Secure Provisioning of Cloud Services
based on SLA Management

SPECS Project - Deliverable 2.3.3

Reference Architecture for Cloud SLA Negotiation: Development and Tests – Final Report

Version no. 1.1
30 April 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.:	D2.3.3
Deliverable title:	Reference Architecture for Cloud SLA Negotiation: Development and Tests – Final Prototype
Deliverable nature:	Report
Dissemination level:	Public
Contractual delivery:	30 April 2016
Actual delivery date:	30 April 2016
Author(s):	Madalina Erascu (IeAT)
Contributors:	Alessandra De Benedictis (CeRICT), Jolanda Modic (XLAB), Damjan Murn (XLAB), Adrian Spataru (IeAT)
Reviewers:	Valentina Casola (CeRICT), Silvio La Porta (EMC)
Task contributing to the deliverable:	T2.3
Total number of pages:	31
Annexes	2

Executive summary

This deliverable is the last of three deliverables (D2.3.1, D2.3.2) presenting the final report on the Negotiation phase and architecture of the SPECS solution.

At the end of year two of the SPECS project the following results of WP2 were presented in the corresponding deliverables, most notably D2.2.2:

- A conceptual model to represent SLAs compliant with the latest outcomes of standards and working groups.
- A metric catalogue compliant with the conceptual model and enriched with the feedback received from the Platform (WP1), the Enforcement module (WP4), and the validation scenarios (WP5).
- An architecture of the module as well as the interaction with external modules.
- A negotiation process which was designed and successively refined based on the feedback received from the Enforcement module and from preliminary prototypes of the Negotiation module (D2.3.1).
- A renegotiation process which was designed and refined based on the feedback received from the Enforcement module which oversees the remediation processes that triggers the renegotiation.
- Aspects related to the security assessment methodologies, namely REM methodology to evaluate the SLA Model of SPECS and QHP methodology that considers uncertainty on qualitative End-Users (EUs) requirements by using quantification of fuzzy numbers.

Moreover, in D2.3.1 we presented a prototype implementation of the *SLO Manager* component of the Negotiation module as well as *Security Reasoner* standalone application.

In D2.3.2, which is released also at month 30, we presented the following information on the three components (*SLO Manager*, *Supply Chain Manager*, *Security Reasoner*) of the Negotiation module: (i) final architecture, (ii) the development activities, (iii) references on the installation procedure and the usage of the prototype components developed in this task, (iv) functional tests results, in accordance with the methodology proposed in D4.5.2, and (v) performance tests results, in accordance with the methodology proposed in D1.5.2.

In this document, we present:

- the final stage of the artifacts developed within the Negotiation module, namely the SLO Manager, Supply Chain Manager, Security Reasoner, SLA conceptual model, SLA machine-readable format, security metrics, catalogue;
- how these artifacts addressed the objective and subobjectives of WP2 of SPECS, and
- how the research activities carried out in the framework of this WP advanced the state of the art in Security SLAs.

Table of contents

Index of figures

Index of tables

1. Introduction

This document describes the final outcomes regarding the Negotiation module developed in SPECS. The content of this deliverable updates the information presented in D2.2.2 and D2.3.1 by adding updated information, changes in the design and implementation, and improvements to the techniques and methods introduced in the second year. It also complements the information from D2.3.2, which was developed simultaneously, by refining the main aspects of the negotiation and renegotiation process; in Section 3 an overview of the negotiation and renegotiation process will be summarized in order to better illustrate the final details on the architecture. A throughout explanations on how the requirements of D2.2.2 were successfully addressed is presented in Section 4.1. In particular, the behaviour, the architecture of all components developed in the Negotiation module are presented in Section 4.2 (*SLO Manager*), Section 4.3 (*Supply Chain Manager*) and Section 4.4 (*Security Reasoner*), while the final list of available API interface will be described in Annexes A and B. We conclude by presenting how the objectives of WP2 were fulfilled.

2. Relationship with other deliverables

This deliverable is based on the existing work on architecture, requirements, use cases, etc., developed in the deliverables released at the end of second year of the project, it constitutes an input for and receives feedback from the deliverables finalizing the architecture, core modules and their interplay, use cases.

The overview of the relationship of this deliverable with others from other WPs is presented in

Figure 1.

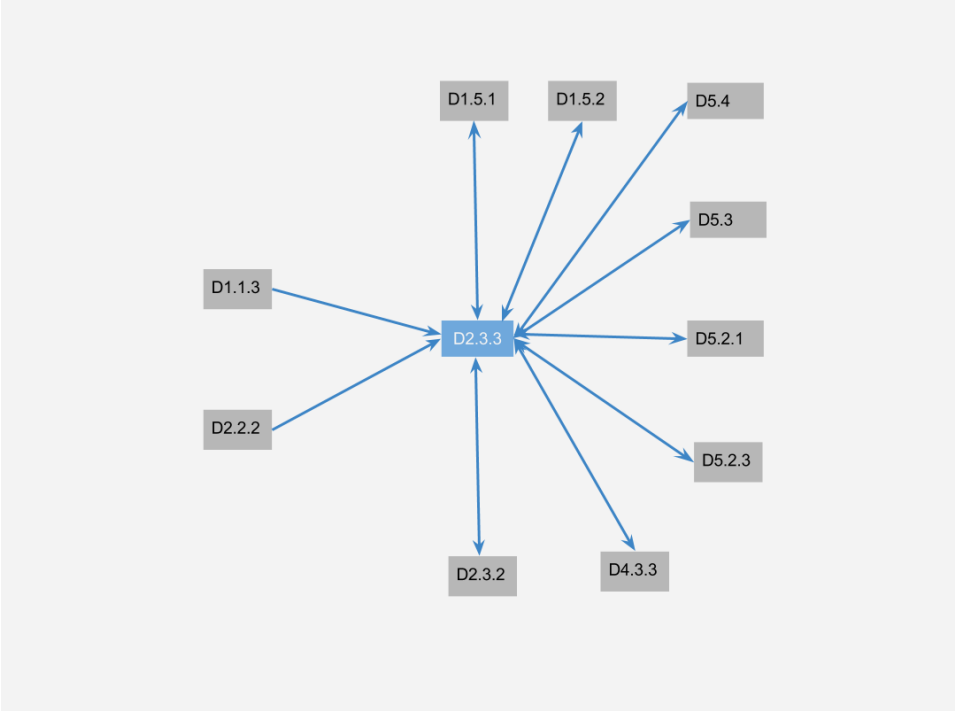

Figure 1. Relationship with other deliverables

In particular, the following deliverables are input for D2.3.3:

- D1.1.3: The final design of the SPECS solution provided input for the implementation of the Negotiation components.
- D2.2.2: The final report on the conceptual framework for Cloud SLA negotiation provided the basis for the negotiation and renegotiation processes as well as for the architecture and development of the Negotiation module.

The rest of the deliverables presented in the picture above are either an input for D2.3.3 or received input from D2.3.3:

- D1.5.1, D1.5.2: In the integration testing and examples, which are reported in these two deliverables, the Negotiation components are used; moreover, these two deliverables were taken into consideration when finalizing the architecture and implementation of the Negotiation module.
- D2.3.2: This deliverable is an input/output deliverable for D2.3.3 as use cases, design, implementation and testing activities were conducted with an iterative approach, in order to benefit of possible feedback.
- D4.3.3: The final prototypes of the Negotiation module takes in consideration the interaction with the SPECS Enforcement components for the building and validation of different supply chains.
- D5.2.1, D5.2.2, D5.3, D5.4: In the evaluation of the scenarios in these deliverables, the Negotiation components are utilized or provided feedback for the scenarios reported.

3. Negotiation and renegotiation processes overview

The SPECS negotiation process, as introduced in D2.2.2, comprises a set of steps that allows to define the security features demanded by EU for a selected service. SPECS also defines a renegotiation process to negotiate a new SLA in case of changes on EU's requirements, violation of a previously negotiated SLA or in case of changes on CSPs.

The negotiation process in SPECS is user centric and is based on the extraction of requirements for a set of services available in the platform. EU starts by selecting the type of service (i.e., secure storage). Each service is associated to a set of possible *capabilities* (understood as security mechanisms such as E2E encryption). Each capability is associated to a set of *controls* and *metrics*, according to the adopted Control Framework and the proposed Security SLA model. The process of negotiation has been designed in a way that EUs define the requested capabilities for a chosen service. Then EUs can choose the expected security level for the controls associated to the chosen capabilities. Expert EUs can also choose specific values for the SLOs associated to controls.

