Secure Provisioning of Cloud Services
based on SLA Management

SPECS Project - Deliverable 1.4.1

Module Shared API and Core Services

Version 1.1
15 February 2016

The activities reported in this deliverable are partially supported
by the European Community's Seventh Framework Programme under grant agreement no. 610795.

Deliverable information

Deliverable no.:	D1.4.1
Deliverable title:	Module Shared API and Core Services
Deliverable nature:	Report
Dissemination level:	Public
Contractual delivery:	31 October 2015
Actual delivery date:	31 October 2015
Author(s):	Massimiliano Rak (CERICT)
Contributors:	Valentina Casola (CERICT), Jolanda Modic (XLAB), Damjan Murn (XLAB), Giancarlo Capone (CERICT), Nicola De Filippo (CeRICT)
Reviewers:	Umberto Villano (CeRICT), Madalina Erascu (IeAT)
Task contributing to the deliverable:	T1.4
Total number of pages:	86

Executive summary

This deliverable presents the final design of the SPECS SLA Platform module and the set of shared APIs that let all Core services cooperate for the management of the SLA life cycle. Indeed, during the second year, almost all the SPECS modules' design evolved thanks to a refinement process compliant with the methodology that is used to gather requirements from the End-Users.
While D1.1.2 introduced a preliminary design of the module at month 12, this document is focused on the refined architecture of the SPECS SLA Platform and of part of the SPECS Vertical layer (i.e., Auditing and User Manager components). The remaining components of the Vertical layer (i.e., Credential Service and Security Tokens), which are dedicated to secure communications, are discussed in a separate deliverable (i.e., D4.4.2). This document is strictly connected to deliverable D1.3, which presented the module API and the interaction protocols.

This document presents:

1. An updated SLA Platform architecture. The main SLA Platform components, designed in a preliminary form at month 12, were slightly updated. We merged some components and deleted others, whose requirements have been superseded or covered by already existing components. We describe the whole set of packages making up the SLA Platform. Furthermore, we also report the updated set of requirements with respect to the ones elicited in D1.2 at month 6 and discuss module responsibilities to cover them.
2. The interactions with other SPECS modules. According to the finalized design of the Interaction protocols and API presented in D1.3, we report the details of all components and their API, to be used by other SPECS components.
3. Vertical Layer services integrated in the SPECS SLA Platform. We describe two of four available Vertical Layers, namely the Auditing component and the User Manager. The other two are illustrated in the D4.2.2 and D4.4.2 deliverables, dedicated to secure communications in SPECS.
4. The SPECS domain model. We introduce the SPECS domain model, that is based on the SLA conceptual model defined in D2.2.2 and includes all concepts related to the management and implementation of the SLA itself; it is used to effectively implement the SLA-based approach provided by SPECS. Furthermore, a summary of the SPECS data model, shared among all API and discussed in D1.3, is reported, too.

Table of contents

Index of figures

Index of tables

1. Introduction

This deliverable aims at presenting the final design and implementation of the SPECS SLA Platform module and the set of shared API that let all core services cooperate on the management of the SLA life cycle. In particular, the deliverable describes all the software components that have been designed for the implementation of the SLA Platform.

As already said in previous deliverables, the SLA Platform is the module devoted to supporting the SLA-based provisioning of services over a common platform (the Enabling Platform). Its components are web applications offering REST APIs that enable the execution of the SPECS Security Services.

Figure 1 shows the component diagram of the SLA Platform and of the Vertical layer. As pointed out in D1.1.3, the Vertical layer offers some cross-cutting services used by the SLA Platform and other modules and has been developed partly in the context of the Enforcement and partly in the context of the SLA Platform. In this document, we present the final implementation of two of the components of the Vertical layer, namely the Auditing component and the User Manager component, while the remaining components are discussed in deliverable 4.4.2, as they provide specific services for secure communication and have been analysed as part of WP4.
The architecture shown in Figure 1 is the result of the updates done to cope with the feedbacks received during the activities performed in the second year, in strict collaboration with SPECS stakeholders and with other activities in WP5.

In order to enable the communication among all modules during the SLA management, the SPECS framework defines and shares a set of data models included in the SPECS Data model. Section 6 presents such data model, while its usage and the meaning of the individual fields are described in details in Deliverable D1.3 along with APIs that use them.

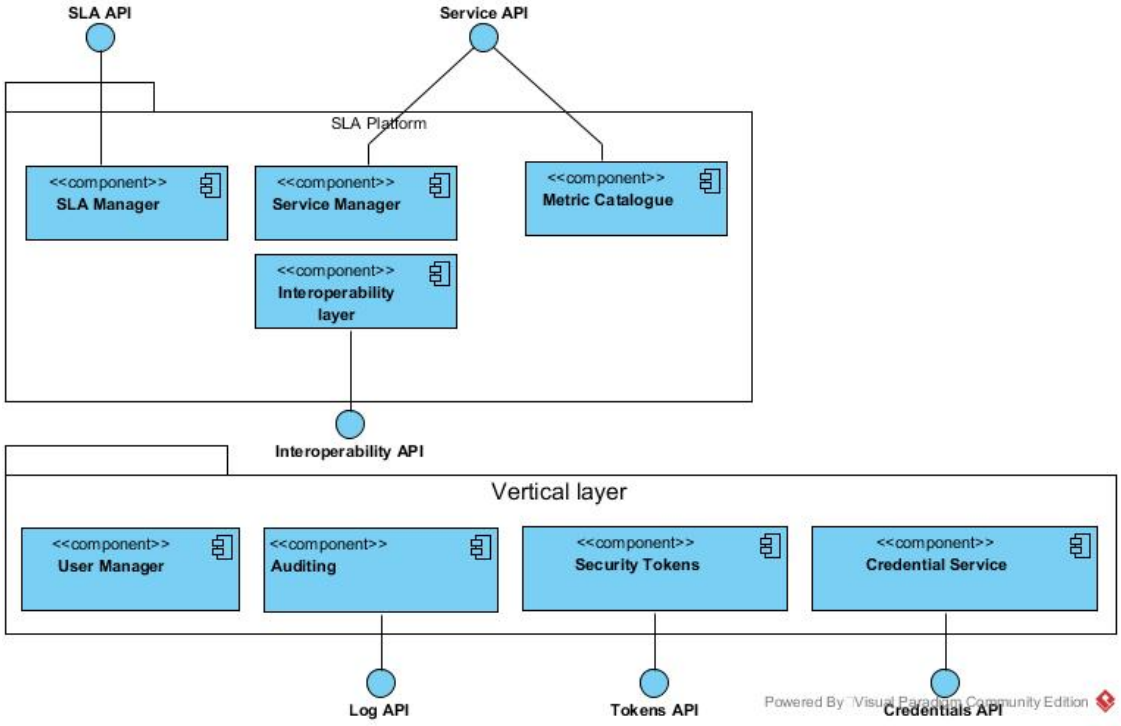

Figure 1: SLA Platform Component Diagrams

The SLA Platform comprises four components, offering three sets of APsI already described in detail in the Annexes A, B and C of Deliverable D1.3:

- **SLA API**: devoted to manage the Service Level Agreement Life Cycle,
- **Services API**: devoted to manage services and associated information in order to compose them to respect SLAs,
- **Interoperability API**: devoted to grant transparent communication among all the modules and components in SPECS.

The SLA Manager component maintains the Security SLAs, represented in the WS-Agreement format enriched with SPECS security extensions, and the information associated to their life cycle (in terms of the state diagram illustrated in D1.1.1). It offers REST SLA API.

The Service Manager and the Security Metrics Catalogue cooperate on offering the Services API. The first component (Service Manager) maintains security mechanisms metadata, in order to enable the enforcement of proper security mechanisms according to an SLA. In practice, it maintains the list of all the artifacts that run on top of the platform and enable their execution. The Enforcement components use the Service Manager in order to identify the proper mechanisms to be set up and provided. The Security Metrics Catalogue, instead, maintains the information related to standard security metrics; the metrics are defined according to the RATAX NIST model[1] and represented in machine-readable format within the SPECS project, as described in D2.2.2.

The Interoperability component is a general-purpose REST gateway that enables transparent communication among different components. Using the Interoperability API it is possible to define virtual interfaces and use the Interoperability component as a collector of all the REST requests, which will be forwarded to the correct components. It facilitates Platform integration and represents an innovation introduced in this refined version of the SLA Platform architecture.

Finally, the Vertical Layer comprises four components able to provide cross cutting services to all SPECS modules: Auditing, User Manager, Credential Service and Security Tokens. As previously said, the Auditing and the User Manager components are presented in this deliverable, while the Credential Service and the Security Tokens have dedicated deliverables where the design and implementation details are reported (D4.2.2 and D4.4.2). The Auditing component is devoted to supporting auditing in all the steps of the SPECS flow, as described in detail in Section 4.1. The User Manager component is devoted to supporting user management on top of the platform and is described in Section 4.2.

In this deliverable, we will provide internal design description for each component included in the architecture. In particular, for each component, we will report in detail use cases, a description of the component, a description of the package with component and class diagrams and, finally, the requirements covered by the component and a short explanation of the coverage. Finally, in Section 7 we report an updated list of requirements with respect to the ones elicited at year 1, we discuss the updates provided to the architecture and, in Annex A, we also report some additional API calls provided by the SLA Platform.
The content of this deliverable is the primary input for deliverable D1.4.2, where implementation, installation, usage and testing details will be reported.

[1] NIST publication - Cloud Computing Service Metrics Description, August 2014

2. Relationship with other deliverables

As shown in Figure 2, this document is strictly connected to deliverables D1.1.2, D1.3 and D4.4.2, where specific components of the SLA Platform are presented.

Specifically, D1.1.2 focused on a preliminary version of the SLA Platform, D1.3 was devoted to module API and the interaction protocols with other Core services and, finally, D4.4.2 presented two of the Vertical Layer services, dedicated to secure communication with SPECS.

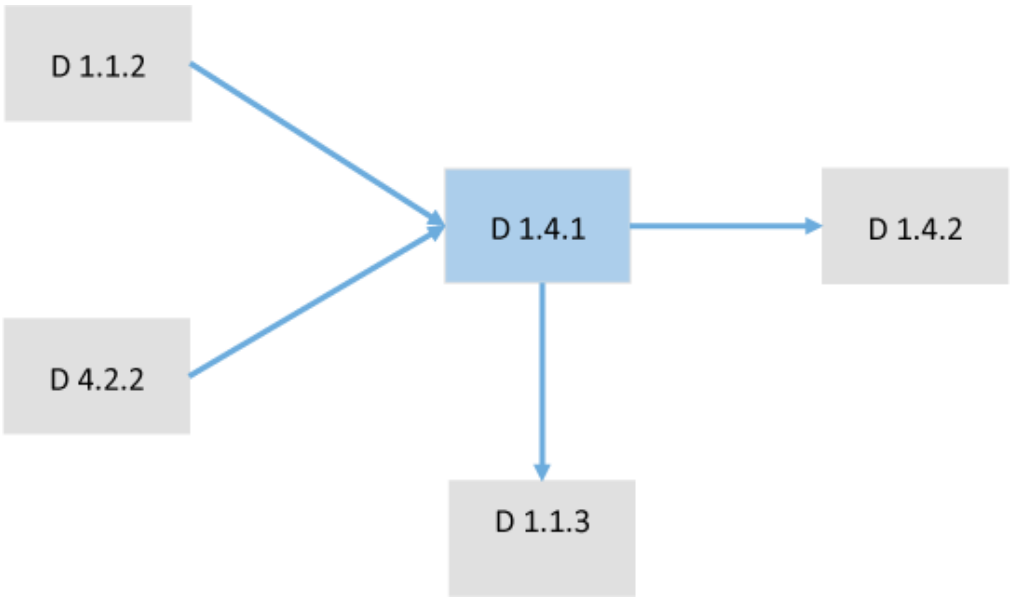

Figure 2: Relationship with other deliverables

As already said, the output of this deliverable is the refined design that is input to the implementation activities that are reported in D1.4.2.

3. SLA Platform

3.1. SLA Manager

The SLA Manager is a component used to manage the SLAs life cycle. Its main task is to store all the SLAs that have been signed by users and to enable any operation on each of them.
Its main design requirement is that all information related to SLAs and their state has to be stored in an SLA Repository, and that specific services should be available to query and to update the contents of such repository. Moreover, whenever events are generated in presence of an alert or a violation, it should allow the update of the state of the involved SLAs.

3.1.1. SLA Manager Use Cases

This section describes all the interactions between a SPECS component and the SLA Manager. The following use case diagram has been created for this purpose.

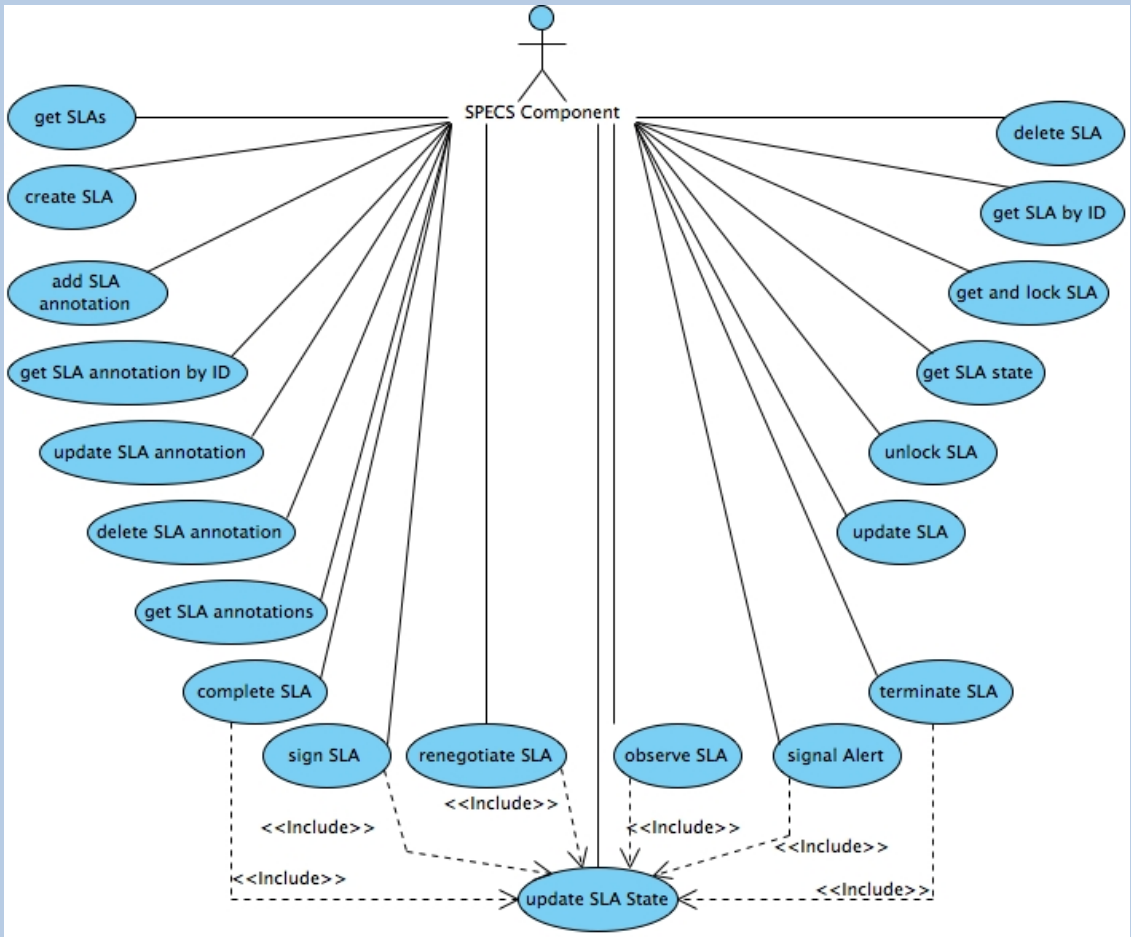

Figure 3: SLA Manager Use Case Diagram

3.1.1.1. Description

The following tables describe in detail the use cases reported in the previous figure, with a simple UML use case template:

Table 1: Create SLA

Use Case Name	Create SLA
Actors	SPECS component
Entry (pre-conditions)	
Exit (post-conditions)	A new SLA is stored into the SLA Manager
Input	The xml that represents the SLA to store
Output	The URI of the created SLA
Flows of events	1 The SPECS component inserts the SLA xml representation 2 The SLA Manager stores the SLA into the database 3 The SPECS component receives the URI of the created SLA
Alternative Flows	
Extensions	
Inclusions	

Table 2: Get SLAs

Use Case Name	Get SLAs
Actors	SPECS component
Entry (pre-conditions)	
Exit (post-conditions)	A collection of SLAs is returned to the SPECS component
Input	
Output	The URI collection of the SLAs
Flows of events	1 The SPECS component executes the GET request 2 The SLA Manager searches all SLAs in the database 3 The SPECS component receives the collection URI of the SLAs
Alternative Flows	3a There are not SLAs stored 1 The returned collection is empty
Extensions	
Inclusions	

Table 3: GetSLA by ID

Use Case Name	Get SLA by ID
Actors	SPECS component
Entry (pre-conditions)	
Exit (post-conditions)	A specific SLA is returned to the SPECS component
Input	The Unique Identifier of the requested SLA
Output	The xml that represents the SLA
Flows of events	1 The SPECS component inserts the ID of the SLA 2 The SPECS component executes the GET request 3 The SLA Manager searches the SLA in the database 4 The SPECS component receives the xml that represents the SLA
Alternative Flows	3a There is no SLA associated to the unique identifier 1 An IllegalArgumentException is thrown
Extensions	
Inclusions	

Table 4: Delete SLA

Use Case Name	Delete SLA
Actors	SPECS component
Entry (pre-conditions)	An SLA is stored into the component
Exit (post-conditions)	A specific SLA is removed
Input	The Unique Identifier of the SLA to remove
Output	The Unique Identifier of the removed SLA
Flows of events	1 The SPECS component inserts the ID of the SLA 2 The SPECS component executes the DELETE request 3 The SLA Manager searches the SLA in the database 4 The SLA Manager removes the SLA from the database 5 The SPECS component receives the Unique Identifier of the SLA
Alternative Flows	3a There is no SLA associated to the unique identifier 1 An IllegalArgumentException is thrown
Extensions	
Inclusions	

Table 5: Get and Lock SLA

Use Case Name	Get and Lock SLA
Actors	SPECS component
Entry (pre-conditions)	An SLA is stored into the component
Exit (post-conditions)	A specific SLA is returned and locked
Input	The Unique Identifier of the SLA to get and lock
Output	The xml that represents the requested SLA
Flows of events	1 The SPECS component inserts the ID of the SLA 2 The SPECS component executes the GET request 3 The SLA Manager searches the SLA in the database 4 The SLA Manager locks the SLA in the database 5 The SPECS component receives the xml that represents the SLA
Alternative Flows	3a There is no SLA associated to the unique identifier 1 An IllegalArgumentException is thrown
Extensions	
Inclusions	

Table 6: Unlock SLA

Use Case Name	Unlock SLA
Actors	SPECS component
Entry (pre-conditions)	An SLA stored into the component is locked
Exit (post-conditions)	A specific SLA is unlocked
Input	The Unique Identifier of the SLA to unlock
Output	The Unique Identifier of the unlocked SLA
Flows of events	1 The SPECS component inserts the ID of the SLA 2 The SPECS component executes the POST request 3 The SLA Manager searches the SLA in the database 4 The SLA Manager unlocks the SLA in the database

	5 The SPECS component receives the Unique Identifier of the unlocked SLA
Alternative Flows	3a There is no SLA associated to the unique identifier 1 An IllegalArgumentException is thrown 4a The SLA is no previously locked 2 An IllegalStateException is thrown
Extensions	
Inclusions	

Table 7: Update SLA

Use Case Name	**Update SLA**
Actors	SPECS component
Entry (pre-conditions)	An SLA is stored into the component
Exit (post-conditions)	The value of a specific SLA is updated
Input	The Unique Identifier of the SLA to update The new value of the SLA
Output	The Unique Identifier of the updated SLA
Flows of events	1 The SPECS component inserts the ID of the SLA 2 The SPECS component inserts the new value of the SLA 3 The SPECS component executes the PUT request 4 The SLA Manager searches the SLA in the database 5 The SLA Manager updates the SLA in the database 6 The SPECS component receives the Unique Identifier of the updated SLA
Alternative Flows	3a There is no SLA associated to the unique identifier 1 An IllegalArgumentException is thrown
Extensions	
Inclusions	

Table 8: Get SLA annotations

Use Case Name	**Get SLA annotations**
Actors	SPECS component
Entry (pre-conditions)	An SLA is stored into the component
Exit (post-conditions)	The collection of the annotations is returned to the SPECS component
Input	The Unique Identifier of the SLA
Output	The URI collection of the Annotations
Flows of events	1 The SPECS component executes the GET request 2 The SLA Manager searches the SLA in the database 3 The SLA Manager searches the Annotations related to the SLA 4 The SPECS component receives the collection URI of the Annotations
Alternative Flows	2a There is no SLA associated to the unique identifier 1 An IllegalArgumentException is thrown 3a There are not stored Annotations for the SLA 1 The returned collection is empty
Extensions	
Inclusions	

Table 9: Get SLA annotation by ID

Use Case Name	Get SLA annotation by ID
Actors	SPECS component
Entry (pre-conditions)	An SLA is stored into the component and an Annotation is added on it
Exit (post-conditions)	The value of the requested annotation
Input	The Unique Identifier of the SLA The Unique Identifier of the Annotation
Output	The value of the Annotation as String
Flows of events	1 The SPECS component inserts the ID of the SLA 2 The SPECS component inserts the ID of the Annotation 3 The SPECS component executes the GET request 4 The SLA Manager searches the SLA in the database 5 The SLA Manager searches the Annotation related to the SLA 6 The SPECS component receives the value of the Annotation
Alternative Flows	5a There is no SLA associated to the unique identifier 1 An IllegalArgumentException is thrown 5a There is no Annotation associated to the unique identifier 1 An IllegalArgumentException is thrown
Extensions	
Inclusions	

