Security Aware Service Composition

Luca Pino, Department of Computer Science

School of Mathematics, Computer Science and Engineering

City University London

Supervisor: Professor George Spanoudakis

Thesis for the degree of Doctor of Philosophy,

May 2015

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank all the people that have supported and assisted me during these years. My appreciation goes to my supervisor Professor George Spanoudakis for providing me the opportunity to work on stimulating project, and for his useful comments, remarks and engagement throughout my experience.

I would like to express my gratitude to all the members of the Department of Computer Science, including the administrative and technical staff that contributed through my journey at City University. In particular amongst all I would like to thank Mark Firman and the Technical Support Team (TST) for their continuous support. Many thanks go also to all the ASSERT4SOA project partners for the great support and collaboration.

I would like to extend my appreciation to the friends that I made during this journey, for their encouragements and for the great time spent together, so thanks Dr. Daniel Wolff, Reinier de Valk, Srikanth Cherla, Muhammad Asad, Maria Krotsiani, Spyros Katopodis, Icamaan da Silva, Dr. Ricardo Contreras, Dr. Evangelia Kalyvianaki and Dr. Christos Kloukinas.

My deepest gratitude goes to my closest friends, now spread all around the world, for their support, advice and patience. So long and thanks for all the fish, Dr. Mayla Brusò, Benedetta Basile, Giulia Borghini and Veronica Varanini.

Last but not least, I am very grateful to my parents and to my girlfriend, Stefania, for their constant love, help and assistance in every day of my life.

Declaration

The author grants powers of discretion to the University Librarian to allow this thesis to be copied in whole or in part without further reference to him. This permission covers only single copies make for study purposes, subject to normal conditions of acknowledgement.

Abstract

Security assurance of Service-Based Systems (SBS) is a necessity and a key challenge in Service Oriented Computing. Several approaches have been introduced in order to take care of the security aspect of SBSs, from the design to the implementation stages. Such solutions, however, require expertise with regards to security languages and technologies or modelling formalisms. Furthermore, existing approaches allow only limited verification of security properties over a service composition, as they focus just on specific properties and require expressing compositions and properties in a model based formalism.

In this thesis we present a unified security aware service composition approach capable of validation of arbitrary security properties. This approach allows SBS designers to build secure applications without the need to learn formal models thanks to security descriptors for services, being they self-appointed or certified by an external third-party.

More specifically, the framework presented in this thesis allows expressing and propagating security requirements expressed for a security composition to requirements for the single activities of the composition, and checking security requirements over security service descriptors. The approach relies on the new core concept of secure composition patterns, modelling proven implications of security requirements within an orchestration pattern. The framework has been implemented and tested extensively in both a SBS design-time and runtime scenario, based respectively on Eclipse BPEL Designer and the Runtime Service Discovery Tool.

Chapter 1

Introduction

1.1 Research Area and Questions

The service-oriented computing (SOC) paradigm is aimed at addressing the need for constructing adaptable interoperable applications involving heterogeneous components over networks, known as *software services*, and offering access to utilities from a broad range of different devices. SOC focuses on interoperability and reuse by promoting the development of applications through composition of software services that might be deployed and running on different computational infrastructures. Technically a software service is a piece of autonomous and self-contained software accessible over a network through a collection of operations that are listed in the service interface.

SOC has been wildly embraced by the software industry [10][12][41][42][93], thanks also to the ability to simplify the communication and integration with business partners and with legacy systems [55][78]. Its adoption as business solution, however, has also raised a number of collateral concerns faced by the engineering, the operations and the business sectors. More specifically, some of the major concerns in the engineering and the operations fields are about studying the quality of

services that are used in service based systems (SBS) and the existence of agreements between service providers and service consumers to regulate them, the ability to discover and compose services and the ability to adapt a SBS dynamically [55].

Service composition, in particular, follows the SOC concept of building software out of existing or new reusable services, in order to provide a new functionality or to automate a task. The new functionality can be made available as a new service that solves more complex problems. The SBS lifecycle benefits from the service composition concept, as service compositions can be used to help both the SBS design phase -allowing the interoperation with more than one service per task- and whilst a SBS is in operation at runtime -allowing discovery and adaptation (i.e., replacement) of service compositions when single services providing a functionality are no longer available.

From the business perspective, however, an additional and very critical concern is about the security of services and the SBSs that use them [55]. Some of the characteristics that make SOC a successful paradigm, in fact, are also the ones that facilitate security attacks: e.g. the network access to services introduce all the security threats of classical distributed systems [102], whilst service interfaces and interoperability features preclude to adopt the concept of security-by-obscurity [17][24][78].

To address security there has been significant research which established (a) new additional stages to the development process in order to take into consideration the security requirements during the design and implementation of a service [29][38][66], (b) special security services that provide the mechanisms to protect other services (security-as-a-service) [16][38][39][76], and (c) extensions to support security for available languages and protocols [72][73][74][75][76][77][109]. Several security

extensions to languages and protocols have been through a process of standardization, following the concept of interoperability of SOC. These extensions introduce solutions to: ensure integrity and confidentiality in the messages exchanged (i.e., WS-Security [77], WS-SecureConversation [74]), provide mechanisms to construct trust relationships between organizations, or "security domains", through the usage of a special security service providing security tokens (i.e., WS-Trust [76]), and provide authentication and authorization of identities between organizations, or "security domains", thanks to federation agreements (i.e., WS-Federation [73], SAML [72]).

An open problem in this field is about *security of service compositions*. To assess the security of a service composition, in fact, the security of the individual services part of the composition must be taken into account, but it is not enough. In order to evaluate the security of the service composition, the order of execution of the composing services and the communications between them must be examined. In this scenario two key questions arise: (1) which security properties can be deducted from the security of the services within a service composition, and (2) how is it possible to require a service composition to preserve a security property?

This work is about assessing and constructing secure compositions of services allowing the support of security at both design and runtime. A possible use of secure service composition is to help the design and development of a SBS that calls different services. In this case *a SBS design tool can also offer some security validation mechanisms* that automatically generates and checks the security properties required by the single services part of the composition executed by the SBS, in order for the SBS to guarantee some more general security requirements on the whole SBS. Support of security at runtime allows automatic adaptation (i.e.,

replacement) of services that are no longer available with services or service compositions that guarantee the same level of security requested from the initial service. In this case, it is very important to support *the discovery and construction of service compositions that preserve the requested security properties*.

Another issue in the security field is how to obtain assurance that a service complies with a given security property (e.g., confidentiality, integrity, availability, authenticity). WS-Policy [109] and WS-SecurityPolicy [75] allow the specification, by the service provider, of which security mechanisms are in place. These policy languages support the development and negotiation of the security aspects of the communication, however they do not provide a general and objective assurance (i.e., based on third-party evaluation) of the security property guaranteed by the service. A proposed solution for this problem is the introduction of security certificates providing assurance that a service complies with a given security property [80].

The service certification approach is based on the traditional concept of software certification that has been used for non-service based software systems and software components. In this paradigm certificates are provided (and signed off) by some certification authority after assessing the compliance of the software with the required security property. For this approach to be effective, the certification authority must be trusted by service consumers and providers. This idea is aimed at providing assurance to all the possible users of the security properties granted by the software. In particular Common Criteria [18] is the international standard for traditional software certification (ISO/IEC 15408:2009 [45]), developed by the governments of Canada, US, UK, France, Germany and Netherlands in order to ensure security of the software used by the government and critical

infrastructures. Since these certificates are produced in order to be checked by IT or government personnel, they are human-readable system-wide documents that can easily exceed the hundreds of pages (e.g. by putting together the Certification Report, Security Target and Protection Profile of a CC certificate: see [19] for some examples).

The service-oriented paradigm, instead, introduces automatic software provisioning with concepts like runtime service discovery, service adaptation and service composition, thanks to a set of machine-readable interfaces. In order to ensure also the security of a SBS, then, certificates should also have a machine-readable equivalent for software services available at runtime and digitally signed as advocated, for example, by the ASSERT4SOA project [5][80].

With such certification scheme in place it is possible to envision a *security aware service discovery mechanism* that would allow also the specification of security requirements in a query to find and sort the services relevant for a task. In particular a security aware service discovery process can find sets of relevant services during the design and the development of a SBS, but it can be also quite useful in the context of run-time replacement. In this case the service discovery system should maintain an updated buffer of relevant services for a query so that the SBS can receive updates and substitute an unavailable or underperforming service with another one at run-time, while maintaining the same security features required to the original service.

Using this certification scheme allows also increasing the level of trust in solutions founded on automatic assessment of the security of service compositions.

1.2 Research Objectives

The overall aim of this thesis is to investigate the problem of assessing the security of service compositions and develop a solution that would enable the generation of secure service compositions out of software services with known (certified) security properties. To address this overall aim, our goal was to construct a framework that automatically infers the security requirements for the services part of a service composition, in order to guarantee general security requirements on the whole composition. This framework is aimed at SBS designers and developers engaged in building applications that require some security constraints.

The research objectives planned to achieve this can be listed as follows:

I. Literature review.

   To provide an analysis of the related works regarding security aware service composition that defines the subject area, its terminology, the existent models and case studies. This analysis should describe the different frameworks and highlight the strengths and weaknesses of each approach, in order to identify the gap that this research intends to fill.

II. Model of secure composition framework.

   To design a framework that allows automated reasoning on the security requirements of a service composition, to be able to generate security requirements for the services part of that composition that would guarantee the general requirements on the composition to hold. The framework shall envision some security patterns depicting abstract inferences that are proved to hold and

computed offline, in order to use those at run-time to ease the reasoning process and make the framework more responsive.

