional object to the Bucket and the array will be extended with that object.

```
{
  "namespace": "test",
  "jsonLdData": {
    "@context": "http://json-ld.org/contexts/person.jsonld",
    "@id": "http://dbpedia.org/resource/John_Lennon",
    "name": "John Lennon",
    "born": "1940-10-09",
    "spouse": "http://dbpedia.org/resource/Cynthia_Lennon"
  }
}
```

Figure 105: Cloud Storage Example: Create Data Object (Semantic)

Read Data Object

The following example shows how to retrieve all objects from the Bucket. Other queries can be created following the SPARQL query language[17].

```
{
    "query": "CONSTRUCT { ?s ?p ?o } WHERE {?s ?p ?o }",
    "queryType": "sparql"
}
```

Figure 106: Cloud Storage Example: Read Data Object (Semantic)

Update Data Object

Information about this operation will be added later.

Delete Data Object

The following example shows how to delete a data object. At the moment it is only possible to delete a whole namespace.

[17] http://www.w3.org/TR/rdf-sparql-query/

```
{
  "query": "test",
  "queryType": "namespace"
}
```

Figure 107: Cloud Storage Example: Delete Data Object (Semantic)

Annex B: Interconnection Bus Configuration Examples

This annex provides additional information regarding the Interconnection Bus and its RESTful interfaces. Specifically, it provides example data showing how to configure different kinds of services depending on the CRUD operation to be carried out.

Create Bucket

This example shows how to create a service in order to create a Bucket in the Cloud Storage. This service is based on the PUT HTTP method.

Definition

The following figure shows the definition of the service:

Create Bucket			
Description	Creates a new Bucket for the user		
Request			
Request URL	**PUT** http://localhost:8182/api/cs/bucket		
JSON Attributes	Name	Required	Possible Values
	bucketId	yes	anystring
	bucketType	yes	SemiStructured Binary Semantic
	accessRights	no	A list of AccessRight Data Transfer Object (see Section 3.4.2.2)
Response			
HTTP Status Code	Value		Description
	200		Bucket created
	400		Bad Request
	404		Bucket Not Found
	409		Bucket exists already
	415		Unsupported Media Type
	502		Database error

Figure 14: RESTful Interface Description for Creating a Bucket

```
{
   "bucketId": "testId",
   "bucketType": "binary",
   "accessRights": [
      {
         "id": "user1",
         "right": "Read"
      }
   ]
}
```

Figure 108: Create Bucket Definition

Configuration

Based on the previous description, it is necessary to introduce the following information in order to configure correctly the Create Bucket service:

- **Name:** CreateBucket
- **HTTP Method:** PUT
- **Signature:** bucket

- **ContentType:** application/json
- **Parameter:** RequestBody.application/json=/SAMRequest/Payload

