ASSERT4SOA

Advanced Security Service cERTificate for SOA

CP - STREP - Grant No. 257351

Framework Requirements

Deliverable D.7.1

Project Title	ASSERT4SOA – Advanced Security Service cERTificate for SOA
Grant Number	257351
Project Type	CP - STREP

Title of Deliverable	Framework Requirements
Subtitle of Deliverable	-
Deliverable Number	D.7.1
Dissemination Level	Consortium
Internal Rev. No.	-
Contractual Delivery Date	30 June 2011 (M9)
Actual Delivery Date	30 June 2011 (M9)
Contributing WPs	WP7
Editor(s)	Renato Menicocci (FUB)
Author(s)	Renato Menicocci (FUB), Vittorio Bagini (FUB), Alessandro Riccardi (FUB), Michele Bezzi (SAP), Antonino Sabetta (SAP), Samuel Paul Kaluvuri (SAP), George Spanoudakis (CITY), Antonio Mana (UMA)
Reviewer(s)	Simona Brugnoni* (IT.TS.SI.), Paolo de Lutiis* (IT.TS.SI.), Marco Anisetti (UNIMI), Claudio Ardagna (UNIMI), Ernesto Damiani (UNIMI), Andres Benjumea (UMA)

* External reviewer

Document Reviews

Date	Version	Reviewer	Comments
10 June 2011	0.8	Simona Brugnoni* (IT.TS.SI.), Paolo de Lutiis* (IT.TS.SI.), Marco Anisetti (UNIMI), Claudio Ardagna (UNIMI), Ernesto Damiani (UNIMI), Andres Benjumea (UMA)	*(omitted)*

* * External reviewers.

Executive Summary

The ASSERT4SOA framework aims at extending the functionalities of current SOA systems, by making it possible, for a service consumer, to lookup and consume services not only based on their functional properties, but also, by expressing the security properties and assurance that the services should provide. A key role in this improvement is played by a digital certificate, called ASSERT, specifically designed to provide assurance about the security properties of a service. The framework, therefore, should provide advanced ASSERT-aware service discovery and matchmaking, as well as a support for managing the lifecycle of ASSERTs, including their issuing and revocation.

The primary objective of this document is to specify use cases and requirements for the ASSERT4SOA framework.

Using standard software engineering approach, we start by identifying and describing a small but representative set of concrete usage scenarios. We consider two scenarios (*ASSERT-Enabled Service Market*, *Proactive Dynamic Discovery*) and multiple use cases, where the extended functionality of the ASSERT4SOA framework would be effective in satisfying security needs of service consumers. From these high-level scenarios, we extract a number of use-cases that correspond to specific actor-system interactions of general applicability. For each of such use cases, we identified the actors involved, and we characterized the underlying interaction as a sequence of steps.

The analysis of these use cases was in turn the starting point for the formulation of a set of *assumptions* (facts that are assumed to be satisfied by the context surrounding the system being specified) and *requirements* (the functionalities and characteristics that the ASSERT4SOA framework must or should exhibit).

In this document, a special emphasis is given to the requirements that are specific to the goals of the ASSERT4SOA framework. A number of requirements that are generally applicable to any service-oriented (or even every distributed) system are also provided, but at a lower level of detail.

Later in the project, it may become necessary to refine and document these general requirements more precisely, and we will do so in the deliverables documenting more advanced stages of the design and development of the framework.

Lastly, to facilitate the reading, this document integrates an Appendix, containing an essential vocabulary with the terms used along the text.

Contents

Chapter 1

Introduction

1.1 Scope and objectives

The objective of Deliverable 7.1 is to elicit and specify the requirements based on a set of general use cases for the ASSERT4SOA framework. In particular, we provide detailed specifications for the aspects that are more specifically related to the objectives of the project, and in particular to those referring to ***certification-specific* requirements**.

Of course, the ASSERT4SOA framework shares many requirements with other frameworks providing support for other non-functional properties, e.g. reliability or performance. These ***generic* requirements**, which apply to different architectures and frameworks, are certainly relevant for the deployment and operation of the ASSERT4SOA framework, but this document addresses them only by providing high-level descriptions. As an example, we capture the need for providing monitoring capabilities by stating that *"[...] non-security-relevant actions (defined by the administrator) SHOULD be captured and made available so as to support runtime monitoring"*. Requirements like this are not ASSERT4SOA-specific, and therefore we do not detail them further. Later in the project, it may become necessary to explicitly document these general requirements, and we may do so in the deliverables documenting more advanced stages of the design and development of the framework.

On the other hand, an entire chapter of the document is dedicated to precisely identifying and documenting the **assumptions** about the functionalities or conditions that are not provided by the framework (and which therefore are not the subject of requirements), but that should be provided by the environment to the framework. If any of these environment functionalities are not available or do not work properly, the functionalities offered by the ASSERT4SOA framework could be prone to either malfunctioning or unavailability.

This document is organized as follows: in the remaining part of Chapter 1, we describe the *methodology* and conventions used for the requirement elicitation.

Chapter 2 describes the two *illustrative scenarios*, showing examples of how the framework would be used in practice[1]. Based on these scenarios, we extracted a set of *use cases of general applicability*, which are described in Chapter 3. The *assumptions* on which the ASSERT4SOA framework relies and that are necessary for its deployment and correct operation, are described in Chapter 4.

Functional and non-functional requirements, derived from the use cases, are detailed in Chapters 5 and 6, respectively. Finally, the document includes an Appendix providing an *essential vocabulary* defining the terms used along the text. The contents of the Appendix are only provided for the sake of clarity, and are actually *preview* (as such, they are subject to change in the coming months) of the working document that will be delivered as part of Deliverable D3.1 ("Requirements for an ontology supporting certificates interoperability"), to be delivered by WP3 at M12 (September 2011).

1.2 Methodology

In this section we briefly describe the methodology we followed in the elicitation of requirements. Essentially, our approach is based on established common practices in requirements engineering [2,3], while striving for simplicity. As a preliminary step, we identified a set of guidelines from the existing literature and accepted practice in requirements engineering. These guidelines refer both to the *process* of requirement elicitation itself, and to the *documentation* of requirements.

Consistently with these guidelines, we started by identifying and describing a small but representative set of concrete usage scenarios, documented in Chapter 2 ("Reference Scenarios"). Starting from these high-level scenarios, we could then extract a number of use cases that correspond to specific actor-system interactions of general applicability. For each of such use cases, we identified the actors involved, and we characterized the underlying interaction as a sequence of steps.

The analysis of these use cases was in turn the starting point for the definition of *assumptions* (facts that are assumed to be satisfied by the context surrounding the system being specified) and *requirements* (the functionalities and characteristics that the ASSERT4SOA framework must or should exhibit).

1.3 Terminology and conventions

We distinguish in this document the notions of "scenario" and "use case". The term *scenario* is used to describe, from the point of view of sample stakeholders, how the system is used in a specific, but representative example. A scenario is therefore more specific (being an example) and more high-level than a use case. A *use case,* on

[1] It is important to remark that the definition of a scenario of specific *industrial* relevance is the goal of Task 7.2 (M9-M18), and it is being worked on, as of this writing. Such a scenario will be used as a reference for the validation of the ASSERT4SOA framework; it will be covered in depth in Deliverable 7.2 (Case Study), due at M18.

the other hand, is intended here as a more generally applicable interaction between one or more actors and the system. A use case may apply to multiple scenarios (e.g., the *user authentication* use case); at the same time, a scenario can refer to multiple use cases (e.g., a scenario may be associated both to a *user authentication* and *system administration* use cases).
Another important remark about the terminology adopted in this document is about the use of the key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL”, which are to be interpreted as described in RFC 2119 [1].
Further conventions as to the naming and documentation of use cases, requirements and assumptions are described in the remainder of this chapter.

1.3.1 Use case identification and classification

The naming convention for use cases follows the template below:

UC_NAME

where NAME is a unique suffix.