III. Prototype of a discovery tool performing security aware service composition.

To design and develop a plugin for a service discovery tool that takes into account security requirements and that is able to automatically build service compositions that would answer a given discovery query. The focus should be on the inference of security requirements from the composition level to the services in the composition, by means of the framework designed as objective II.

IV. Prototype of a design tool for security aware service compositions.

To design and develop a plugin for a SBS design tool that uses the framework results to propagate general security requirements to the single activities in a composition, to be able to automatically query for the services that will be used by the designed system and guarantee the required security at the same time.

V. Evaluation of the discovery tool.

The aim of the evaluation is to assess that the prototype resulting from the above objective behave as it would be expected and in a reasonable amount of time. This translates in some assessments of relevance from end-user developers to be able to evaluate the recall and precision of the system, and performance tests.

1.3 Research Assumptions

To shape the research, some assumptions were made giving some starting points and directions to the work:

- The availability of machine-readable descriptors of service security.

- The service compositions supported by our research are orchestrations, i.e., processes which coordinate individual software services and in which there is a single central coordinator that determines the order of the interaction and acts as an intermediary of all the communications (e.g., it receives results from a service and passes them, or some of them, to other services).

- The prototype of the discovery tool described as objective III makes usage of a service discovery approach supporting the proactive discovery of services at runtime. Such approach has been researched and implemented in a tool, called RSDT [115], by the Software Engineering Group of City University (a more detailed description is in Section 4.3).

- The discovery approach allows service discovery at development time, when there is already an estimated structure of the needed service. This means that the tool won't allow a simple browse of the registry, based only on service names or similar.

- The service discovery is envisioned as an incremental process of refinement of the query based on the discovery results.

- The focus of the research is on the inference of security in service compositions, so the functional composition part of the process might use existing state of the art ideas.

- Finally, the registries and all the parties involved in the discovery process are assumed to be trusted parties. This means that they should comply with a set of security dispositions to assure the security of the process.

1.4 Contributions

This research is aimed to provide a framework for service discovery and composition supporting inference and validation of security requirements. The framework allows constructing dynamically a service composition that respects given security requirements by means of a set of production rules and service discovery. Contributions of this research work include:

- *Design of a service composition mechanism to infer and validate security requirements*

  To handle the question, we introduce the concept of secure composition patterns, i.e., models describing abstract dependencies between the service composition security requirements and the component service security requirements. The dependencies are formally proven in order to ensure the same level of security of the original requirements.

  The patterns can be applied in different steps of a composition lifetime, to discover services guaranteeing the security or to validate the security of an existing composition.

- *Initial set of secure composition patterns and production rules*

  An initial set of secure composition patterns, comprising patterns for integrity, confidentiality and availability, is given to prove and exemplify the approach. The secure composition patterns are encoded into security production rules that can be deployed to a rule-based system.

- *Secure composition inference algorithm*

The process of security requirements inference from the service composition layer to the single composing services is achieved through a recursive algorithm that makes use of the secure composition patterns. This information can be used to assess the security of existing compositions (e.g. during the SBS design time) or to construct secure service compositions (e.g. to replace a service with a composition with the same level of security at either design or runtime).

- *Prototype of a discovery and composition tool supporting security*

  This prototype allows the creation of service compositions during the discovery of a service, and guarantees that the service compositions have the requested level of security. The tool allows adaptation of SBS at both design and runtime. The latter is achieved by taking advantage of the proactive capabilities of the discovery tool.

- *Prototype of a design tool supporting validation and adaptation based on the security of service compositions*

  This prototype allows the validation of the security requirements during an SBS design. The tool shows alternative services or service compositions that comply with the functionality and the security requested. The alternative services or service composition can be used to request automatic adaptation of the designed SBS.

- *Integration with the ASSERT4SOA toolkit*

  The approach and the prototypes have been integrated with the certification framework proposed by the ASSERT4SOA project. The certification framework provides further assurance with respect to

the security of the services, increasing the level of trust in the solutions provided by the presented approach.

- *Evaluation of the approach*

  To prove the feasibility and scalability of the approach, the service discovery and composition performances have been tested.

1.5 Publications

The contributions in this thesis have been submitted to conferences and workshops in order to collect feedback and disseminate the ideas presented to fellow researchers and organizations that work in the field.

In the following you can find a list of the published papers:

- Pino, L., and Spanoudakis, G. (2012, May). Finding secure compositions of software services: Towards a pattern based approach. In *5th IFIP International Conference on New Technologies, Mobility and Security, 2012 (NTMS'12)*, pp. 1-5. IEEE. DOI: 10.1109/NTMS.2012.6208741. [82]

  This paper describes an early version of the framework and introduces the concept of secure composition patterns. In this work the security production rules were encoded in Situation Calculus [62] and it was necessary to retain some information about the services internals (i.e., the “Security related actions on data”).

- Pino, L., and Spanoudakis, G. (2012, June). Constructing secure service compositions with patterns. In *IEEE Eighth World Congress on Services, 2012 (SERVICES'12)*, pp. 184-191. IEEE. DOI: 10.1109/SERVICES.2012.61. [81]

This paper presents an updated version of the framework and introduces an early version of the secure composition algorithm. In this version of the algorithm we were using patterns to address the construction of security composition respecting the required functionality (through the usage of ontologies and OWL-S [61] based patterns), and the security production rules were used to infer the security requirements of the composition.

- Pino, L., Spanoudakis, G., Fuchs, A., and Gürgens, S. (2014, April). Discovering Secure Service Compositions. In *4th International Conference on Cloud Computing and Services Sciences (CLOSER'14)*. DOI: 10.5220/0004855702420253. [84]

  This paper presents an example of a formal proof underlying a secure composition pattern on integrity, allowing trusting the solutions based on such pattern. Furthermore, the paper presents the encoding of the secure composition pattern into security production rules as Drools production rules, which represents our final choice for the encoding of the rules (as Drools is a fast, reliable and widely support rule-based decision system [47]). Finally the paper describes an updated version of the algorithm and presents the service discovery and composition prototype, with some initial evaluation figures.

- Pino, L., Spanoudakis, G., Fuchs, A., and Gürgens, S. (to appear). Generating Secure Service Compositions. In *Cloud Computing and Services Science: Fourth International Conference, CLOSER 2014, Barcelona, Spain, April 3-4, 2014, Revised Selected Papers*. Springer. [85]

This work is an extension of the previous paper that elaborates some further the formalisms used for the proof, based on Security Modelling Framework (SeMF) [37].

- Pino, L., Mahbub, K., and Spanoudakis, G. (2014, November). Designing Secure Service Workflows in BPEL. In *Proceedings of the international conference on Service-Oriented Computing (ICSOC'14)*, pp. 551-559. Springer Berlin Heidelberg. DOI: 10.1007/978-3-662-45391-9_48. [83]

  This paper is focused on the SBS design tool that supports validation and adaptation based on the security of compositions. This work presents the latest version of the security requirement inference algorithm and how this is applied in order to validate the security of portions of BPEL workflows and to adapt secure service compositions in a BPEL workflow.

1.6 Outline

The thesis is organised in 9 chapters as follows.

Chapter 2 presents an analysis of the existing approaches dealing with service discovery, service composition. We focus in particular on the security support in these fields and the existing standards and languages that address the security issue.

In Chapter 3 we summarise the concepts and definitions that are used in this thesis, from the Service Oriented Computing field.

Chapter 4 describes the languages and tools used in the context of this research. More specifically this chapter focuses on the WSDL and BPEL languages, used as the most common service description and service

composition languages, and on Drools, RSDT and Eclipse BPEL Designer, respectively the rule-based system, the runtime discovery platform and the SBS design tool used for the implementation of the prototypes.

Chapter 5 presents the secure composition pattern approach. We introduce some examples of secure composition patterns with the respective proofs. Furthermore, this chapter contains a methodology to encode the patterns into security production rules and the rules corresponding to the patterns presented earlier.

Chapter 6 describes the service composition process based on the secure composition patterns. This process makes usage of the security production rules introduced in Chapter 5. The algorithms underlying the service composition process are presented and discussed through the usage of examples.

Chapter 7 contains the implementation details of the two prototypes that use the secure service composition process in order to offer validation and adaptation of secure service compositions. In particular, it describes how the discovery platform and the SBS design tool presented in Chapter 4 have been extended in order to support security and service compositions.

Chapter 8 presents the setup and the results of the performance evaluation of the discovery and the composition process. The chapter contains the description of the configuration and the scenario, and the explanation of the numerical results obtained from the tests.

Finally, in Chapter 9 we present the conclusions. In this chapter we highlight the contributions of the approach, the implications that this approach has on the field and some topics for future works.

Chapter 2

Literature Review

2.1 Overview

This chapter presents an analysis of the existing works in the fields of service discovery and service composition, focusing in particular on the supported security features. This is followed by an analysis of the standard and languages that support security in the SOC field. Furthermore, all the approaches are summarized and put into relation with the contents of this thesis.

2.2 Service Discovery

In this section we provide an overview of single service discovery techniques, i.e., techniques that support the discovery of a service for a SBS without attempting to formulate complete or partial service compositions. They merely attempt to identify a single service that can fit within a system based on given criteria that this service needs to satisfy. In many cases, these criteria may express conditions that are necessary for the new service to fit within an existing service composition. Also, the discovered service may be a composite service itself. None of these cases, however, is treated as discovery of service compositions in the context of this work as the

discovery process does not attempt to create a new composition. Techniques supporting the discovery of service compositions are overviewed in Section 2.3 below.