Table 10: Add SLA annotation

Use Case Name	Add SLA annotation
Actors	SPECS component
Entry (pre-conditions)	An SLA is stored into the component
Exit (post-conditions)	The URI of the stored annotation
Input	The Unique Identifier of the SLA
Output	The Unique Identifier of the stored Annotation
Flows of events	1 The SPECS component inserts the ID of the SLA 2 The SPECS component inserts the value of the Annotation 3 The SPECS component executes the POST request 4 The SLA Manager searches the SLA in the database 5 The SLA Manager stores the Annotation into the database 6 The SPECS component receives the URI of the created Annotation
Alternative Flows	4a There is no SLA associated to the unique identifier 1 An IllegalArgumentException is thrown
Extensions	
Inclusions	

Table 11: Update SLA annotation

Use Case Name	Update SLA annotation
Actors	SPECS component
Entry (pre-conditions)	An SLA is stored into the component and an Annotation is added on it
Exit (post-conditions)	The URI of the updated annotation
Input	The Unique Identifier of the SLA The Unique Identifier of the Annotation

	The new value of the Annotation
Output	The Unique Identifier of the updated Annotation
Flows of events	1 The SPECS component inserts the ID of the SLA 2 The SPECS component inserts the ID of the Annotation 3 The SPECS component inserts the new value of the Annotation 4 The SPECS component executes the PUT request 5 The SLA Manager searches the SLA in the database 6 The SLA Manager searches the Annotation related to the SLA 7 The SLA Manager updates the value of the found Annotation 8 The SPECS component receives the value of the Annotation
Alternative Flows	5a There is no SLA associated to the unique identifier 1 An IllegalArgumentException is thrown 6a There is no Annotation associated to the unique identifier 1 An IllegalArgumentException is thrown
Extensions	
Inclusions	

Table 12: Delete SLA annotation

Use Case Name	**Delete SLA annotation**
Actors	SPECS component
Entry (pre-conditions)	An SLA is stored into the component and an Annotation is added on it
Exit (post-conditions)	The URI of the removed annotation
Input	The Unique Identifier of the SLA The Unique Identifier of the Annotation
Output	The Unique Identifier of the removed Annotation
Flows of events	1 The SPECS component inserts the ID of the SLA 2 The SPECS component inserts the ID of the Annotation 3 The SPECS component executes the DELETE request 4 The SLA Manager searches the SLA in the database 5 The SLA Manager searches the Annotation related to the SLA 6 The SLA Manager deletes the Annotation 7 The SPECS component receives the unique identifier of the removed Annotation
Alternative Flows	3a There is no SLA associated to the unique identifier 2 An IllegalArgumentException is thrown 6a There is no Annotation associated to the unique identifier 2 An IllegalArgumentException is thrown
Extensions	
Inclusions	

Table 13: Get SLA state

Use Case Name	**Get SLA state**
Actors	SPECS component
Entry (pre-conditions)	An SLA is stored into the component
Exit (post-conditions)	The state of a specific SLA is returned to the SPECS component
Input	The Unique Identifier of the requested SLA
Output	The String value that represents the SLA state

Flows of events	1 The SPECS component inserts the ID of the SLA 2 The SPECS component executes the GET request to know the state 3 The SLA Manager searches the SLA in the database 4 The SLA manager get the state of the SLA 5 The SPECS component receives the string value that represents the SLA state
Alternative Flows	3a There is no SLA associated to the unique identifier 1 An IllegalArgumentException is thrown
Extensions	
Inclusions	

Table 14: Update SLA state

Use Case Name	**Update SLA state**
Actors	SPECS component
Entry (pre-conditions)	An SLA is stored into the component
Exit (post-conditions)	The URI of the updated SLA
Input	The Unique Identifier of the SLA The new value of the state to apply at the SLA
Output	The URI to get the new state of the SLA
Flows of events	1 The SPECS component inserts the ID of the SLA 2 The SPECS component inserts the new value of the state 3 The SPECS component executes the PUT request 4 The SLA Manager searches the SLA in the database 5 The SLA Manager evaluates if the new state is applicable on the SLA 6 The SLA Manager updates the value of the state 7 The SPECS component receives the URI to get the new state
Alternative Flows	3a There is no SLA associated to the unique identifier 1 An IllegalArgumentException is thrown 5a The new state is not applicable on the SLA 1 An IllegalStateException is thrown
Extensions	
Inclusions	

Table 15: Sign SLA

Use Case Name	**Sign SLA**
Actors	SPECS component
Entry (pre-conditions)	An SLA is stored into the component
Exit (post-conditions)	The URI of the updated SLA
Input	The Unique Identifier of the SLA
Output	The URI to get the new state of the SLA
Flows of events	1 The SPECS component inserts the ID of the SLA 2 The SPECS component executes the PUT request 3 The SLA Manager searches the SLA in the database 4 The SLA Manager evaluates if the sign state is applicable on the SLA (only if the SLA is in negotiating or renegotiating state) 5 The SLA Manager updates the value of the state 6 The SPECS component receives the URI to get the new state

Alternative Flows	3a There is no SLA associated to the unique identifier 2 An IllegalArgumentException is thrown 4a Signed state is not applicable on the SLA 2 An IllegalStateException is thrown
Extensions	
Inclusions	Update SLA state

Table 16: Observe SLA

Use Case Name	**Observe SLA**
Actors	SPECS component
Entry (pre-conditions)	An SLA is stored into the component
Exit (post-conditions)	The URI of the updated SLA
Input	The Unique Identifier of the SLA
Output	The URI to get the new state of the SLA
Flows of events	1 The SPECS component inserts the ID of the SLA 2 The SPECS component executes the PUT request 3 The SLA Manager searches the SLA in the database 4 The SLA Manager evaluates if the observe state is applicable on the SLA (only if the SLA is in signed state) 5 The SLA Manager updates the value of the state 6 The SPECS component receives the URI to get the new state
Alternative Flows	3a There is no SLA associated to the unique identifier 3 An IllegalArgumentException is thrown 4a Observed state is not applicable on the SLA 3 An IllegalStateException is thrown
Extensions	
Inclusions	Update SLA state

Table 17: Signal Alert

Use Case Name	**Signal Alert**
Actors	SPECS component
Entry (pre-conditions)	An SLA is stored into the component
Exit (post-conditions)	The URI of the updated SLA
Input	The Unique Identifier of the SLA
Output	The URI to get the new state of the SLA
Flows of events	1 The SPECS component inserts the ID of the SLA 2 The SPECS component executes the PUT request 3 The SLA Manager searches the SLA in the database 4 The SLA Manager evaluates if the alert state is applicable on the SLA (only if the SLA is in observed state) 5 The SLA Manager updates the value of the state 6 The SPECS component receives the URI to get the new state
Alternative Flows	3a There is no SLA associated to the unique identifier 4 An IllegalArgumentException is thrown 4a Alerted state is not applicable on the SLA 4 An IllegalStateException is thrown
Extensions	
Inclusions	Update SLA state

Table 18: Renegotiate SLA

Use Case Name	Renegotiate SLA
Actors	SPECS component
Entry (pre-conditions)	An SLA is stored into the component
Exit (post-conditions)	The URI of the updated SLA
Input	The Unique Identifier of the SLA
Output	The URI to get the new state of the SLA
Flows of events	1 The SPECS component inserts the ID of the SLA 2 The SPECS component executes the PUT request 3 The SLA Manager searches the SLA in the database 4 The SLA Manager evaluates if the renegotiate state is applicable on the SLA (only if the SLA is in observed or alerted or violated state) 5 The SLA Manager updates the value of the state 6 The SPECS component receives the URI to get the new state
Alternative Flows	3a There is no SLA associated to the unique identifier 5 An IllegalArgumentException is thrown 4a Renegotiated state is not applicable on the SLA 5 An IllegalStateException is thrown
Extensions	
Inclusions	Update SLA state

Table 19: Complete SLA

Use Case Name	Complete SLA
Actors	SPECS component
Entry (pre-conditions)	An SLA is stored into the component
Exit (post-conditions)	The URI of the updated SLA
Input	The Unique Identifier of the SLA
Output	The URI to get the new state of the SLA
Flows of events	1 The SPECS component inserts the ID of the SLA 2 The SPECS component executes the PUT request 3 The SLA Manager searches the SLA in the database 4 The SLA Manager evaluates if the complete state is applicable on the SLA (only if the SLA is in observed state) 5 The SLA Manager updates the value of the state 6 The SPECS component receives the URI to get the new state
Alternative Flows	3a There is no SLA associated to the unique identifier 6 An IllegalArgumentException is thrown 4a Completed state is not applicable on the SLA 6 An IllegalStateException is thrown
Extensions	
Inclusions	Update SLA state

Table 20: Terminate SLA

Use Case Name	Terminate SLA
Actors	SPECS component
Entry (pre-conditions)	An SLA is stored into the component
Exit (post-conditions)	The URI of the updated SLA

Input	The Unique Identifier of the SLA
Output	The URI to get the new state of the SLA
Flows of events	1 The SPECS component inserts the ID of the SLA 2 The SPECS component executes the PUT request 3 The SLA Manager searches the SLA in the database 4 The SLA Manager evaluates if the complete state is applicable on the SLA (only if the SLA is in observed or negotiating or renegotiating or remediating state) 5 The SLA Manager updates the value of the state 6 The SPECS component receives the URI to get the new state
Alternative Flows	3a There is no SLA associated to the unique identifier 7 An IllegalArgumentException is thrown 4a Completed state is not applicable on the SLA 7 An IllegalStateException is thrown
Extensions	
Inclusions	Update SLA state

3.1.2. SLA Manager Components

The SLA Manager is made up of two components: the first one represents the back-end that handle the persistence of all data, while the second one represents the front-end that exposes a REST API.

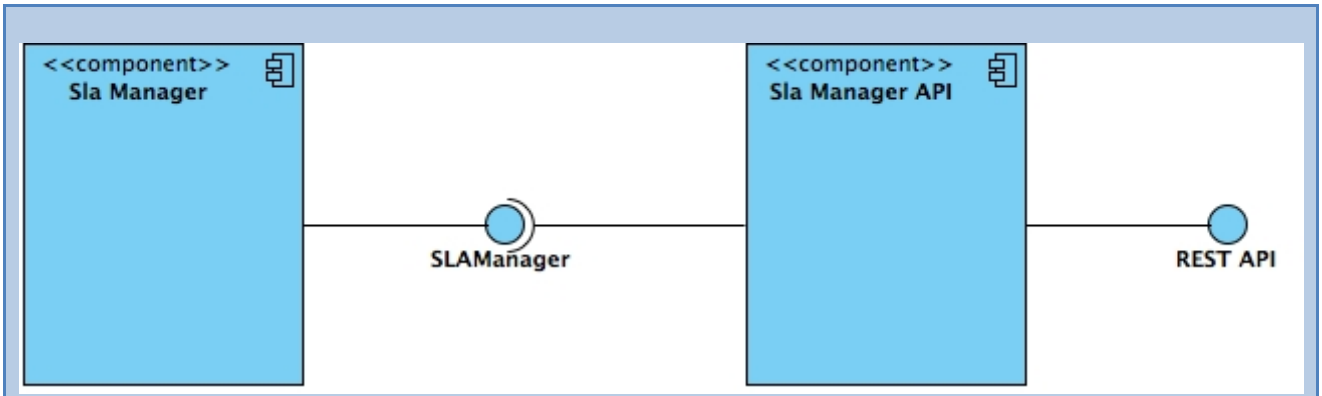

Figure 4: SLA Manager Component Diagram

Figure 4 shows the software organization of the component, including (i) the **SLA Manager API,** which offers the REST SLA API described in Deliverable D1.3, and (ii) the **SLA Manager component**, which acts as a backend, implementing the *SLAManager interface* used by the SLA Manager API component.

Table 21: SLA Manager Interface summarizes the SLA Manager interface used for the communication between the two components.

Table 21: SLA Manager Interface

Method Name	Input	Description	Output
createSLA	The xml representation of the SLA to store	Creates and stores a new SLA into the SLA manager	The URI of the created and stored SLA
getSLAs		Returns the URI of	The collection of SLAs

		all SLAs present in the component	
getSLA	The Unique Identifier of the SLA to be returned	Returns the xml representation of the SLA identified by the ID	The xml representation of the requested SLA
updateSLA	The Unique Identifier of the SLA to be updated and the new xml that represents it	Updates the SLA identified by the ID with the new xml	The URI of the updated SLA
deleteSLA	The Unique Identifier of the SLA to be deleted	Deletes the SLA identified by the ID from the component	The Unique Identifier of the deleted SLA
getAndLockSLA	The Unique Identifier of the SLA to be returned and locked	Returns the requested SLA an locks it to avoid any changes during use	The xml representation of the requested SLA
unlockSLA	The Unique Identifier of the SLA to be unlocked	Unlocks the requested SLA to allow its use	The Unique Identifier of the unlocked SLA
getState	The Unique SLA Identifier you want to get its state	Returns the value of the state of a specified SLA	The state of SLA
signSLA	The Unique Identifier of the SLA to be signed	Changes the state of the SLA to define it as signed by the user	The Unique Identifier of the signed SLA
observeSLA	The Unique Identifier of the SLA to be observed	Changes the state of the SLA to define it as observed	The Unique Identifier of the observed SLA
completeSLA	The Unique Identifier of the SLA to be completed	Changes the state of the SLA to define it as completed	The Unique Identifier of the completed SLA
terminateSLA	The Unique Identifier of the SLA to be terminated	Changes the state of the SLA to define it as terminated	The Unique Identifier of the terminated SLA
signalAlert	The Unique Identifier of the SLA to be signaled	Changes the state of the SLA to define it as alerted	The Unique Identifier of the signaled SLA
reNegotiate	The Unique Identifier of the SLA to be renegotiated	Changes the state of the SLA to define it as renegotiated	The Unique Identifier of the renegotiated SLA
getAnnotations	The Unique SLA Identifier you want to get its annotations	Returns the list of the annotations of a specified SLA	The collection of Annotations
createAnnotation	The Unique Identifier of the SLA and the annotation to be stored	Creates a new annotation on the specified SLA	The URI of the created and stored Annotation

updateAnnotation	The Unique Identifier of the SLA, the Unique Identifier of the annotation to be updated and the new annotation value	Updates the value of a stored SLA with the new one	The URI of the updated Annotation
deleteAnnotation	The Unique Identifier of the SLA and the Unique Identifier of the annotation to be deleted	Deletes the specified annotation from the SLA identified by the ID	The Unique Identifier of the deleted Annotation

The REST API offered by the SLA API component can be found in D1.3; furthermore, a set of new API has been added and they are reported in Annex A.

3.1.3. SLA Manager Classes

The SLA Manager is described through two different class diagrams: the first diagram represents all the packages and classes that compose the SLA Manager component, while the second one represents the packages and the classes of the SLA Manager API component.

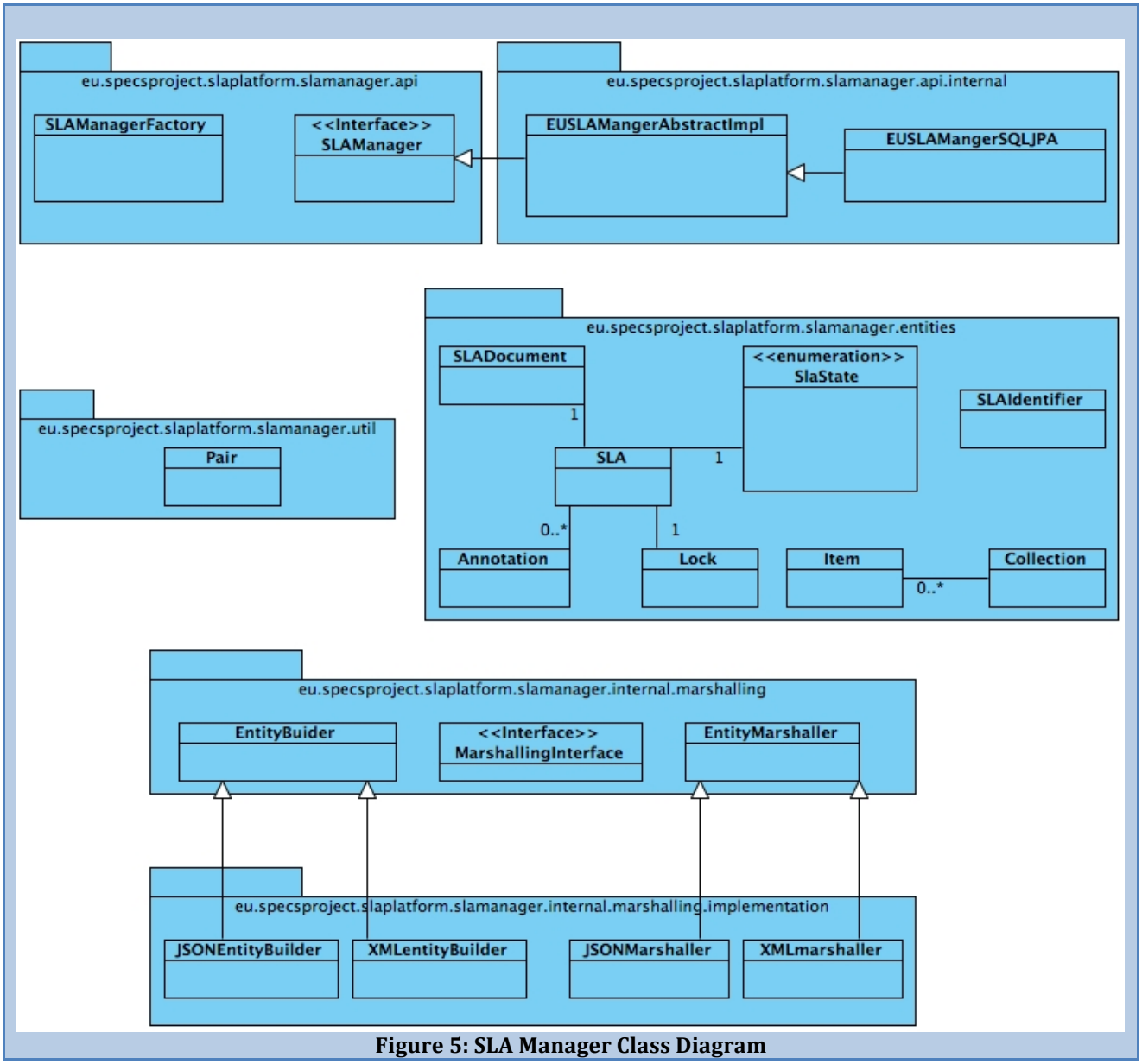

Figure 5: SLA Manager Class Diagram

As described in Figure 5, there are six packages involved, described in the following tables:

Table 22: slamanager package

Package name: eu.specsproject.slaplatform.slamanager	
Description: this package contains the classes useful to have access to all methods that the component exposes	
Class Name	**Description and Goal**
SLAManager	It is an interface that declares all the methods exposes by the component
SLAManagerFactory	It represents the factory useful to have access to all methods that the component exposes. Through the *getSLAManagerInstance* method is possible to access the instance of the class that implements the methods of the SLAManager Interface

Table 23: package slamanager.internal

Package name: eu.specsproject.slaplatform.slamanager.internal	
Description: this package contains the classes that implement the SLAManager interface and manage the SLAs persistence	
Class Name	**Description and Goal**
EUSLAManagerAbstractImpl	It is the implementation of the SLAManager interface. It contains the implementation of all methods declared in the interface; moreover, this class declares same abstract methods useful to manage the persistence of the SLAs
EUSLAManagerSQLJPA	It extends the previous class to implement all the abstract methods declared in it. The goal of this class is to manage the storing and the retrieving of the SLAs to/from a database. Its constructor receives an instance of *EntityManagerFactory* class of javax.persistence package

Table 24: package slamanager.util

Package name: eu.specsproject.slaplatform.slamanager.util	
Description: this package contains a set of utility classes useful to handle the processing of SLA data	
Class Name	**Description and Goal**
Pair	This class implements a hash map to associate two objects (e.g. the locks on SLAs)

Table 25: package slamanager.entities

Package name: eu.specsproject.slaplatform.slamanager.entities	
Description: this package contains the classes that represent the entities managed by the component	
Class Name	**Description and Goal**
Item	It represents an Item of the Collection Entity.
Collection	It represents the structure of the collection returned as response to a method call (e.g., *getSLAs* method).
SLA	This class represents the SLA that is managed by the component. Its structure is used to map the SLA on the database
SLADocument	It represents the content of the SLA as a String. The xml structure of an SLA is managed as an escaped String through this entity.
SLASTATE	This is an enumeration that represents all the possible states that an SLA can assume during its lifecycle.
Annotation	It represents the Annotation that can be added on an SLA. It is composed by a unique identifier and a description.