1.3.2 Use case format

UC_NAME
Description: *A paragraph describing the main idea of the use case and its purpose.*
Actors: *List of entities involved in the use case.*
Preconditions: *List of assumptions that must be met before the use case can start.*
Trigger: *What event triggers the use case.*
Basic flow: *This is the flow of events/states in the "happy day", when there are no errors.*
Alternate Flows: *List of possible alternative flows (the most significant ones) that come to an error condition.*
Postconditions: *List of expected outputs and state the system is supposed to be in when the use case ends.*
Dependencies: *Other use cases whose execution without errors is a required by the use case.*

1.3.3 Assumption identification and classification

The naming convention for assumptions follows the template below:

IDNT.KEY

where IDNT is a unique, four-digit prefix, and the classifier KEY is chosen among the following: ASS_FUN (functional), ASS_PRF (performance), ASS_SEC (security), ASS_USB (usability), ASS_ONF (other non-functional). No two assumptions can have the same prefix. Although such prefix is sufficient to uniquely identify an assumption, the full name MUST always be used (in order to easily distinguish among assumptions of different categories and to recognize an assumption identifier as such).

1.3.4 Relationship among assumptions

Relationships among assumptions are described using the following keywords: REQUIRES X (to indicate that the assumption has X as a pre-assumption), CONFLICTS X (to indicate that the assumption is in conflict with assumption X), SIMILAR X (to indicate that the assumption expresses a similar assumption as X), RELATED X (to indicate that the assumption is related in some other way to X; use sparingly, and only if none of the previous apply).

1.3.5 Assumption priority

Prioritising assumptions: use the keywords HIGH, MEDIUM, LOW. HIGH means that this assumption is critical to the success of ASSERT4SOA; MEDIUM means that the assumption is not critical although it is highly desirable in order to demonstrate the merit of ASSERT4SOA to its full extent; LOW indicates that the assumption is relative to an additional feature or feature improvement that is desirable although not fundamental.

1.3.6 Assumption format

IDNT.KEY
Description: *A one-sentence description of the assumption.*
Event/use case: *Reference to a use case or event (use case step).*
Rationale: *Explain why this assumption is needed.*
Source: *Name of partner(s) who introduced this assumption.*
Dependencies: *A list of other assumptions that have some relationships with this one, especially pre-assumptions, conflicts, similarity.*
Priority: *Priority of this assumption.*

1.3.7 Requirement identification and classification

The naming convention for requirements follows the template below:

IDNT.KEY

where IDNT is a unique, four-digit prefix, and the classifier KEY is chosen among the following: FUN (functional), PRF (performance), SEC (security), USB (usability), ONF (other non-functional). No two requirements can have the same prefix. Although such prefix is sufficient to uniquely identify a requirement, the full name MUST always be used (in order to easily distinguish among requirements of different categories and to recognize a requirement identifier as such).

As a non-normative guideline, it is advised to use numbers from 5000 and above for non-functional requirements.

1.3.8 Relationship among requirements

Relationships among requirements are described using the following keywords: REQUIRES X (to indicate that the requisite has X as a prerequisite), CONFLICTS

X (to indicate that the requisite is in conflict with requisite X), SIMILAR X (to indicate that the requisite expresses a similar constraint as X), RELATED X (to indicate that the requisite is related in some other way to X; use sparingly, and only if none of the previous apply).

NOTE: The CONFLICTS and SIMILAR keywords were only applicable to intermediate versions of this document. Before the delivery, all CONFLICTS have been resolved, and all SIMILAR requirements have been compared and merged or rephrased in order to address specific (non-overlapping) aspects.

1.3.9 Requirement priority

Prioritising requirements: use the keywords HIGH, MEDIUM, LOW. HIGH means that this requirement is critical to the success of ASSERT4SOA; MEDIUM means that the requirement is not critical although it is highly desirable in order to demonstrate the merit of ASSERT4SOA to its full extent; LOW indicates that the requirement represents an additional feature or feature improvement that is desirable although not fundamental.
As mentioned earlier, it is worth noting that we took a pragmatic approach in the requirement elicitation process, trying to emphasize the requirements that are specific to the goals of the ASSERT4SOA framework. A number of requirements that are generally applicable to any service-oriented (or even every distributed) system could have been included, but this would have unnecessarily cluttered the document, obfuscating the content that is essential for the project.

1.3.10 Requirement format

IDNT.KEY
Description: *A one-sentence description of the requirement.*
Event/use case: *Reference to a use case or event (use case step).*
Rationale: *Explain why this requirement is needed.*
Source: *Name of partner(s) who introduced this requirement.*
Dependencies: *A list of other requirements/assumptions that have some relationships with this one, especially pre-requisites, conflicts, similarity.*
Priority: *Priority of this requirement.*

Chapter 2

Reference scenarios

2.1 Introduction

This chapter presents a set of scenarios where the functionalities and the mechanisms of ASSERT4SOA are used in different contexts.

2.2 ASSERT-enabled service marketplace

We consider a scenario where a client may access a web-based user interface to browse a catalog of applications. As an example, we may assume the client needs to find a secure storage service. The secure storage service must provide the typical file storage functionalities: it must allow users to remotely store, modify, retrieve, delete files, and browse folder directories. The service must also allow for a special administrator role, which can be used, e.g., to audit the history of actions/events, in order to guarantee accountability.

The client uses the ASSERT4SOA framework, through the web-based interface, to locate a service that matches his/her functional needs, as well as his requirements about security assurance. Several candidates exist on the market; some of them offer functionalities that match the client's request; only a few of them can also satisfy the client's requested security properties.

The scenario includes several players:

1. *the service provider* providing the secure storage service to its customers;
2. *client*: using the ASSERT4SOA framework to locate the secure storage service, and then consuming such service.
3. *the ASSERT4SOA framework,* a service itself, to which clients can express their needs (functional and non-functional) as a query, and they can obtain results (i.e., a ranked list of services that match the requirements as expressed in the query).

In response to the client query, ASSERT4SOA framework returns a set of results, sorting them in decreasing order of fit w.r.t. to the query. The client is presented the list of matching services displayed in a GUI, whereby he/she can pick the one candidate that he/she prefers (not necessarily the first of the list, meaning that he/she could apply some other criteria, in addition to those expressed in the query).

The system, periodically, sends updates to the GUI displaying the results, providing a new set of results that reflects changes in the available candidate services (e.g., when a new service is published, or when the characteristics of a formerly available service change).

The client may not specify explicitly all his/her preferences; instead, these could be loaded from a client-side preference store (so that recurring choices about security properties are reused across queries).

Use Cases used in this scenario:

- UC_Authentication
- UC_ReactiveDynamicServiceDiscovery
- UC_ProactiveDynamicServiceDiscovery

2.3 Proactive dynamic discovery

An online Investment Brokering Service (IBS) offers its clients the capability of making investment decisions and executing relevant transactions, including the creation and management of stock portfolios and the purchase and selling of shares through them, online. To provide these capabilities, IBS uses a number of third party services including:

- a (portfolio) *data storage and maintenance service*,
- services providing of different types of financial information related to investment decisions and are available on an enquiry mode, live data stream mode or combinations of the two (*financial information services*), and
- services enabling the execution of stock purchase and selling requests through the systems of different stock exchanges that are accessible through IDS (*stock exchange services*).

Key requirements of IBS, with regards to the external services that it deploys, include the high availability and integrity of the financial information services. The high availability of such services is important since the customers of IDS need continual access to financial information on a 365/24/7 basis in order to be able to make investment decisions. They also need to be certain that the information available to them through such services is accurate in the sense that what they receive through IBS is exactly the same information that was produced by the source service that IBS is using and has not been compromised by some external agent either at source or along the communication line. Further requirements of the IBS that are important for its customers relate to the confidentiality of the customer portfolio data stored in the external storage services that IDS uses and the integrity and confidentiality of the stock purchase and selling information that it provides to its associated stock exchange services.

The dependence of IBS on a number of external services has made it necessary to incorporate mechanisms for its dynamic adaptation in case that some of these services become unavailable or fail to deliver according to IBS's requirements. Particular consideration has, for example, been given to the need to adapt quickly to changes in the financial information services. Such changes may relate to delays in live information streams (e.g. at peak times) or even total service unavailability at critical trading hours. When conditions like these arise, IBS designers would like to have a means for discovering alternative financial information services that satisfy not only a set of functional and interface requirements enabling their dynamic binding into IBS but also key security conditions (e.g., certified service identity, certified integrity of provided live data streams etc).