The techniques that we overview are classified into groups depending on the main characteristics of the algorithmic approach deployed for service discovery. According to this criterion, techniques are grouped into:

- *Text matching service discovery* – These are techniques that make use of information retrieval techniques. In this group, discovery criteria are expressed as keywords which are subsequently matched with textual or structural descriptions of services. Typically, such techniques are deployed for early and design time service discovery.

- *Semantic service discovery* – These are techniques that assume descriptions of services that have been expressed in an ontology or annotated with links to ontological descriptions. Such techniques make use of the ontological descriptions during the matching process in order to improve the precision and completeness of the discovery process.

- *Graph matching techniques* – These are techniques that make use of different types of graph matching techniques (e.g. weighted bipartite graph matching, graph transformations, etc.) without relying on any form of ontology or formal reasoning of semantic service descriptions.

A summary of representative techniques in each of the above categories is provided below.

2.2.1 Text matching service discovery

Keyword-based retrieval underpins some service registries available on the Internet (e.g. *Xignite* [112] and *WebserviceX* [111]) and some basic built-in clients for development platforms (e.g. *jUDDI GUI* [103], *Eclipse Web Services Explorer* [26]). These approaches may also offer discovery through service categories and the use of tagged service descriptions. Text based service discovery is easy to use, due to the simplicity in the expression of the discovery queries. It is also useful in static service discovery, where the developers of SBS are usually concerned with finding a service that fits their requirements or the requirements of an application being designed. However, it cannot offer the matching precision that is required in dynamic service discovery that is executed to support automatic service replacement in applications at runtime. This is because in the analysis and design stages of SBSs, it is often useful to identify even services that do not match perfectly with what is required as a means of exploring alternative solutions and considering alterative designs and implementation paths for the application. At runtime, however, when the design of the overall application and its coordination logic have been fixed, the imprecision that typically characterizes keyword-based techniques is not acceptable, as decisions about replacing the partner services of a system with alternatives identified during the discovery process, in many cases, need to be taken in an automated manner.

2.2.2 Semantic service discovery

Semantic service discovery techniques constitute a significant approach to service discovery that is based on explicit representations of the semantics of services and logic reasoning techniques that analyse these representations. There has been a vast number of techniques that realise the semantic service discovery approach, including [50][56].

A system realising the semantic approach is OWLS-MX [50]. OWLS-MX uses logic based approximate matching and information retrieval techniques.

A semantic approach has also been advocated in [56], where a service discovery prototype that uses a Description Logic reasoner to match service discovery requests with ontology based service descriptions expressed in DAML-S.

Despite some experimental evidence showing acceptable precision and recall over competitors, however, the semantic approaches do not appear to be adequate for dynamic service discovery. This is because the ontological matches do not necessarily coincide with behavioural and interface matching at the level required for dynamic service discovery.

2.2.3 Graph matching techniques

Other approaches for service discovery consider graph transformation rules [49], or behavioural matching [32][67][92]. The work in [49] is limited since it cannot account for changes in the order or names of the parameters. In [92], the authors use service behaviour signatures to improve service discovery. In AOWS [33][96] the functional and quality characteristics of components and services are described as aspects and discovery is based on a formal analysis and validation of these descriptions. The work in [67] advocates the use of behavioural specifications represented as BPEL for service discovery for resolving ambiguities between requests and services and uses a tree-alignment algorithm to identify matches between request and services.

Graph matching underpins also the Runtime Service Discovery Tool (RSDT) [115][116] developed within City University of London. This system uses graph morphism detection algorithms to match service

interfaces and graph search algorithms to identify the compatibility of behavioural discovery criteria with behavioural service description models expressed in BPEL. Furthermore, this tool offers the capability to subscribe queries in order to have them executed and maintained proactively, in order to offer timely runtime service discovery to SBSs.

2.2.4 Context awareness

Several approaches have also been proposed to support context awareness in service discovery [11][20][79][113]. In [20], context information is represented by key-value pairs attached to the edges of a graph representing service classifications. This approach does not integrate context information with behavioural and quality matching. Furthermore, the context information is stored explicitly in a service repository that must be updated following context changes. In [9] queries, services and context information are expressed in ontologies. The approach in [11] focuses on user context information (e.g. location and time) and uses it to discover the most appropriate network operator before making phone calls. The work in [113] locates components based on context-aware browsing. The above context-aware approaches support simple conditions regarding context information in service discovery, do not fully integrate context with behavioural criteria in service discovery, and have limited applicability since they depend on the use of specific ontologies for the expression of context conditions.

2.2.5 Summary

In summary, most of the proposed approaches support service discovery based on limited sets of service criteria and using a reactive approach for query execution. Unlike them, RSDT supports dynamic service discovery based on a comprehensive set of service and application criteria

including but not limited to structural, functional, quality, and contextual characteristics. This tool supports both reactive and proactive service discovery, resulting in more efficient service replacement during the execution of a SBS.

Due to these reasons, RSDT was selected as a reasonable choice to be the basis for developing support for handling security related criteria and handling compositions as part of service discovery. In particular the extensible support for query criteria to any XML service description allows matching any form of security property specification, whilst the proactive service discovery support allows obtaining timely results even when the computation may require some time, as while performing service composition.

Approach	Algorithm	Service descr. language	QoS support
Xignite [112]	Keyword-based	WSDL	No
WebserviceX [111]	Keyword-based	WSDL	No
OWLS-MX [50]	Semantic (logic-based and IR)	OWL-S	No
Li, L. et al. [56]	Semantic (logic-based)	DAML-S	No
Mikhaiel, R. et al. [67]	Graph-based (tree alignment)	BPEL	No
Shen, Z. et al. [92]	Graph-based (RE-tree [15])	Behaviour signatures (new language)	No
AWOS [33][96]	Graph-based	AOWSDL [95] (new language)	Yes
RSDT [115][116]	Graph-based (morphism detection)	WSDL, BPEL and XMLs	Yes
Cuddy, S. et al. [20]	Context graph-based	Not explained	No
Beeri, C. et al. [9]	Context graph-based	BPEL	No
Bormann, F. et al. [11]	Context	n/a – not on WS	No
Ye, Y. et al. [113]	Context	n/a – not on WS	No

Table 2.1: Summary of the single service discovery approaches.

2.3 Service Composition

If the basic single service discovery fails to find the requested functionality, there is another way a discovery platform can try to fulfil the request: to compose an ad-hoc service on the fly by discovering and combining some services that provide the different parts of the functionality.

This additional step in the discovery process can be realized with the aid of different approaches arising from several areas of research (formal methods, automated reasoning, semantic computing, distributed systems, etc.) [8]. This wide range of possibilities offers a lot of solutions that can satisfy different types of discovery (static or dynamic, with human intervention or not), based mostly on which parts of the composition process can be automated. In particular we categorize the works in this area within two main groups:

- Approaches which automate the translation of the service query into workflows containing activity placeholders that need to be bounded to concrete services;
- Approaches that focus on automating the service discovery, adaptation and binding, when the workflow is already available.

2.3.1 Definition of Service Workflows

The approach of automating the phase of finding or building a new service composition is a step in the direction of dynamicity and it also answers to problems of complexity, response-time and scalability of a manual approach. The problem is typically to find or construct a workflow (or plan) that can satisfy the requirements. This step is usually followed by

the discovery and adaptation of services in order to instantiate the service composition, as explained in Section 2.3.2.

A solution to this problem is using reference process models, in domains in which such models exist, in order to generate a set of standard workflows that offer specific business functionalities. Some examples of such standard process models are the Health Level 7 (HL7) in the health domain [40], SWIFT used by financial institutions [98], and Electronic Data Interchange (EDI) [70], RosettaNet [34], IBM BPM Industry Packs [43] that specify models for a variety of fields (e.g., manufacturing, telecommunications, …).

Whenever a process model does not yet exist, however, the need to construct a workflow ad-hoc arises. An early work on this is SAHARA [60][87], a framework to compose services in a Wide-Area network, where the approach is not specific on services, rather it composes more generic data operators. The "composition path" (i.e., the workflow) is built by running the shortest path algorithm on the graph of the operator space. They propose to build domain-specific graphs and to cache popular results to limit the size of the graphs, but the solution isn't scalable in a more general context without the notion of local and wide-area paths. Furthermore the data operator point of view is a little restricting, not allowing for example a service to just retrieve information or to compose data from/to different services.

Another work from the same period is SWORD [86], a toolkit for efficient service composition. In this work a service is represented as a rule expressing that given certain inputs, the service will provide a certain output. These rules are expressed using Entity-Relationship assertions and are elaborated through a rule-based Expert System to generate plans, given the pre-conditions and post-conditions of the requirement. They allow only

simple queries, by not allowing arbitrary joins (like “find all pairs of movies with the same director”) and not providing arithmetic/function symbols, to maintain an efficient and simple model.

Most recent works, however, prefer to use standard languages to describe services (and composition requests), in particular OWL-S (and DAML-S). The reasons are mostly business-related and include: (a) in this way developers don’t need to learn further languages, (b) it simplifies the process of integrating an existent service discovery platform and (c) to avoid the error-prone (manual) process of converting the service descriptions in another language.

A framework for the automated service composition is described in [58] and it uses the services’ DAML-S description (DAML-S is a predecessor of OWL-S). In particular, the approach of this work is to try to find a single service corresponding to the high-level goal requested by the user, in case this step fails then a repository of abstract workflows is interrogated. Only if also this other step fails the framework tries to build a new composition, by chaining services through their input-output and precondition-effect descriptions. The matching of IOPEs (i.e., Input, Output, Precondition and Effect) is provided by a specific component that admits the composition of the I/O data from different services, allowing the creation of complex compositions.