Once EUs requirements have been obtained the negotiation module searches among CSPs and SPECS services for those combinations that matches with requirements. The negotiation module trusts on the Enforcement module to build a set of supply chains. Each supply chain is the result of a combination of a CSP with one or more SPECS services. These supply chains are then used to build a set of SLA offers. To this end, the Negotiation module uses SLA templates which are used to define an SLA for each supply chain defined by the Enforcement module. During this process the Enforcement module will have checked that the supply chains created are valid (for example, checking that the combination of some CSP with one of the selected SPECS services is possible).

The resulting set of valid SLA offers are then passed to the reasoner component of the negotiation module. The SLAs offers are evaluated, compared with the EU requirements and ranked. As part of the negotiation process the valid SLA offers are also signed by CSPs, as a guarantee that they are able to provide the selected service. With this step the EU is provided with a ranked list of SLA offers. The EU can choose one of them to sign it. The signed SLA is then enforced and monitored.

The next sequence diagram shows the negotiation process further detailing it with respect to the one introduced in D2.2.2. It includes details of the interfaces invoked between the negotiation module and the rest of the modules and details on the mechanism to retrieve SLA templates to build SLA offers and manage EU requirements.

This diagram hides the internals of the process to generate supply chains. This process will be detailed in D4.3.3.

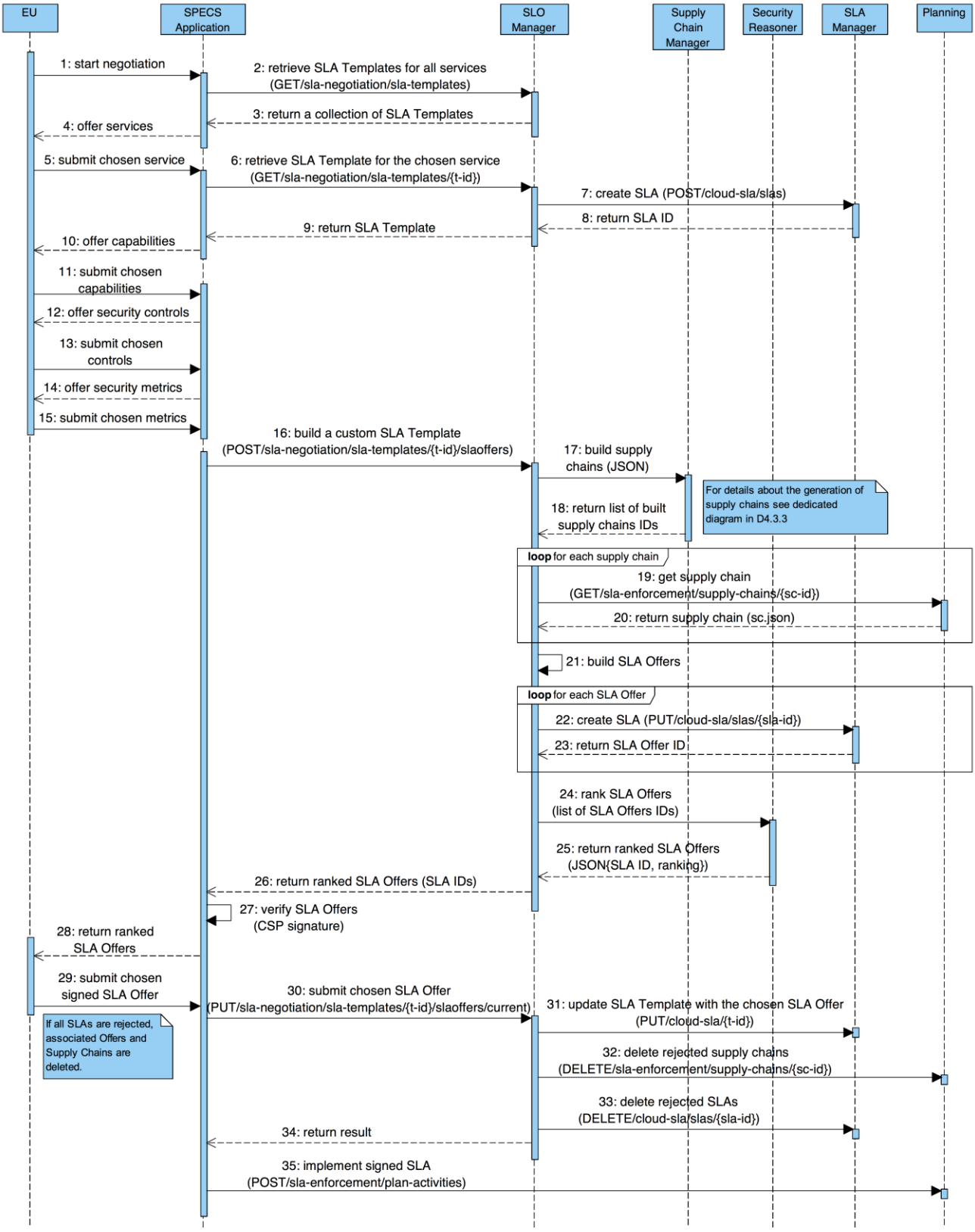

Figure 2. Detailed negotiation process

The aforementioned flow to negotiate SLAs is reused to renegotiate them. The process of renegotiation is triggered either by EUs by the CSPs/SPECS.

The first case occurs when the EU asks for different requirements. The sequence diagram of Figure 3 details this process.

Changing the requirements (controls or even capabilities) invalidates the previously signed SLA and as a result a new one can be negotiated. In this process the previously signed SLA is retrieved from the platform, which is used to prompt the EU with her/his previous preference, so that she/he can modify only some aspects of the SLA. Once the new requirements have been retrieved, the negotiation process is repeated again, with the difference that the invalid SLA has to be withdrawn from the system.

The other case of renegotiation is shown in Figure 4. It happens when an SLA has been violated. Two situations can provoke the violation of an SLA:

1. The detection (at monitoring time) of an unfulfillment of an SLO that cannot be remediated by Enforcement,
2. The CSP changing some aspect of the service that derive in the unfulfillment of the previous commitments.

In this case the negotiation process is repeated again. Same as for the EU triggered renegotiation, the previously enforced SLA is also withdrawn from the system.