The ASSERT4SOA framework will support the adaptation of IBS in cases like these by offering the designers of IBS the possibility of specifying queries for discovering financial information services and using the framework at runtime to execute these queries in order to discover suitable services that could be bound into IBS. Such queries will typically include functional, interface, quality and security conditions that financial services should meet in order to be suitable for IBS. Furthermore such queries should be executable in reactive or proactive mode.

In reactive mode, IBS will send the relevant to query to the ASSERT4SOA framework when one or more of its financial information services becomes unavailable or fails to satisfy required operational conditions. In response the ASSERT4SOA framework will return alternative financial information services or even compositions of services constructed on-fly if no single service meeting the required conditions is available. Consider for example, a case where a financial information service used by IBS for getting live news streams about initial public offerings (IPOs) in the UK is lost. The search for alternative services to it could identify candidate services that can offer exactly the same information or services that offer this information but cover IPOs across the entire European Union. In the former case, the result of the discovery process would be the single alternative IPO service itself whilst in the latter a composition of the located IPO services and a filtering service maintaining only UK IPOs would need to be constructed and be made available to the IBS.

Typically, the execution of service discovery in reactive mode would take time that could not guarantee an "almost real-time" response to IBS. Hence, depending on the criticality of the required service, it might be necessary to execute the discovery process in a proactive mode, i.e., executing it continually and in parallel with the operation of IBS even when there is no problem with the services that it deploys in order to discover alternative services for them and have them available when a need arises. The ASSERT4SOA framework will support the latter mode of execution of the discovery process by enabling applications like IBS to subscribe to it queries that will be continually executed and maintain records of the discovered services for future use.

Use Cases used in this scenario:

- UC_ReactiveDynamicServiceDiscovery
- UC_ProactiveDynamicServiceDiscovery
- UC_ProactiveCheckForServiceRegistryUpdates

- UC_ProactiveBufferFilling

Chapter 3

Use cases

3.1 Introduction to use cases

This chapter presents the reference use cases for the ASSERT4SOA framework:

- UC_ReactiveDynamicServiceDiscovery
- UC_ProactiveDynamicServiceDiscovery
- UC_ProactiveCheckForServiceRegistryUpdates
- UC_ProactiveBufferFilling
- UC_Authentication
- UC_Administration
- UC_ASSERTManagement

The first four use cases focus on the main functionality of the ASSERT4SOA framework, which is service discovery with extended discovery criteria. The fifth use case covers the authentication of the framework users[2], while the sixth use case covers the administration of the ASSERT4SOA framework. The last use case covers the ASSERT lifecycle management operations executed by ASSERT issuers via the ASSERT4SOA framework.

For the considered use cases, the reference actors are:

- Service-based application (SBA)[3]

[2] The framework users are: Service-based application (SBA), ASSERT issuer, framework Administrator.

[3] In this document we assume that the discovery functionalities offered by the ASSERT4Soa framework will be consumed by a service-based application (SBA), through a programmatic interface. Of course, this does not exclude scenarios where humans (as opposed to programs) query the framework; these scenarios however can still be described in terms of interactions between a SBA providing the interface through which human actor supply queries and consume results, and the ASSERT4Soa framework.

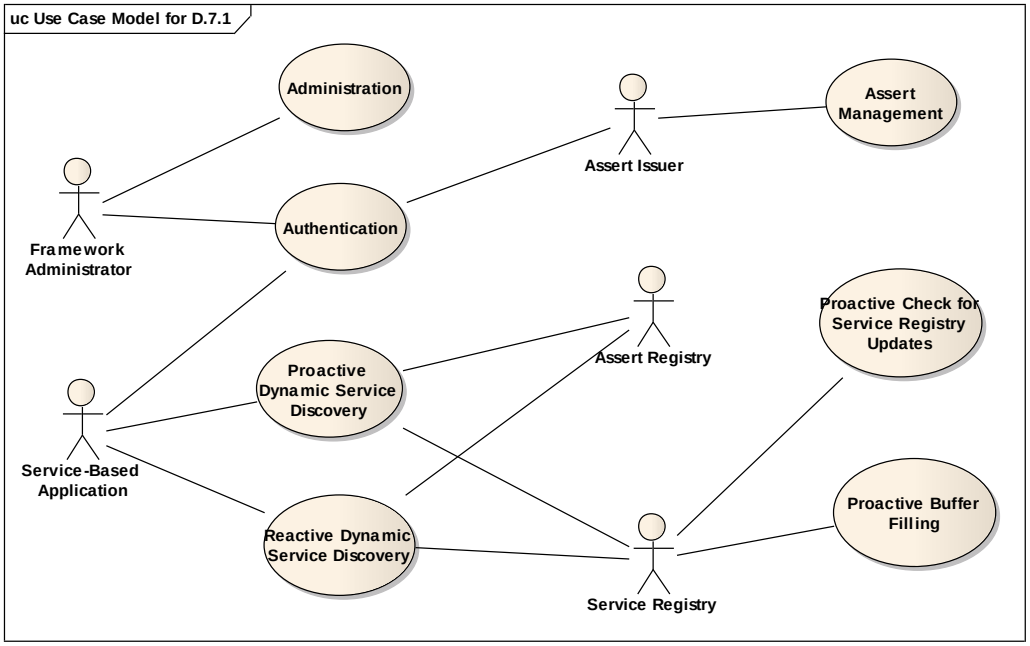

Figure 1. Use cases

- ASSERT Issuer
- Framework Administrator
- Service Registry
- ASSERT Registry

A first view of the use cases along with the corresponding actors is given in the use case model picture. SBA is the client of the service discovery with extended discovery criteria. The ASSERT Issuer is the manager of the ASSERT lifecycle (including issuing and revocation). The framework Administrator is the trusted entity in charge of the execution of administration functions. The Service Registry is the reference service repository accessed by the ASSERT4SOA framework during service discovery. The ASSERT Registry is the reference ASSERT repository. The ASSERT Registry is populated, via the ASSERT4SOA framework, by the ASSERT Issuer with the results of the ASSERT lifecycle management operations and accessed by the ASSERT4SOA framework during service discovery.

The Service Registry and the ASSERT Registry are assumed to be available to the ASSERT4SOA framework as external functionalities.

For the actual ASSERT lifecycle management operations (including issuing and revocation) the ASSERT Issuer depends on input and functionalities external to the ASSERT4SOA framework. For the above operation, the ASSERT4SOA framework supports the ASSERT Issuer for the generation of data to be stored in the ASSERT Registry.

3.2 Use case: Reactive Dynamic Service Discovery

UC_ReactiveDynamicServiceDiscovery

Description: This use case describes how an SBA uses the ASSERT4SOA framework in order to retrieve a list of services matching its needs in a reactive mode.

Actors: SBA, Service Registry, ASSERT Registry.

Preconditions: An ASSERT4SOA framework interface is available to the SBA, a Service Registry and an ASSERT Registry are available to the ASSERT4SOA framework. A service discovery query describing the criteria that should be met by potential candidate services has been specified.

Trigger: The SBA sends a Service Discovery Query to the ASSERT4SOA framework through the appropriate interface.

Basic flow:

1. The SBA sends a Service Discovery Query to the ASSERT4SOA framework (See step A in the picture)
2. The ASSERT4SOA framework receives the query and syntactically checks it
3. The ASSERT4SOA framework sends a response like "request under processing" (See step A' in the picture).
4. The ASSERT4SOA framework formats and sends a registry query (See step C in the picture)
5. The ASSERT4SOA framework receives a response from the service registry containing a set of services matching the criteria specified in the registry query (See step D in the picture). For each service entry in the response, the service's ASSERTs are retrieved (see steps E and F in the picture) and verified if necessary according to the registry query.
6. The ASSERT4SOA framework computes the degree of fit of the candidate services.
7. The ASSERT4SOA framework sends a response to the SBA containing a list of services matching the discovery criteria of the Service Discovery Query, sorted by the sorting criteria specified in this query (See step B in the picture).