In [88][89] the DAML-S Service Profile of each service is converted in extralogical axioms of propositional Linear Logic. The service composition request is then specified as a Linear Logic sequent and the system uses a theorem prover to check if the request can be satisfied by a composition of services. If a composition is possible, then a process calculus representation of it is generated from the proof and it is possible to request a workflow model (DAML-S Service Profile or BPEL4WS). Non-functional properties,

like security and QoS ones, are taken in consideration as well as the functional ones, thanks to inference rules.

CoSMoS [30][31] is a semantic-based model for services and compositions that is slightly different from OWL-S since it allows also semantic annotation of operation “concepts” (in addition of I/O) that cannot be defined as data types. In this context they introduce SeGSeC: a service composition mechanism that supports CoSMoS (i.e., semantic annotations). In this work the services must be described in CoSMoS/WSDL and the service request can be written in natural language: the tool then translates it into a CoSMoS semantic graph representation. The composition starts with the discovery of the service for the initial concept in the request (the one that provides the goal output) and then goes on by finding the services that provide the inputs for the initial service, using also the semantic information. At the end of the composition process the workflow is checked to guarantee that it satisfies the semantic request; otherwise the tool tries to find other compositions.

One of the most recent works in the field is DynamiCoS [53][94], a framework for dynamic service composition that supports requests in natural language (but also in a formal language based on OWL) and functional and non-functional properties. The first step of the composition in this framework is the service discovery, based on semantic concepts. The semantic connections between the I/O of the discovered services are stored in a Casual Link Matrix (CLM); so then the composition is built starting from the requested output searching backwards for compatible services through the CLM.

The framework itself does not include the service discovery component and the necessary interpreters to convert the service descriptions in the internal formalism (it is claimed that the approach can be applied with

OWL-S Service Profile, WSMO Capability Model or SA-WSDL specification).

Approach	Algorithm	Service description language	Allowed WF patterns	Security support
SAHARA [60][87]	Graph-based	n/a - not only on WS	Choice	No
SWORD [86]	Logic-based (rule-based system)	Based on ER-model (new language)	Parallel	No
Majithia, S. et al. [58]	Backward chain of I/O	DAML-S	-	No
Rao, J. [88][89]	Logic-based (theorem proving)	DAML-S	Choice and parallel	As service goals and constraints
CoSMoS / SeGSeC [30][31]	Semantic graph-based	CoSMoS/WSDL (new language)	Choice	No
DynamiCoS [53][94]	Semantic graph-based	Internal, needs interpreters (new language)	-	Yes / not explained

Table 2.2: Summary of approaches supporting automated construction of service compositions.

2.3.2 Instantiation of Service Workflows

This research area focuses on finding, given a workflow, the most suitable services for the activities in the workflow or, in case no perfect

matches are available, to adapt the workflow to consider the services that behave in a very similar way to the activities involved. The “most suitable” unit of measure is in the majority of the works based on the semantic correlation. Some other works focus on reaching the best QoS, after a first selection of services.

The earliest work on this subject is eFlow [14]: a platform for composition of services. This platform offers means of describing the workflow of the services through the GUI (by defining flow graphs) or through a simple composition language (an XML language called CSDL: Composite Service Definition Language) that allows dynamic discovery of services or dynamic selection and instantiation (with possibilities of multiple instantiations) of services from a list. The discovery is obtained by executing generic XQL queries on the repository of the service descriptions as the platform allows any XML format for the service descriptions. The obtained dynamic composition, however, isn’t guaranteed to be working correctly: the framework is built just to compose but it doesn’t perform any verification after the composition.

An example of work that extensively uses semantic computing is [64], an ontology-based framework for automatic service composition. The desired workflow, with semantic annotations, is described through a language called CSSL (Composite Service Specification Language). Syntactic, semantic and qualitative composability rules are used to select the services for the composition. In particular the service WSDLs must be augmented with semantic properties from the DAML+OIL ontology presented in the paper. An interesting feature from this work is the introduction of three measures for the selection between the different resulting compositions, called ranking, relevance and completeness (in

particular the first two measures are calculated on the basis of stored templates).

Another work on automatic composition based on ontologies is [51]. In this work the service request is defined with TWFO (Transactional WorkFlow Ontology), an ontology used to describe workflows with transaction support. The main difference with other works is that the service registry must also contain the workflow of the services used in the discovery process (expressed in TWFO as well). Then, after the candidate services are found through the DAML-S registry, the system tries to compose the workflow of each service in the requested workflow (called Master Workflow). The work does not go too much into details on the discovery process.

Regarding the automatic service composition based on QoS criteria, it should be noted that this kind of approach needs, in addition to the workflow of the composite service, the list of the compatible services for each activity as input. So, since a list of services has been already discovered, the matter to solve is reduced to just aggregate the different QoS data to find the best composition. A work in this area is [46], that uses some of the workflow patterns from [107] to define aggregation functions for QoS criteria. The patterns are used to do a stepwise graph reduction, and for every step the aggregated value of the QoS criteria is calculated.

Another work on QoS composition, even though quite domain specific, is SpiderNet [35][36], a framework for QoS assurance and load balancing of multimedia service compositions. The input of their tool is the composition of functionalities (a function graph) and a QoS requirement vector. Then the service composition is achieved through the bounded composition probing protocol: at each step a probe is sent from the actual service node to the most promising of its neighbours, to look for the next functions. Each node

provides as a result the list of the service components that implement the desired functions and the statistical QoS (the assumption is that the nodes are cooperative and trustworthy). The drawbacks of this work are that the composition process is slow and that the algorithm is based on probing the network, so it can be used only on bounded networks.

A more complete approach is given in METEOR-S [1][97], where the semantic and the QoS approaches are combined in a single automatic service composition framework. METEOR-S is more broadly a framework for the complete lifecycle of semantic web services; the particular component that deals with service composition is called MWSCF (METEOR-S Web Service Composition Framework).

The definition of the desired workflow is made through a specific GUI tool, where the user (service designer) should also associate each activity to a discovery URL. Then the framework ranks the services on two dimensions: the semantic matching and the QoS criteria matching. The service designer can specify the weights of each criterion to have control on the service selection process. The framework is not able to automatically generate an executable but it needs some user intervention for the data binding.

Approach	Matching approach	Input format (workflow)	Service description	Security support
eFlow [14]	n/a	CSDL and XQL	XML	No
Medjahed, B. et al. [64]	Syntactic and semantic logic-based	CSSL	WSDL with semantic in DAML+OIL	Privacy and encryption
Korhonen, J. et al. [51]	Ontology-based reasoning	TWFO	DAML-S and TWFO	No
Jaeger, M.C. et al. [46]	QoS aggregation (minimize function)	Workflow (+candidate services)	n/a	Encryption
SpiderNet [35][36]	Network probing	Function graph and QoS req. vector	Function names	No
METEOR-S [1][97]	Semantic and QoS ranking	BPEL-like, generated through a GUI	WSDL (+semantics), WSEL (QoS)	No

Table 2.3: Summary of the automated service discovery in service composition approaches.

2.3.3 Summary

The works on Service Composition comprise results from a wide area of fields. A variety of languages have been used to encode service descriptions and workflows, with no standard being embraced by the

community. The recent trend, however, has been to adopt semantic-aware specifications and solutions.

In more detail, we have presented the works in this field by categorizing them in the ones that discover or build a composition plan that can provide a given functionality, and the ones that instantiate workflow plans with services that will collaborate to implement the given functionality. In the context of this thesis we do not wish to address directly the former, as our assumption is that any approach in the literature may be used, but we enhance the solutions for the latter with extended security support. The described works that handle the instantiation of service workflows, in fact, present a very limited support for security, as described in more detail in Section 2.4.4.

2.4 Security in Service Oriented Computing

In this section we provide an overview of how the security problem has been addressed in the service-oriented computing (SOC) field. In particular, we are going to describe first the security standards and solutions that have been introduced in the SOC field and then how security has been handled in the context of Service Discovery, Service Composition and SBS Design.

2.4.1 Security Languages and Standards

In order to deal with security issues that used to hold back a wider usage of services, the SOC community has actively worked on standardizing a set of languages and protocols that would help the development of secure services and SBS.

WS-Security [77] is an OASIS standard that extends SOAP in order to allow enforcing confidentiality, integrity and non-repudiation on XML messages, thanks to encryption, signature and identifying security token

capabilities. This standard does not provide a complete solution for security, however is used as a building block for further protocols. WS-SecureConversation [74] extends the use cases of WS-Security providing a way to establish security contexts for multiple message exchanges, reducing the overhead introduced by key negotiation.

WS-Trust [76] is another OASIS standard that introduces the mechanisms to manage security tokens in order to build trust relationships between organizations, or "security domains". WS-Trust defines the process of issuing, renewing and validating of security tokens by the Security Token Service, the key exchange process and the format of the message used for each one of these operations.

SAML [72] and WS-Federation [73] are two OASIS standards for identity federation specifications that provide the means for shared authentication and authorization of identities between organizations, or "security domains", thanks to federation agreements (e.g., single sign-on mechanisms).

WS-Policy [109] is a W3C recommendation that allows the specification, by the service provider, of which QoS or security policies are in place. WS-SecurityPolicy [75] is an OASIS standard used to specify and negotiate security policies, based on WS-Policy, that can defined on a wide range of technologies, from transport layer security to the usage of protocols specified by WS-Security, WS-Trust and so on.