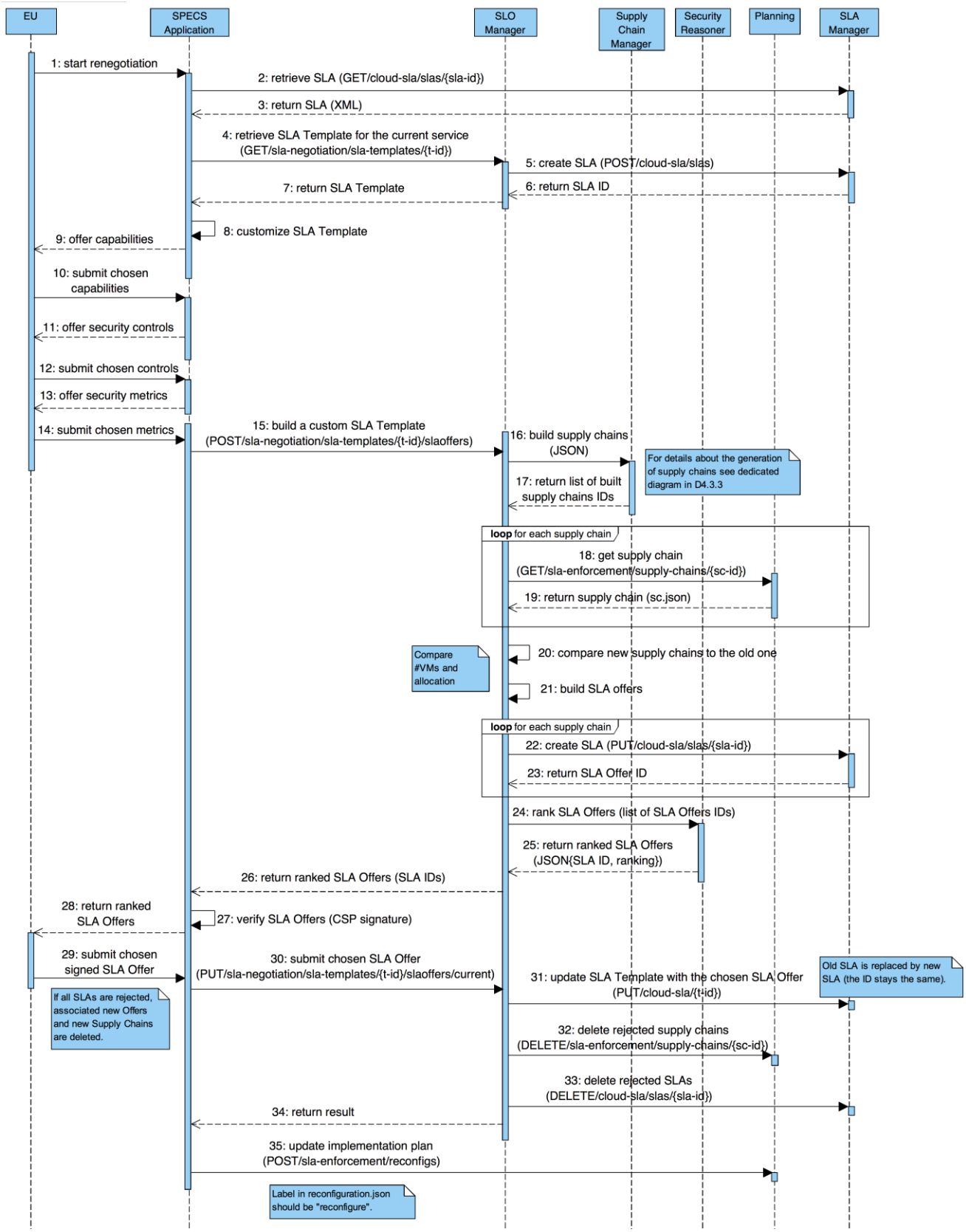

Figure 3. Detailed End-User triggered renegotiation process

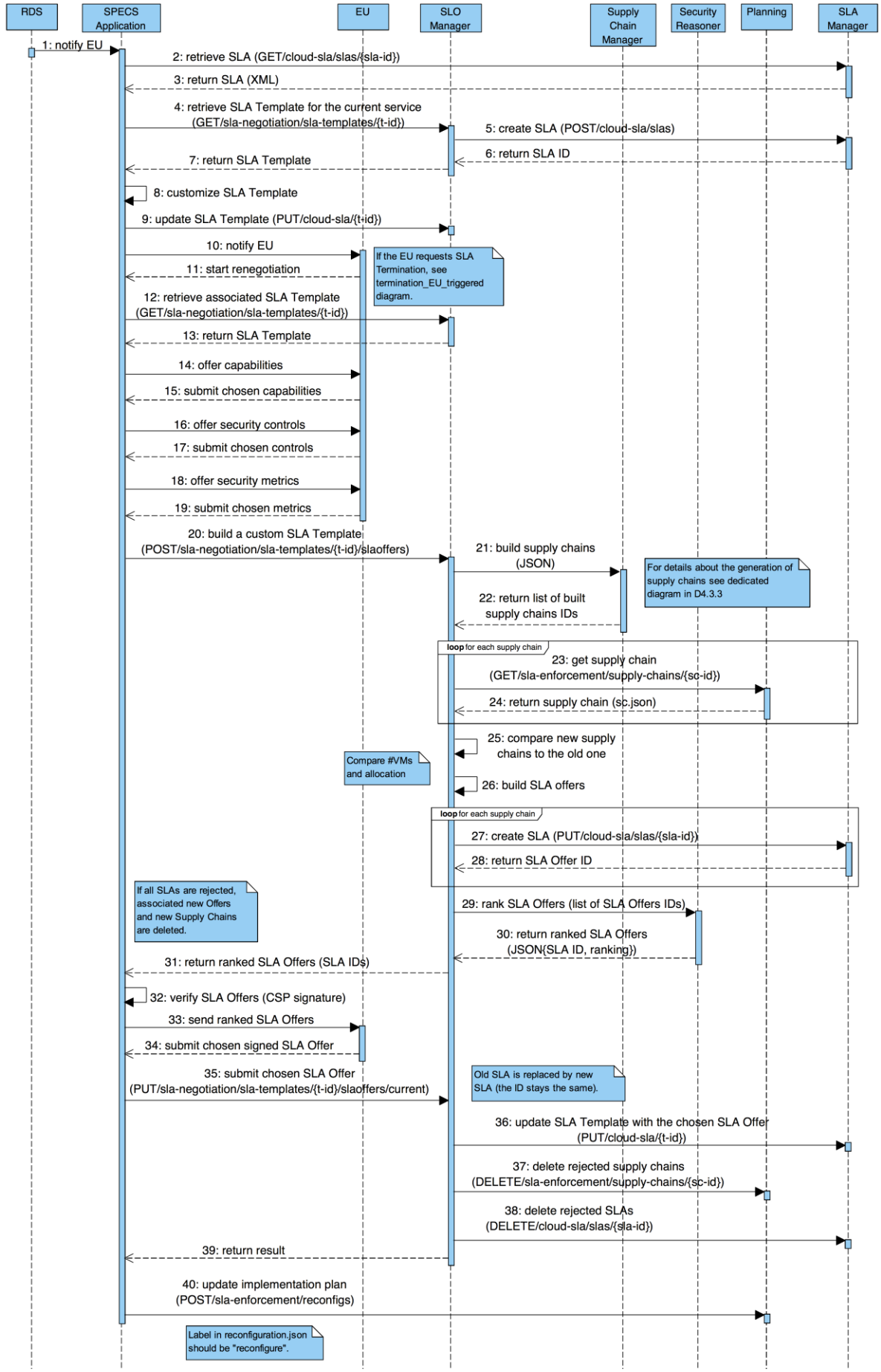

Figure 4. Detailed CSP triggered renegotiation process

4. Negotiation Module

In this section we first present the final architecture of the Negotiation module and then the artifacts (software components and models) developed within WP2 and then shows how these have covered the list of requirements (Section 4.1).

The Negotiation module comprises a list of artifacts (components and models) as follows.

component:SLOManager: the *SLO Manager* is the component that offers the Negotiation and Renegotiation API to the SPECS Application. It orchestrates the entire negotiation and renegotiation processes. It manages the creation of *SLA Templates*, it triggers generation of supply chains according to the EUs security requirements, and invokes evaluation and ranking of the *SLA Offers* that are built according to the supply chains. The complete description of the component is available in Section 4.2 while the final implementation details are reported in the prototype deliverable D2.3.2.

component:SupplyChainManager: the *Supply Chain Manager* is the component in charge of building supply chains according to the set of security requirements chosen by the EU. The creation of supply chains is supported by the Enforcement module (through the Planning component); for further details see D4.3.2. The complete description of the component is available in Section 4.3 while the final implementation details are reported in the prototype deliverable D2.3.2.

component:SecurityReasoner: the *Security Reasoner* component evaluates and ranks the *SLA Offers* created during the negotiation and renegotiation process. The evaluation is done by using security assessment techniques that apply quantification algorithms to reason about the level of security provided by each of the *SLA Offer* with respect to the EU requirements. The complete description of the component is available in Section 4.4 while the final implementation details are reported in the prototype deliverable D2.3.2.

The final architecture of the Negotiation module is presented in Figure 5.

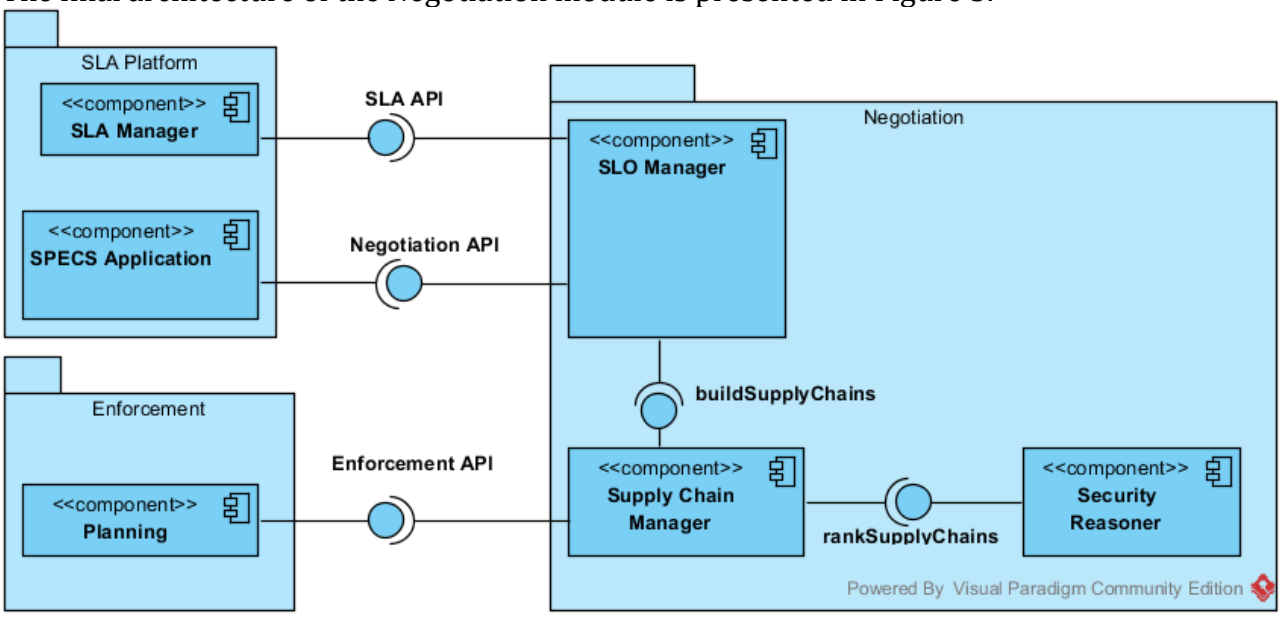

Figure 5. Negotiation module architecture

The models developed within this module contributed to the reasoning about Security SLAs, in particular the process of negotiating Service Level Objectives (SLOs). At state of the art the main issues were:

1. A common language, understandable by both Cloud Service Providers an Consumers, allowing quantitative representation of security, via SLOs;
2. Automation the provisioning of security mechanisms able to grant desired security features (activities of WP4 - Enforcement module);
3. Continuously monitoring the services in order to verify the fulfillment of specified Security SLOs (activities of WP3 – Monitoring module).

The first issue was solved within SPECS by proposing model:SLAConceptualModel (Figure 6) and model:SLAMachineReadableFormat (Figure 7), see D2.2.2 for a detailed description.

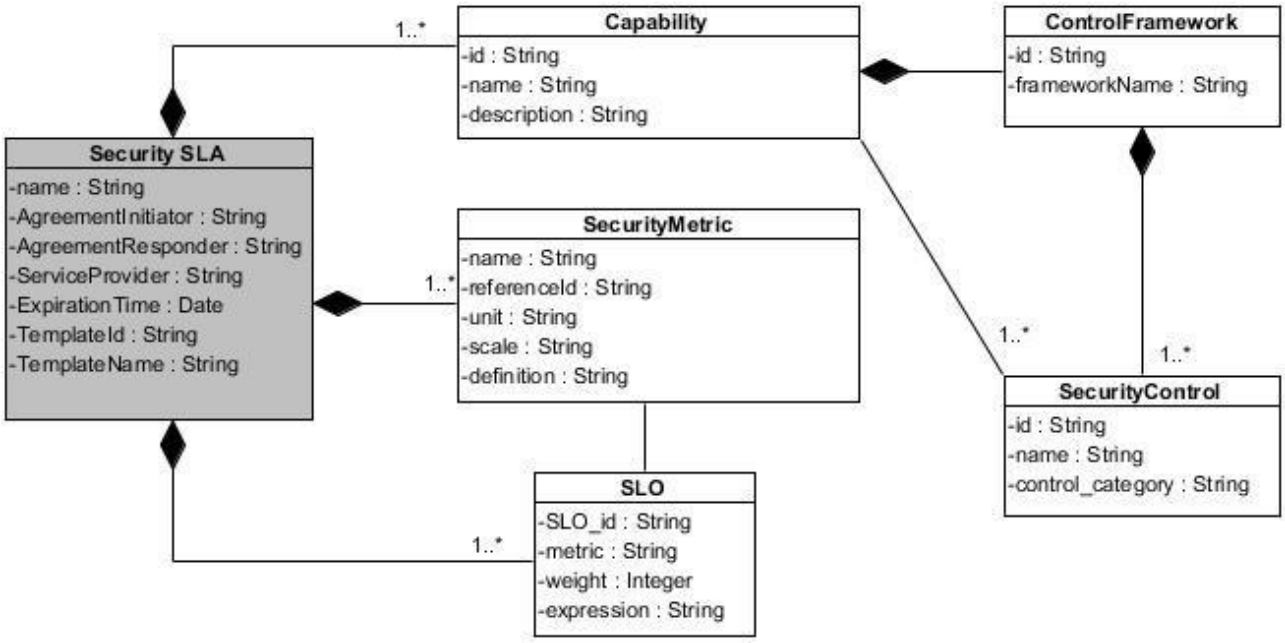

Figure 6. Security SLA conceptual model

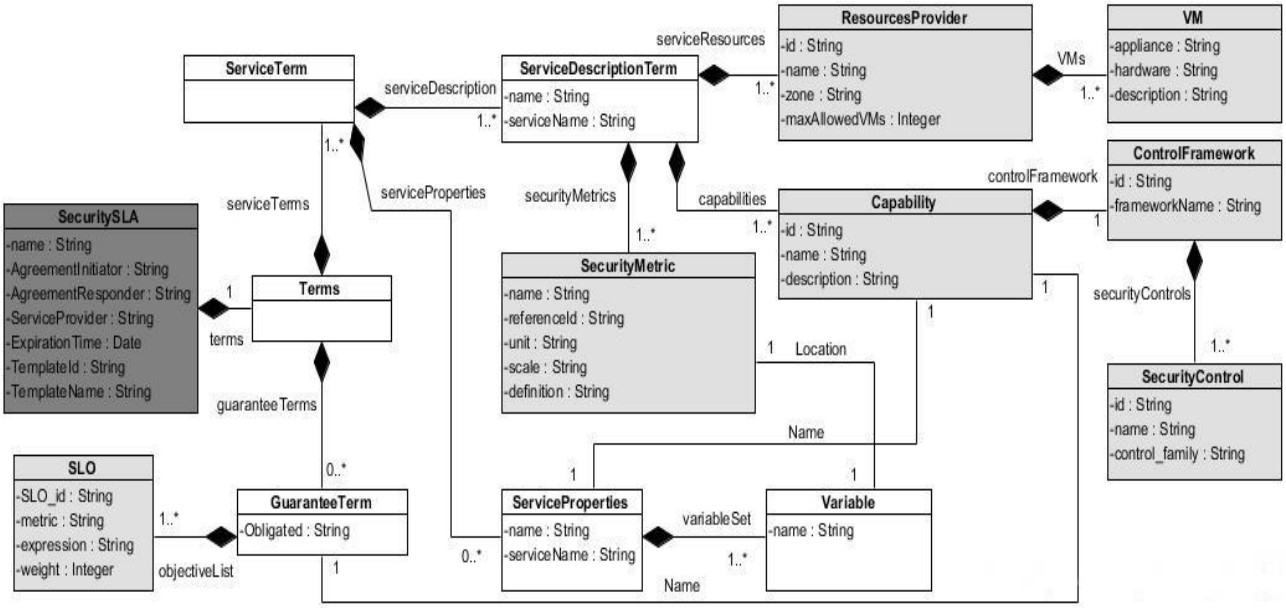

Figure 7. SLA machine-readable format

Another important artifact developed for the purpose of negotiation is the security metrics catalogue (model:SecurityMetricsCatalogue). This catalogue provides a list of quantitative and qualitative metrics which are measurable hence their values can be negotiated, monitored and enforced. These metrics rely on control categories from relevant standards such as NIST [1] and Cloud Security Alliance's CCM [2].

4.1. List of requirements covered and discussion

In the framework of WP2, the following artifacts (components and conceptual models) were developed:

- component:SLOManager
- component:SupplyChainManager
- component:SecurityReasoner
- model:SLAConceptualModel
- model:SLAMachineReadableFormat
- model:SecurityMetricsCatalogue

In the other two deliverables of this task (D2.3.1 and D2.3.2), we reported how the requirements of the Negotiation module were covered by the component artifacts (see D2.3.1 for the coverage at M18, respectively D2.3.2 for the coverage at M30).
In Table 1 of this deliverable, we report the final analysis of the requirements, with respect to their coverage by Negotiation artifacts.

REQ_ID	Requirement	Comment
SLANEG_R1	*SLA language should support specification of required cloud resources*	The conceptual model allows resources to be specified at the metric level, or higher. The machine readable format allows resources to be specified at the metric level, or higher. The SLA Lifecycle has been designed in an extensible way so as to support composition.
SLANEG_R2	*SLA language should support simple composition*	The conceptual model allows security commitments, composed from several Cloud Service Providers (CSPs), to be specified. The machine readable format is extensible to allow composition of services.
SLANEG_R3	*The negotiation process should support composite cloud services*	The evaluation techniques are based on aggregation rules that allow composition. The machine readable format is extensible to allow composition of services.
SLANEG_R4	*Negotiated Service Level Objectives (SLOs) should be monitorable and enforceable*	The conceptual model defines measurable metrics. The machine readable format defines measurable metrics. The negotiation API allows the configuration of the monitoring policies through the enforcement.