Alternate Flows:

AF1: If in step 2 of the Basic Flow the syntactical check does not pass, the ASSERT4SOA framework returns an error message to the SBA.

AF2: If in step 5, no single service satisfying the registry query is found and the service discovery query allows for service composition:

5.1 The ASSERT4SOA framework retrieves patterns of service composition that are known to satisfy the security discovery criteria of the discovery query and uses them to create alternative queries for discovering single services that could fit into the patterns (the alternative queries incorporate security criteria as required by the identified patterns in order for the compositions built by them to satisfy the security criteria of the original query).

5.2 The ASSERT4SOA framework executes the new queries and uses their results to build up compositions of services that satisfy the criteria in the query. Each composition is returned as a candidate composite service

AF3: If in step 5.1, no composition patterns exist, the framework informs SBA and the use case terminates.

AF4: If at step 7, there are no services having the minimum required degree of fit with the discovery query, the framework should inform the SBA about the closest matching services OR just inform that no services match the security properties (according to the preferences express as part of the query).

Postconditions:

PC1: In case of Basic Flow, the ASSERT4SOA framework returns a list of services matching SBA discovery criteria and the audit trail is updated.

PC2: In case of alternate flow AF1, the ASSERT4SOA framework returns an error message to the SBA and the audit trail is updated.

PC3: In case of alternate flow AF2, the ASSERT4SOA framework returns pointers to on-fly composed service compositions that can be invoked by the SBA.

PC4: In case of alternate flow AF3, the ASSERT4SOA framework returns to SBA an indication that no single services or service compositions could be found.

Dependencies: None.

3.3 Use case: Proactive Dynamic Service Discovery

UC_ProactiveDynamicServiceDiscovery

Description: This use case describes how an SBA uses the ASSERT4SOA framework in order to retrieve a list of services matching its needs in a proactive mode.

Actors: SBA, Service Registry, ASSERT Registry.

Preconditions:

An ASSERT4SOA framework interface is available to the SBA

A Service Registry and an ASSERT Registry are available to the ASSERT4SOA framework.

Trigger: The SBA sends a Service Discovery Query to the ASSERT4SOA framework through the appropriate interface.

Basic flow:

1. SBA subscribes to the ASSERT4SOA framework a query to be proactively executed for discovering alternative services for one of its partner services.
2. The ASSERT4SOA framework receives the query and syntactically checks it
3. The ASSERT4SOA framework sends a notification that the service discovery query has been successfully subscribed to the list of queries that will be proactively executed on behalf of the particular SBA.
4. The ASSERT4SOA framework formats and sends a registry query (See step C in the picture)
5. The ASSERT4SOA framework receives a response from the service registry containing a set of services matching the criteria specified in the registry query (See step D in the picture). For each service entry in the response, the service's ASSERTs are retrieved (see steps E and F in the picture) and verified if necessary according to the registry query.
6. The ASSERT4SOA framework computes the degree of fit of the candidate services that have been received from the registry with the discovery criteria.

7. The ASSERT4SOA framework sorts the candidate services (whether composite or not) according to the sorting criteria specified in the service discovery query.
8. The ASSERT4SOA framework selects the best N of the ranked services and builds a service buffer for the particular query containing them.
9. If SBA requests a service discovered by the specific query, the ASSERT4SOA framework returns the first service in the buffer built for the query

Alternate Flows:

AF1: If in step 2 of the Basic Flow the syntactical check does not pass, the ASSERT4SOA framework returns an error message to the SBA

AF2: If in step 5, no single service satisfying the registry query is found and the service discovery query allows for service composition

5.1 The ASSERT4SOA framework retrieves patterns of service composition that are known to satisfy the security discovery criteria of the discovery query and uses them to create alternative queries for discovering single services that could fit into the patterns (the alternative queries incorporate security criteria as required by the identified patterns in order for the compositions built by them to satisfy the security criteria of the original query).

5.2 The ASSERT4SOA framework executes the new queries and uses their results to build up compositions of services that satisfy the criteria in the query. Each composition is returned as a candidate composite service.

AF3: If in step 5.1, no composition patterns exist, the framework sends a notification to the SBA that an initial attempt to build a buffer for the query has failed.

Postconditions:

PC1: In the case where the basic flow has been executed, the ASSERT4SOA framework has built a buffer with candidate services for the query.

PC2: In case of alternate flow AF3, the ASSERT4SOA framework returns to SBA an indication that that an initial attempt to build a buffer for the query has failed.

Dependencies: None.

3.4 Use case: Proactive Check for Service Registry Updates

UC_ProactiveCheckForServiceRegistryUpdates

Description: This use case describes how the ASSERT4SOA framework checks for service registry updates to ensure that the buffer services built for proactive dynamic service discovery are maintained in an up-to-date state.

Actors: Service Registry

Preconditions:

At least one valid service discovery query has been submitted by an SBA for execution in proactive mode.

Trigger: A periodic event signifying the need to check external registries has occurred.

Basic flow:

1. The ASSERT4SOA framework checks external service registries for new services

2. For each new service that is found
   2.1. For each subscribed buffer query
      2.1.1. If the new service satisfies the query, it is ranked against the service discovery criteria of the query and if it is better than any of the services already in the buffer, the new service is inserted at the appropriate position into the buffer

Alternate Flows: None.
Postconditions:
PC1: The buffers maintained for the subscribed service discovery queries are updated as needed by service registry updates.
Dependencies: None.

3.5 Use case: Proactive Buffer Filling

UC_ProactiveBufferFilling
Description: This use case describes how the ASSERT4SOA framework executes service discovery queries proactively to ensure that service buffers have the required minimum number of services in them.
Actors: Service Registry
Preconditions: At least one valid service discovery query has been submitted by an SBA for execution in proactive mode.
Trigger: A size of a service discovery query buffer falls below the minimum threshold specified in the query.
Basic flow:
1. The ASSERT4SOA framework executes the use case UC_ProactiveCheckForServiceDiscoveryUpdates for the buffer whose size has fallen below the minimum size.

Alternate Flows: None.
Postconditions: see UC_ProactiveCheckForServiceDiscoveryUpdates.
Dependencies: UC_ProactiveCheckForServiceDiscoveryUpdates.

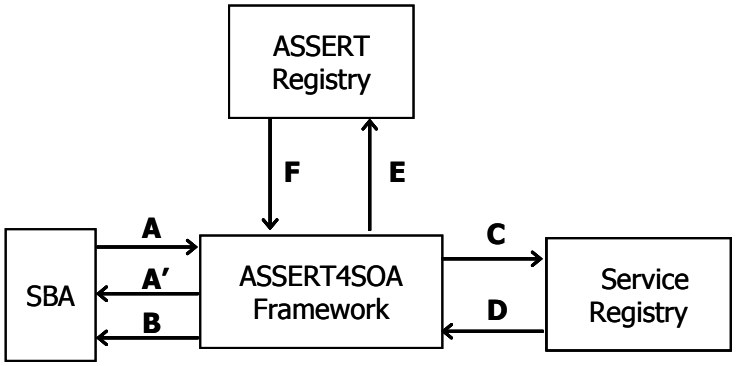

Figure 2. Service discovery: interactions of the framework with the actors.

3.6 Use case: Authentication

UC_Authentication

Description: This use case describes how a user is authenticated by the ASSERT4SOA framework.
Actors: SBA, ASSERT Issuer, Framework Administrator.
Preconditions: An ASSERT4SOA framework interface/frontend is available to the user
Trigger: The user connects to the ASSERT4SOA framework interface.
Basic flow:
1. The user sends its authentication data to the ASSERT4SOA framework
2. The ASSERT4SOA framework checks the authentication data
3. The ASSERT4SOA framework sends a positive response to the authenticated user

Alternate Flows:
AF1: If in step 2 the authentication check has negative result, the ASSERT4SOA framework returns an error message.
Postconditions:
PC1: After the authentication process, both in basic and alternate flow, the audit trail is updated.
Dependencies: None.