Common Criteria (CC) [18] is the international standard for traditional software certification (ISO/IEC 15408:2009 [45]), developed by the governments of Canada, US, UK, France, Germany and Netherlands in order to ensure security of the software used by the government and critical infrastructures. In CC vendors ask testing laboratories to evaluate their

software in order to check if it meets the security functional and assurance requirements (SFRs and SARs) they claim. After testing, CC certificates can be released produced by certification bodies in order to be checked by the users; for this reason the certificates are human-readable system-wide documents that can easily exceed the hundreds of pages (e.g. by putting together the Certification Report, Security Target and Protection Profile of a CC certificate: see [19] for some examples).

The ASSERT4SOA project [5][80] aimed to solve the shortcomings of traditional software certification in the SOC field. In particular, the proposed approach advocates the usage of machine-readable security certificates available at runtime and digitally signed by trusted third-parties (Certification Authorities, or CA). Each security certificate describes a security property that has been verified to hold for a given service. The certificates can be seen as additional service descriptions placed in service registries and available to clients and to service discovery processes.

2.4.1.1 Summary

The standards introduced in the SOC field encompass a number of mechanisms to support a wide range of security requirements, offering solutions to service developers for their implementations.

As the approach presented in this thesis is meant to assess security of service compositions, it does not need to go in the level of detail of the security implementation for a service. In fact, from the point of view of a SBS designer or a service user, all the concepts introduced by the WS-* standards are very fine grained, as they are used at a technical level to solve a problem, but they might be far too complicated for potential clients. In this sense, service users might want to know which security properties hold for a service without the need of the information about how the property is

achieved, e.g. knowing that a piece of data is treated with *confidentiality*, or that the service is *available* at the 99.99% of the time.

Furthermore, users cannot always know, and therefore trust, the Service Provider that offers a service and the level of security he/she declares. Security certificates offer third-party security guarantee for service users, without the need to investigate the exact mechanism that assure the requested security property.

For these reasons the implementation of our work uses security certificates, since our approach is mainly directed to SBS designers; however our solution can support any kind of security descriptor.

2.4.2 Security Design and Implementation

For the design and implementation of secure services, the research has been focused on: (a) additional stages to the development process in order to take into consideration the security requirements during the design and implementation of a service [29][38][66], and (b) special security services that provide the mechanisms to protect other services (security-as-a-service) [16][38][39][76].

[29] introduces the usage of a formal framework called SI*/Secure Tropos during the Early Requirements Engineering phase in order to model and analyse security requirements. The requirements are then used to produce a Secure BPEL workflow that goes through an iterative process of refinement. [66] proposes to use SecureUML in order to encode security requirements at design time. Then, before implementing the solution, an additional step is introduces in order to investigate the SOA Security Meta-model. At this stage a set of Security Pattern is used to convert the security requirements into security constraints that describe how to achieve the security requirement. PWSSec [38] describes a set of complementary stages

to be added to service development phases in order to support security. In particular WSSecReq is a first design phase aimed to specify the security requirements, WSSecArch is a phase where the requirements are used in conjunction with security architectural patterns in order to define the security architecture, and WSSecTech is the final design phase where a set of WS security standards (see the previous section) are identified starting from the security architecture designed in the previous stage. In particular during the WSSecArch stage, security services are added to the architecture in order to support a required security mechanism.

AO4BPEL [16] allows the integration of security specifications in a BPEL process. These specifications are then used to indicate security functionalities that are offered by a special Security Service, and integrate them in the AO4BPEL process. Sectet [39] is a framework for the implementation of security patterns from design to the implementation of an orchestration. Sectet enables the design of orchestrations as UML message flow diagrams, which are converted into workflows and used to generate stubs for actual orchestrations. In orchestrations, services are wrapped by Policy Enforcement Points, whose purpose is to provide the required security properties.

2.4.2.1 Summary

In order to ease the application of security measures to new services, the described approaches introduce new design phases and new security services. These approaches, however, differ from the work presented in this thesis as they aim to support security for a service, not a service composition. Furthermore, the processes described in the literature often require human intervention or additional security services, whilst our approach does not introduce such requirements.

2.4.3 Security aware Service Discovery

In this section we provide an overview of single service discovery techniques supporting security constraints, i.e., techniques that perform the discovery of a service without attempting to perform service composition (even though a discovered service may be a composite service itself). Frameworks supporting the discovery of service compositions are overviewed in Section 2.4.4 below.

In [105] the authors describe an approach to Web Service discovery based on privacy preferences. The preferences are specified as part of privacy policies (architecturally placed with service descriptions in a central service repository). The privacy descriptions consist of a vocabulary for properties including terms for *disclosure, openness* and *anonymity*. The process of applying the privacy-aware policy for the web services is accomplished in several stages. First, a client sends their preferences to a discovery agent. Then, a correspondence will be established between the user's interests and the web service privacy policies. Finally, the degree of user confidence to the privacy-aware policies on services is evaluated and a selection is made based upon the confidence levels obtained.

Similarly, the work presented in [48] uses policies described in extended service descriptions for authorization and privacy for semantic web services. The descriptions are proposed ontologies to annotate OWL-S input and output parameters with respect to their security characteristics, including encryption and digital signatures. Several extensions to OWL-S are proposed in the form of objects. First an *information object* which itself is extended to support either *encrypted information* or *signed information*. The approach also adds a series of policy types to OWL-S including a *PrivacyPolicy*, *ConfidentialityPolicy* and *AuthorizationPolicy*. The authors describe a design-time "best service selection" process. They also discuss

how this may be used for run-time compliance checking, but allude to the difficulties of trusting what providers offer as descriptions and what the services they provide actually undertake in execution.

[106] introduces a context-aware service discovery approach that makes usage of security policies. A threat analysis is given for the service discovery process, which led to the specification of security policies. The service client or the service provider can enforce security policies in order to restrict the access to their respective profiles during service discovery.

FSSD [69] is a decentralized peer-to-peer service discovery protocol that allows users to adjust their degrees of collaboration, security and privacy. In particular this work investigates the trade-off between these three characteristics and introduces a common secure trust overlay that may work in multiple administrative domains and that is independent of network and security infrastructures.

2.4.3.1 Summary

Existing research has focused largely on two sub-areas: the first being service discovery driven by some specific security or privacy constraint and second, the security of the service discovery mechanisms.

In our work we are interested in the former, which is yet to see a comprehensive solution in the literature, as typically only subsets of security properties are supported. In our approach, instead, we extend an existing service discovery tool with capabilities to match any security property with a service security description, in order to support security aware discovery to be used when instantiating a service composition.

2.4.4 Security aware Service Composition

The efforts to provide security in service compositions can be summarized in two categories, the ones that merely verify that an existing service composition guarantee a given security property, and the ones that take security properties into account during the construction of a service composition.

2.4.4.1 Verification of Service Compositions security properties

Among the works focusing on security in service composition particular relevance has been given to verification, through model checking, of already existent compositions' security. The service composition can be checked for flaws at design time or in a later stage of development, usually after concrete services are associated with each task. To perform the check the composition is modelled with formal languages and the requirements are expressed as properties on the model.

The design time verification is applied on a specification of the system. To encourage the use of this kind of verification, the language of the required specification is conventionally a common language of the Software Engineering area, usually UML.

Works meant for design time verification of security properties, like [22] and [23], usually support the system definition in UML (or similar tools), since it is a common language used in the Software Engineering area, encouraging in this way the use of this kind of approach. In particular [23] has an unusual approach with respect to the normal verification since they add the concept of patterns. Basically, the first step in their approach is to express security design patterns (i.e., design patterns of best practices to achieve some security goal) in UML sequence diagram. These patterns are

then converted into the formal language CCS [68] through some rules. The model checking of the security properties can be done, in the end, on compositions of these security patterns, to verify if the security properties are preserved.

A more general work in the verification of security properties in service composition is in [6][7]. They introduce a calculus (a typed extension of λ-calculus) to describe and compose services. In particular their language can be used to describe a model and check the security-related activities (access events, e.g. writing a file, opening a socket connection) of a service composition. The main remark on this work is that there's no description of the modelling phase, leaving to the reader the burden of planning how to convert the services into their language.

Another work in the verification of service composition area is Aniketos [4]. This work introduces a set of security patterns that are defined as design patterns that guarantee some security goal. These patterns are used to secure software during the design phase, which is human based, and to monitor changes related to the requested security policy at runtime.

2.4.4.2 Security aware Definition and Instantiation of Service Workflows

Another point of view regarding security in service composition is to obtain the guarantee that a composition respects some security policies directly from the discovery process, when an automatic composition of services approach is used.

A work that falls into this category is [54], where planning techniques are used to compose workflows compliant with some lattice-based access control models (e.g. multi-level secure systems). The focus is on how to

find efficient algorithms for workflow planning, even though in the limited case of sequences of operators.

In [13] the authors describe an approach to security conscious web service composition through the declaration of security constraints required on service provision and of the constraints declared by service providers. Security constraints are declared in SAML assertions [72]. Examples are provided for both authentication and authorisation assertions although a common security ontology is not provided. The architecture of using the constraints specified is based upon a Web Service brokering model. A Secure WS-Broker (SWS-Broker) is used to manage service requests and sets of security constraints, identifies a well-known business process (i.e., the workflow) compatible with the request from a library and tries to instantiate such workflow with appropriate services that respect also the security constraints. The approach also provides an implementation of the broker consisting of a workflow modeller, service locator, security matchmaker and WS-BPEL generator. The security matchmaker builds a tree structure of the path of security considerations (from the constraints applied to the workflow) and analyses the possible composition paths and security constraints from discovered services. WS-Agreement nodes are also generated as part of service message structures to express the constraints applied.

2.4.4.3 Summary

Approaches allowing verification of service composition security require modelling the service composition, its services and the security property to check them through formal languages, in order to be able to say if the security property is satisfied.