		The Supply Chain Manager parses an SLA to prepare the input for the Planning which builds supply chains. The SLA Lifecycle considers the monitoring and enforcement of the SLA.
SLANEG_R5	*Support the evaluation of trade-offs*	The conceptual model is designed to support dependencies among elements. The machine readable format is designed to support dependencies among elements.
SLANEG_R6	*Evidence associated with measured SLO*	The conceptual model allows resources to be specified at the metric level, or higher, that are monitorable and can provide evidence of the measurement. The machine readable format allows resources to be specified at the metric level, or higher, that are monitorable and can provide an evidence of the measurement. The metric catalogue is compliant with the conceptual model and the conceptual model allows resources to be specified at the metric level, or higher, that are monitorable and can provide evidence of the measurement.
SLANEG_R7	*Interactive and customer centric process*	The conceptual model allows qualitative and quantitative requirements from customers to be specified. The machine readable format allows qualitative and quantitative requirements from customers to be specified. The Negotiation protocol allows EUs preferences to be considered to negotiate the level of security demanded.
SLANEG_R8	*Specification of customer's security requirements*	The conceptual model allows qualitative and quantitative requirements from customers to be specified. The *SLAMachineReadableFormat* allows qualitative and quantitative requirements from customers to be specified. The Negotiation protocol allows EUs to negotiate the level of security demanded, specifying their preferences. The metric catalogue is compliant with the conceptual model specification.

SLANEG_R9	*Reasoning about security SLOs in cloud SLA*	The conceptual model is used by the assessment algorithms. The machine readable format is compliant with the conceptual model.
SLANEG_R10	*Follow standards and industrial-accepted practices*	The conceptual model is compliant with current standards: CSA CCM, ISO 19086, NIST 800.53 and NIST RATAX. The machine readable format is compliant with the conceptual model.
SLANEG_R11	*Mapping the user's security requirements to the CSP's offered SLOs*	The conceptual model allows qualitative and quantitative requirements from customers to be specified. The machine readable format is compliant with the conceptual model. The metric catalogue is compliant with the conceptual model specification.
SLANEG_R12	*Adoption of a conceptual model for security SLOs*	The conceptual model was created. The machine readable format is compliant with the conceptual model.
SLANEG_R13	*Security SLO should be measurable in the real-world*	The conceptual model is compliant with current standards: CSA CCM, ISO 19086, NIST 800.53 and NIST RATAX. The machine readable format is compliant with the conceptual model. SCM parses a set of SLOs to prepare the input for the Planning which builds supply chains.
SLANEG_R13	*Security SLO should be measurable in the real-world*	The conceptual model is compliant with current standards: CSA CCM, ISO 19086, NIST 800.53 and NIST RATAX. The machine readable format is compliant with the conceptual model. SCM parses a set of SLOs to prepare the input for the Planning which builds supply chains.
SLANEG_R16	*Only measurable security SLO's can be negotiated*	The conceptual model specifies security at the metric level and measurable metrics are compatible with the model. The machine readable format is compliant with the conceptual model. The negotiation protocol can negotiate measurable and unmeasurable parameters.

		The negotiation API can negotiate measurable and unmeasurable parameters. The metric catalogue is compliant with the conceptual model.
SLANEG_R18	*Management of Alerts on agreed SLA's*	The conceptual model allows the SLOs to be defined with the thresholds that must be fulfilled. The machine readable format is compliant with the conceptual model. The SLA Lifecycle considers the management of events that may entail violations or alerts
SLANEG_R19	*SLO representation using a machine-readable SLA specification*	The conceptual model can be represented using a machine readable format. A machine readable format was created.
SLANEG_R20	*Security metrics might have quantitative or qualitative values.*	The conceptual model allows EUs requirements to be represented, both qualitative and quantitative. The machine readable format is compliant with the conceptual model. The metric catalogue is compliant with the conceptual model.
SLANEG_R21	*Ordered values for security metrics.*	The conceptual model is compatible with the order relationship. The machine readable format is compliant with the conceptual model. The metric catalogue is compliant with the conceptual model.
SLANEG_R22	*Security metrics operators*	The conceptual model is compatible with operators for metric values. The machine readable format is compliant with the conceptual model. The metric catalogue is compliant with the conceptual model.
SLANEG_R23	*Output of a successful negotiation process*	The conceptual model represents the SLA hierarchy to negotiate. The machine readable format is compliant with the conceptual model. The negotiation protocol uses the machine readable representation created. The negotiation API uses the machine readable representation created.

		The negotiation protocol uses the machine readable representation created.
SLANEG_R24	*Independence from Interaction models*	The conceptual model can be reused in other domains. The machine readable format can be reused in other domains. The negotiation protocol format can be reused in other domains.
SLANEG_R25	*Renegotiation triggered by CSP or the EU*	The conceptual model is also used to manage the renegotiation. The machine readable format is compliant with the conceptual model. The negotiation protocol is also used to manage the renegotiation. The negotiation API is also used to manage the renegotiation.
SLANEG_R26	*Input for renegotiation*	The conceptual model is used to manage the renegotiation.
SLANEG_R27	*Output of a successful renegotiation*	The conceptual model is used to manage the renegotiation.
SLANEG_R28	*Human-assessment of security metrics*	The conceptual model allows any type of metrics to be integrated. The machine readable format allows any type of metrics to be integrated. The negotiation protocol allows any type of metrics to be integrated. The metric catalogue allows any type of metric to be included.
SLANEG_R29	*Uncertainty/assurance of performed measurements*	The conceptual model allows any type of metrics to be integrated. The machine readable format allows any type of metrics to be integrated.
SLANEG_R30	*Remediation through SLA renegotiation*	RDS triggers renegotiation if no other remediation actions recover from SLA violation.
SLANEG_R31	*Alerts/violations affecting multiple elements of the secure SLA hierarchy*	RDS considers metric interrelationships when identifying optimal remediation actions.

SLANEG_R32	*Platform repositories*	The conceptual model is used as the format to store SLAs in the repository. The machine readable format is used as the format to store SLAs in the repository. The metric catalogue is part of the SLAs that are stored in the repository. The SLA Lifecycle considers the storage of SLAs in the repository.
SLANEG_R33	*SLA Management*	The SLA Lifecycle was created to model the SLA Management.
SLANEG_R34	*Representing security requirements of non-expert users*	The negotiation protocol uses the machine readable representation created.
SLAPL_R14	*Search CSP SLA*	The conceptual model and the machine readable format permit SLAs from CSPs to be represented.
SLAPL_R21	*Get SLA*	The conceptual model and the machine readable format permit SLAs from CSPs to be represented.
SLAPL_R33	*Sign SLA*	The negotiation protocol and the negotiation API allows the chosen SLA to be signed by EUs.

Table 1 Coverage of Negotiation Module Requirements

4.2. SLO Manager

SLO Manager is a core component of the Negotiation module and of the SPECS solution, although it is not visible to the users. It consists of a RESTful web service that captures EU security requirements, translates them into a machine-readable format to be used by the other Negotiation module components and SPECS modules, and, finally, translates the result of the negotiation into *SLA Offers*, from which the EU must choose one to be signed.

4.2.1. *SLO Manager* Behaviour

SLO Manager provides support for negotiating *SLA Offers*. At M18, in D2.3.1, we presented the behaviour of this component, constructing an *SLA Offer* starting from a *ServiceDescriptionTerm*. The security capabilities contained in the *ServiceDescriptionTerm* have been used to construct the *SLOs based* on the user input, which together constructed the SLA for which offers had to be generated. Following the definition of the Negotiation and Renegotiation flow, given at M24, in D2.2.2, *SLOManager* maintains a collection of *SLA Templates* describing default configurations for security metrics and SLOs. Such a *SLA Template* can be retrieved and customized based on the user's needs before starting a negotiation process. The negotiation process starts with the creation of a new SLA in the *SLA Manager*. Once the negotiation started the user can request offers for requested requirements. This process involves the *Supply Chain Manager*, which generates the supply chains consisting of the resources needed to fulfil the requirements. For each supply chain received, *SLO Manager* generates a *SLA Offer* and creates the associated SLA in the *SLA Manager*. When the user selects one of the offers to be implemented, *SLO Manager* deletes the unused SLAs and supply chains from the *SLA Manager* and *Planning* components, respectively, and updates the value of the initial created SLA with the value of the selected offer. In case the SLA needs to be renegotiated, the implemented SLA is deleted from the *SLA Manager*, and the template is updated in the *SLO Manager*, after which the usual negotiation process can

take place. From this step ahead, there is no distinction between negotiation and renegotiation of an SLA from the point of view of *SLO Manager*. Depending on external events, such as a CSP triggering renegotiation, due to an unsupported service, the associated *SLA Template* will be deleted or updated, but *SLO Manager* is agnostic to the context in which the operation is made. After the update has been made on the associated *SLA Template*, triggering renegotiation, the process continues as a normal negotiation process: the *SLA Template is* retrieved, *SLA Offers* are generated for it, and one is accepted. In the case that the EU wants to terminate his/her SLA, this can be done via the SPECS Application, not requiring the negotiation module.