3.7 Use case: Administration

UC_Administration

Description: This use case describes how an authenticated framework administrator interacts with the ASSERT4SOA framework for executing administration functions
Actors: Framework administrator.
Preconditions: The framework administrator is authenticated to the ASSERT4SOA framework (as per UC_Authentication).
Trigger: A framework administrator needs to execute some administration functions.
Basic flow:
1. The ASSERT4SOA framework shows the set of functions the framework administrator is authorized to execute.
2. The framework administrator selects and executes the needed functions.
3. The ASSERT4SOA framework shows a trace of the executed functions.

Alternate Flows: None.
Postconditions:
PC1: The audit trail is updated.
Dependencies: UC_Authentication.

3.8 Use case: ASSERT Management

UC_ASSERTManagement

Description: This use case describes how an authenticated ASSERT issuer interacts with the ASSERT4SOA framework for managing the ASSERT lifecycle (issuing, revoking) and for updating the ASSERT Registry.
Actors: ASSERT Issuer, ASSERT Registry.
Preconditions: The ASSERT issuer is authenticated to the ASSERT4SOA framework (as per UC_Authentication).
Trigger: An ASSERT issuer needs to update the ASSERT Registry with the results of some ASSERT lifecycle management operation.
Basic flow:

1. The ASSERT4SOA framework shows the set of operations the ASSERT issuer is authorized to perform.
2. The ASSERT4SOA framework updates the ASSERT Registry according to the ASSERT issuer request (registration of new ASSERT, revocation of an existing ASSERT).

Alternate Flows:

AF1: If in step 2 the ASSERT Registry is not available, the ASSERT4SOA framework returns an error message.

Postconditions:
PC1: The audit trail is updated in both basic and alternate flow
Dependencies: UC_Authentication.

Chapter 4

Assumptions

4.1 Assumptions about external functionalities

8001.ASS_FUN
Description: The ASSERT4SOA Framework can access Service Registries.
Event/use case: All use cases related to discovery.
Rationale: Needed for the search for services meeting the service discovery query.
Dependencies: None.
Priority: HIGH.

8002.ASS_FUN
Description: Under a registry query, a service registry responds either with a null response (no registered service meets the registry query) or with the list of the services meeting the registry query, where for each service the list of the associated ASSERTs is also provided.
Event/use case: All use cases related to discovery.
Rationale: The basic ASSERT4SOA framework operation is the search for services meeting the service discovery query.
Dependencies: REQUIRES: 8001.ASS_FUN.
Priority: HIGH.

8003.ASS_SEC
Description: Service registries are operated by identified entities who are responsible for the registry contents and operation.
Event/use case: All use cases related to discovery.
Rationale: Role/responsibility separation between ASSERT4SOA framework and service registry.
Dependencies: REQUIRES: 8001.ASS_FUN.
Priority: HIGH.

8004.ASS_FUN

Description: The ASSERT4SOA Framework can access ASSERT Registries.
Event/use case: UC_ASSERTManagement, all use cases related to discovery.
Rationale: Needed for the search for services meeting the service discovery query and for the notification of the ASSERT lifecycle management actions.
Dependencies: None.
Priority: HIGH.

8005.ASS_SEC

Description: ASSERT registries are operated by identified entities, who are responsible for the registry contents and operation.
Event/use case: UC_ASSERTManagement, all use cases related to discovery.
Rationale: Role/responsibility separation between ASSERT4SOA framework and ASSERT registry.
Dependencies: REQUIRES: 8004.ASS_FUN.
Priority: HIGH.

8006.ASS_SEC

Description: The results of the actions related to the ASSERT lifecycle management (including issuing and revocation), performed by the ASSERT Issuer, are stored in the ASSERT registry.
Event/use case: UC_ASSERTManagement.
Rationale: The ASSERT registry must be considered as the authoritative repository of ASSERTS.
Dependencies: REQUIRES: 8004.ASS_FUN.
Priority: HIGH.

8007.ASS_SEC

Description: The ASSERT registry updates executed by an ASSERT Issuer include data allowing an automatic verification of integrity and authenticity of origin of the single update.
Event/use case: UC_ASSERTManagement.
Rationale: The ASSERT registry must be considered as the authoritative repository of ASSERTS.
Dependencies: REQUIRES: 8006.ASS_SEC.
Priority: HIGH.

4.2 Assumptions about internal roles

8008.ASS_SEC
Description: The framework Administrator is not careless, wilfully negligent or hostile.
Event/use case: UC_Administration.
Rationale: Needed to support the correct operation, configuration and maintenance of the ASSERT4SOA framework.
Dependencies: None.
Priority: HIGH.

Chapter 5

Functional requirements

4001.FUN
Description: The ASSERT4SOA framework MUST provide support for multiple types of ASSERTs corresponding to different types of evidences available at issuing time.
Event/use case: UC_ASSERTManagement, all use cases related to discovery.
Rationale: To offer coverage of the different types of ASSERTs defined.
Dependencies: REQUIRES: 4003.FUN.
Priority: HIGH.

4002.FUN
Description: The ASSERT4SOA framework MUST provide support for certificates issued according to existing security certification criteria.
Event/use case: UC_ASSERTManagement, all use cases related to discovery.
Rationale: To offer coverage of existing certificates about security properties of services.
Dependencies: None.
Priority: HIGH.

4003.FUN
Description: The ASSERT language MUST support the expression of the following (minimal) contents of an ASSERT: Unique identifier of the ASSERT itself; Unique identifier of the ASSERT issuer issuing the ASSERT; the type of the ASSERT; binding to the ASSERTed Service; list of security properties certified by the ASSERT; Validity period of the ASSERT.
Event/use case: UC_ASSERTManagement, all use cases related to discovery.
Rationale: Support to the automatic verification of the validity of an ASSERT.
Dependencies: None.
Priority: HIGH.

4004.FUN
Description: ASSERTs MUST include data to automatically verify their integrity and their authenticity of origin.
Event/use case: All use cases related to discovery.
Rationale: Both malicious and non-malicious actions leading to release a bogus ASSERT within an ASSERT4SOA framework have to be considered.
Dependencies: None.
Priority: HIGH.

4005.FUN
Description: The ASSERT4SOA framework MUST verify the binding between an ASSERT and the service it refers to.
Event/use case: All use cases related to discovery.
Rationale: To support the trustworthiness of the response provided by the ASSERT4SOA framework to service discovery queries.
Dependencies: REQUIRES: 4003.FUN.
Priority: HIGH.

4006.FUN
Description: The ASSERT4SOA framework SHOULD support the verification of credentials provided by ASSERT issuers about their competence.
Event/use case: UC_ASSERTManagement, all use cases related to discovery.
Rationale: This would support the SBA and the framework itself in establishing the trustworthiness of the ASSERT issuer (e.g., to distinguish an issuer accredited by a suitable accreditation scheme).
Dependencies: None.
Priority: MEDIUM.

4007.FUN
Description: The ASSERT4SOA framework MUST include automated tools (or programmatic interfaces) for checking the revocation status of an ASSERT.
Event/use case: All use cases related to discovery.
Rationale: The check of the validity of an ASSERT must be available as an operation that can be included in an automated process.
Dependencies: REQUIRES: 8007.ASS_SEC.
Priority: HIGH.

4008.FUN
Description: In response to a service discovery query, the ASSERT4SOA framework MUST include a service in the results if and only if the following conditions hold for each ASSERT corresponding to that service:
- The time of check is included in the validity period specified in the ASSERT.
- The ASSERT signature is valid and the issuer is trusted.
- The identity of the ASSERT issuer is verified according to a suitable uniform resource identification scheme.
- The service binding specified in the ASSERT is verified.
- The properties specified in the ASSERT and the way these have been assessed satisfy the service discovery query.

Event/use case: All use cases related to discovery.

Rationale: All the specified conditions support the reliability of the response provided to SBA.
Dependencies: REQUIRES: 4003.FUN, 4004.FUN.
Priority: HIGH.