Lock	This entity is used to manage the lock state of an SLA. It is composed of a unique identifier and a timestamp that represents the instance of lock.
SLAIdentifier	It represents the Unique Identifier of an SLA. This entity is used as reference to the SLA currently managed by the component

Table 26: slamanager.internal.marshalling

Package name: eu.specsproject.slaplatform.slamanager.internal.marshalling	
Description: this package contains all the interfaces and the abstract classes useful to manage the marshalling and the unmarshalling of the entities	
Class Name	**Description and Goal**
EntityBuilder	It is an abstract class that declares the unmarshal method to be implemented to unmarshal an entity
EntityMarshaller	It is an abstract class that declares the marshal method to be implemented to marshal an entity
MarshallingInterface	It is the interface to be implemented from the entities that have to be marshal and so unmarshal

Table 27: slamanager.internal.marshalling.implementation

Package name: eu.specsproject.slaplatform.slamanager.internal.marshalling.implementation	
Description: this package contains all the classes that extend the classes of the previous package *eu.specsproject.slaplatform.slamanager.internal.marshalling* to implement the methods useful to marshaling and unmarshalling operations	
Class Name	**Description and Goal**
JSONentityBuilder	This class extends the EntityBuilder abstract and so it implements the unmarshall method for the JSON entities
JSONentityMarshaller	This class extends the EntityMarshaller abstract and so it implements the marshall method for the JSON entities
XMLentityBuilder	This class extends the EntityBuilder abstract and so it implements the unmarshall method for the XML entities
XMLentityMarshaller	This class extends the EntityMarshaller abstract and so it implements the marshall method for the XML entities

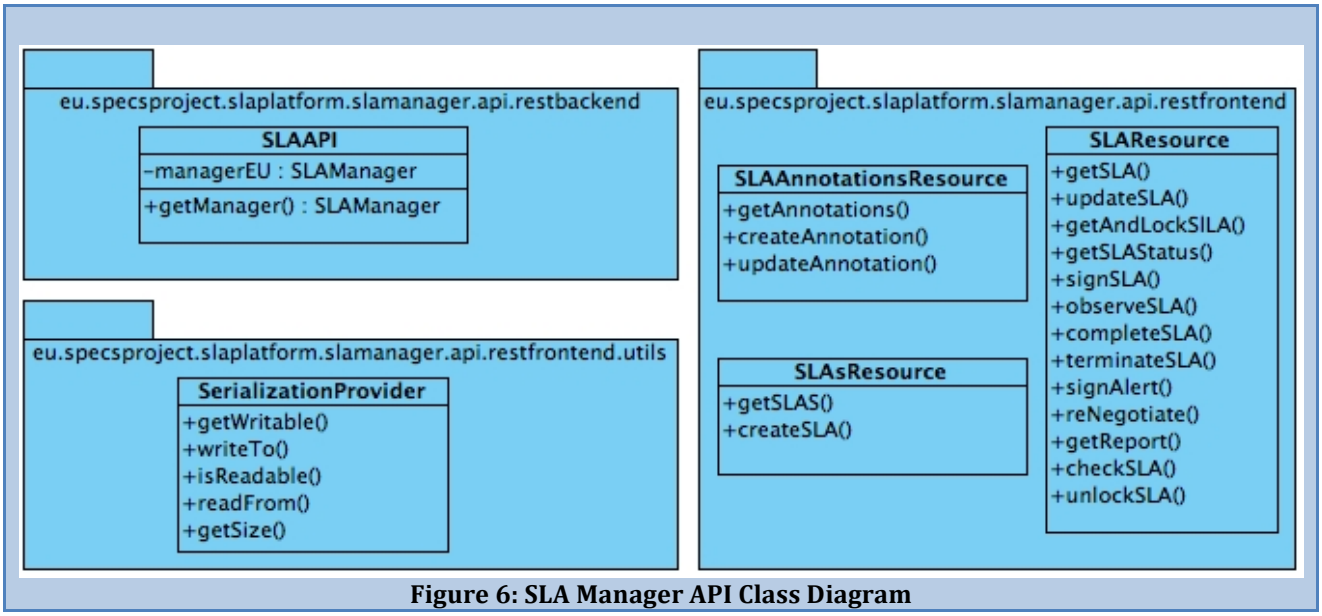

Figure 6: SLA Manager API Class Diagram

Figure 6 presents the class diagram of the SLA Manager API component. The SLA Manager API Class Diagram is composed of three packages that are described below:

Table 28: package slamanager.api.restbackend

Package name: eu.specsproject.slaplatform.slamanager.api.restbackend	
Description: this package contains the class that allows to access to the SLAManager in order to use all methods exposes by the back-end	
Class Name	**Description and Goal**
SLAAPI	It is a class that contains a method to get an instance of the *EUSLAManagerAbstractImpl* class defined in the back-end to use all the functions declared in the *SLAManager* interface

Table 29: package slamanager.api.restfrontend

Package name: eu.specsproject.slaplatform.slamanager.api.restfrontend	
Description: this package contains all the classes that represent the REST API resources. Each class implements the API useful to access the specific resource that it represents	
Class Name	**Description and Goal**
SLAsResources	It implements the collection of stored SLAs and offers methods to manage the collection.
SLAResouce	It implements the resource associated to a specific SLA and offers methods to manage it.
SLAAnnotationsResource	It implements the resource associated to the annotations of a specific SLA and offers methods to manage them.

Table 30: package slamanager.api.restfrontend.utils

Package name: eu.specsproject.slaplatform.slamanager.api.restfrontend.utils	
Description: this package contains the class useful to marshal and unmarshal the entities	
Class Name	**Description and Goal**
SerializationProvider	This class implements two interfaces:

	MessageBodyReader<T> useful to handle inbound entity representation-to-Java deserialization, and *MessageBodyWriter<T>* useful to handle the outbound entity Java-to-representation serialization.

3.1.4. List of requirements covered and discussion

The following table lists all requirements for the SLA Manager component, with a short description of the coverage is reported.

Table 31: SLA Manager requirements list

Requirements	Description	Coverage description
SLAPL_R10	**Get CSP SLA**	Through the SLA Manager it is possible to retrieve a CSP SLA using the REST API call (method: GET, URI: /cloud-sla/slas/{sla-id})
SLAPL_R11	**Add CSP SLA to repository**	Through the SLA Manager it is possible to store a new CSP SLA using the REST API call (method: POST, URI: /cloud-sla/slas/) where the body contains the value of the CSP SLA to store
SLAPL_R12	**Delete CSP SLA from repository**	Through the SLA Manager it is possible to remove a CSP SLA using the REST API call (method: DELETE, URI: /cloud-sla/slas/{sla-id})
SLAPL_R13	**Update CSP SLA in repository**	Through the SLA Manager it is possible to update the value of a CSP SLA using the REST API call (method: PUT, URI: /cloud-sla/slas/{sla-id}) where the body contains the new value of the CSP SLA
SLAPL_R14	**Search CSP SLA**	DEPRECATED (SLA Repository removed from design)
SLAPL_R17	**Create SPECS SLA repository**	DEPRECATED (SLA Repository removed from design)
SLAPL_R19	**Query SLA details**	Through the SLA Manager it is possible to retrieve an SLA detail using the REST API call (method: GET, URI: /cloud-sla/slas/{sla-id})
SLAPL_R20	**Query SLA status**	Through the SLA Manager it is possible to retrieve an SLA status using the REST API call (method: GET, URI: /cloud-sla/slas/{sla-id}/status)
SLAPL_R21	**Get SLA**	Through the SLA Manager it is possible to retrieve an SLA using the REST API call (method: GET, URI: /cloud-sla/slas/{sla-id})
SLAPL_R22	**Retrieve SLA document**	Through the SLA Manager it is possible to retrieve an SLA document using the REST API call (method: GET, URI: /cloud-sla/slas/{sla-id})
SLAPL_R23	**Search SLA**	UNCOVERED YET (SLA API enables annotations and state representation, but still it does not enable to retrieve an SLA according to its state).

SLAPL_R24	**Check SLA**	SLA Manager enables to verify SLA state and annotations
SLAPL_R25	**Annotate SLA**	Through the SLA Manager it is possible to add an annotation on an SLA using the REST API call (method: POST, URI: /cloud-sla/slas/{sla-id}/annotations) where the body contains the value of the annotation to add
SLAPL_R26	**Store SLA**	Through the SLA Manager it is possible to store a new SLA using the REST API call (method: POST, URI: /cloud-sla/slas/) where the body contains the value of the SLA to store
SLAPL_R27	**Create SLA**	Through the SLA Manager it is possible to create a new SLA using the REST API call (method: POST, URI: /cloud-sla/slas/) where the body contains the value of the SLA to create
SLAPL_R28	**Create SLA from template**	SLA Manager enables to store both SLA Offers and SLA templates. The management of coherence among the initial template and the SLA offers is delegated to the SLO Manager
SLAPL_R29	**Complete SLA**	Through the SLA Manager it is possible to define an SLA as completed using the REST API call (method: POST, URI: /cloud-sla/slas//{sla-id}/complete)
SLAPL_R30	**Get SLA status**	Through the SLA Manager it is possible to retrieve an SLA status using the REST API call (method: GET, URI: /cloud-sla/slas//{sla-id}/status)
SLAPL_R31	**Update SLA status**	Through the SLA Manager it is possible to update the state of an SLA using the Java method changeState that checks if the new state is applicable to the SLA and, if it is so, updates the state
SLAPL_R32	**Terminate SLA**	Through the SLA Manager it is possible to define an SLA as terminated using the REST API call (method: POST, URI: /cloud-sla/slas//{sla-id}/terminate)
SLAPL_R33	**Sign SLA**	Through the SLA Manager it is possible to define an SLA as signed using the REST API call (method: POST, URI: /cloud-sla/slas//{sla-id}/sign)
SLAPL_R34	**Change SLA**	SLA Manager enables SLA Updates (for what regards states and annotations)
CERT_R1	**Ensure that Platform complies with relevant security standard and best practices requirements**	SLA Manager maintains the SLA state according to the SPECS SLA Life Cycle and SLA Security Model, which is built according to the relevant standards. REST API is built according as much as possible according to HATEOS principles. Platform Security must be granted according to deployment choices. For additional consideration check D1.1.3, section 7
CERT_R2	**Ensure that Platform complies with legal and regulatory requirements**	Platform implementation is made according to best practices. Compliance to legal and regulatory requirements depends on the concrete deployment of the solution. The SPECS portfolio proposes solutions compliant with regulatory requirements. For additional consideration check D1.1.3, section 7
CERT_R3	**Ensure that for the specific SPECS Applications, the relevant certifications are applied**	This not functional requirement should be fulfilled by the SPECS Platform's owner to ensure that the scope of the SPECS Application has adopted relevant certifications in the context of the service provided.

3.2. Services Manager

The Services Manager is a component used to store all the information about the Security Mechanisms and Security Capabilities, and it enables the CRUD operations on each of them. The main design requirement is that it should be possible to get all the Security Mechanisms that are able to enforce or monitor one or more metrics, and satisfy one or more capabilities. Moreover, the component allows the end user to get the Metadata related to each Security Mechanism.

3.2.1. Services Manager Use Cases

This section describes all the interactions between a User API and the Services Manager. The use case diagram in Figure 7 has been created for this purpose.

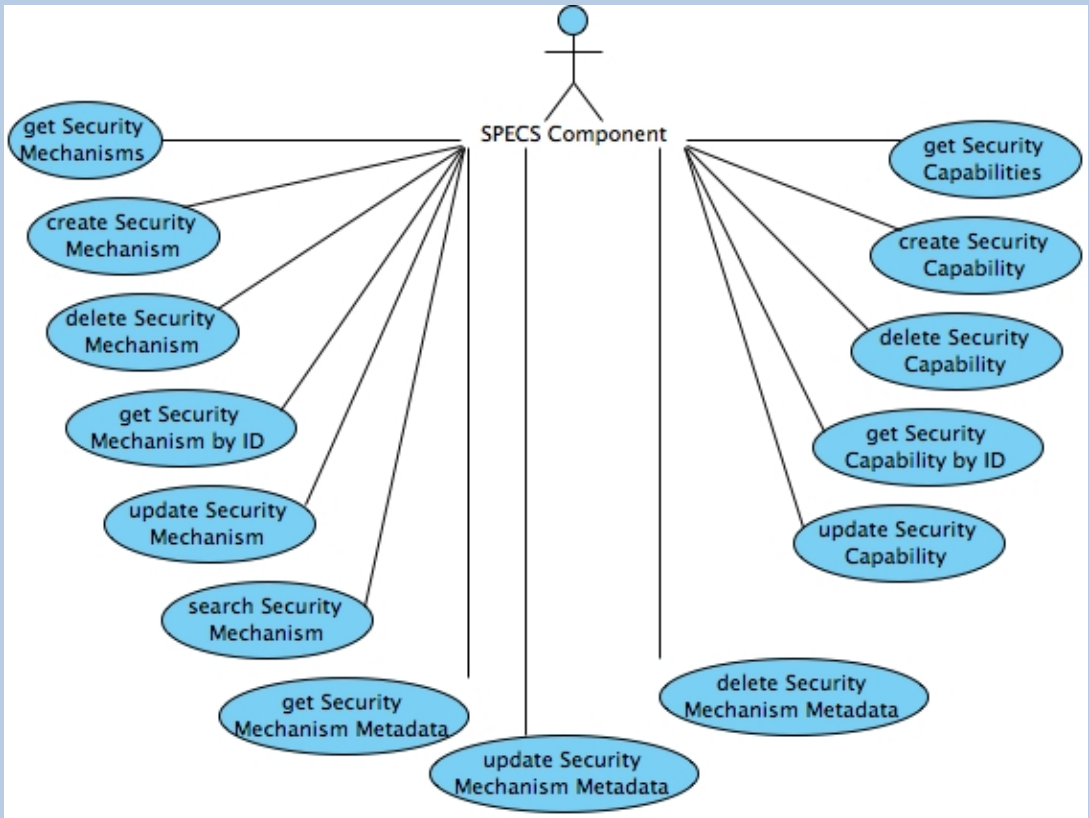

Figure 7: Services Manager Use Case Diagram

3.2.1.1. Description

The following tables describe in detail the use cases reported in the previous figure, following a simple UML case study template:

Table 32: Create Security Mechanism

Use Case Name	Create Security Mechanism
Actors	SPECS component
Entry (pre-conditions)	
Exit (post-conditions)	A new Security Mechanism is stored into the Services Manager
Input	The JSON that represents the Security Mechanism to store

Output	The URI of the created Security Mechanism
Flows of events	1 The SPECS component inserts the Security Mechanism JSON representation 2 The Services Manager stores the Security Mechanism into the database 3 The SPECS component receives the URI of the created Security Mechanism
Alternative Flows	
Extensions	
Inclusions	

Table 33: Get Security Mechanisms

Use Case Name	Get Security Mechanisms
Actors	SPECS component
Entry (pre-conditions)	
Exit (post-conditions)	A collection of Security Mechanism is returned to the SPECS component
Input	
Output	The URI collection of the Security Mechanisms
Flows of events	1 The SPECS component executes the GET request 2 The Services Manager searches all Security Mechanisms in the database 3 The SPECS component receives the collection URI of the Security Mechanisms
Alternative Flows	3a There are no stored Security Mechanisms 1 The returned collection is empty
Extensions	
Inclusions	

Table 34: Get Security Mechanism by ID

Use Case Name	Get Security Mechanism by ID
Actors	SPECS component
Entry (pre-conditions)	
Exit (post-conditions)	A specific Security Mechanism is returned to the SPECS component
Input	The Unique Identifier of the requested Security Mechanism
Output	The JSON that represents the Security Mechanism
Flows of events	1 The SPECS component inserts the ID of the Security Mechanism 2 The SPECS component executes the GET request 3 The Services Manager searches the Security Mechanism in the database 4 The SPECS component receives the JSON that represents the Security Mechanism
Alternative Flows	3a There is no Security Mechanism associated to the unique identifier 1 An IllegalArgumentException is thrown
Extensions	
Inclusions	

Table 35: Search Security Mechanism

Use Case Name	Search Security Mechanism
Actors	SPECS component
Entry (pre-conditions)	
Exit (post-conditions)	A collection of Security Mechanism is returned to the SPECS component
Input	The query parameters to search the Security Mechanisms
Output	The URI collection of the found Security Mechanisms
Flows of events	1 The SPECS component inserts the query parameters to search the Security Mechanism (in terms of enforced and/or monitored metrics and covered capabilities) 2 The SPECS component executes the GET request 3 The Services Manager searches all the Security Mechanisms that match the query parameters 4 The SPECS component receives the collection URI of the Security Mechanisms
Alternative Flows	3a There are no stored Security Mechanisms that match the query parameters 1 The returned collection is empty
Extensions	
Inclusions	

Table 36: Delete Security Mechanism

Use Case Name	Delete Security Mechanism
Actors	SPECS component
Entry (pre-conditions)	A Security Mechanism is stored in the component
Exit (post-conditions)	A specific Security Mechanism is removed
Input	The Unique Identifier of the Security Mechanism to be removed
Output	The Unique Identifier of the removed Security Mechanism
Flows of events	1 The SPECS component inserts the ID of the Security Mechanism 2 The SPECS component executes the DELETE request 3 The Services Manager searches the Security Mechanism in the database 4 The Services Manager removes the Security Mechanism from the database 5 The SPECS component receives the Unique Identifier of the Security Mechanism
Alternative Flows	3a There is no Security Mechanism associated to the unique identifier 1 An IllegalArgumentException is thrown
Extensions	
Inclusions	

Table 37: Update Security Mechanism

Use Case Name	Update Security Mechanism
Actors	SPECS component

Entry (pre-conditions)	A Security Mechanism is stored in the component
Exit (post-conditions)	The value of a specific Security Mechanism is updated
Input	The Unique Identifier of the Security Mechanism to update The new value of the Security Mechanism
Output	The Unique Identifier of the updated Security Mechanism
Flows of events	1 The SPECS component inserts the ID of the Security Mechanism 2 The SPECS component inserts the new value of the Security Mechanism 3 The SPECS component executes the PUT request 4 The Services Manager searches the Security Mechanism in the database 5 The Services Manager updates the Security Mechanism in the database 6 The SPECS component receives the Unique Identifier of the updated Security Mechanism
Alternative Flows	4a There is no Security Mechanism associated to the unique identifier 1 An IllegalArgumentException is thrown
Extensions	
Inclusions	

Table 38: Get Security Mechanism Metadata

Use Case Name	**Get Security Mechanism Metadata**
Actors	SPECS component
Entry (pre-conditions)	A Security Mechanism is stored in the component
Exit (post-conditions)	The Metadata value is returned to the SPECS component
Input	The Unique Identifier of the Security Mechanism
Output	The JSON that represents the Metadata
Flows of events	1 The SPECS component inserts the ID of the Security Mechanism 2 The SPECS component executes the GET request 3 The Services Manager searches the Security Mechanism in the database 4 The SPECS component receives the metadata of the Security Mechanism
Alternative Flows	3a There is no Security Mechanism associated to the unique identifier 1 An IllegalArgumentException is thrown
Extensions	
Inclusions	

Table 39: Update Security Mechanism Metadata

Use Case Name	**Update Security Mechanism Metadata**
Actors	SPECS component
Entry (pre-conditions)	A Security Mechanism is stored in the component and it has a metadata value
Exit (post-conditions)	The URI of the updated Security Mechanism The new Metadata value

Input	The Unique Identifier of the Security Mechanism
Output	The Unique Identifier of the updated Metadata
Flows of events	1 The SPECS component inserts the ID of the Security Mechanism 2 The SPECS component inserts the new value of the Metadata 3 The SPECS component executes the PUT request 4 The Services Manager searches the Security Mechanism in the database 5 The Services Manager updates the value of the found Metadata with the new one 6 The SPECS component receives the URI of the Security Mechanism
Alternative Flows	6a There is no Security Mechanism associated to the unique identifier 1 An IllegalArgumentException is thrown
Extensions	
Inclusions	

Table 40: Delete Security Mechanism Metadata

Use Case Name	**Delete Security Mechanism Metadata**
Actors	SPECS component
Entry (pre-conditions)	A Security Mechanism is stored in the component and it has a metadata value
Exit (post-conditions)	The URI of the removed metadata
Input	The Unique Identifier of the Security Mechanism
Output	The Unique Identifier of the removed Metadata
Flows of events	1 The SPECS component inserts the ID of the Security Mechanism 2 The SPECS component executes the DELETE request 3 The Services Manager searches the Security Mechanism in the database 4 The Services Manager checks if the Security Mechanism has a metadata 5 The Services Manager deletes the Metadata 6 The SPECS component receives the unique identifier of the removed Metadata
Alternative Flows	3a There is no Security Mechanism associated to the unique identifier 1 An IllegalArgumentException is thrown 6a There is no Metadata associated to the Security Mechanism 2 An IllegalArgumentException is thrown
Extensions	
Inclusions	

Table 41: Create Security Capability

Use Case Name	**Create Security Capability**
Actors	SPECS component
Entry (pre-conditions)	

Exit (post-conditions)	A new Security Capability is stored in the Services Manager
Input	The JSON that represents the Security Capability to store
Output	The URI of the created Security Capability
Flows of events	1 The SPECS component inserts the Security Capability JSON representation 2 The Services Manager stores the Security Capability into the database 3 The SPECS component receives the URI of the created Security Capability
Alternative Flows	
Extensions	
Inclusions	

Table 42: Get Security Capabilities

Use Case Name	**Get Security Capabilities**
Actors	SPECS component
Entry (pre-conditions)	
Exit (post-conditions)	A collection of Security Capability is returned to the SPECS component
Input	
Output	The URI collection of the Security Capabilities
Flows of events	1 The SPECS component executes the GET request 2 The Services Manager searches all Security Capabilities in the database 3 The SPECS component receives the collection URI of the Security Capabilities
Alternative Flows	3a There are no stored Security Capabilities 1 The returned collection is empty
Extensions	
Inclusions	

Table 43: Get Security Capability by ID

Use Case Name	**Get Security Capability by ID**
Actors	SPECS component
Entry (pre-conditions)	
Exit (post-conditions)	A specific Security Capability is returned to the SPECS component
Input	The Unique Identifier of the requested Security Capability
Output	The JSON that represents the Security Capability
Flows of events	1 The SPECS component inserts the ID of the Security Capability 2 The SPECS component executes the GET request 3 The Services Manager searches the Security Capability in the database 4 The SPECS component receives the JSON that represents the Security Capability
Alternative Flows	3a There is no Security Capability associated to the unique identifier 1 An IllegalArgumentException is thrown
Extensions	

Inclusions	

Table 44: Delete Security Capability

Use Case Name	**Delete Security Capability**
Actors	SPECS component
Entry (pre-conditions)	A Security Capability is stored in the component
Exit (post-conditions)	A specific Security Capability is removed
Input	The Unique Identifier of the Security Capability to remove
Output	The Unique Identifier of the removed Security Capability
Flows of events	1 The SPECS component inserts the ID of the Security Capability 2 The SPECS component executes the DELETE request 3 The Services Manager searches the Security Capability in the database 4 The Services Manager removes the Security Capability from the database 5 The SPECS component receives the Unique Identifier of the Security Capability
Alternative Flows	3a There is no Security Capability associated to the unique identifier 1 An IllegalArgumentException is thrown
Extensions	
Inclusions	

Table 45: Update Security Capability

Use Case Name	**Update Security Capability**
Actors	SPECS component
Entry (pre-conditions)	A Security Capability is stored in the component
Exit (post-conditions)	The value of a specific Security Capability is updated
Input	The Unique Identifier of the Security Capability to update The new value of the Security Capability
Output	The Unique Identifier of the updated Security Capability
Flows of events	1 The SPECS component inserts the ID of the Security Capability 2 The SPECS component inserts the new value of the Security Capability 3 The SPECS component executes the PUT request 4 The Services Manager searches the Security Capability in the database 5 The Services Manager updates the Security Capability in the database 6 The SPECS component receives the Unique Identifier of the updated Security Capability
Alternative Flows	4a There is no Security Capability associated to the unique identifier 1 An IllegalArgumentException is thrown
Extensions	
Inclusions	

3.2.2. Services Manager Components

The Service Manager is made up of three components. The first one represents the data model that is useful to define the structure of the entities used by the Service Manager (it will be described in Section 6)i. The second component represents the back-end that is useful to handle the persistence of all data, while the third one represents the front-end that exposes the REST API.