The works that handle security during the definition and the instantiation of service workflows, instead, add a security aware discovery dimension during the composition of the services. Furthermore, as summarized in Table 2.2 and Table 2.3, some of the works in Section 2.3 allow the expression of few security properties in the request as well as non-functional properties.

Our approach does not require the knowledge of any formal language or knowledge about the internals of the services, as services are software components that may be available from an external provider unwilling to share information about the service internals. Furthermore model based approaches are usually specialised to verify a specific security property, whilst our approach allows the inference and validation of any security property, given that a formal proof of composition results exists.

Finally, in almost all the approaches taken into consideration, security properties are specified and checked only against single services in the composition, not giving information on the overall security of the composition. In our approach, instead, it is possible to require a security property over an entire service composition in order to be used to infer which security properties are required by the services part of the composition. This allows treating a service composition as a single service, permitting in this way the substitution of a single service that has a set of security requirements with a service composition that is generated online and that respects the same security requirements of the original service.

2.4.5 Security aware Design of SBS

The definition and verification of security requirements is an aspect that is not only important during the design and development of services,

but also fundamental when designing SBS. The following approaches define some initial steps in the direction of security aware design of SBS.

One direction of research [27][29] is to define new languages for the specification of security requirements over a BPEL. These approaches can be seen as a first step to support the design of secure SBSs, but lack of appropriate editors to aid the use of the new language at design time.

The work in [29] introduces a language to specify high-level security requirements in a business process description. This specification language is a BPEL dialect that abstracts low level details about the security implementation, allowing devising secure workflows at design time. The approach presented in [27] focuses on the definition of a language for specifying security policies in order to simplify the verification when a BPEL business process is used in different enterprises. The policies apply on single services part of a business process, so no security requirement can be formulated for the composition as a whole.

The Sec-MoSC (Security for Model-oriented Service Composition) tool [100] is an extension of the Eclipse BPMN Modeller that allows to design BPMN business processes and to add security requirements to them. In this approach, security requirements are expressed by (i) the security property category (e.g., Confidentiality, Integrity, …), called NF-Attribute, (ii) the level of the property (i.e., High, Medium, Low), called NF-Statement and (iii) the security mechanism that implements the property (e.g., Confidentiality can be implemented by UseCryptography, that has properties about Encryption Type, Algorithm, Encrypted Message Parts and Key Length), called NF-Action. After selecting the NF-Attribute and NF-Statement for a BPMN element, a default set of NF-Actions that implements the requested property are automatically added in the security requirement.

The tool needs human intervention to associate services to the BPMN activities, but it filters the service repository based on the NF-Attributes and NF-Statements in the security requirements to ease this task. The user needs also to add manually the data mappings and the predicates for the control flow (e.g., loops and conditions for decision commands) to enable the encoding of the BPMN process, which does not contain this kind of information, into an executable BPEL process. An interesting feature of this approach is the usage of an auxiliary security engine during the execution of the BPEL process that performs the NF-Actions that were required to the BPMN process itself.

Another similar approach, on QoS requirements instead of security ones, can be found in the METEOR-S project (see Section 2.3.2). The project made available several tools to manage annotations in WSDLs and UDDI registries, as all the declarations in the WSDLs must be linked to ontological concepts. These tools can be used to semantically annotate an abstract BPEL process, allowing also the specification of QoS requirements. The annotations are then used to discover appropriate services for the BPEL process, using an enhanced UDDI registry. The demo of this approach [57] shows, however, that the generation of the abstract BPEL process is external to their toolset (they use a BPEL design tool from IBM called BPWS4J Editor) and that the BPEL must be imported into the METEOR-S tool for the annotation to take place. Furthermore, this approach requires extensive semantic annotation of all the services in the registry, their behaviour and inputs/output.

2.4.5.1 Summary

In order to allow the specification of security requirements during the design of a SBS, some works in the literature define new languages for the specification of security requirements over a BPEL. These approaches are

an interesting first step to taking care of security at design time, however further efforts might be required in order for SBS designer to use them, e.g., provide some user friendly editors. Furthermore such approaches might benefit by allowing the specification, inference or the validation of security requirements over pieces of the composition, instead of allowing only the specification over single activities.

Other works in the area provide editors to guide the SBS designer to specify security requirements for single activities in a workflow and bound appropriate services that respect the requirements. The approach presented in this thesis is meant to extend this, allowing the SBS designer to use a single tool in order to (a) design the (executable) BPEL process for an SBS, (b) specify security requirements for single activities or workflow fragments, (c) automatically infer the security requirements over the single activities part of a workflow from the security requirements on the entire workflow, (c) validate the security requirements by checking services security descriptors, (d) discover alternative services or build service composition that satisfy the requirements, and (e) automatically replace a service with an alternative service or service composition.

Chapter 3

Conceptual Foundations

3.1 Overview

In this chapter we cover the conceptual foundations that underpin the research outcomes of this thesis by giving definitions and explaining the relations between concepts used in our approach. The focus of this chapter will be on software services and service related topics, e.g. service discovery, service composition and cloud computing.

In the context of this work we are going to use the terms *Software Service* and *service* interchangeably.

3.2 Software Service

Software Services are the basic components of the *Service-Oriented Architecture* (SOA) paradigm, i.e., an emerging paradigm that employs services to support rapid and simple development, usage and composition of distributed applications:

"A *service* in *SOA* is an exposed piece of functionality with three essential properties. A SOA-based service is a self-contained (i.e., the service maintains its own state) and platform-independent (i.e., the interface

contract to the service is platform independent) service that can be dynamically located and invoked. The primary value of SOA is that it enables the reuse of existing services, either as standalone or as part of composite applications that perform more complex functions by orchestrating numerous services and pieces of information. A simple service is reused in different ways and combined with other services to perform a specific business function." [78]

In other words a *Software Service* is a self-contained piece of interoperable software exposed over a network that might be accessed programmatically by other software applications through the discovery and invocation of a specific public *interface*. By using or combining different software services a software application can provide larger and more complex functionalities; we call such applications *Service Based Systems* (SBS).

"The *service interface* part defines service functionality visible to the external world and provides the means to access this functionality. The service describes its own interface characteristics, i.e., the operations available, the parameters, data typing, and the access protocols, in such a way that other software modules can determine what it does, how to invoke its functionality, and what result to expect in return." [78]

Software services are developed and offered to users (i.e., *Service Clients*) by entities called *Service Providers*. When a *Service Provider* makes available a service, it publishes also a *service interface* description in order to define how potential clients can access the service. *Service Registries* are collections of service interfaces (from one or more service providers) that allow clients to look up for the service they need. The decoupling of these concepts allows a service to be used by other entities than the *Service Provider*, i.e., the *Service Clients*, to build their own *SBS*.

Furthermore, it allows using a service for applications that may have been unforeseen by the *Service Provider*.

3.2.1 Web Service

One of the most common implementations of the Software Service concept is called *Web Service*. The definition of *Web Service*, as given by the leading standard organization for the Web, the World Wide Web Consortium (W3C), is the following:

"A *Web service* is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards." [110]

While being a little outdated (as SOAP is no longer the only protocol used to exchange messages by *Web Services*), this definition provides a good introduction to the spectrum of technologies that characterize *Web Services*. In essence *Web Services* are *Software Services* that can be described, used and coordinated through XML artefacts and that are conveyed through web-related standards as HTTP. In particular XML encodings promote interoperability, as their text-based representation is platform-independent.

In more detail, *Web Services* have their service interface described in WSDL, an XML-based language that allows definition of types, operations and bindings, and they can expose bindings to a variety of architectures, most notably based on SOAP or REST. SOAP is a protocol that uses XML-based messages to exchange data, whilst REST is an architectural style that allows for stateless and cacheable services (called *RESTful services*) and

that uses a more compact representation, thanks to the usage of the HTTP methods (i.e., GET, PUT, POST, DELETE).

3.3 Service Discovery

The process of finding a service suitable for the client's needs is called *Service Discovery*. More formally, *Service Discovery* can be defined as the act of locating software services that meet a set of discovery criteria, by matching the criteria against the service interfaces that are published in service registries.

The requirements of the service client are called discovery criteria and they are used to guide the *Service Discovery* process. In particular the logical combination of criteria sent to a service discovery platform in order to obtain a list of compatible services is called *Service Query*.

Different types of *Service Discovery* can be distinguished at different phases of a SBS lifecycle. The *Static Service Discovery* is used at design or development time in order to bound software services during the implementation of a SBS. The human designer of the SBS surveys the results of the static service discovery, and potentially requests new discovery processes iteratively, until the designer finds the best suitable service for the application. The *Dynamic Service Discovery*, instead, can be requested at runtime by a SBS in order to bind to the most appropriate service during execution. This can happen either because the SBS has been left purposely unbound at design time or because one or more services bound to it failed to satisfy the requirements, so they should be substituted. In the dynamic service discovery scenario, the SBS designer has to specify the *Service Query* that must be used by the SBS to request the discovery of services, in this way some application can even avoid requesting for human intervention during the replacement process.

3.4 Service Composition

As mentioned above, a set of services might be combined to achieve a more complex functionality; the product of such process is called a *Service Composition*:

"Composite services (and processes) integrate multiple services – and put together new business functions – by combining new and existing application assets in a logical flow. Service composition combines services following a certain composition pattern to achieve a business goal, solve a problem, or provide new service functions. The definition of composite services requires coordinating the flow of control and information between the component services." [78]

There are two techniques that allow the definition of a *Service Composition*; these are called *Service Orchestration* and *Service Choreography*.