4009.FUN
Description: Service discovery MUST be based on a query language supporting the expression of conditions about the required structure, behaviour, quality and security properties of services.
Event/use case: All use cases related to discovery.
Rationale: Service Discovery must be based on a comprehensive query language enabling the expression of all the different criteria for service discovery that might be required for locating services at runtime.
Dependencies: None.
Priority: HIGH.

4010.FUN
Description: The query language for service discovery MUST allow the specification of the relative significance of the different conditions used in queries.
Event/use case: All use cases related to discovery.
Rationale: The relative significance of querying criteria enables service based application owners to tailor the discovery process to the exact needs of their applications and establishes a scheme for selecting amongst identified services.
Dependencies: REQUIRES: 4009.FUN.
Priority: HIGH.

4011.FUN
Description: The query language for service discovery MUST allow the distinction between confidential and non-confidential discovery criteria.
Event/use case: All use cases related to discovery.
Rationale: Querying criteria can reveal the internal structure of a service based applications in ways that make them vulnerable. Since the discovery infrastructure might be realised by including functionalities offered by third parties, the classification of some querying criteria as confidential dictates their execution by parties that are trusted.
Dependencies: REQUIRES: 4009.FUN.
Priority: MEDIUM.

4012.FUN
Description: The query language for service discovery MUST allow the specification of conditions in queries indicating whether the result of the discovery process should include only atomic services, compositions of services, or both.
Event/use case: All use cases related to discovery.
Rationale: In some cases there might be no atomic services satisfying a query and hence attempting to find a composition of services matching it might be required. However, compositions of services established at runtime may be less robust than atomic services and hence the owners of service based applications should be given a means to indicate whether they wish to bind dynamically formed service compositions into their systems.

Dependencies: REQUIRES: 4009.FUN.
Priority: MEDIUM.

4013.FUN
Description: The query language for service discovery SHOULD allow the specification of conditions regarding the formation of service compositions from single services.
Event/use case: All use cases related to discovery.
Rationale: Different environments/clients could impose different requirements as to the level of security of the build up of service compositions on-the-fly. These requirements will be formulated as part of the query, in such a way that the framework can use them to guide the composition process.
Dependencies: REQUIRES: 4009.FUN, 4012.FUN.
Priority: LOW.

4014.FUN
Description: The query language for service discovery MUST allow the specification of the parts of service descriptions that the answer to a query should contain.
Event/use case: All use cases related to discovery.
Rationale: Service based applications may need to filter further the results of service discovery based on the identified information for services.
Dependencies: REQUIRES: 4009.FUN.
Priority: MEDIUM.

4015.FUN
Description: The query language for service discovery SHOULD allow the specification of the mode of the notification of query results distinguishing between at least two cases**:** (a) the case where a query is executed once and its result is returned immediately, and (b) the case where notifications of services matching the query are produced when new (or amended) services matching the query arise.
Event/use case: All use cases related to discovery.
Rationale: Service-based applications may require to be notified when (1) new services matching a query are found, or (2) when previously matching services cease to match the query (because their characteristics have changed, ore because they are no longer available), or (3) when new matches are found.
Dependencies: REQUIRES: 4009.FUN.
Priority: MEDIUM.

4016.FUN
Description: Replies to service discovery queries MUST include the set of descriptive elements of the services matching the query specified in the query itself.
Event/use case: All use cases related to discovery
Rationale: Owners of service-based applications should be able to retrieve the descriptive elements of services matching their needs.
Dependencies: REQUIRES: 4013.FUN.
Priority: MEDIUM.

4017.FUN
Description: Service Discovery MUST produce replies to queries including the invocation endpoints of the services matching them.
Event/use case: All use cases related to discovery.
Rationale: Service invocation endpoints are necessary for service binding.
Dependencies: NA.
Priority: HIGH.

4018.FUN
Description: Service Discovery SHOULD return compositions of services matching a query, if compositions are required or allowed by the query.
Event/use case: All use cases related to discovery.
Rationale: In some cases there might be no atomic services satisfying a query and hence attempting to find a composition of services matching it might be required. However, compositions of services established at runtime may be less robust than atomic services and hence the owners of service based applications should be given a means to indicate whether they wish to bind dynamically formed service compositions into their systems.
Dependencies: REQUIRES: 4012.FUN.
Priority: HIGH.

4019.FUN
Description: The choice between single service and composite services SHOULD be transparent to the SBA, unless otherwise specified in the query.
Event/use case: All use cases related to discovery.
Rationale: An application that requires a service substitution may want only the binding for the new service and it may not be able to compose/understand composition.
Dependencies: REQUIRES: 4018.FUN.
Priority: LOW.

4020.FUN
Description: Service Discovery MAY create (if possible given the certificates of individual services) a *virtual* on-fly certificate to express the properties of a composition and to show that such a composition matches the query.
Event/use case: All use cases related to discovery.
Rationale: An application that requires a service substitution with some security properties may need evidence regarding the satisfaction of the given security properties by service composition.
Dependencies: REQUIRES: 4013.FUN, 4018.FUN.
Priority: MEDIUM.

4021.FUN
Description: The query language MUST support the expression of conditions regarding all the different parts of an ASSERT.
Event/use case: All use cases related to discovery.
Rationale: Different parts of service certificates (e.g. certificate type, producer, underlying evidence, status) may be important for the assessment of the certificate value.
Dependencies: REQUIRES: 4009.FUN, 4010.FUN.
Priority: HIGH.

4022.FUN
Description: Service Discovery MUST produce ranked (partially-ordered) sets of candidate services and/or service compositions as indicated by the query.
Event/use case: All use cases related to discovery.
Rationale: Enables service selection.
Dependencies: REQUIRES: 4009.FUN, 4010.FUN.
Priority: HIGH.

4023.FUN
Description: Service Discovery SHOULD produce notifications about new service-query matches, if necessary according to the query.
Event/use case: All use cases related to discovery.
Rationale: To adapt to the availability of the services. See also 4015.FUN.
Dependencies: REQUIRES: 4015.FUN.
Priority: HIGH.

4024.FUN
Description: In reactive service discovery, it should be possible to modify the discovery queries for a service based application whilst the application is in operation.
Event/use case: All use cases related to discovery.
Rationale: To adapt to the availability of the services.
Dependencies: NA.
Priority: LOW.

4025.FUN
Description: If specified in the service discovery query, the corresponding response provided by the ASSERT4SOA framework MUST include data allowing for an automatic verification of both the integrity of contents and the authenticity of origin of the response itself.
Event/use case: All use case related to discovery.
Rationale: Both malicious and non-malicious actions leading to release a bogus ASSERT4SOA framework response have to be counteracted.
Dependencies: None.
Priority: HIGH.

4026.FUN
Description: The ASSERT4SOA framework SHALL allow an authenticated framework Administrator to perform administrative functions, including the following ones:
- maintain security attributes of ASSERT4SOA framework users
- configure the operations allowed to authenticated ASSERT issuers (*ASSERT management functions*)
- configure which operations are allowed to non-authenticated SBAs (*open SBA functions*)
- configure which operations are allowed only to authenticated SBAs (*closed SBA functions*)
- configure the type of events to be audited
- access audit data
- maintain lists of service registries and ASSERT registries
- monitor system performance and availability.

Event/use case: UC_Administration.
Rationale: The ASSERT4SOA framework should be configurable and maintainable by an authenticated framework administrator.
Dependencies: REQUIRES: 6003.SEC.
Priority: HIGH.

4027.FUN
Description: The ASSERT4SOA framework SHALL allow an authenticated ASSERT issuer to perform ASSERT management functions, such as the storage in ASSERT registries of results of actions related to the ASSERT lifecycle management (including issuing and revocation).
Rationale: Support the ASSERT issuers in populating and updating the ASSERT registries.
Dependencies: REQUIRES: 6004.SEC.
Priority: HIGH.

4028.FUN
Description: The ASSERT4SOA framework SHALL allow an SBA to perform the operations such as the following (according to the configuration defined by the framework administrator):
- send service discovery query
- subscribe/unsubscribe to queries
- update queries

Event/use case: All use cases related to discovery.
Rationale: These are key functionalities of the framework.
Dependencies: REQUIRES 6005.SEC.
Priority: HIGH.