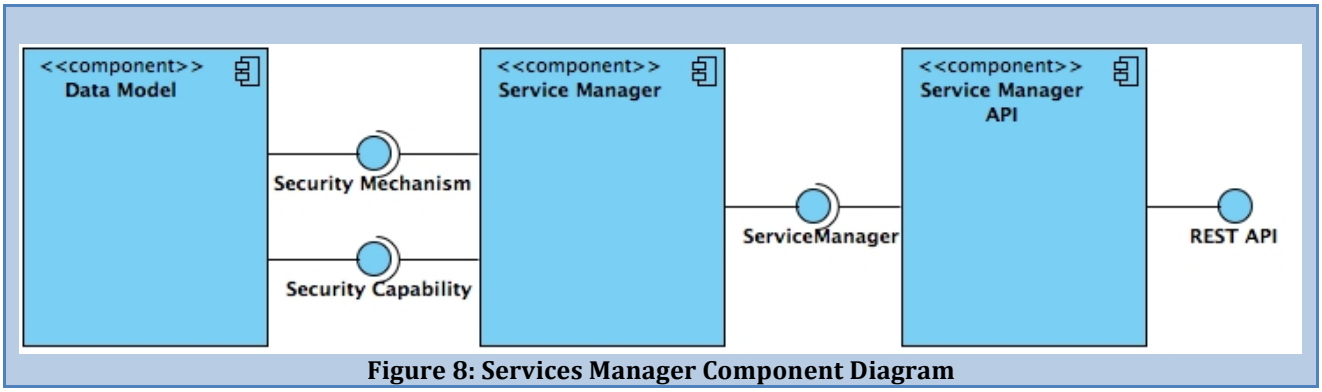

Figure 8: Services Manager Component Diagram

Figure 8 presents the software organization of the component, which is composed of: (i) **Service Manager API,** which offers the REST SLA API described in Deliverable D1.3, (ii) **Service Manager component**, which acts as a backend, implementing the *ServiceManager interface* used by the Service Manager API component, and (iii) **Data Model**, which exposes the entity that are used from the Service Manager component.

Table 46: Service Manager Interface summarizes the Service Manager interface used for communication between the two components.

Table 46: Service Manager Interface

Method Name	Input	Description	Output
createSM	The JSON formatted representation of the Security Mechanism to store	Creates and stores a new Security Mechanism into the manager	The URI of the created and stored Security Mechanism
searchSMs		Returns the URI of all Security Mechanisms present in the component	The collection of Security Mechanisms
retrieveSM	The Unique Identifier of the Security Mechanism to be returned	Returns the JSON with the representation of the Security Mechanism identified by the UI	The JSON representation of the requested Security Mechanism
updateSM	The Unique Identifier of the Security Mechanism to be updated and the new JSON that represents	Updates the Security Mechanism identified by the ID with the new JSON	The URI of the updated Security Mechanism

	it		
removeSM	The Unique Identifier of the Security Mechanism to be deleted	Deletes the Security Mechanism identified by the ID from the component	The Unique Identifier of the deleted Security Mechanism
retrieveSMMetadata	The Unique Security Mechanism Identifier you want to get its metadata	Returns the value of the metadata of a specified Security Mechanism	The metadata of the Security Mechanism
createSC	The JSON formatted representation of the Security Capability to store	Creates and stores a new Security Capability into the manager	The URI of the created and stored Security Capability
searchSCs		Returns the URI of all Security Capabilities present in the component	The collection of Security Capabilities
retrieveSC	The Unique Identifier of the Security Capability to be returned	Returns the JSON with the representation of the Security Capability identified by the ID	The JSON representation of the requested Security Capability
updateSC	The Unique Identifier of the Security Capability to be updated and the new JSON that represents it	Updates the Security Capability identified by the ID with the new JSON	The URI of the updated Security Capability
removeSC	The Unique Identifier of the Security Capability to be deleted	Deletes the Security Capability identified by the ID from the component	The Unique Identifier of the deleted Security Capability

The detailed explanation of the REST API offered by the Service API component can be found in Deliverable 1.3.

3.2.3. Services Manager Classes

The Services Manager is described through three different class diagrams: the first diagram represents all the packages and the classes that compose the Data Model, the second diagram represents the structure of the Services Manager component, while the third diagram represents the packages and the classes of the Services Manager API component.

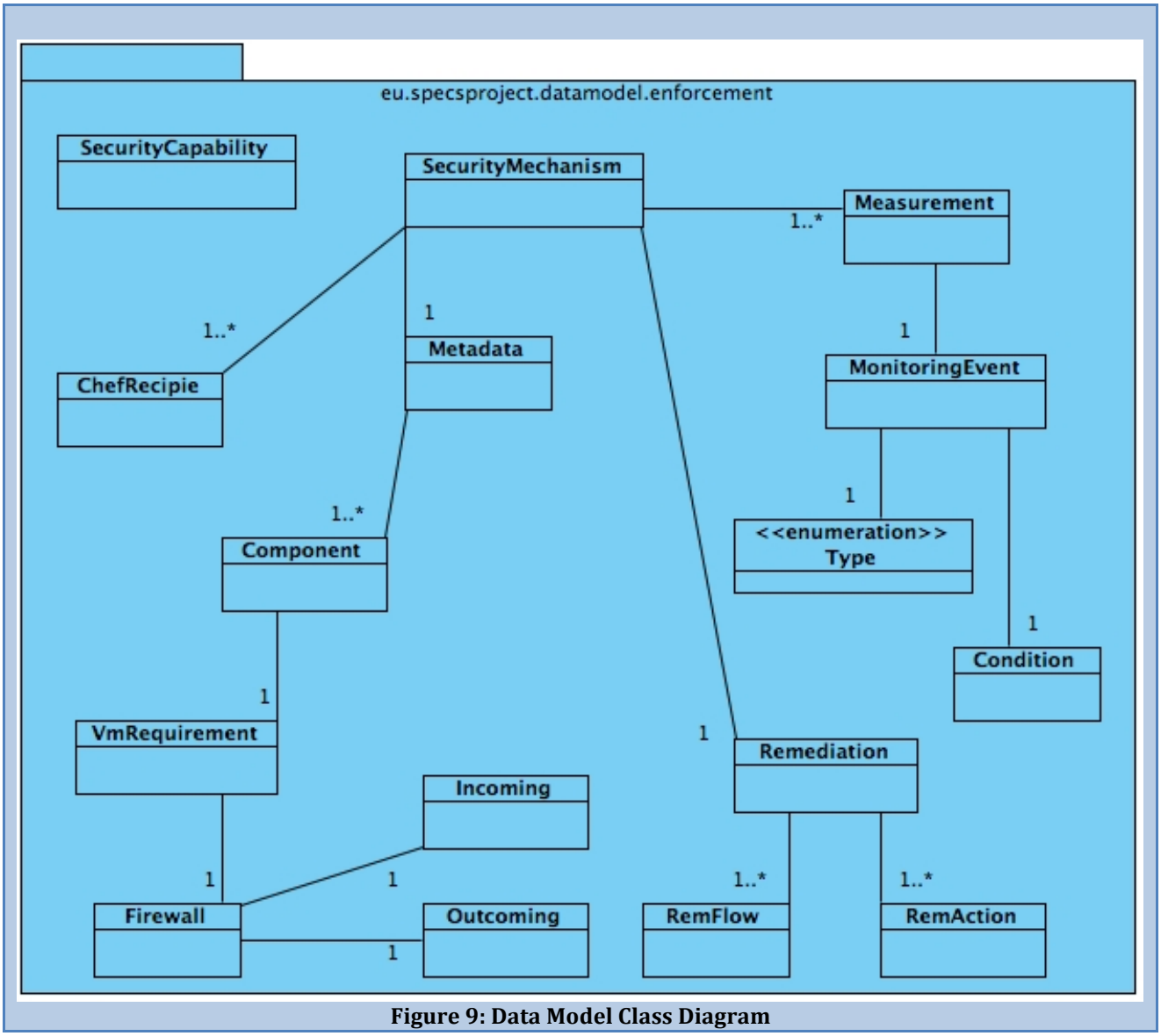

Figure 9: Data Model Class Diagram

Figure 9 shows the entities used by the Services Manager back-end component. In particular, both the used entities are in the same package, which is described in the following table:

Table 47: package datamodel.enforcement

Package name: eu.specsproject.datamodel.enforcement	
Description: this package contains the classes that represents the entities managed by the component	
Class Name	**Description and Goal**
SecurityCapability	It represents the structure of the Security Capability managed by the Services Manager component
SecurityMechanism	It represents the structure of the Security Mechanism managed by the Services Manager component
ChefRecipe	This class contains the information about the chef recipe that have to be used in order to install the security mechanism. One Security

	Mechanism can have one or several ChefRecipe
Metadata	It represents the metadata of the Security Mechanism. Its has an association with the Security Mechanism
Component	This class contains information about the element that compose the Security Mechanism. A metadata can have one or more components
VMrequirement	It represents the requirement of the component to be installed
Firewall	This entity is used to manage possible rules to be set on the firewall. It is composed of Incoming and Outcoming information
Remediation	It represents the remediation that can be performed for the security mechanism. Each remediation has one or several remediation flows (RemFlow) and one or several remediation actions (RemAction)
Measurement	This class represents the measurement that can be used to evaluate the state of the security mechanism. One Security Mechanism can have one or several Measurements
MonitoringEvent	The monitoring event class contains information about the events monitored on the Security Mechanism. Each event is composed of a type (Type) and a component (Component)

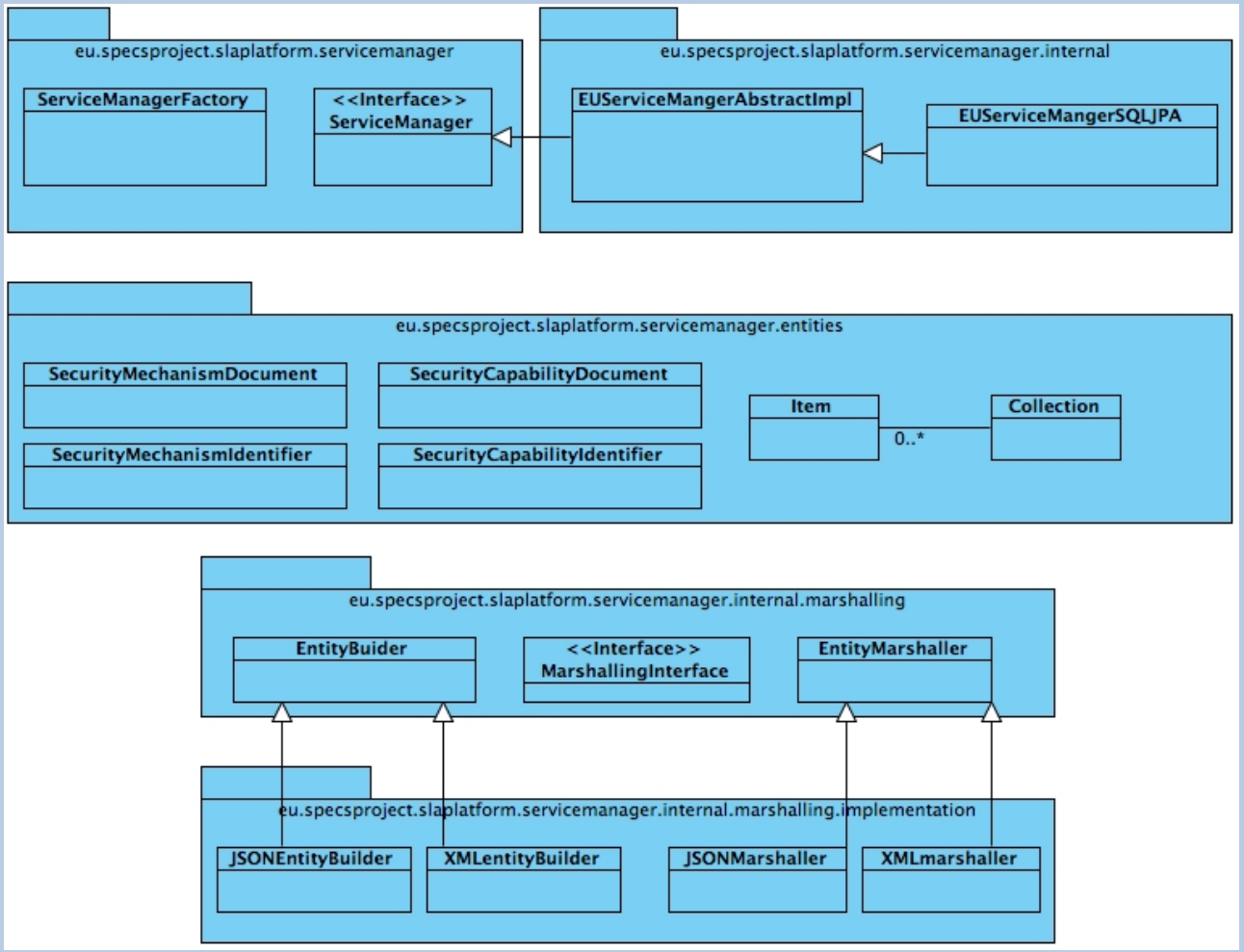

Figure 10: Services Manager Class Diagram

As described in Figure 10, Services Manager is composed of five packages, whose details are shown in the following tables:

Table 48: slamanager package

Package name: eu.specsproject.slaplatform.servicemanager	
Description: this package contains the classes useful to have access to all methods that the component exposes	
Class Name	**Description and Goal**
ServiceManager	It is an interface that declares all the methods exposed by the component
ServiceManagerFactory	It represents the factory useful to have access to all methods that the component exposes. Through the *getServiceManagerInstance* method it is possible to access the instance of the class that implements the methods of the ServiceManager Interface

Table 49: package slamanager.internal

Package name: eu.specsproject.slaplatform.servicemanager.internal
Description: this package contains the classes that implement the ServiceManager interface

and manage the Security Mechanisms and Security Capabilities persistence	
Class Name	**Description and Goal**
EUServiceManagerAbstractImpl	It is the implementation of the ServiceManager interface. It contains the implementation of all methods declared in the interface; moreover, this class declares same abstract methods useful to manage the persistence of the Security Mechanisms and Security Capabilities
EUServiceManagerSQLJPA	It extends the previous class to implement all the abstract methods declared in it. The goal of this class is to manage the storing and the retrieving of the Security Mechanisms and Capabilities to/from a database. Its constructor receives an instance of *EntityManagerFactory* class of javax.persistence package

Table 50: package servicemanager.entities

Package name: eu.specsproject.slaplatform.servicemanager.entities	
Description: this package contains the classes that represents the entities managed by the component	
Class Name	**Description and Goal**
Collection	It represents the structure of the collection returned as response tof a REST call (e.g., *searchSMs* method)
Item	It represents an Item of the Collection Entity.
SecurityMechanismDocument	It represents the content of the SecurityMechanism as a String. The JSON structure of a Security Mechanism is managed as a String through this entity.
SecurityMechanismIdentifier	It represents the Unique Identifier of a Security Mechanism. This entity is used as reference to the Security Mechanism currently managed by the component
SecurityCapabilityDocument	It represents the content of the SecurityCapability as a String. The JSON structure of a Security Capability is managed as a String through this entity.
SecurityCapabilityIdentifier	It represents the Unique Identifier of a Security Capability. This entity is used as reference to the Security Capability currently managed by the component

Table 51: servicemanager.internal.marshalling

Package name: eu.specsproject.slaplatform.servicemanager.internal.marshalling
Description: this package contains all the interfaces and the abstract classes useful to manage

the marshalling and the unmarshalling of the entities	
Class Name	**Description and Goal**
EntityBuilder	It is an abstract class that declares the unmarshal method to unmarshal an entity
EntityMarshaller	It is an abstract class that declares the marshal method to marshal an entity
MarshallingInterface	It is the interface to be implemented from the entities that have to be marshal and unmarshal

Table 52: servicemanager.internal.marshalling.implementation

Package name: eu.specsproject.slaplatform. servicemanager.internal.marshalling.implementation	
Description: this package contains all the classes that extend the classes of the previous package *eu.specsproject.slaplatform.servicemanager.internal.marshalling* to implement the methods useful for the marshaling and unmarshalling operations	
Class Name	**Description and Goal**
JSONentityBuilder	This class extends the EntityBuilder abstract and so it implements the unmarshall method for the JSON entities
JSONentityMarshaller	This class extends the EntityMarshaller abstract and so it implements the marshall method for the JSON entities
XMLentityBuilder	This class extends the EntityBuilder abstract and so it implements the unmarshall method for the XML entities
XMLentityMarshaller	This class extends the EntityMarshaller abstract and so it implements the marshall method for the XML entities

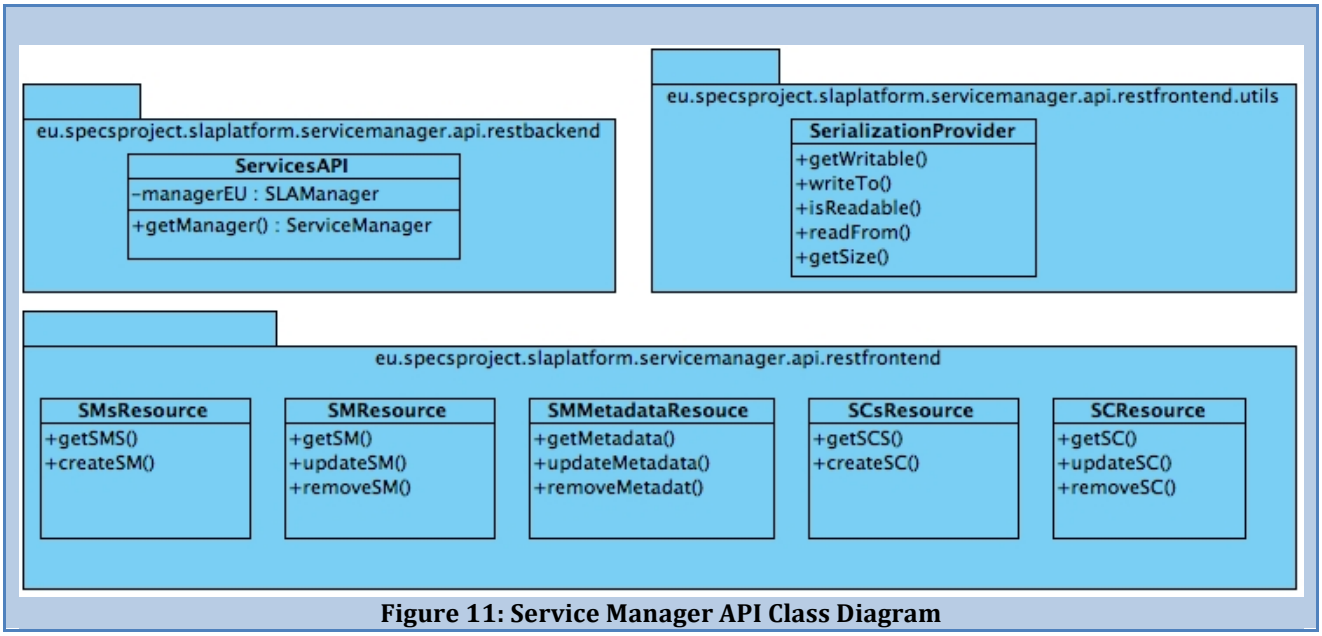

Figure 11: Service Manager API Class Diagram

The class diagram of the Service Manager API component is shown in Figure 11. The Service Manager API Class Diagram is composed of three packages that are described below:

Table 53: package servicemanager.api.restbackend

Package name: eu.specsproject.slaplatform.servicemanager.api.restbackend	
Description: this package contains the class that allows to access to the ServiceManager in order to use all methods exposed by the back end	
Class Name	**Description and Goal**
ServicesAPI	It is a class that contains a method to get an instance of the *EUServiceManagerAbstractImpl* class defined in the back-end to use all the functions declared in the *ServiceManager* interface

Table 54: package servicemanager.api.restfrontend

Package name: eu.specsproject.slaplatform.servicemanager.api.restfrontend	
Description: this package contains all the classes that represents the REST API resources. Each class implements all the API useful to access at the specific resource that it represents	
Class Name	**Description and Goal**
SMsResources	It implements the collection of stored Security Mechanisms and offers methods to manage the collection.
SMResouce	It implements the resource associated to a specific Security Mechanism and offers methods to manage it.
SMMetadataResource	It implements the resource associated to the metadata of a specific Security Mechanism and offers methods to manage them.
SCsResources	It implements the collection of stored Security Capabilities and offers methods to manage the collection.