"*Orchestration* describes how Web services can interact with each other at the message level, including the business logic and execution order of the interactions from the perspective and under control of a single endpoint. […] With orchestration, the business process interactions are always controlled from the (private) perspective of one of the business parties involved in the process." [78]

In other words, a *Service Orchestration* is a *Service Composition* controlled by a single entity, called coordinator, which executes a process that uses software services in order to accomplish a business objective.

Service Choreography is defined instead as:

"*Choreography* is typically associated with the public (globally visible) message exchanges, rules of interaction, and agreements that occur between multiple business process endpoints, rather than a specific business process that is executed by a single party. Choreography tracks the sequence of messages that may involve multiple parties and multiple sources, including customers, suppliers, and partners, where each party involved in the process describes the part it plays in the interaction and no party "owns" the conversation." [78]

This is very similar to the *Service Orchestration*, but instead of describing the instructions for a single party perspective, the *Service Choreography* requires a description of all the interactions between all the parties involved in the composition in order to accomplish the goal. In this sense, *Service Choreography* can be seen also as the collection of all the *Service Orchestrations* of each involved party.

The approach presented in this work focuses on *Service Orchestration*, so when referring to *Service Composition* we hint at compositions obtained through *Service Orchestration*, if not stated otherwise.

3.4.1 Business Process Management

Many concepts used in the *Service Orchestration* field are taken from the *Business Process Management* area. *Business Process Management* focuses on modelling workflows and processes within an organization.

In this context, a *Business Process* is defined as the collection of structurally linked activities that realise a business goal. A *Workflow* is a model of the procedural steps through which documents, products or tasks have to pass to carry out the *Business Process*, where the procedural steps may be in some occasions automatable. In particular when the steps are

limited to requests to Software Services, a *Workflow* corresponds to a *Service Orchestration*.

A concept from the *Business Process Management* field that is used in this work is the one proposed by van Der Aalst, W.M. et al., called *Workflow Patterns* [107]. The *Workflow Patterns* are a collection of design patterns describing the control flow dependencies between activities in a *Workflow*.

3.5 Cloud Computing

Cloud Computing is a recent paradigm to share resources in order to provide scalable services:

"*Cloud computing* is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." [65]

In particular, based on the offered resources, the provision can be distinguished in three models:

i. *Infrastructure as a Service* (IaaS). The cloud provider offers computing resources, including but not limited to physical and virtual machines, storage, firewalls and load balancers. Some examples include Amazon EC2, Rackspace Cloud, Google Compute Engine.

ii. *Platform as a Service* (PaaS). The cloud provider offers a software platform in order to allow deployment of services without the need to manage the underlying infrastructure and its scalability. The

platform usually includes operating system, databases, servers and execution environments. Some examples include Google App Engine, Microsoft Windows Azure, Salesforce Force.com.

iii. *Software as a Service* (SaaS). The cloud provider offers applications or (web) services on demand, running on the platform. Examples include Google Apps, Microsoft Office 365, Salesforce AppExchange, Xignite Market Data Cloud.

Furthermore clouds can be classified in *Private*, *Public* or *Hybrid Clouds* if the infrastructure is, respectively, for the exclusive usage of a single organization, provisioned to the general public, or a composition of distinct cloud infrastructures.

Chapter 4

Enabling Languages, Techniques and Tools

4.1 Overview

In this chapter we present the languages, techniques and tools that are used to implement this research. In particular WSDL and BPEL are languages commonly used in the Web Services area to define respectively service interfaces and business processes (i.e., orchestrations). Drools is a rule-based system we use to encode the patterns that underpin this thesis. RSDT and BPEL Designer are respectively a (proactive) service discovery tool and a service orchestration designer tool that we augmented with the security capabilities offered by our approach.

4.2 Web Services Languages

4.2.1 WSDL

The Web Services Description Language (WSDL) is the service interface description language, based on XML, which allows the definition of the operations and messages that can be sent to and received from a service, and the protocols and the addresses to contact the service.

The current version of the specification, 2.0, is a W3C recommendation [108]. As most of the tools and languages, however, currently support only WSDL version 1.1, we are going to describe this version instead of the latest one.

As shown in Table 4.1, a WSDL document is composed of five elements: `types`, `message`, `portType`, `binding` and `service`. The `types` element allows listing the data type definitions used by the service. The `message` element allows defining the data communicated by the service, using the types previously declared. The `portType` element lists the operations supported by the service. The `binding` element allows the specification of the protocol and data format specification for the abstract service operations described in the port type part. Finally the `service` element specifies the address to contact the service, called *service endpoint*.

```
<definitions xmlns=... name="HelloWorld">

  <types>
    <xsd:schema ...>
      <xsd:element name="RequestType">
        <xsd:complexType><xsd:sequence>
            <xsd:element name="in" type="xsd:string"/>
        </xsd:sequence></xsd:complexType>
      </xsd:element>
      <xsd:element name="ResponseType">
        <xsd:complexType><xsd:sequence>
            <xsd:element name="out" type="xsd:string"/>
        </xsd:sequence></xsd:complexType>
      </xsd:element>
    </xsd:schema>
  </types>

  <message name="OperationRequest">
    <part element="RequestType" name="parameters"/>
  </message>
  <message name="OperationResponse">
    <part element="ResponseType" name="parameters"/>
  </message>

  <portType name="HelloWorld">
    <operation name="Operation">
      <input message="OperationRequest"/>
      <output message="OperationResponse"/>
    </operation>
  </portType>

  <binding name="HelloWorldSOAP" type="HelloWorld">
    <soap:binding style="document"
                  transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="Operation">
      <soap:operation
                  soapAction="http://www.example.org/Operation"/>
      <input><soap:body use="literal"/></input>
      <output><soap:body use="literal"/></output>
    </operation>
  </binding>

  <service name="HelloWorld">
    <port binding="HelloWorldSOAP" name="HelloWorldSOAP">
      <soap:address location="http://www.example.org/"/>
    </port>
  </service>
</definitions>
```

Table 4.1: Example of a WSDL

4.2.2 BPEL

BPEL (short for WS-BPEL, Web Services Business Process Execution Language) is an XML based orchestration language that allows defining business processes that interact with external services. The interaction with external services is described through partner links, i.e., connectors between the service ports, as specified in the WSDL, and the business process.

The current version of BPEL, 2.0, is an OASIS standard and allows the specification of both abstract and executable business processes.

BPEL activities can be discriminated in three categories:

I. Activities that control the process flow. These include `sequence`, `if-else`, `while`, `repeatUntil`, `forEach`, `flow`, `pick`, `wait` and `exit`.

II. Activities that perform the actions of the process, i.e., web service invocation (`invoke`), assigning values to variables (`assign`) and receive and reply messages (`receive` and `reply`).

III. Management activities, such as fault generation (`throw`) and handling (`faultHandlers`).

4.3 Drools

Drools is a production rule system that allows rule reasoning for object-oriented languages. As in most rule engines, the production rules in Drools are used to derive information from data facts, usually stored in a Knowledge Base (KB). This reasoning process is based on the Rete algorithm [28], i.e., a pattern-matching algorithm for that is able to scale well for large numbers of data facts and rules.

A production rule in Drools has two parts. The first part is a set of conditions and the second part is a list of actions. When a rule is applied, the Drools rule engine checks, through pattern-matching, whether the conditions of the rule match on the facts in the KB and, if they do, the list of the actions of the rule are executed as a consequence. Table 4.2 presents the overall structure of Drools rules.

The conditions of a rule are expressed as patterns on the objects that encode the facts in the Drools KB. The patterns can be connected through a set of logical operators (e.g., `and`, `or`, `not`, `exists`, `forall`) and when no operator is explicitly declared, the engine assumes and uses the "`and`" operation as a default. The syntax is pretty flexible (especially w.r.t. the common programming languages), as most of the punctuation, double quotes and newlines are optional.

```
rule "name"
    when
        conditions
    then
        actions
end
```

Table 4.2: Drools rule structure

A pattern defines an object type and a set of constraints on the data of the objects that can match it. When an object that matches the object type in a pattern and the related set of constraints is found, the pattern is evaluated as `true`. In addition, it is possible to declare a variable (usually prefixed with a dollar sign to make it more easily identifiable) that can be subsequently used to refer to the matched object (or object field in the conditions). This is done by prefixing the variable name (followed by a colon) to the pattern.

```
$redApple : Apple( color == "Red" )
(Bowl( contents contains $redApple )
or Fridge( contents contains $redApple ))
```

Table 4.3: Example of Drools conditions

The conditions in Table 4.3, for example, activate the rule for each red apple found in the Knowledge Base that is contained in a bowl or in a fridge. The example includes also the "contains" operator used in a constraint: this operator checks if the specified value is contained in an array, list or set.

The actions in the consequence part are usually meant to modify the Knowledge Base by inserting, retracting or updating the objects in it. This is encoded through the keywords "insert", "update" or "retract" followed with the object to modify in parenthesis.

```
rule "Thrash expired bananas"
    when
        $expiredBanana : Banana( color == "Black" )
        $fridge : Fridge( contents contains $expiredBanana ))
    then
        $fridge.getContents().remove($expiredBanana);
        update($fridge);
        insert(new Bin($expiredBanana));
end
```

Table 4.4: Example of a Drools rule

The rule in Table 4.4 gives an example of a complete rule. This rule is activated ("fired") against all the black bananas contained in a fridge. Each black banana is then removed from the fridge and put into a new bin. The updated fridge and the new bin are reported to the KB because these new facts could lead to the activation of another rule.

4.4 Runtime Service Discovery Tool

The Runtime Service Discovery Tool (RSDT) is a discovery framework that has been developed at City University to support the discovery of services at runtime [115].