4.2.2. *SLO Manager* Architecture

The *SLO Manager* component is responsible for the negotiation process, by maintaining a collection of *SLA Templates* which can be adjusted and used to generate *SLA Offers*, from which one can be selected to be implemented. For the creation of *SLA Offers*, several SPECS components are used: *SLA Manager* is used to store details of the currently negotiated SLA; the *Planning* component is used through *Supply Chain Manager* to generate supply chains for negotiated security capabilities; and *Security Reasoner* is used to rank the *SLA Offers.*
The architecture of *SLO Manager* is composed of three layers, designed to address the afore-mentioned functionality; it is presented in Figure 8.
The **Persistence Layer** uses the SPECS Data Model artefact, in order to translate XML data into Java data structures used for storing the XML fields, which provide access to database operations and indexing over specific fields. Additionally, other components can use the same objects, to avoid serialization/deserialization issues. MongoDB is used to store *SLA Templates* and a mapping between them and negotiated *SLA Offers*.
The **Components Interaction Layer** is represented by the interaction with the interfaces exposed by other SPECS components; the logic and implementation for accessing, updating and deleting external resources through their REST API are implemented at this layer. The logic of supply chain generation is implemented in *Supply Chain Manager* artefact, the creation of supply chains being made as a method invocation.

The **Resource Presentation Layer** is responsible for the REST API implementation, including the implementation for CRUD [1] operations over *SLA Templates*, as well as the logic for generating, presenting and implementing *SLA Offers*.

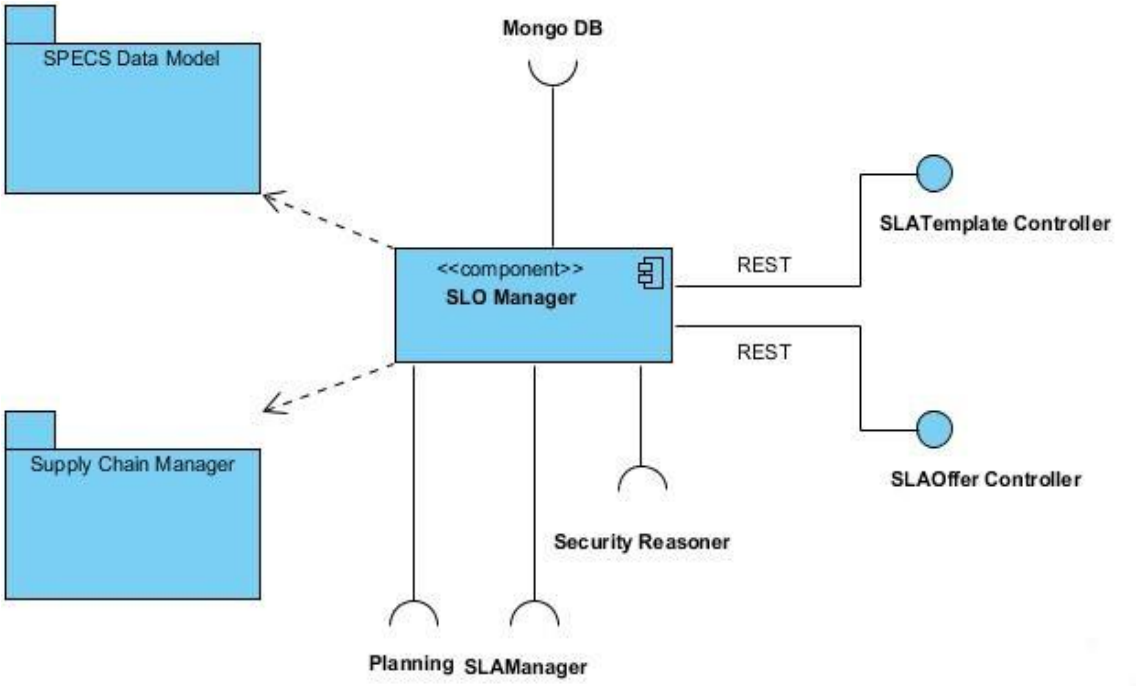

Figure 8. *SLO Manager* architecture

4.2.3. Negotiation API Updates

In D2.3.1 we described the preliminary version of the Negotiation API, namely the RESTful interface, which provided access to the *SLO Manager* component. That interface provided functionalities for manipulating security capabilities, service description terms (SDTs), security metrics, Service Level Objectives (SLOs) and uses the functionalities of other modules, e.g. *SLA Platform (*stores the security capabilities together with their metadata*)*, *Supply Chain Manager* (stores valid supply chains), *Security Reasoner* (stores ranked and valid supply chains).

Compared to the prototype available at M18, the *SLO Manager* REST API required some modifications, due to the final requirements of the SPECS solution architecture and modules interactions protocols. Therefore, the current list of REST API methods manipulates:

- *SLA Templates* (retrieval and deletion of *SLA Templates* collection, retrieval of an *SLA Template* with a certain id, creation of a new *SLA Template,* update and deletion of an *SLA Template* with a certain id), and
- *SLA Offers* (retrieval and deployment of *SLA Offers* constructed for the *SLA Template* with a certain id, retrieval of Currently employed *SLA Offers*).

Note that the renegotiation logic is embedded in the Negotiation API. If renegotiation is triggered by the EU, the current SLA is deleted from *SLA Platform* and a new negotiation process starts based on the *SLA Template* the SLA was generated. In the case of a CSP triggered renegotiation because a service can no longer be provided, after the deletion of the current SLA, the *SLA Template* used for its generation is updated with regard to the supported services and a new negotiation process starts based on the updated *SLA Template*.

[1] CRUD – Create, Read, Update, Delete

In Annex B - Negotiation API we present the complete description of the REST API associated to the *SLO Manager*.

4.3. Supply Chain Manager

As discussed in Section 3, the Negotiation module in SPECS oversees (i) the elicitation of EU's requirements and (ii) their translation into a set of ranked *SLA Offers*. In order to prepare a set of *SLA Offers* that comply with EU's requirements, on one side, and CSPs' and SPECS' offers, on the other side, the Negotiation module relies on the Enforcement module, to analyse the input and generate a set of feasible supply chains. The *Supply Chain Manager* component has a Client that invokes functionalities of the Enforcement module in order to build feasible supply chains. In particular, it parses the *SLA Template* customized by the *SLO Manager* with the EU's requirements, and prepares the input for the Enforcement module.

In the next subsections, we introduce the process orchestrated by the *Supply Chain Manager* component in detail, and present the architecture and the interface.

4.3.1. *Supply Chain Manager* Behaviour

As previously said, *SLO Manager* component customizes an *SLA Template* with security controls and security metrics for the security service chosen by the EU. The customized *SLA Template* is, then, used by the *Supply Chain Manager* component to prepare the input for the Enforcement module (in particular, for the Planning component), to trigger the generation of feasible supply chains, and to return the list of their IDs to the *SLO Manager*. The detailed process is depicted in the sequence diagram in Figure 9.