SCResouce	It implements the resource associated to a specific Security Capability and offers methods to manage it.

Table 55: package slamanager.api.restfrontend.utils

Package name: eu.specsproject.slaplatform.servicemanager.api.restfrontend.utils	
Description: this package contains the class useful to marshal and unmarshal the entities	
Class Name	**Description and Goal**
SerializationProvider	This class implements two interfaces: *MessageBodyReader<T>* useful to handle inbound entity representation-to-Java deserialization, and *MessageBodyWriter<T>* useful to handle the outbound entity Java-to-representation serialization.

3.2.4. List of requirements covered and discussion

The following table lists all requirements for the Service Manager component with a short description of the coverage.

Table 56: Service Manager requirements list

Requirements	Description	Coverage description
SLAPL_R1	**Create service repository**	Service Repository is created together with Service Manager component and its persistence solution. It is created by enabling platform through the dedicated recipe
SLAPL_R2	**Create CSP SLA repository**	DEPRECATED (SLA Repository removed from design) CSP SLAs can be maintained using the SLA manager.
SLAPL_R5	**Get service from repository**	Through the Service Manager and/or the Security Metrics Catalogue it is possible to retrieve a service using the REST API call (method: GET, URI: /cloud-sla/service-type/{service-id})
SLAPL_R6	**Add service to repository**	Through the Service Manager and/or the Security Metrics Catalogue it is possible to store a new service using the REST API call (method: POST, URI: /cloud-sla/service-type) where the body contains the new service
SLAPL_R7	**Delete service from repository**	Through the Service Manager and/or the Security Metrics Catalogue it is possible to remove a service using the REST API call (method: DELETE, URI: cloud-sla/service-type/{service-id})
SLAPL_R8	**Update service in repository**	Through the Service Manager and/or the Security Metrics Catalogue it is possible to update the value of a service using the REST API call (method: PUT, URI: /cloud-sla/service-type/{service-id}) where the body contains the new value of the service
SLAPL_R9	**Search service in repository**	Through the Service Manager and/or the Security Metrics Catalogue it is possible to search a service that has certain

		features using the REST API call (method: GET, URI: /cloud-sla/service-type/{service-id}) where the query params define the requested features

3.3. Security Metrics Catalogue

The Security Metrics Catalogue is a component used to store and manage all the information about the Security Metrics. It enables the CRUD operations on each of them.
The main design requirement is that should be possible to get all the Security Metrics starting from their unique identifier (Reference ID). Moreover, the component allows the end user to get an xml file that represents a specific Security Metric. Finally, the end user has the possibility to extract an SQLite file (.db extention) that contains all the Security Metrics stored in the component.

3.3.1. Security Metrics Catalogue Use Cases

The interactions between a generic SPECS component and the Security Metrics catalogue will be described in the following subsections. Figure 12 illustrates the use case diagrams related to this component.

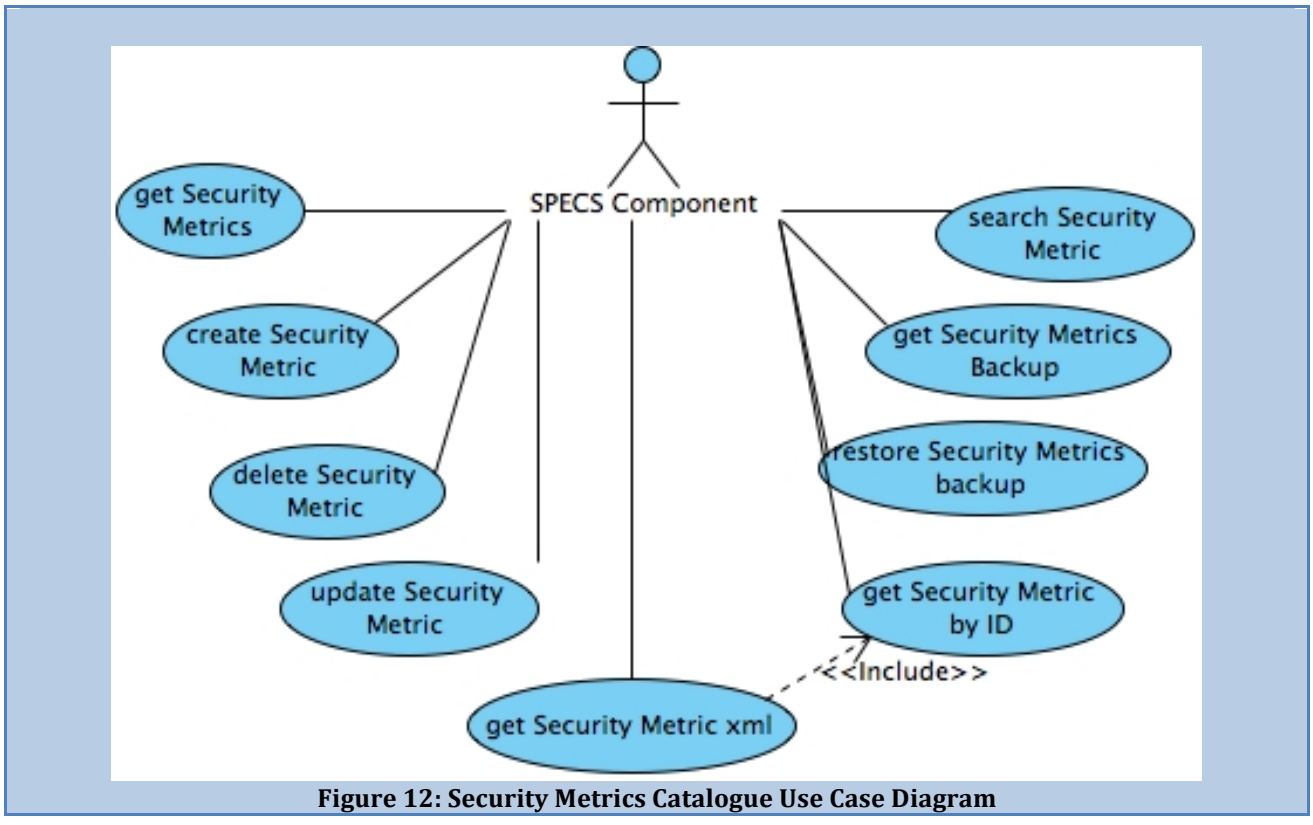

Figure 12: Security Metrics Catalogue Use Case Diagram

3.3.1.1. Description

The following tables describe in detail the use cases reported in the previous figure:

Table 57: Create Security Metric

Use Case Name	Create Security Metric
Actors	SPECS component
Entry (pre-conditions)	
Exit (post-conditions)	A new Security Metric is stored into the Security Metrics Catalogue
Input	The JSON that contains the xml representation the Security Metric to store
Output	The URI of the created Security Metric
Flows of events	1 The SPECS component inserts the Security Metric JSON representation 2 The Security Metrics Catalogue stores the Security Metric into the database 3 The SPECS component receives the URI of the created Security Metric
Alternative Flows	
Extensions	
Inclusions	

Table 58: Get Security Metrics

Use Case Name	Get Security Metrics
Actors	SPECS component
Entry (pre-conditions)	
Exit (post-conditions)	A collection of Security Metric is returned to the SPECS component
Input	
Output	The URI collection of the Security Metrics
Flows of events	1 The SPECS component executes the GET request 2 The Security Metrics Catalogue Manager searches all Security Metrics in the database 3 The SPECS component receives the collection URI of the Security Metrics
Alternative Flows	3a There are no stored Security Metrics 1 The returned collection is empty
Extensions	
Inclusions	

Table 59: Get Security Metric by ID

Use Case Name	Get Security Metric by ID
Actors	SPECS component
Entry (pre-conditions)	
Exit (post-conditions)	A specific Security Metric is returned to the SPECS component
Input	The Unique Identifier of the requested Security Metric
Output	The JSON that contains the xml representation of the Security Metric
Flows of events	1 The SPECS component inserts the ID of the Security Metric 2 The SPECS component executes the GET request 3 The Security Metrics Catalogue searches the Security Metric in the database 4 The SPECS component receives the JSON that represents the

	Security Metric
Alternative Flows	3a There is no Security Metric associated to the unique identifier 1 An IllegalArgumentException is thrown
Extensions	
Inclusions	

Table 60: Delete Security Metric

Use Case Name	**Delete Security Metric**
Actors	SPECS component
Entry (pre-conditions)	A Security Metric is stored into the component
Exit (post-conditions)	A specific Security Metric is removed
Input	The Unique Identifier of the Security Metric to be removed
Output	The Unique Identifier of the removed Security Metric
Flows of events	1 The SPECS component inserts the ID of the Security Metric 2 The SPECS component executes the DELETE request 3 The Security Metrics Catalogue searches the Security Metric in the database 4 The Security Metrics Catalogue removes the Security Metric from the database 5 The SPECS component receives the Unique Identifier of the Security Metric
Alternative Flows	3a There is no Security Metric associated to the unique identifier 1 An IllegalArgumentException is thrown
Extensions	
Inclusions	

Table 61: Update Security Metric

Use Case Name	**Update Security Metric**
Actors	SPECS component
Entry (pre-conditions)	A Security Metric is stored into the component
Exit (post-conditions)	The value of a specific Security Metric is updated
Input	The Unique Identifier of the Security Metric to update the new value of the Security Metric
Output	The Unique Identifier of the updated Security Metric
Flows of events	1 The SPECS component inserts the ID of the Security Metric 2 The SPECS component inserts the new value of the SLA Security Metric 3 The SPECS component executes the PUT request 4 The Security Metrics Catalogue searches the Security Metric in the database 5 The Security Metrics Catalogue updates the Security Metric in the database 6 The SPECS component receives the Unique Identifier of the updated Security Metric
Alternative Flows	3a There is no Security Metric associated to the unique identifier 1 An IllegalArgumentException is thrown
Extensions	
Inclusions	

Table 62: Search Security Metrics

Use Case Name	Search Security Metrics
Actors	SPECS component
Entry (pre-conditions)	
Exit (post-conditions)	A collection of Security Metric is returned to the SPECS component
Input	The query parameter to search the Security Metric, it can be *abstract* or *concrete*
Output	The URI collection of the found Security Metrics
Flows of events	1 The SPECS component inserts the query parameter to search the Security Metric (in terms of *type of metrics*, abstract or concrete) 2 The SPECS component executes the GET request 3 The Security Metrics Catalogue searches all the Security Metrics that match the query parameter 4 The SPECS component receives the collection URI of the Security Metrics
Alternative Flows	3a There are no stored Security Metrics that match the query parameter value 1 The returned collection is empty
Extensions	
Inclusions	

Table 63: Get Security Metrics Backup

Use Case Name	Get Security Metrics Backup
Actors	SPECS component
Entry (pre-conditions)	A database of security metrics exists
Exit (post-conditions)	The file that represents the metric database backup is returned to the SPECS component
Input	The name of the database to return
Output	The metric database backup
Flows of events	1 The SPECS component inserts the NAME of the database 2 The SPECS component executes the GET request to get the backup 3 The Security Metrics Catalogue searches the database with the correct name 4 The SPECS component receives the file that represents the metrics backup
Alternative Flows	3a There is no database associated with the name 1 An IllegalArgumentException is thrown
Extensions	
Inclusions	

Table 64: Restore Security Metrics Backup

Use Case Name	Restore Security Metrics Backup
Actors	SPECS component
Entry (pre-conditions)	A database of security metrics exists

Exit (post-conditions)	A back-up of a metric database is restored in the component
Input	The file that represents the database backup to be restored
Output	A Boolean that represents the result of the operation
Flows of events	1 The SPECS component inserts the file of the backup to be restored 2 The SPECS component executes the PUT request 3 The Security Metrics Catalogue searches the database with the correct name to be restored 4 The Security Metrics Catalogue updates the file with the new one 5 The SPECS component receives the result of the operation
Alternative Flows	3a There is no database associated with the name 1 An IllegalArgumentException is thrown
Extensions	
Inclusions	

Table 65: Get Security Metric xml

Use Case Name	**Get Security Metric xml**
Actors	SPECS component
Entry (pre-conditions)	A Security Metric is stored into the component
Exit (post-conditions)	The xml that represents a specific SLA is returned to the SPECS component
Input	The Unique Identifier of the requested Security Metric
Output	The xml file that represents the Security Metric
Flows of events	1 The SPECS component inserts the ID of the Security Metric 2 The SPECS component executes the GET request to get the xml file 3 The Security Metrics Catalogue searches the Security Metric in the database 4 The Security Metrics Catalogue get the Security Metric and make the xml file 5 The SPECS component receives the xml file that represents the Security Metric
Alternative Flows	3a There is no Security Metric associated to the unique identifier 1 An IllegalArgumentException is thrown
Extensions	
Inclusions	Get Security Metric by ID

3.3.2. Security Metrics Catalogue Components

The Security Metrics Catalogue is made up of two components: the first one represents the back-end that is used to handle the persistence of all data, while the second one represents the front-end that exposes the REST API.

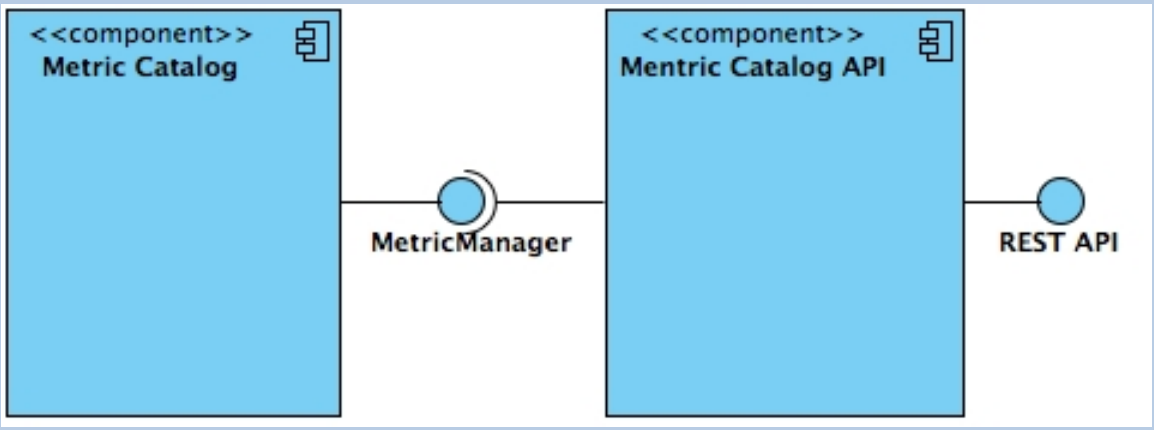

Figure 13: Security Metrics Catalogue Component Diagram

Figure 13 shows the software organization of the component, which is composed of: (i) **Metric Catalogue API,** which offers the REST SLA API described in Deliverable D1.3, and (ii) **Metric Catalogue component**, which acts as a backend, implementing the *MetricManager* interface used by the Metric Catalogue API component.

Table 64 summarizes the Metric Catalogue interface used for the communication between the two components.

Table 66: Security Metrics Catalogue Interface

Method Name	Input	Description	Output
createSMT	A JSON that contains the xml representation of the Security Metric to store	Creates and stores a new Security Metric into the manager	The URI of the created and stored Security Metric
getSMTs		Returns the URI of all Security Metrics present in the component	The collection of Security Metrics
getSMT	The Unique Identifier of the Security Metric to be returned	Returns a JSON that contains the xml representation of the Security Metric identified by the UI	The JSON representation of the requested Security Metric
updateSMT	The Unique Identifier of the Security Metric to be updated and the new JSON that represents it	Updates the Security Metric identified by the ID with the new JSON value	The URI of the updated Security Metric
deleteSMT	The Unique Identifier of the Security Metric to be deleted	Deletes the Security Metric identified by the ID from the component	The Unique Identifier of the deleted Security Metric
getMetricsBackup	The name of the	Returns a file that	The backup of the

	database to be returned	represents the backup of the requested database	metrics database
restoreMetricsBackup	The file that represents the backup to restore and the name of database to restore	Changes the file that represents the database with the new one	The result of the operation (true/false)

The REST API offered by the Service API component can be found in Deliverable D1.3; furthermore, a set of new API has been added and they are reported in Annex A.

3.3.3. Security Metrics Catalogue Classes

The Security Metrics Catalogue is described through two class diagrams. The first diagram represents all the packages and the classes that compose the Metric Catalogue component, while the second diagram represents the packages and the classes of the Metric Catalogue API component.

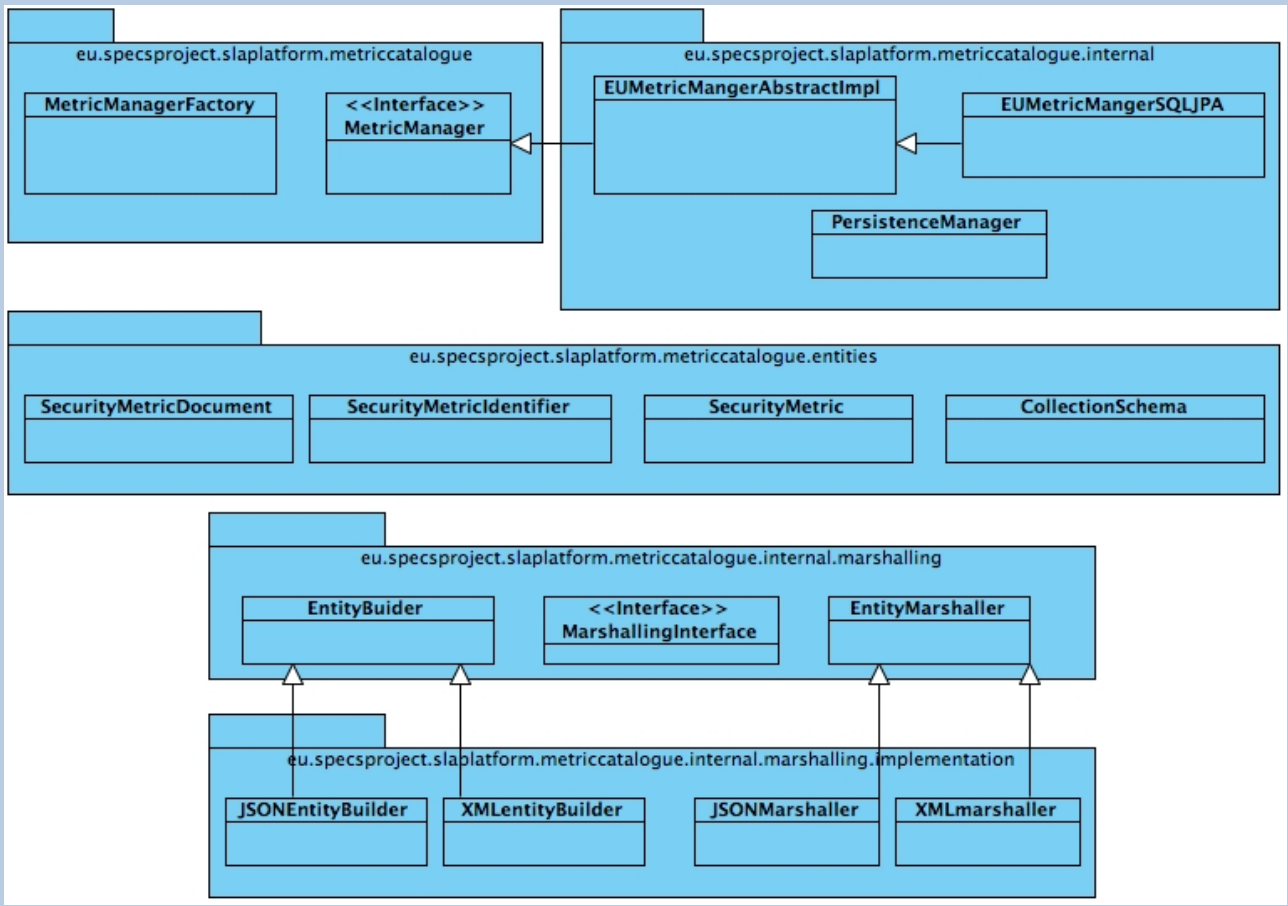

Figure 14: Metric Catalogue Class Diagram

As described in the 'Metric Catalogue Class Diagram' figure, the Security Metrics Catalogue Manager is composed of five packages whose details are shown in the following tables:

Table 67: metriccatalogue package

Package name: eu.specsproject.slaplatform.metriccatalogue	
Description: this package contains the classes useful to have access to all methods that the component exposes	
Class Name	**Description and Goal**
MetricManager	It is an interface that declares all the methods exposed by the component
MetricManagerFactory	It represents the factory useful to have access to all methods that the component exposes. Through the *getMetricManagerInstance* method it .is possible to access the instance of the class that implements the methods of the MetricManager Interface

Table 68: package metriccatalogue.internal

Package name: eu.specsproject.slaplatform.metriccatalogue.internal	
Description: this package contains the classes that implement the MetricManager interface and manage the Security Metrics persistence	
Class Name	**Description and Goal**
EUMetricManagerAbstractImpl	It is the implementation of the MetricManager interface. It contains the implementation of all methods declared in the interface; moreover. this class declares same abstract methods useful to manage the persistence of the Security Metrics
EUMetricManagerSQLJPA	It extends the previous class to implement all the abstract methods declared in it. The goal of this class is to manage the storing and the retrieving of the Security Metrics to/from a database. Its constructor receives an instance of *EntityManagerFactory* class of javax.persistence package

Table 69: package metriccatalogue.entities

Package name: eu.specsproject.slaplatform.metriccatalogue.entities	
Description: this package contains the classes that represents the entities managed by the component	
Class Name	**Description and Goal**
CollectionSchema	It represents the structure of the collection returned as response of a REST call (e.g. *getSMTS* method).
SecurityMetric	This class represents the Security Metric that is managed by the component. Its structure is used to map the Security Metric on the database
SecurityMetricDocument	It represents the content of the Security Metric as a String. The xml structure of a Security Metric is managed as an escaped String through this entity.