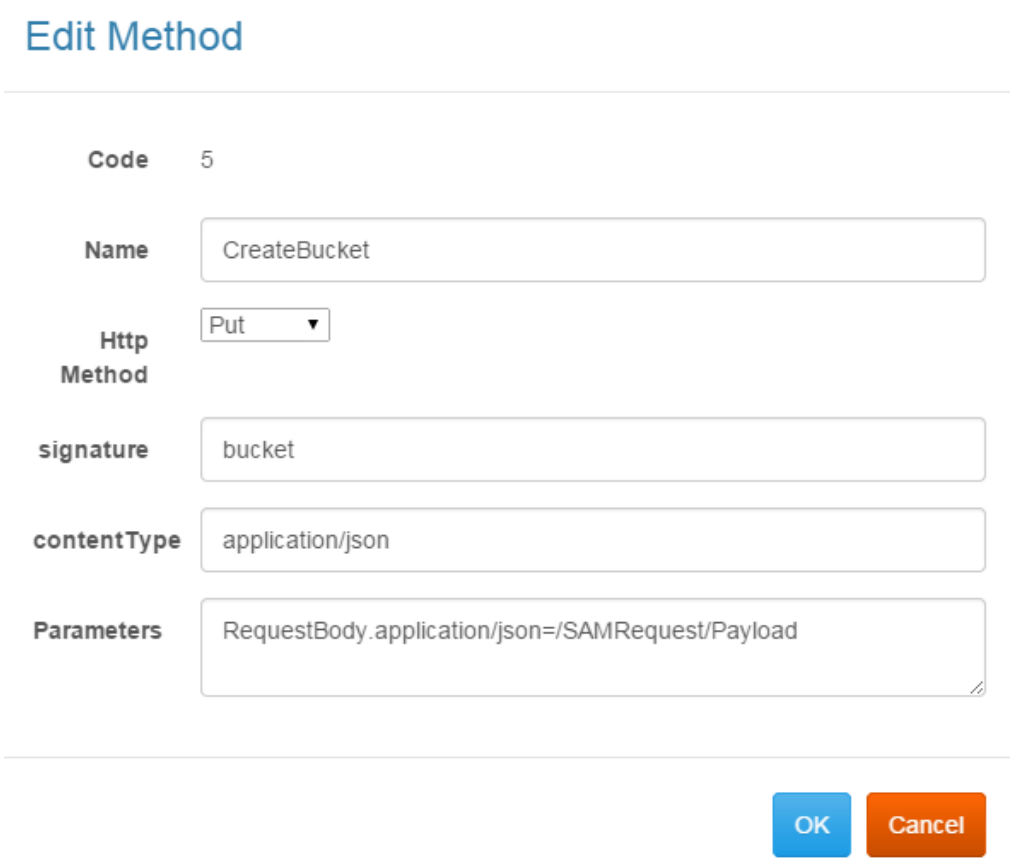

Figure 109: Create Method “Create Bucket”

SAM Message

Based on Section 4.4.2.1 the following SAM Request message will be used to create a new Bucket in the Cloud Storage.

```
<SAMRequest>
  <Header>
    <Source>ContentGateways</Source>
    <Destination>CloudStorage</Destination>
    <Service>CreateBucket</Service>
    <Topic> </Topic>
    <Parameters>
   </Parameters>
  </Header>
  <Payload>
{
   "bucketId": "testId",
   "bucketType": "Binary",
   "accessRights": [
      {
         "userId": "user1",
         "right": "Read"
      }
   ]
}

  </Payload>
</SAMRequest>
```

Figure 110: SAM Request to Create a Bucket

Get Brand and Consumer Rules

This example shows how to create a service in order to get the Brand and Consumer Rules based on the owner. This service is based on the GET HTTP method.

Definition

In this example the service definition is the following:

```
http://<domain>/rules?owner=BDS
```

Configuration

Based on the previous description, it is necessary to introduce the following information in order to configure correctly the Get Brand and Consumer Rules service:

- **Name:** GetRules
- **HTTP Method:** GET
- **Signature:** rules?owner={sowner}
- **ContentType:** application/json
- **Parameter:** UrlSegment.sowner=/SAMRequest/Header/Parameters/Parameter[Name='param1']/Value