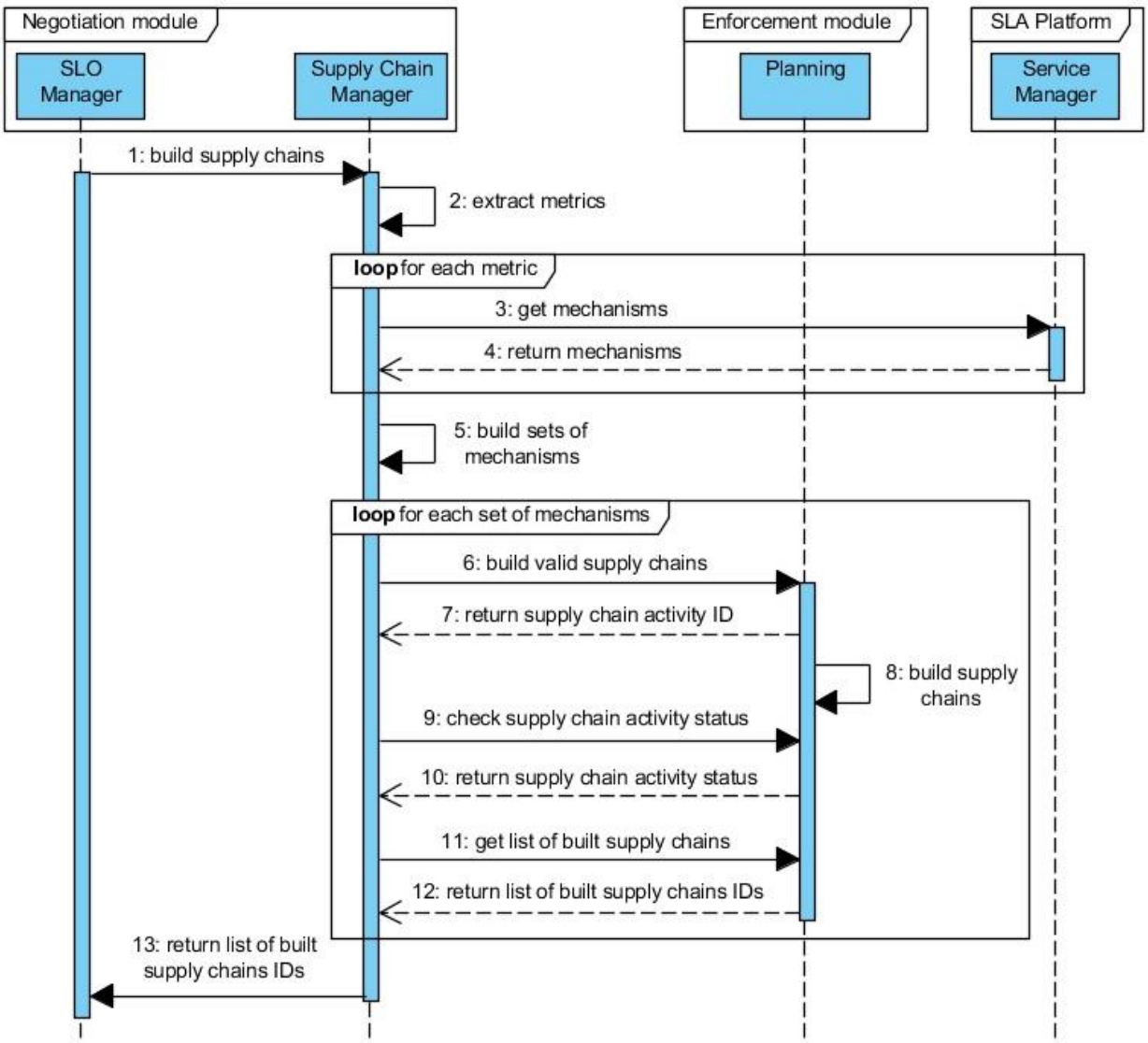

Figure 9. *Supply Chain Manager* Sequence Diagram

The process starts (Step 1 in the diagram) with the *SLO Manager* component triggering the *Supply Chain Manager* to parse the *SLA Template* and to prepare the input for the Enforcement module. The *SLA Template* contains the information about the security service, controls, and metrics, selected by the EU, to be enforced on top of the selected service. The *Supply Chain Manager* extracts the list of security metrics (Step 2), to identify SPECS security mechanisms that are able to enforce and monitor the metrics. To this end, the *Supply Chain Manager* uses the information provided by the Service Manager component (Steps 3 and 4). Since each security metric can be enforced and monitored with more than one security mechanisms, the *Supply Chain Manager* prepares all possible combinations of available security mechanisms which implement the SLA (Step 5). For each set, the *Supply Chain Manager* then triggers the Planning component to build feasible supply chains (Step 6). The Planning component creates a supply chain activity object that contains all information about the process itself (e.g., the ID of the *SLA Template*, the set of security mechanisms for which the supply chains are being generated, the state of the process, etc.). Its ID is then returned to the *Supply Chain Manager* (Step 7) and the generation of the supply chains begins (Step 8). The *Supply Chain Manager* continuously checks the status of the supply chain activity (Steps 9 and 10), and when the status is *Completed*, the list of IDs of the generated supply chains (stored by the Enforcement module) can be retrieved (Steps 11 and 12). At the end of this process, the *Supply Chain Manager* collects

the IDs of all generated supply chains (for all sets of security mechanisms) and returns them to the *SLO Manager* component (Step 13).

For the steps covered by the Enforcement module (Step 8), see deliverables D4.3.2 and D4.3.3. For details about the REST APIs (for calls to the Enforcement module and the Service Manager in Steps 3, 6, 9, and 11) and resources (the supply chain activity object), see deliverable D1.3.

4.3.1. *Supply Chain Manager* Architecture

The *Supply Chain Manager* is implemented as a Java library and is packed into a Java archive (JAR) file. The *SLO Manager* uses the *Supply Chain Manager* as a dependency and calls its Java API directly.

4.3.2. *Supply Chain Manager* Interface

The *Supply Chain Manager* component offers a simple Java API which is used by the SLO Manager:

```
SupplyChainManager(String serviceManagerApiAddress, String planningApiAddress)
List<SupplyChain> buildSupplyChains(AgreementOffer agreementOffer) throws
SupplyChainManagerException
```

The *Supply Chain Manager* constructor accepts two parameters: address of the Service manager and address of the Planning component. The method *buildSupplyChains* accepts *SLA Template* as an AgreementOffer object and returns a list of built supply chains objects. In case anything goes wrong, the method throws *SupplyChainManagerException*.

4.4. Security Reasoner

4.4.1. *Security Reasoner* Behaviour

The *Security Reasoner* is a SPECS component devoted to ranking Security SLAs. In particular, the component offers functionalities to evaluate and rank Cloud Service Providers based on the publicly available information, regarding the security controls they put in place (cf. the CAIQ initiative by CSA). Building on this knowledge, as discussed in D2.2.2, the *Security Reasoner* enables *per-service SLA Offers* to be ranked, by taking into account the CSPs that will host the service resources (i.e., the virtual machines) and the specific security capabilities offered, both included in the SLA.
In the default SPECS flow (see deliverable D1.3), *the Security Reasoner* component will be used only by the *Supply Chain Manager* in order to rank the *SLA Offers* generated during the negotiation process. Currently, the component is available, also, to the SPECS Owner, in order to help him/her compare available providers based on specific evaluation criteria. Figure 10. *Security Reasoner* Use Cases summarizes the component use cases.

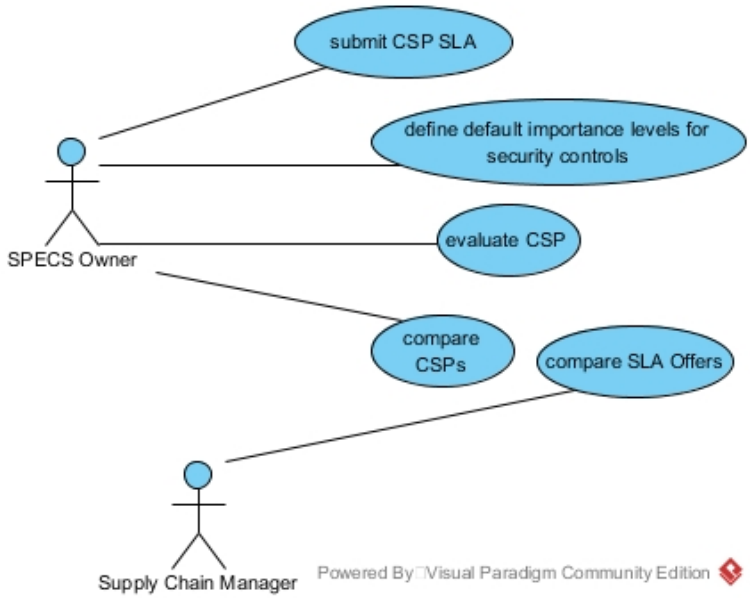

Figure 10. *Security Reasoner* Use Cases

As shown, the only use case related to the Supply Chain Manager actor is the comparison of *SLA Offers*, carried out by means of the techniques illustrated in D2.2.2. Conversely, for the SPECS Owner, several functionalities are supported:

- *Submit CSP SLA*: when the SPECS Owner wants to support a new CSP (e.g., a new provider from which the broker can acquire resources), he/she has to upload the security-related information associated with such a new CSP. Due to the need for concrete information on the security offered by a CSP, we adopt the information retrieved from the CAIQ and consider it as the provider's SLA. Therefore, the *Security Reasoner* offers a simple web interface that enables the CAIQ responses for the new CSPs to be submitted.
- *Define default importance levels for security controls:* CAIQ entries refer to specific controls, which are all given the same importance by default. In order to support a different weighting strategy, we allow the SPECS Owner to update the default importance levels for security controls.
- *Evaluate CSP*: the SPECS Owner can evaluate a specific provider in order to obtain a score for each control family that expresses the level of coverage of the family.
- *Compare CSPs*: the SPECS Owner can compare a set of providers based on specific criteria.