SecurityMetricIdentifier	It represents the Unique Identifier of a Security Metric. This entity is used as reference to the Security Metric currently managed by the component

Table 70: metriccatalogue.internal.marshalling

Package name: eu.specsproject.slaplatform. metriccatalogue.internal.marshalling	
Description: this package contains all the interfaces and the abstract classes useful to manage the marshalling and the unmarshalling of the entities	
Class Name	**Description and Goal**
EntityBuilder	It is an abstract class that declare the unmarshal method to unmarshal an entity
EntityMarshaller	It is an abstract class that declare the marshal method to marshal an entity
MarshallingInterface	It is the interface to be implemented from the entities that have to be marshalled and unmarshalled

Table 71: metriccatalogue.internal.marshalling.implementation

Package name: eu.specsproject.slaplatform. metriccatalogue.internal.marshalling.implementation	
Description: this package contains all the classes that extend the classes of the previous package *eu.specsproject.slaplatform.* metriccatalogue*.internal.marshalling* to implements the methods useful to the marshaling and unmarshalling operations	
Class Name	**Description and Goal**
JSONentityBuilder	This class extends the EntityBuilder abstract and so it implements the unmarshall method for the JSON entities
JSONentityMarshaller	This class extends the EntityMarshaller abstract and so it implements the marshall method for the JSON entities
XMLentityBuilder	This class extends the EntityBuilder abstract and so it implements the unmarshall method for the XML entities
XMLentityMarshaller	This class extends the EntityMarshaller abstract and so it implements the marshall method for the XML entities

Is now presented the class diagram related to the Metric Catalogue API component.

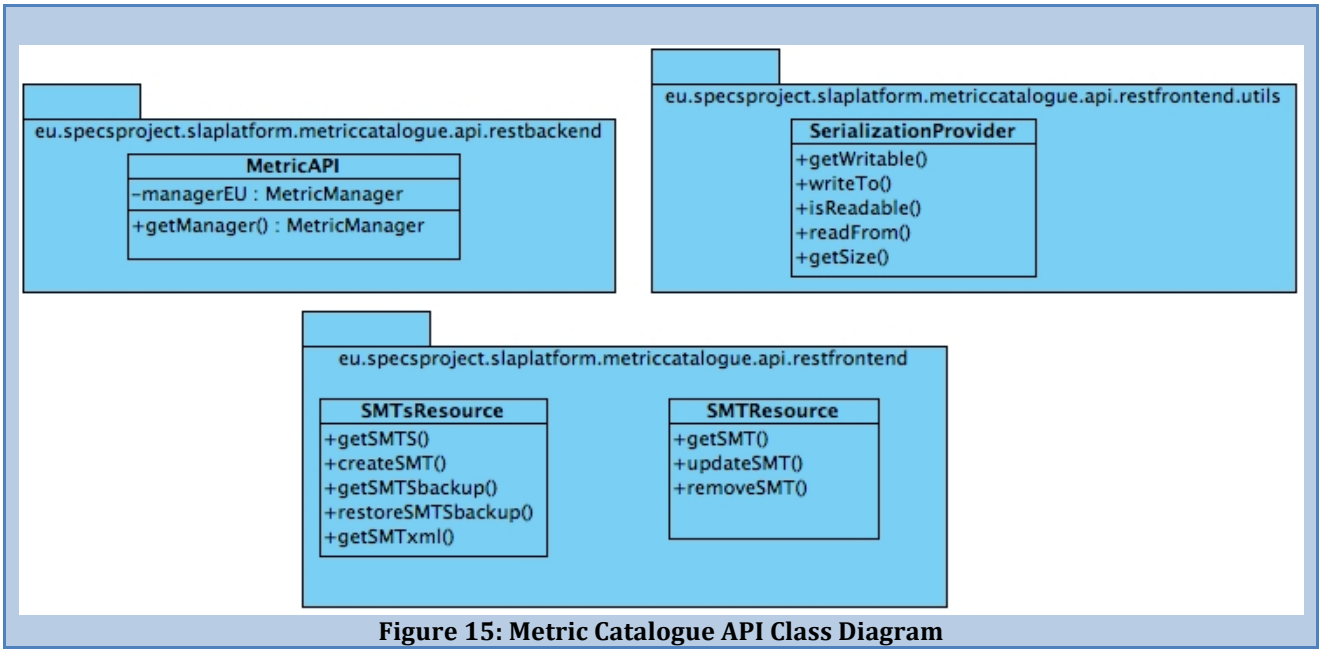

Figure 15: Metric Catalogue API Class Diagram

The Metric Catalogue API Class Diagram is composed by three packages, which are described below:

Table 72: package metriccatalogue.api.restbackend

Package name: eu.specsproject.slaplatform.metriccatalogue.api.restbackend	
Description: this package contains the class that allows to access to the MetricManager in order to use all methods exposed by the back end	
Class Name	**Description and Goal**
MetricAPI	It is a class that contains a method to get an instance of the *EUMetricManagerAbstractImpl* class defined in the back-end to use all the functions declared in the *MetricManager* interface

Table 73: package metriccatalogue.api.restfrontend

Package name: eu.specsproject.slaplatform.metriccatalogue.api.restfrontend	
Description: this package contains all the classes that represents the REST API resources. Each class implements all the API useful to access the specific resource that its represents	
Class Name	**Description and Goal**
SMTsResources	It implements the collection of stored Security Metrics and offers methods to manage the collection.
SMTResouce	It implements the resource associated to a specific Security Metric and offers methods to manage it.

Table 74: package slamanager.api.restfrontend.utils

Package name: eu.specsproject.slaplatform.metriccatlogue.api.restfrontend.utils	
Description: this package contains the class useful to marshal and unmarshal the entities	
Class Name	**Description and Goal**
SerializationProvider	This class implements two interfaces:

	MessageBodyReader<T> useful to handle inbound entity representation-to-Java deserialization, and *MessageBodyWriter<T>* useful to handle the outbound entity Java-to-representation serialization.

3.3.4. List of requirements covered and discussion

The following table lists all requirements for the Security Metrics Catalogue component with a short description of the coverage.

Table 75: Security Metrics Catalogue requirements list

Requirements	Description	Coverage description
SLAPL_R5	**Get service from repository**	Through the Service Manager and/or the Security Metrics Catalogue it is possible to retrieve a service using the REST API call (method: GET, URI: /cloud-sla/service-type/{service-id})
SLAPL_R6	**Add service to repository**	Through the Service Manager and/or the Security Metrics Catalogue it is possible to store a new service using the REST API call (method: POST, URI: /cloud-sla/service-type/) where the body contains the new service
SLAPL_R7	**Delete service from repository**	Through the Service Manager and/or the Security Metrics Catalogue it is possible to remove a service using the REST API call (method: PUT, DELETE: /cloud-sla/service-type/{service-id})
SLAPL_R8	**Update service in repository**	Through the Service Manager and/or the Security Metrics Catalogue it is possible to update the value of a service using the REST API call (method: PUT, URI: /cloud-sla/service-type/{service-id}) where the body contains the new value of the service
SLAPL_R9	**Search service in repository**	Through the Service Manager and/or the Security Metrics Catalogue it is possible to search a service that has certain features using the REST API call (method: GET, URI: /cloud-sla/service-type/{service-id}) where the query params define the requested features

3.4. Interoperability layer

3.4.1. Interoperability layer Use Cases

The Interoperability layer offers functionalities for enabling a transparent communication among different modules. In practice, it acts as a gateway by intercepting all REST calls and by redirecting them to the right component. This is accomplished by defining a proper virtual interface that associates a set of REST calls to a specific endpoint (i.e., a concrete URL address).

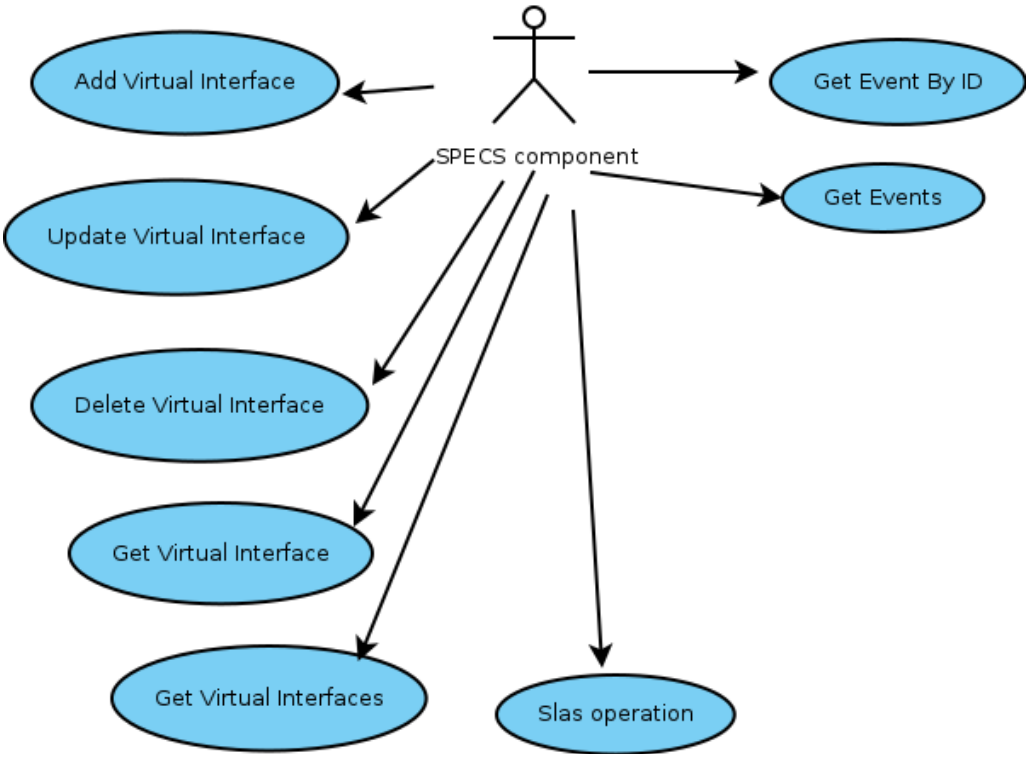

Figure 16: Interoperability Layer Use Case Diagram

3.4.1.1. Description

Table 76: Add virtual Interface

Use Case Name	Add Virtual Interface
Actors	SPECS component
Entry (pre-conditions)	
Exit (post-conditions)	A specific virtual interface is added
Input	Virtual interface
Output	The Unique Identifier of the virtual interface
Flows of events	The SPECS component inserts the virtual interface data The virtual interface manager inserts the virtual interface The SPECS component returns the virtual interface ID
Alternative Flows	
Extensions	
Inclusions	

Table 77: Update virtual Interface

Use Case Name	Update Virtual Interface
Actors	SPECS component
Entry (pre-conditions)	Existence of a virtual interface with specified id
Exit (post-conditions)	A specific virtual interface is updated
Input	The new Virtual interface value
Output	The Unique Identifier of the virtual interface
Flows of events	1. The SPECS component inserts the new virtual interface data

	2. The virtual interface manager searches the virtual interface with specified id 3. The virtual interface manager updates the virtual interface 4. The SPECS component returns the virtual interface ID
Alternative Flows	3a If the Virtual Interface Manger does not find the virtual interface, the SPECS component returns a NOT FOUND
Extensions	
Inclusions	

Table 78: Get virtual Interface

Use Case Name	**Get Virtual Interface**
Actors	SPECS component
Entry (pre-conditions)	Existence of a virtual Interface
Exit (post-conditions)	Returns specific virtual interface
Input	Virtual interface id
Output	Empty virtual interface
Flows of events	1. The SPECS component inserts the virtual interface id 2. The virtual interface manager gets the virtual interface 3. The SPECS component returns the virtual interface
Alternative Flows	
Extensions	
Inclusions	

Table 79: Get virtual Interfaces

Use Case Name	**Get Virtual Interfaces**
Actors	SPECS component
Entry (pre-conditions)	
Exit (post-conditions)	Return collections of virtual interface id
Input	
Output	
Flows of events	1. The SPECS component gets the virtual interfaces 2. The virtual interface manager gets the virtual interfaces id 3. The SPECS component returns a collection of virtual interface id
Alternative Flows	
Extensions	
Inclusions	

Table 80: : Get Events

Use Case Name	**Get Events**
Actors	SPECS component
Entry (pre-conditions)	Existence of event
Exit (post-conditions)	
Input	
Output	

Flows of events	1. The SPECS component gets the interoperability events 4. The event manager gets the virtual interfaces events id 5. The SPECS component returns a collection of virtual interface events id
Alternative Flows	
Extensions	
Inclusions	

Table 81: Get Event by id

Use Case Name	Get Event By Id
Actors	SPECS component
Entry (pre-conditions)	Existence of event
Exit (post-conditions)	
Input	Event id
Output	
Flows of events	1. The SPECS component gets the interoperability events 2. The event manager gets the virtual interfaces event with specified id 3. The SPECS component returns the event
Alternative Flows	
Extensions	
Inclusions	

Table 82: SLA operation

Use Case Name	Get Event By Id
Actors	SPECS component
Entry (pre-conditions)	Existence of Other Components
Exit (post-conditions)	See the Other Components API
Input	See the Other Components API
Output	See the Other Components API
Flows of events	1. The SPECS component gets the SLA operation 2. The ProxyController calls the “real” url 3. The SPECS component return the “real” response
Alternative Flows	
Extensions	
Inclusions	

The detailed explanation of the REST API offered by the Interoperability layer component can be found in Deliverable D1.3.

3.4.2. Interoperability Layer Classes

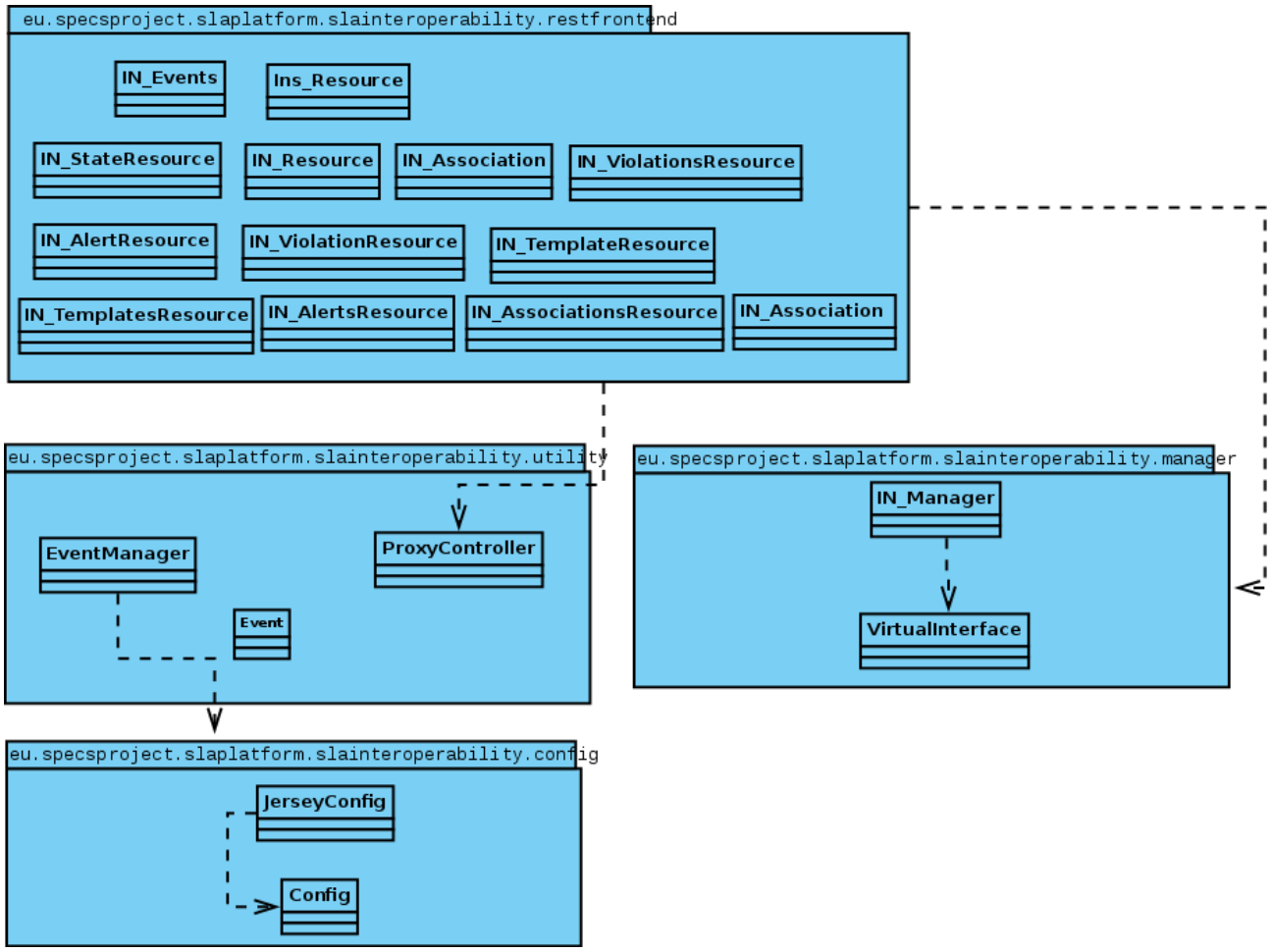

Figure 17: Interoperability layer Class Diagram

The Interoperability layer is described through the previous class diagram that represents all the packages and the classes making up this layer.

The packages are described below:

Table 83: eu.specsproject.slaplatform.slainteroperability.restfrontend

Package name: eu.specsproject.slaplatform.slainteroperability.restfrontend	
Description: this package contains the class that allows to access to the Interoperability layer	
Class Name	**Description and Goal**
INs_Resources	This class is the entry point for this layer
IN_Resource	It explode the sub-path of the Ins_Resource path
IN_Events	This class contains the event REST API
IN_AlertsResource	It contains the Alerts REST interface
IN_AlertResource	It contains the Alert REST interface
IN_Association	It contains the Association REST interface
IN_AssociationResources	It contains the Associations REST interface
IN_Violation	It contains the REST violation interface
IN_ViolationResources	It contains the REST violations interface
IN_Template	It contains the REST template interface

IN_TemplateResources	it contains the REST templates interface
IN_StateResources	it contains the REST state interface

Table 84: eu.specsproject.slaplatform.slainteroperability.utility

Package name: eu.specsproject.slaplatform.slainteroperability.utility	
Description: this package contains the classes that implement the manager logic	
Class Name	**Description and Goal**
ProxyController	This class implements a Singleton pattern for the proxy logic
Event_Manager	This class implements a Singleton pattern for the proxy logic
Event	This class contains the event definition

Table 85: eu.specsproject.slaplatform.slainteroperability.config

Package name: eu.specsproject.slaplatform.slainteroperability.config	
Description: this package contains the class that allows to configure the interoperability layer	
Class Name	**Description and Goal**
Config	This class is the configuration class
JerseyConfig	This class loads the configuration

Table 86: eu.specsproject.slaplatform.slainteroperability.manager

Package name: eu.specsproject.slaplatform.slainteroperability.manager	
Description: this package contains the class that allows to configure the ineroperability layer	
Class Name	**Description and Goal**
IN_Manager	This class contains the REST API to manage the virtual interface
Virtual Interface	The virtual interface class

3.4.3. List of requirements covered and discussion

The following table lists all requirements for the Interoperability layer component and a short description of the coverage is reported.

Table 87: Interoprability layer requirements list

Requirements	Description	Coverage description
Interoperability SLAPL_R45	**Enable transparent communication among modules**	Each module should be able to run independently and possibly use transparently the API offered by other modules. The SLA Platform should be able to decouple

		communication when needed.

4. Vertical Layer

4.1. Auditing

The Auditing component is devoted to logging/auditing different kinds of events. It offers these services to all components and all modules of the SPECS framework.

4.1.1. Auditing Use Cases

This section presents interactions between a generic SPECS component and the Auditing component. Figure 18 illustrates the use case diagrams related to this component.

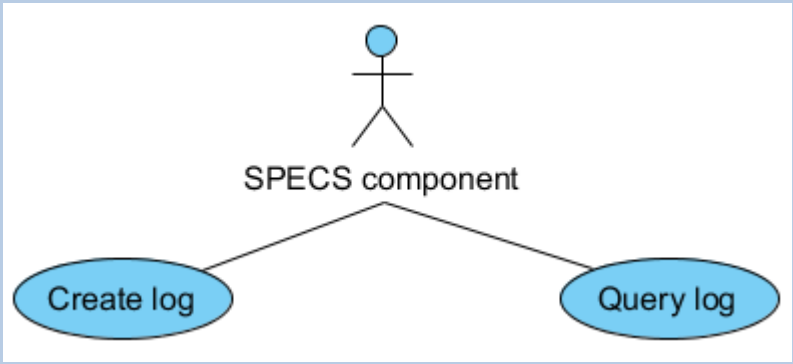

Figure 18: Security Metrics Catalogue Use Case Diagram

4.1.2. Auditing Components

The Auditing component comprises the following subcomponents: the first one represents the back-end that is useful to handle the persistence of all data, while the second one represents the front-end that exposes the REST API.