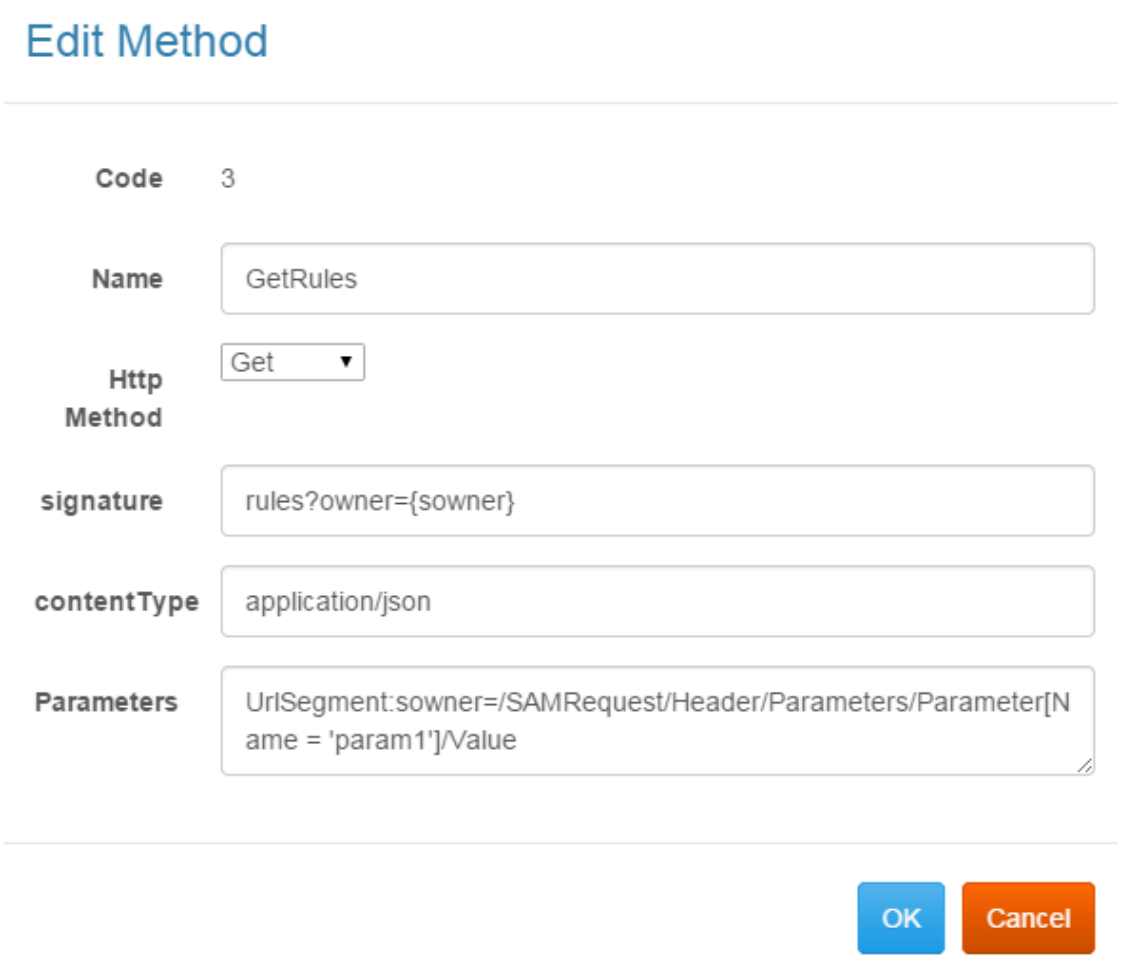

Figure 111: Create Method “Get Brand and Consumer Rules”

SAM Message

Based on Section 4.4.2.1, the following SAM Request message will be used to get the Brand and Consumer Rules filtered by owner.

```
<SAMRequest>
  <Header>
    <Source>ContentGateways</Source>
    <Destination>BrandAndConsumer</Destination>
    <Service>GetRules</Service>
    <Topic> </Topic>
    <Parameters>
     <Parameter>
       <Name>param1</Name>
       <Value>BDS</Value>
    </Parameter>
   </Parameters>
  </Header>
  <Payload>
  </Payload>
</SAMRequest>
```

Figure 112: SAM Request to Get Brand and Consumer Rules

How to Test it

One of the easiest ways to test the communication is by installing the Advanced Rest Client Extension[18] in Chrome. Using this extension, it is possible to test the service following these steps:

1. Introduce the SAMCL service URL (see D4.2.1)
2. Introduce the SAM Message XML in the Payload
3. Click on “Send” button
4. The response will be show in the bottom of the page.

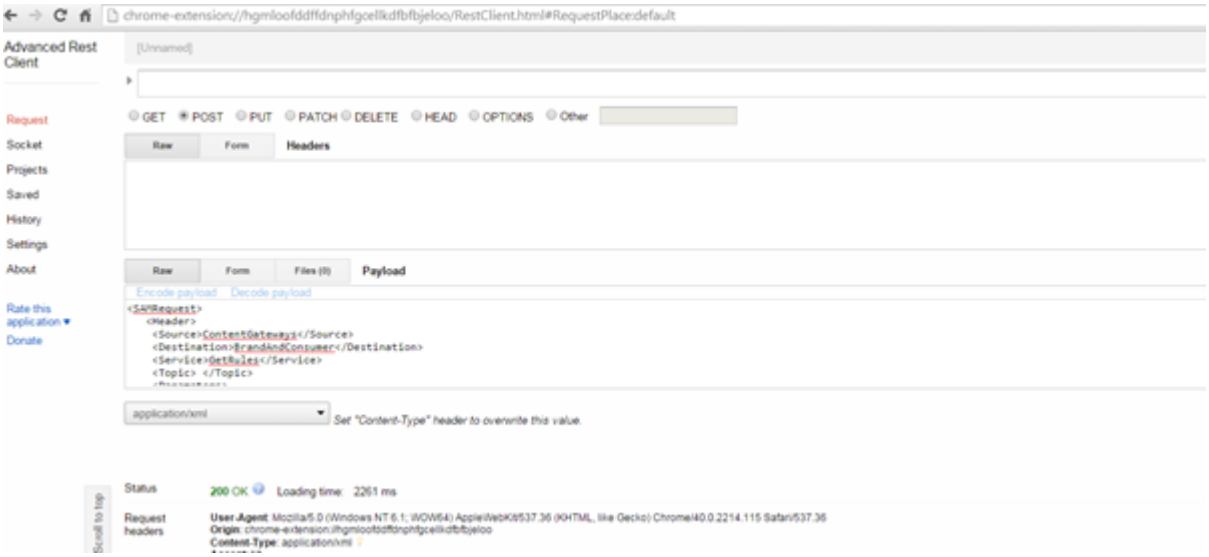

Figure 113: Advanced Rest Client Extension Example

[18] Advanced Rest Client Extension