4.4.2. *Security Reasoner* Architecture

The updated *Security Reasoner* Architecture is depicted in Figure 11 The first release of the *Security Reasoner*, described in D2.3.1, was accessible as a standalone SPECS Application. In the current version, the functionalities of the *Security Reasoner* are now available through the Evaluation REST API, later described, and have been integrated into the SPECS flow.

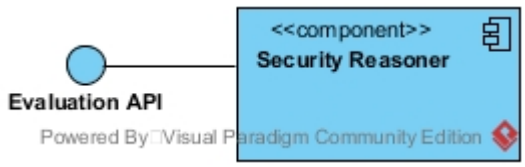

Figure 11: Security Reasoner architecture

4.4.3. *Security Reasoner* REST API

The Security Reasoner *Evaluation* REST API provides the functionalities to:

- *Manage* (access, update, delete) the Consensus Assessment Initiative Questionnaire (CAIQ) data of cloud providers, represented in a tree-like structure;
- *Manage* (access, update, delete) *judgements*, which define the weights assigned to each element of the CAIQ structure;
- *Associate* a judgement with a CAIQ;
- *Evaluate CAIQs* based on a judgement;
- *Rank CAIQs*;

The complete documentation of the Evaluation API is illustrated in Annex A – Evaluation API.

5. Conclusions

The SPECS project aimed at designing and implementing a framework for the management of the whole SLA life cycle (Negotiation, Monitoring, Enforcement), intended to build applications (SPECS Applications) devoted to offering services to SPECS Customers, whose security features are stated in and granted by a Security SLA.

A potential user of our Negotiation module might wonder what elements from the negotiation/renegotiation processes might include latency. One would be the interaction of with the EU because she/he has to specify his/her security requirements and decide which SLA Offer suits his requirements the best. Second would be in the creation and validation of the supply chains via calls to the Enforcement module. More precisely, checking the availability of resources in order to determine the valid supply chains might lead to latency that depends on things such as the availability of the cloud resources and services.

As stated in the DoW and detailed in D2.1.2, the main issues related to the adoption of Security SLAs are:

- representing security features so that it is understandable by both customers and providers and measurable (by means of verifiable security-related Service Level Objectives (SLOs)),
- automating the provisioning of security mechanisms able to grant desired security features (by means of a security-driven resource allocation process), and
- continuously monitoring the services in order to verify the fulfillment of specified Security SLOs (by means of Cloud security monitoring solutions).

In [3], [4], we proposed to face the Security SLA life cycle management with a framework able to enrich Cloud applications with security features. We presented a novel Security SLA model which was applied to a security-driven planning process that can be adopted to determine the (optimum) deployment of security-related software components. It was applied in a read case study from our EMC partner, see [5].

Other impediments for the wide adoption of Security SLAs is that the security features offered by different Cloud Service Providers (CSPs) are very similar and one does not have an overview of these differences and how they imply the costs. Moreover there might be differences between security related services therefore more challenges occur, e.g. specification of security requirements taking service dependencies into consideration and determining which CSP can satisfy these requirements.
In [6], we proposed a framework which automatically detects conflicts resulting from inconsistent customer requirements, provides an explanation for the detected conflicts allowing customers to resolve these conflicts. Moreover, it assesses the security level provided by various CSPs and ranks the CSPs according to the desired customer requirements. The framework was tested on case studies from CSA partner and Trust and Assurance Registry.

Lack of security assurance and transparency on Cloud Services is another impediment for the adoption of Security SLAs.
In [7] we developed two evaluation techniques, namely QPT and QHP, for conducting the quantitative assessment and analysis of the Security SLA based security level provided by CSPs with respect to a set of Cloud Customer security requirements. These proposed techniques help to improve the security requirements specifications by introducing a flexible and simple methodology that allows Customers to identify and represent their specific security needs.

In order to address the issues above, within WP2 we proposed and developed a set of artifacts, mainly models and components, which were also designed with the goal of *allowing user-centric negotiation of Cloud SLA* by proposing a solution assisting End Users to negotiate security features effectively with a set of CSP, by understanding the resulting trade-offs.

In the table below we present the list of objectives associated to the task T2.3 and report the outcomes which verify the benefits of the results achieved in this task in the entire duration of the project.

Objective	Result
SO2.1: Design of **user-centric Cloud SLA negotiation solution** for security parameters	We developed an innovative solution able to automatically negotiate and configure cloud resources to (optimally) deploy security-related software components for the enforcement of the Security SLOs included in the SLA [3], [4]. This was possible due to the development of a *conceptual model for SLAs*, a *machine-readable format* for representing them as well as *security metrics catalogue* standards-compliant which allow matching customers security requirements reported in an SLA with a set of security mechanisms offered as a service (Security-as-a-Service).
SO2.2: Develop the techniques to systematically **evaluate the trade-offs** related with offered security in Cloud SLA	One important phase of Negotiation is the generation of supply chains. One supply chain comprises one CSP and the security capabilities offered by SPECS enhancing some specific security features according to the EU's preferences. A supply chain is materialized in an *SLA Offer* to be signed by the EU. In order to assist the EU in choosing the SLA Offer suitable to his/her security needs, we designed the *Security Reasoner.* It is based on two assessment algorithms that are able to compare EU security requirements with respect to the security controls provided by CSPs. SPECS has designed two algorithms: • REM [8] that uses aggregation techniques to perform an evaluation of the security level provided by a provider. • A fuzzy logic [6], [7] based security assessment methodology based on fuzzy-AHP that is able to manage uncertainty of EU's requirements to provide a multi-layered comparison of the security provided by providers and requirements demanded by EUs.
SO2.3: Provide a **reference implementation** of security negotiation services for Cloud SLA	The Negotiation module architecture [9] is composed of three components: *SLO Manager*, *Supply Chain Manager*, *Security Reasoner*. *SLO Manager* is in charge of managing the entire negotiation and renegotiation processes in a user-centric manner. The *Supply Chain Manager* is in charge of building supply chains according to the set of security requirements chosen by the End User. Finally the *Security Reasoner* component evaluates and ranks the *SLA Offers* created during the negotiation and renegotiation process ensuring the evaluation of the trade-offs related with offered SLAs.

Table 2. WP2 objectives and results

6. Bibliography

[1] "(Draft) Cloud Computing: Cloud Service Metrics Description", http://www.nist.gov/itl/cloud/upload/RATAX-CloudServiceMetricsDescription-DRAFT-20141111.pdf.

[2] Cloud Security Alliance, "Cloud Control Matrix v3.0.", https://cloudsecurityalliance.org/download/cloud-controls-matrix-v3/ .

[3] V. Casola, A. De Benedictis, M. Erascu, J. Modic, and M. Rak, "Automatically Enforcing Security SLAs in the Cloud," *IEEE Trans. Serv. Comput.*, no. Special Issue on Security and Dependability of Cloud Systems and Services, 2016.

[4] A. De Benedictis, M. Rak, and U. Villano, "SLAs for Cloud Applications: Agreement Protocol and REST-based Implementation," *Int. J. Grid Util. Comput. by Inderscience*, 2016.

[5] V. Casola, M. Rak, S. La Portailvio, and A. Byrne, "Providing Security SLA in next generation Data Centers with SPECS: the EMC Case Study," in *CLOSER*, 2016.

[6] A. Taha, P. Metzler, R. Trapero, J. Luna, and N. Suri, "Identifying and Utilizing Dependencies Across Cloud Security Services," in *AsiaCCS*, 2016.

[7] J. Luna, A. Taha, R. Trapero, and N. Suri, "Quantitative Reasoning About Cloud Security Using Service Level Agreements," *IEEE Trans. Cloud Comput.*, no. 99, 2015.

[8] V. Casola, A. Mazzeo, N. Mazzocca, and M. Rak, "A SLA evaluation methodology in service Oriented Architectures," *Adv. Inf. Secur.*, vol. 23, pp. 119–130, 2006.

[9] V. Casola, A. De Benedictis, M. Rak, and U. Villano, "SLA-based Secure Cloud Application Development: the SPECS Framework," in *MICAS*, 2015.