With respect to the design and the role of the component reported in D4.2.2, only the format of the logged events slightly changed. Summary of changes is provided in the following table.

Year 1	Year 2
• Activations/deactivations of components. • Activations/deactivations of security services. • SLA impact report. • Alert/violation root cause report. • Priority queue report.	• Activations/deactivations of components. • Activations/deactivations of security services. • Diagnosed monitoring events. • Remediation results. • Notifications sent to the EU.
Comments	
SLA impact report, alert/violation root cause report, and priority queue report are included in the Diagnosed monitoring events log. After detailing remediation process in task T4.3, remediation results and notifications sent to the EU are additionally logged.	

Table 88. Changes of audit events with respect to year 1

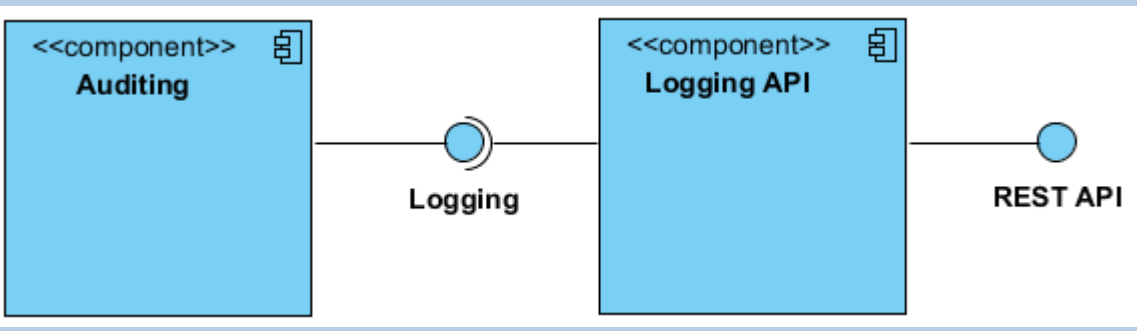

Figure 19: Auditing Component Diagram

Figure 19 presents the software organization of the component, which is composed by two components: (i) **Auditing API,** which offers the REST Logging API described in Deliverable D1.3, implementing the RESTful principles and (ii) **Auditing component**, which acts as a backend, implementing the *Logging interface* used by the Logging API component.

Table 89 summarizes the Logging interface used for the communication among the two components.

Table 89: Logging Interface

Method Name	Input	Description	Output
createCompActivities	The JSON representation of the CompActivities audit event to store	Creates and stores a new CompActivities audit event into the Auditing	The URI of the created and stored audit event
getCompActivities		Returns the URI of all CompActivities audit events present in the Auditing component	The collection of CompActivities
getCompActivity	The Unique Identifier of the CompActivity to be returned	Returns the JSON representation of the CompActivity identified by the UI	The JSON representation of the requested CompActivity
createServActivities	The JSON representation of the ServActivities audit event to store	Creates and stores a new ServActivities audit event into the Auditing	The URI of the created and stored audit event
getServActivities		Returns the URI of all ServActivities audit events present in the Auditing component	The collection of ServActivities
getServActivity	The Unique Identifier of the ServActivity to be returned	Returns the JSON representation of the ServActivity identified by the UI	The JSON representation of the requested ServActivity
createDiagMonEvents	The JSON representation of the DiagMonEvents audit event to store	Creates and stores a new DiagMonEvents audit event into the Auditing	The URI of the created and stored audit event

getDiagMonEvents		Returns the URI of all DiagMonEvents audit events present in the Auditing component	The collection of DiagMonEvents
getDiagMonEvent	The Unique Identifier of the DiagMonEvent to be returned	Returns the JSON representation of the DiagMonEvent identified by the UI	The JSON representation of the requested DiagMonEvents
createRemResults	The JSON representation of the RemResults audit event to store	Creates and stores a new RemResults audit event into the Auditing	The URI of the created and stored audit event
getRemResults		Returns the URI of all RemResults audit events present in the Auditing component	The collection of RemResults
getRemResult	The Unique Identifier of the RemResult to be returned	Returns the JSON representation of the RemResult identified by the UI	The JSON representation of the requested RemResult
createEuNotifications	The JSON representation of the EuNotifications audit event to store	Creates and stores a new EuNotifications audit event into the Auditing	The URI of the created and stored audit event
getEuNotifications		Returns the URI of all EuNotifications audit events present in the Auditing component	The collection of EuNotifications
getEuNotification	The Unique Identifier of the EuNotification to be returned	Returns the JSON representation of the EuNotification identified by the UI	The JSON representation of the requested EuNotification

The detailed explanation of the REST API offered by the Logging API component can be found in D1.3.

4.1.3. List of requirements covered and discussion

The following table lists all (direct or indirect) requirements for the Auditing component. The current prototype of the Auditing component implements 11 out of 14 (~ 77%) associated requirements. In the last year of the project we might work on improving functionalities related to requirements *ENF_AUD_R3*, *ENF_AUD_R4*, and *ENF_AUD_R5*. Any changes that might occur will be reported in D4.3.3 at M30.

Table 90: Auditing component requirements list

REQ ID	Description	Coverage description
ENF_PLAN_R6	**Log component activation and deactivation**	The Planning component is able to report about its activation or deactivation for accountability purposes.

ENF_IMPL_R6	**Log service activation**	The Implementation component is able to log a successful activation of each security service related to a certain SLO in an SLA.
ENF_IMPL_R8	**Log component activation or deactivation**	The Implementation component is able to report about its activation or deactivation for accountability purposes.
ENF_DIAG_R9	**Log component activation or deactivation**	The Diagnosis component must is to log its activation or deactivation for accountability purposes.
ENF_DIAG_R11	**Log SLA impact**	When all SLOs affected by a monitoring event are identified, and the severity of the impact of the monitoring event has been determined, the Diagnosis component is able to log this information.
ENF_DIAG_R12	**Classify event**	The Diagnosis component is able to classify a monitoring event with regard to each affected SLA, based on the information provided by the Monitoring component and the affected SLOs and SLAs.
ENF_DIAG_R14	**Log root cause**	The Diagnosis component is able to log the information about the root cause of a monitoring event.
ENF_DIAG_R17	**Log priority queue**	The Diagnosis component is able to log the information about the priority queue.
ENF_REM_R2	**Log component activation or deactivation**	The Remediation Decision System component is able to log its activation or deactivation.
ENF_AUD_R1	**Create log**	The Auditing component is able to create different types of logs (e.g., activation/deactivation of a component, service activation, priority queue, etc.).
ENF_AUD_R2	**Query log**	The Auditing component enables all components of the SPECS framework to query for different types of logs (retrieving logs from the database using different search criteria).
ENF_AUD_R3	**Support different communication technologies**	The Auditing component has to support different (software) communication technologies (REST, Thrift, etc.).
ENF_AUD_R4	**Support log correlation**	The Auditing component has to support correlation among different logs, i.e., to consolidate logs created due to the same request or event into a workflow.
ENF_AUD_R5	**Support different databases**	The Auditing component has to support the use of different databases.

All implementation details for the Auditing components are available in D1.4.2 and on BitBucket [1].

4.2. User Manager

The User Manager is a component devoted to provide authentication and authorization mechanisms to control access to SPECS services through a SPECS application. The component is able to enforce a role-based authorization mechanism based on the standard XACML framework [2].

4.2.1. User Manager Use Cases

The interactions between a generic SPECS component and the User Manager component will be described in the following subsections. Figure 20: User Manager Use Case DiagramFigure 20 illustrates the use case diagrams related to this component.

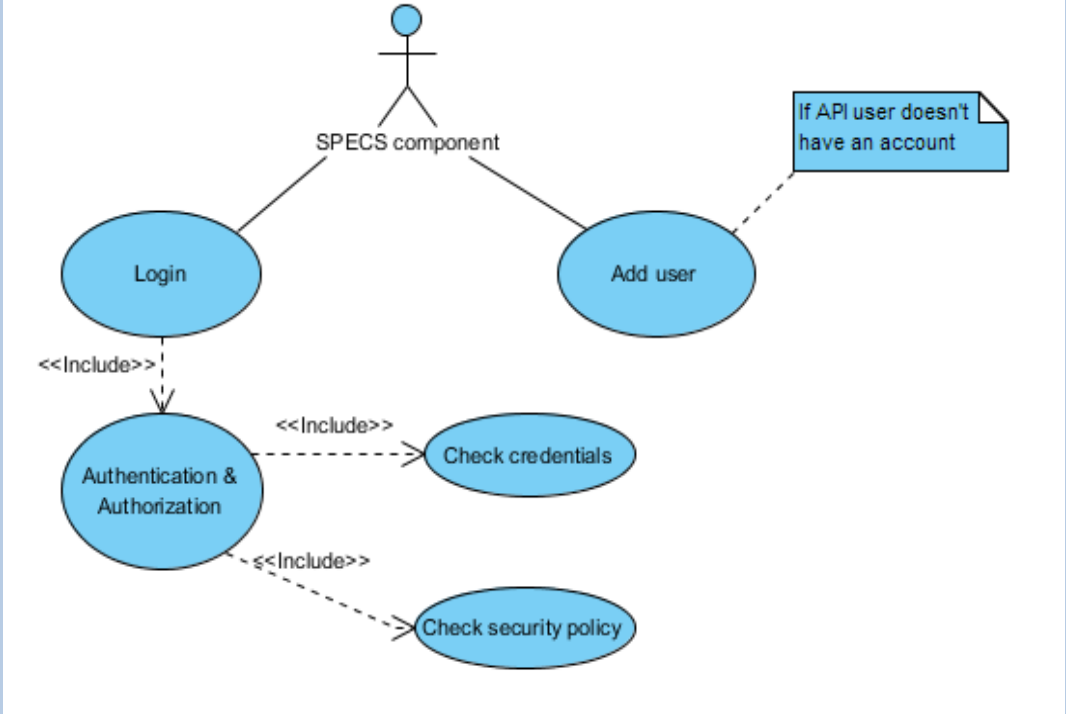

Figure 20: User Manager Use Case Diagram

4.2.1.1. Description

The following tables describe in detail the use cases reported in the previous figure:

Table 91: Login

Use Case Name	Login
Actors	SPECS component
Entry (pre-conditions)	SPECS actor that uses the component must have a registered account and provide its role
Exit (post-conditions)	
Input	SPECS credentials (username, password, role)
Output	
Flows of events	1. SPECS component inserts username, password and role, and executes the login 2. Authentication phase: the credentials are checked to verify that the account exists in the user repository and that password and role are correct. 3. Authorization phase: it receives the request and, reading policy files, checks if the role has a permission to access to the resource. 4. Response = PERMIT
Alternative Flows	2a Login Error: the account entered does not exist or the password and/or role are wrong; an error message is returned. 4a DENY/INDETERMINATE/NOT APPLICABLE: the account has not a permission to access, or the policy is not applicable; an error message is returned.

Extensions	
Inclusions	Authentication & Authorization, which in turn include Check credentials and Check security policy.

Table 92: Add user

Use Case Name	Add user
Actors	SPECS component
Entry (pre-conditions)	SPECS actor that uses the component does not have a SPECS account
Exit (post-conditions)	
Input	Username, Password, Role
Output	New SPECS Account
Flows of events	1. SPECS component inserts into form username, password and role, and clicks on "Signup" button to submit it 2. The application checks if username already exists in the user repository. 3. The application creates a new SPECS account in user repository. 4. A success message is generated.
Alternative Flows	2a The username already exists; an error message is returned.
Extensions	
Inclusions	

4.2.2. User Manager Components

The User Manager is made up of two main packages to provide Authentication and Authorization functionalities, respectively. The User Manager offers just one API, the User API. On the backend, we adopted an LDAP server to store all user credentials and a policy repository that acts as Policy Administration Point, where the security administrator can upload a new access control policy based on XACML.

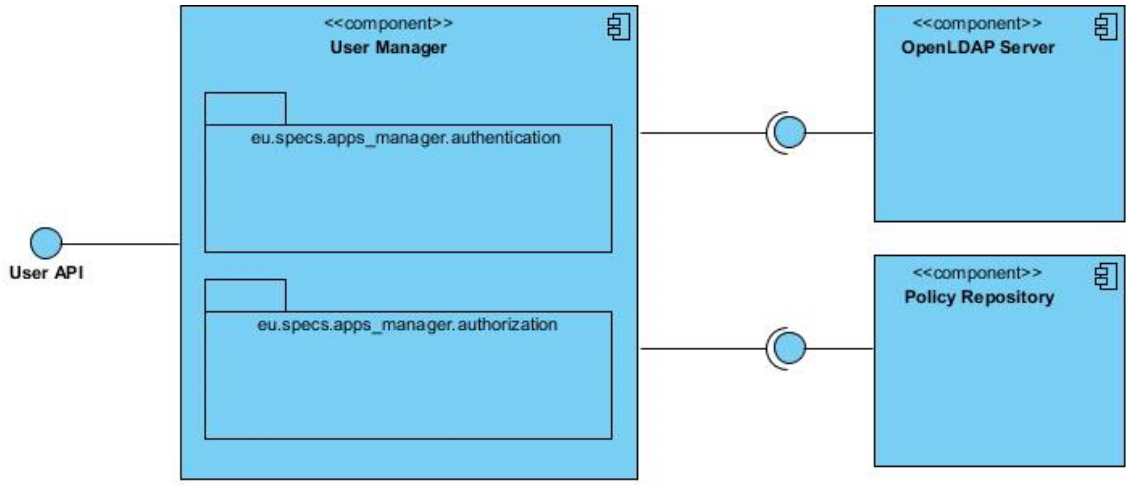

Figure 21: User Manager Component Diagram

4.2.3. User Manager Classes

The User Manager component is described through two class diagrams. The first diagram represents all the classes that compose the Authentication package, while the second diagram represents the classes of the Authorization package.

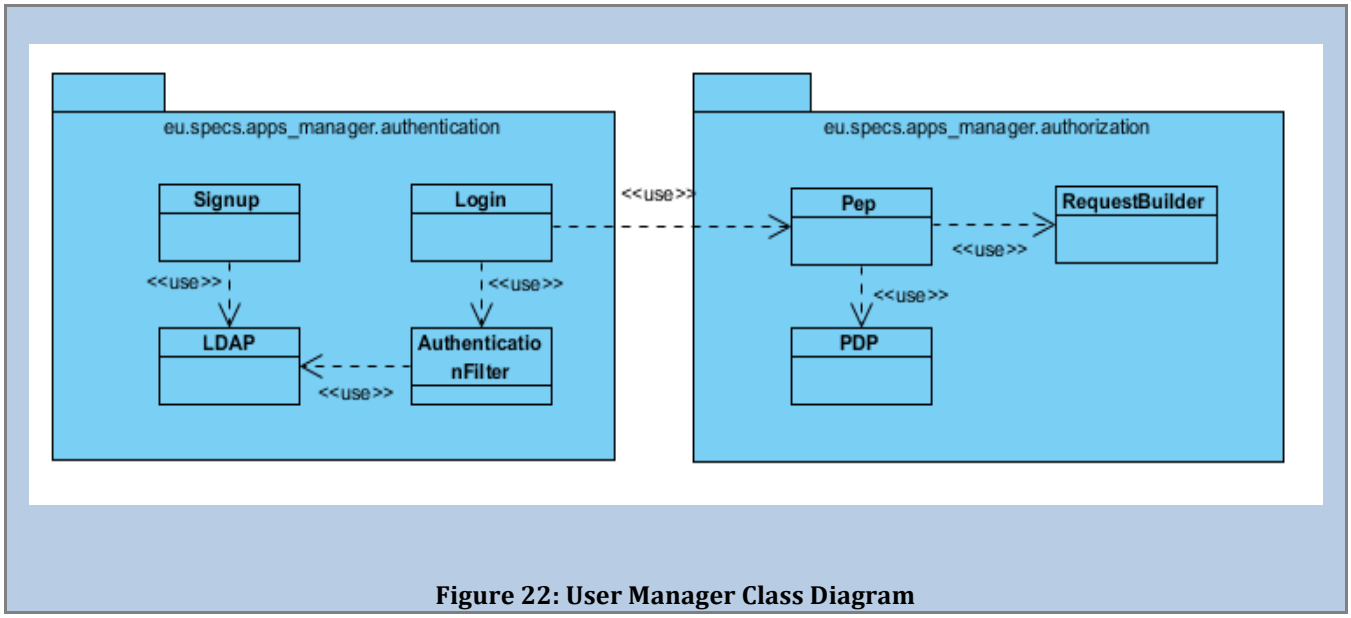

Figure 22: User Manager Class Diagram

The two packages are described in the following tables:

Table 93: User Manager package Authentication

Package name: eu.specs.apps_manager.authentication	
Description: this package contains the classes used to execute authentication and signup mechanisms	
Class Name	**Description and Goal**
LDAP	Utility class used to connect OpenLDAP server and, accessing it with Manager credentials, implements procedures to validate login and account creation.
Login	Class that implements a Servlet for the login process, activated by the "submit" button of the login form. It uses two filters for authentication and authorization.
AuthenticationFilter	Class that implements the first Servlet Filter of Login, activated by the "submit" button of the login form. It implements authentication mechanism, through the use of the validate login function of LDAP class.
Signup	Class that implements a Servlet for the signup process, activated by the "submit" button of the signup form. It uses the account creation function of LDAP class to implement this procedure.

Table 94: User Manager package Authorization

Package name: eu.specs.apps_manager.authorization	
Description: this package contains the classes used to execute authorization mechanism	
Class Name	**Description and Goal**
PEP	Class that implements the second Servlet Filter of Login, activated by the "submit" button of the login form. It implements Policy Enforcement Point of XACML authorization mechanism, using Sun's XACML library and an utility class to build XACML request.
RequestBuilder	Utility class used by PEP class to build a XACML request.
PDP	Class that implements Policy Decision Point of XACML mechanism, used to evaluate the request, take a decision and return a XACML response containing it. Class imported by Sun's XACML [Rif:http://www.java2s.com/Code/Jar/s/Downloadsunxacmljar.htm]

4.2.4. List of requirements covered and discussion

The following table lists all requirements for the User Manager component. The current prototype of the User Manager component implements 5 out of 9 (~ 55%) associated requirements. In the last year of the project we might work on improving functionalities to improve the set of functionalities dedicated to the management of the User Manager, as manage users and manage policies related to requirements *SLAPL_R38*, *SLAPL_R39*, *SLAPL_R43* and *SLAPL_R44*. Any improvement to the current implementation will be reported in D4.3.3 at M30.

Table 95: User Manager component requirements list

REQ ID	***Description***	***Coverage description***
SLAPL_R18	**Create user repository**	User Repository is managed by User manager Component (Vertical Layer). It is created by the associated recipes.
SLAPL_R37	**Add user**	We implemented the Sign up method in the Authentication package
SLAPL_R38	**Delete user**	We implemented the Delete user method but it is not yet offered
SLAPL_R39	**Manage security policy**	Not yet implemented, manually add and modify policies
SLAPL_R40	**Authenticate and authorize user**	We used openLDAP as a User Directory with secret credentials, and the XACML framework to authorize users according to the ABAC/RBAC model
SLAPL_R41	**Check credentials**	We implemented the Validate Login method in the Authentication package

SLAPL_R42	**Check authorization policy**	We used the evaluate method provided by the PDP implementation of SUN XACML library, in the Authorization package
SLAPL_R43	**Get user info**	Not yet implemented
SLAPL_R44	***Audit user access***	Not yet implemented

5. The SPECS Domain Model

A high-level view of the SPECS domain model is provided in the UML diagram of Figure 23. The depicted diagram, based on the WS-Agreement representation of SLAs, extends such model to include all the concepts needed to:

- ***declare*** the functional and non-functional (i.e., security-related) features provided by a service in an SLA (useful for negotiation),
- ***quantify*** the level of security provided with a service (useful for evaluation and monitoring),
- ***automatically provision*** a service based on the functional and non-functional requirements stated in an SLA (useful for enforcement and remediation).