4029.FUN

Description: All security-relevant events MUST be detected and logged in a secure audit trail. The list of security-relevant events includes:

- any successful user attempt to authenticate as a framework administrator
- any action executed by an authenticated framework administrator
- any unsuccessful user attempt to authenticate as a framework administrator
- any successful user attempt to authenticate as an SBA
- any unsuccessful user attempt to authenticate as an SBA
- any request presented by an authenticated SBA
- any successful user attempt to authenticate as an ASSERT issuer
- any unsuccessful user attempt to authenticate as an ASSERT issuer
- any request presented by an authenticated ASSERT issuer
- any unsuccessful framework attempt to access service registries or ASSERT registries

Event/use case: All use cases.
Rationale: This is standard practice for detecting and contrasting attempts to unauthorised access/use of restricted functionalities of the framework. Additionally, the audit trail is the basic means to provide accountability.
Dependencies: REQUIRES: 6003.SEC, 6004.SEC, 6005.SEC, 4026.FUN.
Priority: HIGH.

4030.FUN

Description: All non-security-relevant actions (defined by the administrator) SHOULD be captured and made available so as to support runtime monitoring.
Event/use case: All use cases (potentially).
Rationale: The framework must be *monitorable* (e.g., to ensure correct operation, acceptable performance, etc.).
Dependencies: NA.
Priority: MEDIUM.

Chapter 6

Non-functional requirements

6.1 Security requirements

6001.SEC
Description: The confidentiality and integrity of communications between an SBA and the ASSERT4SOA framework must be assured.
Event/use case: All use cases related to discovery.
Rationale: All communications between clients and the ASSERT4SOA framework may contain sensitive information, and therefore they must be secured against malicious access and modification in order to ensure both confidentiality and integrity.
Dependencies: NA.
Priority: MEDIUM.

6002.SEC
Description: Service Discovery SHOULD guarantee the non-disclosure of the parts of the query that the client marked as confidential to functional element of the framework that are provided by (untrusted) third-parties.
Event/use case: All use cases related to discovery.
Rationale: Some functionalities of the framework might be provided by third-party services. Since clients may disclose sensitive information with the query, they may want to keep confidential some of its parts, by constraining their disclosure to third parties.
Dependencies: REQUIRES: 4011.FUN.
Priority: MEDIUM.

6003.SEC
Description: The ASSERT4SOA framework MUST allow the access to administration functions only to authenticated framework administrators.
Event/use case: UC_Authentication, UC_Administration.
Rationale: Malicious framework Administrators actions have to be contrasted.
Dependencies: None.
Priority: HIGH.

6004.SEC
Description: The ASSERT4SOA framework MUST allow the access to ASSERT management functions only to authenticated ASSERT issuers.
Event/use case: UC_Authentication, UC_ASSERTManagement.
Rationale: ASSERT management operations have to be protected.
Dependencies: None.
Priority: HIGH.

6005.SEC
Description: The ASSERT4SOA framework MUST allow the access to closed SBA functions only to authenticated SBAs.
Event/use case: UC_Authentication, all use cases related to service discovery.
Rationale: Closed SBA functions have to be protected. Although framework administrators may want to allow the SBAs to use the ASSERT4SOA framework without being authenticated first (i.e., as anonymous users), for the sake of flexibility, it must be possible to support scenarios where authentication is required.
Dependencies: None.
Priority: HIGH.

6006.SEC
Description: The Service Discovery process SHOULD be documented in such a way that it can be certified.
Event/use case: All use cases related to discovery.
Rationale: The Service Discovery operates on security issues, so it should guarantee that it operates in a secure way.
Dependencies: NA.
Priority: LOW.

6.2 Performance requirements

6007.PRF
Description: Service discovery must have “near real-time” performance.
Event/use case: UC_ReactiveDynamicServiceDiscovery.
Rationale: The presence of the ASSERT4SOA framework must be transparent to service clients, and in particular it must not introduce excessive delays in order to ensure the efficiency of SBAs.
Dependencies: NA.
Priority: MEDIUM.

6.3 Other technical requirements

6008.ONF
Description: The framework MUST interact with the backend registries through pluggable registry adapters; it MUST be possible to interface to additional registries just by developing a new dedicated adapter.
Event/use case: UC_ASSERTManagement, all use cases related to discovery.
Rationale: The operation of the system must be decoupled from concrete registry implementations, by implementing the framework as an open-ended system.
Dependencies: NA.
Priority: HIGH.

Bibliography

[1] Scott Bradner. Key words for use in RFCs to indicate requirement levels, <http://www.ietf.org/rfc/rfc2119.txt>, 1997.

[2] Software Engineering Standards Committee of the IEEE Computer Society. IEEE Recommended Practice for Software Requirements Specifications, IEEE Std 830-1998, 1998

[3] R. Pressman. Software Engineering: A Practitioner's Approach. McGraw-Hill, 4th edition, 1997.

Appendix A

Vocabulary

ASSERT4SOA certificate: see ASSERT.

ASSERT: A digitally signed, machine-readable document containing a list of statements about the security properties possessed by a service, together with the relevant details about the way these have been assessed.

ASSERT lifecycle: The specification of the different phases of the life of the ASSERT, including Issuing and Revoking.

ASSERT type: One of the following: *Evidence-based ASSERT (A.k.a. ASSERT-E*); *Model-based ASSERT (A.k.a. ASSERT-M)*; and *Ontology-based ASSERT (A.k.a. ASSERT-O).*

ASSERT registry: A repository of ASSERTs of software services.

Accredited ASSERT Issuer: The ASSERT issuer accredited according to defined criteria.

ASSERT issuer: The originator of an ASSERT.

ASSERT registration: The process of registering the ASSERT of a service in the ASSERT Registry

ASSERT Profiles: Mechanisms to describe certification schemes (e.g. CC EAL4, coverISO9001, SAS70, Common criteria). The goal is to allow the homogenous representation of specific certification schemes.

ASSERT Language: Language to capture the semantics of certification activities and results for services and service–oriented applications, in a machine-processable way. In particular, the language shall enable the comparison of certificates and the selection of services based on different aspects contained in the certificates. Comparison will not only be binary for equality, but will support richer relations and the establishing of partial orderings, which will happen according to some dimension of the certificate (issuer, properties, method of assessment, strength of evidences, etc.)

ASSERT certification process: In the context of the ASSERT language:

Identification, abstraction and expression of the aspects that characterize a certification process used certified the service or service–oriented application.

ASSERT certification results: In the context of the ASSERT language: Identification, abstraction and expression of the aspects that characterize the possible ASSERT certification results. The nature of these results is very heterogeneous. Therefore, means for establishing relations between different results are an essential part of this model. To better understand the nature of the relations we are considering here we can use the following example: a proven evidence (e.g., Service A has successfully passed a penetration testing) contributes to (increases the trust on the fulfilment of) a property (e.g., confidentiality of the data managed by Service A), therefore we need to consider these relations when designing the language.

Assumptions/Dependencies: In the context of the ASSERT language: Refer to the runtime-checkable conditions for the ASSERT to be considered valid.

ASSERT Ontology: The ontology supporting the description of ASSERTs (ASSERT-E, ASSERT-O and ASSERT-M) in the ASSERT4SOA Framework. It also describes a set of terms required to formalize an ASSERT-O.

Binding: Mechanisms to connect the ASSERT with the service, in order to ensure that the properties included in the ASSERT actually correspond to that specific service.

Behavioural discovery criteria: Discovery criteria regarding the behavioural properties required from a software service.

Candidate service: A service whose match with a given criterion is to be established; candidate services are the input to matchmakers.

Category of (security) properties: Grouping of properties referring to the same high-level security characteristic (e.g., confidentiality, integrity, authenticity,....).

Certificate discovery criteria: Discovery criteria regarding the different types of ASSERT certificates required from a software service. Such criteria depend on the type of the certificate and cover the different descriptive elements of the machine-readable representations of different types of certificates.

Dynamic service composition: A service composition process that is carried out whilst an SBA is in operation at runtime.

Dynamic service discovery: A service discovery process that is carried out whilst an SBA is in operation at runtime.