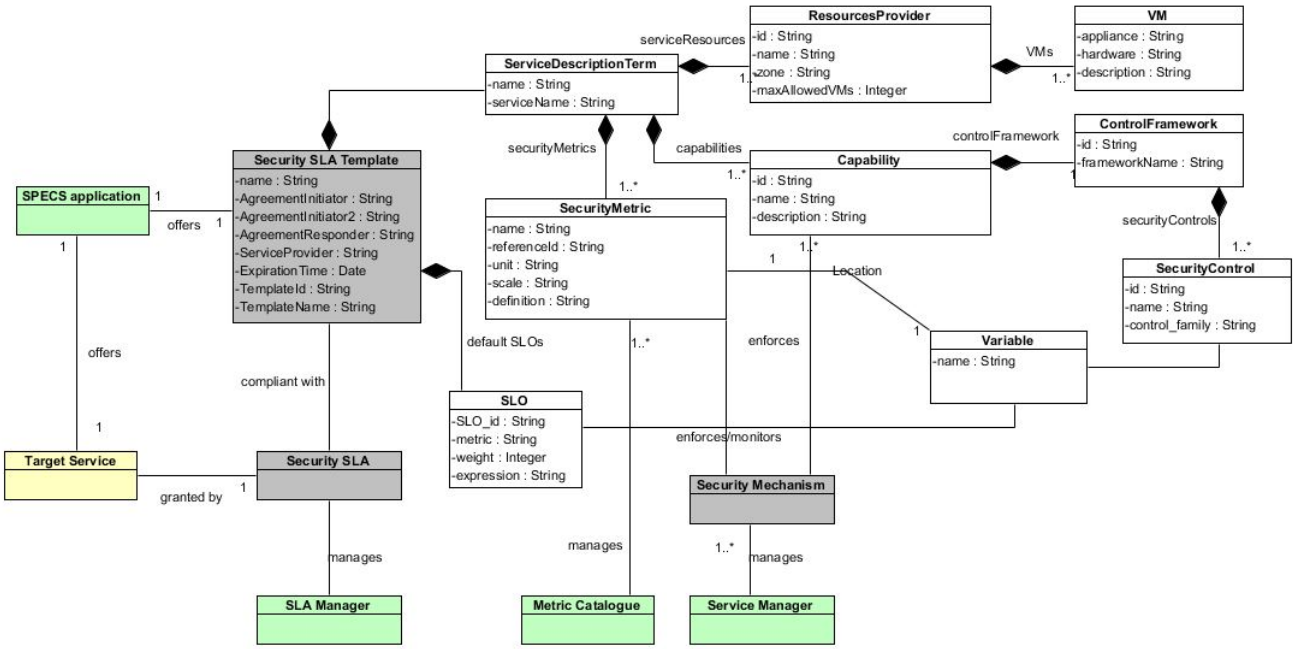

Figure 23: SPECS domain model

The *declaration* of functional and security-related terms is enabled by the following concepts:

- the *service description term* identifies the type of service to acquire (e.g. a storage service);
- the *resources provider* and the *virtual machines* (VM) concepts identify the specific cloud resources to acquire from external CSPs and to enhance with security features;
- the *capabilities* identify the security features to enforce on the acquired cloud service. They are expressed in terms of *sets of security controls,* in compliance with NIST definition [3] and are enforced through proper software mechanisms, implemented by software components offered by the SPECS framework. Our model supports different security controls definitions, included in different *security control frameworks*, also represented in the diagram.

The declaration of security-related terms in an SLA enables the *automatic evaluation* of the security level associated with a service and the comparison of different offers. This task, performed during negotiation to rank different available offers, is accomplished in SPECS by a

Security Reasoner. The reader is referred to deliverable D2.2.2 for a complete discussion of the requirements of the Security Reasoner and of the techniques used for security evaluation.

The *quantification* of the level of security associated with a service is useful not only for the evaluation task, but also for specifying the guarantees associated with a certain offer and for monitoring that such guarantees are met. With respect to this, the framework models the following concepts:

- the *security metrics* are used to specify *measurable* aspects related to the declared security capabilities, in order to enable their monitoring (e.g., "availability" is a measurable attribute that can be used to monitor the correct provisioning of a "resilience to attacks" capability). Moreover, in SPECS, security metrics also include *configurable* parameters associated with the enforcement of some of the declared capabilities (e.g., "scanning frequency" is a configuration parameter related to a "vulnerability detection" capability);
- the *SLOs* are logic conditions defined over security metrics (e.g., "availability >99.9"). SLOs are negotiated by End-users and state the desired level of security; they are used to configure the services being delivered and are monitored during system operation.

It should be noted that the concepts defined above belong to the *SPECS conceptual model*, described in deliverable D2.2.2 and targeted to the declaration and evaluation of security, on which WP2 is focused. Indeed, deliverable D2.2.2 also presents the machine-readable format for the representation of a Security SLA based on such conceptual model. The reader is referred to it for an extensive discussion of the topic.

The SPECS domain model, as mentioned, includes other additional concepts related to the management and implementation of the SLA itself. As shown, a Security SLA is related to a specific *target service,* for which it states the related security guarantees. The target service is delivered to End-users through a *SPECS application,* relying upon a specific *SLA Template* to which all offered SLAs are compliant. Templates summarize all the features that can be offered to End-users through the application. They are stored in the Negotiation module and are used as a guideline during negotiation in compliance with the WS-Agreement standard: when the End-user negotiates security features with a SPECS application, he/she basically selects a subset of the available features and obtains corresponding SLA Offers built according to what can be actually delivered. Hence, an SLA Offer can be seen as an instance of the set of possible SLAs represented by a template.
The Security SLA life cycle is managed by the SPECS Platform's *SLA Manager*, which offers proper API to store, retrieve and update Security SLAs and their state.
The enforcement of security capabilities and the monitoring of related security metrics (as specified in the SLOs) are performed by software tools called *Security Mechanisms*: they are selected, deployed and configured during the Implementation phase. All the information needed to automate the deployment and execution of security mechanisms is maintained by the *Service Manager*, which also provides the basic functionalities to broker cloud services from external providers.
Finally, the *Security Metrics Catalogue* maintains, in a machine-readable format, the information used for the definition of the security metrics. In our model, metrics are defined according to the NIST RATAX framework [3] and Cloud Control Matrix [4].

6. The SPECS Data Model

The SPECS framework consists of a set of independent components, which need a common data model to interact with each other. The SPECS data model is a Java library that contains common Java classes shared among SPECS components. The interaction among SPECS components goes through REST API. In this interaction, the Java objects are serialized to JSON or XML and deserialized back to the same Java class on the other side.

The SPECS data model contains the following models:

- SLA API data model, which represents:
  - a description of the possible states that an SLA can go through during its life-cycle;
  - a description of the possible transitions among SLA states;
  - a description of the possible annotations that can be added to a resource.
- Service API data models, which consist of the following data models:
  - Security mechanisms/monitoring systems data model;
  - Security mechanisms/monitoring systems metadata data model: the metadata is needed for activation and configuration of security mechanisms and monitoring systems;
  - Security capabilities data model: security capabilities are fundamental to define the security features introduced by the activation of a given security mechanism;
  - Security metrics data model: security metrics are fundamental to monitor an SLA and are measured by specific monitoring systems and associated with specific security controls
- Interoperability API data model, which consists of the following data model:
  - Virtual Interface data model
- Negotiation API data models, which represent:
  - a description of the activity related to building a valid Service description term according to the provided identifier;
  - a description of the activity related to building a valid Service level objective according to the provided identifier;
  - a description of the activity related to building an SLA Template according to the provided identifier.
- Enforcement API data models, which consist of the following data models:
  - Supply Chain Activity: represents a description of the activity related to building all valid supply chains according to the provided input;
  - Supply Chain: represents the set of cloud resources and the set of security mechanisms' components needed to implement an SLA;
  - Notification: represents information about the occurrence of a monitoring event;
  - Diagnosis Activity: represents the information associated to a Notification;
  - Planning Activity: represents all information needed to build an implementation plan for an SLA;
  - Planning Activity Update data model: represents the information about a renegotiated SLA and the associated new supply chain;
  - Implementation Plan: represents the resources and their configurations needed to implement an SLA;

  - Implementation Activity: represents all information about an SLA implementation process;
  - Reconfiguration: represents a required reconfiguration of an existing Implementation Plan;
  - Remediation Plan: represents a set of actions required for an SLA remediation;
  - Remediation Activity: represents all information related to an SLA remediation process.
- Log API data models which:
  - contain all information related to activations and deactivations of SPECS components;
  - contain all information related to services deployed by SPECS;
  - contain all information related to detected and analysed monitoring events;
  - contain all information related an SLA remediation process.

Details of these data models, their JSON and XML schemas and their usage can be found in deliverable D1.3 and on the project's Bitbucket site https://bitbucket.org/specs-team/specs-utility-data-model

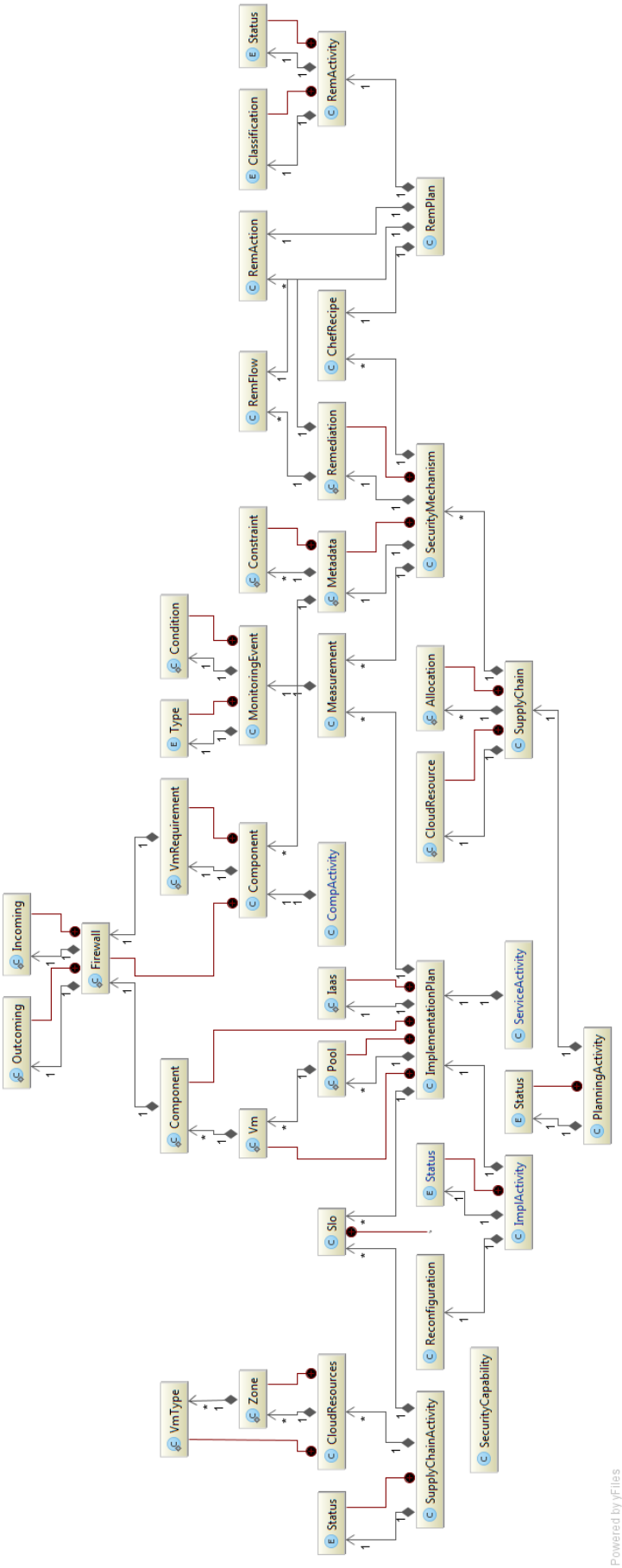

Figure 24: SPECS data model

7. Year 2 - SLA Platform Design Updates

In D1.1.2 we described a preliminary architecture of the SLA Platform as resulted by the requirements elicited during the first year. Thanks to the feedbacks from stakeholders during the implementation phase, it has been slightly modified. In particular, in the previous version we considered three additional components in the Vertical Layer: Notification Hub, Storage Manager and SLA Repository.
Thanks to the feedback received during the second year, the architecture was simplified; indeed, the elicited requirements are now covered by a smaller subset of components.

7.1. Updates on Requirements

In this following table we report the revised list of requirements associated to the SPECS components that will implement them.

Table 96: SPECS components requirements list

SLA Platform Module	***SPECS Software components***			
Requirements	***SLA Manager***	***Service Manager***	***Interoperability Layer (new)***	***Security Metrics Catalogue[2]***
SLAPL_R1		X		
SLAPL_R2	DEPRECATED			
SLAPL_R3	Covered by other module (Monitoring- Archiver)			
SLAPL_R4				
SLAPL_R5		X		X
SLAPL_R6		X		X
SLAPL_R7		X		X
SLAPL_R8		X		X
SLAPL_R9		X		X
SLAPL_R10	X			
SLAPL_R11	X			
SLAPL_R12	X			
SLAPL_R13	X			
SLAPL_R14	X			
SLAPL_R15	Covered by other modules (Monitoring- Archiver and CTP Exporter)			
SLAPL_R16	Covered by other modules (Monitoring- Archiver)			
SLAPL_R17	DEPRECATED			
SLAPL_R18	Covered by Vertical Layer (User manager)			
SLAPL_R19	X			
SLAPL_R20	X			
SLAPL_R21	X			
SLAPL_R22	X			
SLAPL_R23	X			
SLAPL_R24	X			
SLAPL_R25	X			
SLAPL_R26	X			
SLAPL_R27	X			
SLAPL_R28	X			

[2]

SLAPL_R29	X			
SLAPL_R30	X			
SLAPL_R31	X			
SLAPL_R32	X			
SLAPL_R33	X			
SLAPL_R34	X			
SLAPL_R35	Covered by other modules (Monitoring & Enforcement)			
SLAPL_R36	Covered by other modules (Monitoring & Enforcement)			
SLAPL_R45			X	
CERT_R1	X			
CERT_R2	X			
CERT_R3	X			

Description on the current implementation of these requirements will be reported in D1.4.2., according to the development plans.

7.2. Updates on Components

As anticipated, we removed from the design three components: Notification Hub, Storage Manager and SLA Repository, because their functionalities are no more needed, or are included in other components.

7.2.1. Notification Hub

The first design introduced the Notification Hub in order to manage asynchronous communication in the flow, due to the generation of alerts and violations of monitoring module.
After the implementation of the Monitoring core, we noticed that the EventHub, used to collect all the information from the monitoring adapters, was able to manage all the asynchronous communication managed inside the SPECS framework. Communication between monitoring module and Enforcement happens using the REST API described in D1.3, and can be implemented using common synchronous interactions.

As a consequence, the new design assumes that all asynchronous communication happens through the monitoring core, using the SPECS event format. The Notification Hub is no more used.

7.2.2. Storage Manager

The Storage Manager was introduced in order to obtain easily persistence backends to be used by the other components. In practice, the goal of the Storage Manager was to setup databases needed by to the core components.
The SPECS implementation founds on the adoption of the Chef deployment solution, which is used by both enforcement module and enabling platform to setup mechanisms and core components.
The Storage-as-a-Service case study introduced the need to provide a (secure) DBaaS mechanism in the Enforcement module. According to such needs. we simply shared the database setup recipes developed for enforcement in a set of utility recipe used by both mechanisms ad core components. In such a way, the Storage Manager is no more needed as an independent component in the SPECS architecture.

7.2.3. SLA Repository

The SLA Repository should be used to collect Cloud Service Provider public SLAs inside the SPECS platform, in order to enable their comparison. The only source of information for the (security) SLA repository is the CSA STAR program. There is no additional concrete information that can be collected at the state of the art. The CSA STAR repository is available publicly, but CSA maintain the intellectual property and data cannot be used without explicit consent of the CSA.

Consequently, even if CSA (partner of the project) has no objection to the use of the STAR repository in the project, it was useless to offer a solution to replicate the data, which cannot be used outside the project. The SLA Repository adopted is the STAR repository itself, and our evaluation techniques offer the tools to directly import data from the official repo for customer-oriented direct evaluation.

The SPECS framework adopts as single SLA Repository the STAR repository (used by the STAR Watch solution in the SPECS portfolio), and has no need for a dedicated component implementing a new repository.

8. Conclusions

The present deliverable is an evolution of the SLA Platform architecture presented in D1.1.2, and introduces a detailed description of SLA Platform use cases, component and class diagrams to describe the final version of the SLA Platform module. The activities performed in this second year, and above all the discussions with SPECS stakeholders and End-users enable to further detail both the requirements that were identified in D1.2 and the architectural design, where many components were aggregated and simplified with respect to the original idea. This deliverable also presents the final SPECS domain model that is based on the SLA conceptual model introduced in D2.2.2 and provides a conceptual view of the SPECS SLA-based approach.

Finally, this deliverable describes in details two of four available Vertical Layer services, these are cross-cutting mechanisms that offer security mechanisms intended to protect the SPECS platform and not target services; here we report an updated description of the Auditing and the User manager components, whose internal design was refined with respect to the previous version introduced in D4.2.2, too.

This document served as input for tasks related to the final implementation of the SLA Platform module; it is worth noticing that all the SLA Platform components are available and their implementation and usage description is reported in D1.4.2.

9. References

[1] SPECS, "*SPECS Enforcement Auditing*", 2015. [Online]. Available: https://bitbucket.org/specs-team/specs-enforcement-auditing.

[2] eXtensible Access Control Markup Language (XACML) Version 3.0 [Online]. Available: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.pdf

[3] Cloud Computing Service Metrics Description, NIST. Available at: http://www.nist.gov/itl/cloud/upload/RATAX-CloudServiceMetricsDescription-DRAFT-20141111.pdf

[4] Cloud Control Matrix v3.0.1, CSA. Available at: https://cloudsecurityalliance.org/download/cloud-controls-matrix-v3-0-1/

Annex A
SLA Platform additional API calls

In this Annex, we report some additional API calls that were added to the APIs offered by the SLA Platform (i.e., the SLA API and the Services API) to reflect the changes to the architecture.

1. SLA API updates

Resource URI	**/cloud-sla/slas/{sla-id}/lock**	
GET	*Description*	It retrieves the SLA identified by the sla-id parameter and locks it.
	Query string	Not supported
	Request body	Empty
	Response body	Media type: ✓ Application/xml http://www.specs-project.eu/resources/schemas/SLAtemplate/SLAtemplate.xsd ✓ (SLA XML description)
	Response Codes Semantics	✓ *404 Not Found:* the specified sla has not been found.
	Notes	

Resource URI	**/cloud-sla/slas/{sla-id}/unlock**	
POST	*Description*	It unlocks the SLA identified by the sla-id parameter.
	Query string	Not supported
	Request body	Empty
	Response body	Empty
	Response Codes Semantics	✓ *404 Not Found:* the specified sla has not been found.
	Notes	

Resource URI	**/cloud-sla/slas/{sla-id}/sign**	
POST	*Description*	It updates the SLA identified by the sla-id parameter with a new one and it puts its state in signed.
	Query string	Not supported
	Request body	Media type: ✓ Application/xml http://www.specs-project.eu/schemas/SLAtemplate/SLAtemplate.xsd ✓ (SLA XML description)
	Response body	Empty
	Response Codes Semantics	✓ *409 Conflict:* the call cannot be completed due to the current state of the SLA ✓ *404 Not Found:* the specified sla has not been found.
	Notes	This call is allowed only if the SLA identified by the resource URI is either in the *negotiating* state or in the *re-negotiating*

		state.

Resource URI	**/cloud-sla/slas/{sla-id}/observe**	
POST	*Description*	It updates the SLA identified by the sla-id parameter and it puts its state in observed.
	Query string	Not supported
	Request body	Empty
	Response body	Empty
	Response Codes Semantics	✓ *409 Conflict:* the call cannot be completed due to the current state of the SLA ✓ *404 Not Found:* the specified sla has not been found.
	Notes	This call is allowed only if the SLA identified by the resource URI is either in the *signed* state.

Resource URI	**/cloud-sla/slas/{sla-id}/complete**	
POST	*Description*	It updates the SLA identified by the sla-id parameter and it puts its state in completed.
	Query string	Not supported
	Request body	Empty
	Response body	Empty
	Response Codes Semantics	✓ *409 Conflict:* the call cannot be completed due to the current state of the SLA ✓ *404 Not Found:* the specified sla has not been found.
	Notes	This call is allowed only if the SLA identified by the resource URI is either in the *observed* state.

Resource URI	**/cloud-sla/slas/{sla-id}/terminate**	
POST	*Description*	It updates the SLA identified by the sla-id parameter and it puts its state in terminated.
	Query string	Not supported
	Request body	Empty
	Response body	Empty
	Response Codes Semantics	✓ *409 Conflict:* the call cannot be completed due to the current state of the SLA ✓ *404 Not Found:* the specified sla has not been found.
	Notes	This call is allowed only if the SLA identified by the resource URI is either in the *negotiating* or *observed* or *redressing* or *renegotiating* or *remediating* state.

Resource URI	**/cloud-sla/slas/{sla-id}/signalAlert**

POST	*Description*	It updates the SLA identified by the sla-id parameter and it puts its state in alerted.
	Query string	Not supported
	Request body	Empty
	Response body	Empty
	Response Codes Semantics	✓ *409 Conflict:* the call cannot be completed due to the current state of the SLA ✓ *404 Not Found:* the specified sla has not been found.
	Notes	This call is allowed only if the SLA identified by the resource URI is either in the *observed* state.

Resource URI	**/cloud-sla/slas/{sla-id}/reNegotiate**	
POST	*Description*	It updates the SLA identified by the sla-id parameter and it puts its state in violated.
	Query string	Not supported
	Request body	Empty
	Response body	Empty
	Response Codes Semantics	✓ *409 Conflict:* the call cannot be completed due to the current state of the SLA ✓ *404 Not Found:* the specified sla has not been found.
	Notes	This call is allowed only if the SLA identified by the resource URI is either in the *observed* state.

2. Services API updates

Resource URI	**/cloud-sla/security-metrics/backup/{dbname}.db**	
GET	*Description*	It returns a backup of the Metric Catalogue associated at the database identified by dbname parameter.
	Query string	Not supported
	Request body	Empty
	Response body	Media type: ✓ Application/octect-stream
	Response Codes Semantics	✓ *404 Not Found:* the specified database has not been found.
	Notes	

Resource URI	**/cloud-sla/security-metrics/{metric_id}.xml**	
GET	*Description*	It returns an xml file that contains the Security Metrc identified by metric_id parameter.
	Query string	Not supported
	Request body	Empty
	Response body	Media type: ✓ Application/octect-stream
	Response Codes Semantics	✓ *404 Not Found:* the specified Security Metric has not been found.
	Notes	

Resource URI	**/cloud-sla/security-metrics/restore/{dbname}.db**	
POST	*Description*	It restore a backup of the Metric Catalogue associated at the database identified by dbname parameter.
	Query string	Not supported
	Request body	Media type: ✓ Application/octect-stream
	Response body	Empty
	Response Codes Semantics	✓ *404 Not Found:* the specified database to restore has not been found.
	Notes	