Discovery request: A request addressed to a Service Discovery system; the request is composed of a Query and optionally of a set of (stored) preferences.

Evidence-based ASSERT *(A.k.a. ASSERT-E)*: An ASSERT in which the

assessment of the properties is based on the execution of tests. This type of ASSERTS is used to capture claims like "the evaluation of the User Data Protection property of the ASSERTED Service was based on a mixture of manual and automated tests based on the ATE Test suite and the CC Tests. All tests ran successfully and produced the expected results".

Fit: In matchmaking, it is the **d**egree to which a candidate service matches a selection criterion.

Framework administrator: Human being whose role is to configure and manage the operation of the ASSERT4SOA framework.

Matching service: A service that matches a given selection criterion; matching services are the output of matchmakers.

Matchmaker: An element of the service discovery system that matches specific types of service discovery criteria within a service discovery query with descriptions of a given set of services, and produces a partial or full order of the services in the set based on their combined degree of match with the given set of criteria. A matchmaker may also produce individual matching scores indicating the level of fit of the individual services in the input set with the given discovery criteria.

Model-based ASSERT *(A.k.a. ASSERT-M)*: An ASSERT in which the assessment of the properties is based on the creation and analysis of a formal model. This type of ASSERTS is used to capture claims like "the evaluation of the Communications Security property of the ASSERTED Service was based on the development of a formal model using formalism X. Analyses based on model checking revealed no vulnerability using a Dolev-Yao attacker model".

Nature (of a statement): The type of assessment through which statement was obtained (e.g., "evidence-based nature", "model-based nature"). Note: properties do not have a nature, it is the way statements are grounded on an assessment procedure that gives them an evidence-based or a model-based nature.

Ontology-based ASSERT *(A.k.a. ASSERT-O)*: An ASSERT in which the claims about the properties are simply stated by the authority with the support of the ASSERRT Ontology. This type of ASSERTS is used to capture claims like "the User Data Protection property of the ASSERTED Service is backed by an insurance policy provided by X, and defined in <Link-to-ontology>".

Overall service matching score: An overall score indicating the degree of fit of a service to all the criteria expressed in a service discovery query.

Proactive service discovery: A dynamic service discovery process that is executed continually in order to identify ranked sets of services that satisfy different service discovery queries and could be used by an SBA if a need for them

arises. Proactive service discovery is based on discovery queries that are subscribed by SBAs and may be executed periodically at runtime and/or upon the occurrence of other kinds of triggering events (e.g., changes in service descriptions, changes in security certificates of services, emergence of new services). Proactive service discovery maintains a set of services that are known to satisfy a service discovery query and may be selected for binding upon request without any need to check the fit of the services with query itself at the time of the request.

Pre-selected services: A set of services that are known to satisfy a given service discovery query and have been selected by a proactive service discovery process.

Partial service matching score: A score indicating the degree of fit of a service to a subset of the criteria expressed in a service discovery query.

Quality-of-Service (QoS) discovery criteria: Discovery criteria regarding the quality properties required from a software service. They may also be referred to as "QoS discovery criteria" in an abbreviated form.

Query engine: The component of SDS that is responsible for the execution of service discovery queries.

Reactive service discovery: A dynamic service discovery process that is executed following an explicit request from an SBA at runtime when one or more services that are deployed by it fail to satisfy its needs. In reactive service discovery, following the identification and ranking of services that satisfy the given discovery criteria, the discovery process returns information that would enable the binding of the service that has the best fit with the criteria.

Registry query: An expression used to specify to a service registry a logical combination of Structural (or interface) discovery criteria, Behavioural discovery criteria, and Quality-of-Service (QoS) discovery criteria.

Service-based application (SBA): A software application that makes use of software services. A service-based application may also be referred to as "SBA" in an abbreviated form.

Service discovery: The process of identifying software services that satisfy different discovery criteria. Depending on the discovery criteria, this process may involve only the discovery of services as published in registries and/or the construction of compositions of such services on-fly.

Static service discovery: A service discovery process that is carried out whilst developing a service-based application. Any software services identified during this process may be bound to the service based application prior to its deployment.

Service discovery criteria: Criteria regarding the properties that services should have in order to be appropriate for an SBA.

Structural (or interface) discovery criteria: Discovery criteria regarding the

interface required from a software service.

Security discovery criteria: Discovery criteria regarding the security properties required from a software service.

Service discovery query: An expression specifying a logical combination of different service discovery criteria, which need to be considered in an instance of the discovery process. A service discovery query should also specify preferences amongst the service discovery criteria and whether discovery of services as published in the registries o and/or such service composition should be attempted in trying to locate services satisfying it, and whether the query should be executed in a reactive or proactive mode (see proactive and reactive service discovery).

Service discovery query language (aka "query language"): The language for expressing service discovery queries.

Service composition: The process of generating compositions of software services which satisfy given discovery criteria.

Static service composition: A service composition process that is carried out whilst developing a service-based application. Any service compositions identified during this process may be bound to the service based application prior to its deployment.

Service composition patterns: Abstract and parametric models of service compositions that can be instantiated to generate concrete executable compositions of services.

Secure service composition patterns: Service composition patterns that provably satisfy given security properties when instantiated by services which are known to have certain security properties.

Secure service composition: The process of identifying compositions of software services based solely on secure service composition patterns.

Service discovery system (aka SDS): The component in the ASSERT4SOA infrastructure that realises the different forms of the service discovery and composition process and enables the binding of the identified services or service composition to an SBA. The service identification system interacts with service registries and different types of matchmakers.

Static service discovery: A service discovery process that is carried out whilst developing a service-based application. Any software services identified during this process may be bound to the service based application prior to its deployment.

Service discovery criteria: Criteria regarding the properties that services should have in order to be appropriate for an SBA.

Structural (or interface) discovery criteria: Discovery criteria regarding the interface required from a software service.

Service registry: A repository of descriptions of software services.

Service description: A description of some facet of a software service that is stored in a service registry

Service interface description: A service description giving information about the interface of a service, i.e., the set of operations supported by the services and a full specification of the signatures of these operations.

Service endpoint description: A service description providing information about the endpoint from where a service can be invoked.

Service behavioural description: A service description providing information about the behavioural properties of a service.

Service quality description: A service description providing information about the quality properties of a service.

Service matchmaking: The process of matching a service description with a service discovery query.

(Security) Statement: It is the claim that a given security property is possessed by a given service. A statement is made and signed by a certification authority and refers (is bound) to a specific service.

Stored (security) preferences: a set of stored selection criteria that a service consumer may want to reuse across multiple queries.

Security property: The definition of a (security) characteristic of a service about which a statement can be formulated. In an ASSERT, it represents a security quality (such as: Confidentiality, Integrity, Robustness, and Authentication) of the evaluated service or of a part of it (e.g. a parameter of an operation or the internal data flow). Security properties can also be used in queries (opposed to using them in ASSERTs). In this case they represent a quality that is required by the service consumer in order to select a specific service. Properties have relations:

- Relations can form a hierarchy, but also other types of graphs;
- Properties can be related to other concepts (using the the ontology) like "Threats" and "Solutions";
- These relations are not "part" of the property description. Instead, we envision that they will be captured by the secure ASSERT4SOA Ontology;
- Properties can also be related to "Tests" and "Certification Schemes", but in this case this relation is intrinsic to the property, in case the property is defined in terms of those.

Selection of services: When an application sends a query for a service, it will normally include the expected functionality and the set of security properties to be fulfilled by the selected service. The query may return one of these two possibilities: 1) a service which provides this functionality and fulfils the requested properties, or 2) a set of services (let's assume for now that the sets are selected ONLY if they fit to a "pattern"), which provides this functionality and fulfils the properties. There are three important aspects related to the Service matching:

- *Functional:* The service functionality must be equivalent to the required functionality;
- *Security:* The services are attached with ASSERTs that certify their security

properties; The combination of ASSERTS is known a priory to ensure a property for the combined service (this is what the service orchestration patters do).

- *Structural:* The pattern offers the possibility of being combined with other services to provide the required functionality.