The framework supports the discovery of single services based on criteria regarding the interface, behaviour and quality of services, in a reactive or a proactive mode, i.e., when a need for finding a service at arises (reactive mode) or continually in order to maintain up-to-date sets of candidate services that could be used to replace the constituent services of an SBS when any of these services fails (proactive mode).

4.4.1 Architecture

The approach to service discovery is shown in Figure 4.1. The framework accepts service discovery queries from SBSs, and finds services in external service registries that satisfy the conditions of the queries. Queries can be submitted for execution in reactive (PULL) or proactive (PUSH) mode.

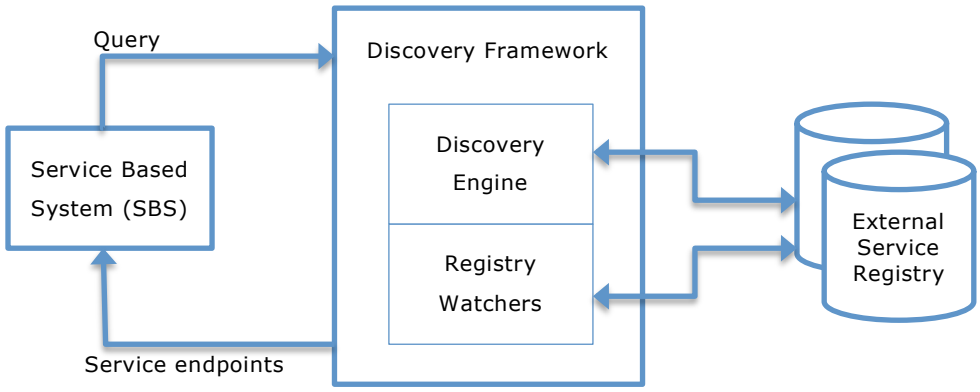


Figure 4.1: Discovery Framework structure

The framework includes a Discovery Engine that is responsible for the retrieving individual service descriptions from external service registries and matching them with the queries. It also includes Registry Watchers which poll external registries periodically to check if there are new services or amended service descriptions that would alter the candidate sets of services that are maintained for queries executed in proactive mode.

4.4.2 Discovery process

The discovery process starts when the Discovery Engine receives a query that should be used for discovering replacement services for one of the partner services of an SBS. Queries are expressed in an XML based language, called SerDiQueL. The discovery engine executes the received query at least once (in proactive mode multiple executions may be triggered by changes in the services) and returns any services that match the discovery criteria of the query. Any services and that match with the discovery criteria of the query at this stage are used to update a Candidate Service Set. This set is used as a cache of replacement services for the partner service that was associated with the query in the first place and any subsequent service replacement request will retrieve the first service from this set.

It should also be noted that the initial formation of the Candidate Service Set is followed by ordering the elements of this set in descending order of the degree of match that they have with these criteria.

Certain parts of the overall discovery process can be also triggered by events other than a request for the execution of a query. These events are:

- service replacement requests resulting in removal of the first service in the Candidate Service Set in order to use it in the SBS;

- publications of new security descriptions for one of the services in the candidate service set that should trigger the re-evaluation of the security related criteria for a candidate set that has been built for a query executed in proactive mode and possibly a re-ordering of this set; and

- changes in the descriptions of services in the service registries or the publication of new services in them that can lead to the execution of queries executed in the proactive mode.

4.4.3 Query Language

The queries of the discovery framework are expressed in SerDiQueL [101], an XML-based language that allows the specification of interface, behavioural, and QoS conditions about the services to be discovered.

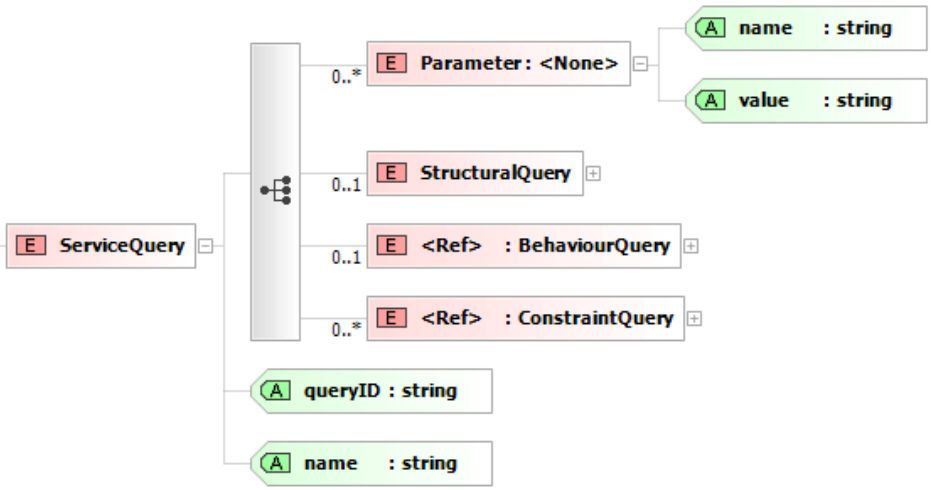


Figure 4.2: Overview of the schema of SerDiQueL

The top-level schema of SerDiQueL is shown in Figure 4.2. Each query has a name, a query ID, a set of parameters and a set of conditions. In

particular the parameter `mode` allow to specify if the query has to be executed in the reactive (`PULL`) or in the proactive (`PUSH`) mode.

The `StructuralQuery` part of the query contains the structural description of the service being discovered, i.e., the WSDL specification.

```
<BehaviourQuery>
  <Requires>
    <MemberDescription ID="login" synchronous="true"
                       opName="BankTransferService.login" />
    <MemberDescription ID="credit" synchronous="true"
                       opName="BankTransferService.credit" />
    <MemberDescription ID="transferAmount" synchronous="true"
                       opName="BankTransferService.transferAmount" />
    <MemberDescription ID="debit" synchronous="true"
                       opName="BankTransferService.debit" />
    <MemberDescription ID="balance" synchronous="true"
                       opName="BankTransferService.getBalance" />
    <MemberDescription ID="logout" synchronous="true"
                       opName="BankTransferService.logout" />
  </Requires>
  <Expression>
    <Condition>
      <GuaranteedMember IDREF="login" />
    </Condition>
  </Expression>
  <LogicalOperator operator="AND" />
  <Expression>
    <Condition>
      <Sequence ID="pay">
        <Member IDREF="credit" />
        <Member IDREF="transferAmount" />
        <Member IDREF="debit" />
        <Member IDREF="balance" />
      </Sequence>
    </Condition>
    <Condition>
      <OccursBefore immediate="false" guaranteed="false">
        <Member1 IDREF="login" />
        <Member2 IDREF="pay" />
      </OccursBefore>
    </Condition>
  </Expression>
</BehaviourQuery>
```

Table 4.5: Example of behavioural conditions of a SerDiQueL query

The `BehaviourQuery` part, instead, contains the behaviour of the client, in terms of operation calls and their ordering, as expected by the service. An example of a `BehaviourQuery` is shown in Table 4.5. The query in the example requires the existence of a set of operations (i.e., `login`, `credit`, `transferAmount`, `debit`, `getBalance`, `logout`). Furthermore every trace of interaction with this service must include a `login` (the `GuaranteedMember` condition). The last condition in the example specifies that a payment, composed as a sequence of `credit`, `transferAmount`, `debit` and `balance`, must be always preceded by a `login`.

The `ConstraintQuery` part allows to specify a set of constraint on any kind of service description (or facet). Table 4.6 shows an example of `ConstraintQuery` on a quality of service facet (QoS). The first constraint is required to match (`HARD` constraint) and checks if the organisation name is CITY. The second constraint, instead, doesn't have necessarily to match, as it is used for ordering the resulting set of services (SOFT constraint). This constraint is composed by two conditions joined by the AND operator, checking that the service is available from 00:00 till 24:00.

```
<ConstraintQuery name="C1" type="HARD">
  <LogicalExpression><Condition relation="EQUAL-TO">
      <Operand1>
        <NonContextOperand facetName="QoS" facetType="QoS">
          //QoSCharacteristic[Name="Organisation"]/Constant
        </NonContextOperand>
      </Operand1>
      <Operand2><Constant type="STRING">CITY</Constant></Operand2>
  </Condition></LogicalExpression>
</ConstraintQuery>

<ConstraintQuery name="C2" type="SOFT">
  <LogicalExpression>
    <Condition relation="EQUAL-TO">
      <Operand1>
        <NonContextOperand facetName="QoS" facetType="QoS">
          //QoSCharacteristic[Name="Availability"]/Metrics
                    /Metric[Name="OpenTime"][Unit="Hours"]/MinValue
        </NonContextOperand>
      </Operand1>
      <Operand2><Constant type="STRING">00:00</Constant></Operand2>
    </Condition>

    <LogicalOperator>AND</LogicalOperator>

    <LogicalExpression>
      <Condition relation="EQUAL-TO">
        <Operand1>
          <NonContextOperand facetName="QoS" facetType="QoS">
            //QoSCharacteristic[Name="Availability"]/Metrics
                    /Metric[Name="OpenTime"][Unit="Hours"]/MaxValue
        </NonContextOperand>
        </Operand1>
        <Operand2><Constant type="STRING">24:00</Constant></Operand2>
      </Condition>
    </LogicalExpression>
  </LogicalExpression>
</ConstraintQuery>
```

Table 4.6: Example of constraint conditions of a SerDiQueL query

4.5 Eclipse BPEL Designer

BPEL Designer is a plugin for the Eclipse IDE that offers a visual representation for reading and editing WS-BPEL 2.0 processes, allowing the specification of SBS based on BPEL. The editor provides graphical

representation of BPEL constructs and processes using shapes, icons, forms and wizards to guide the user.

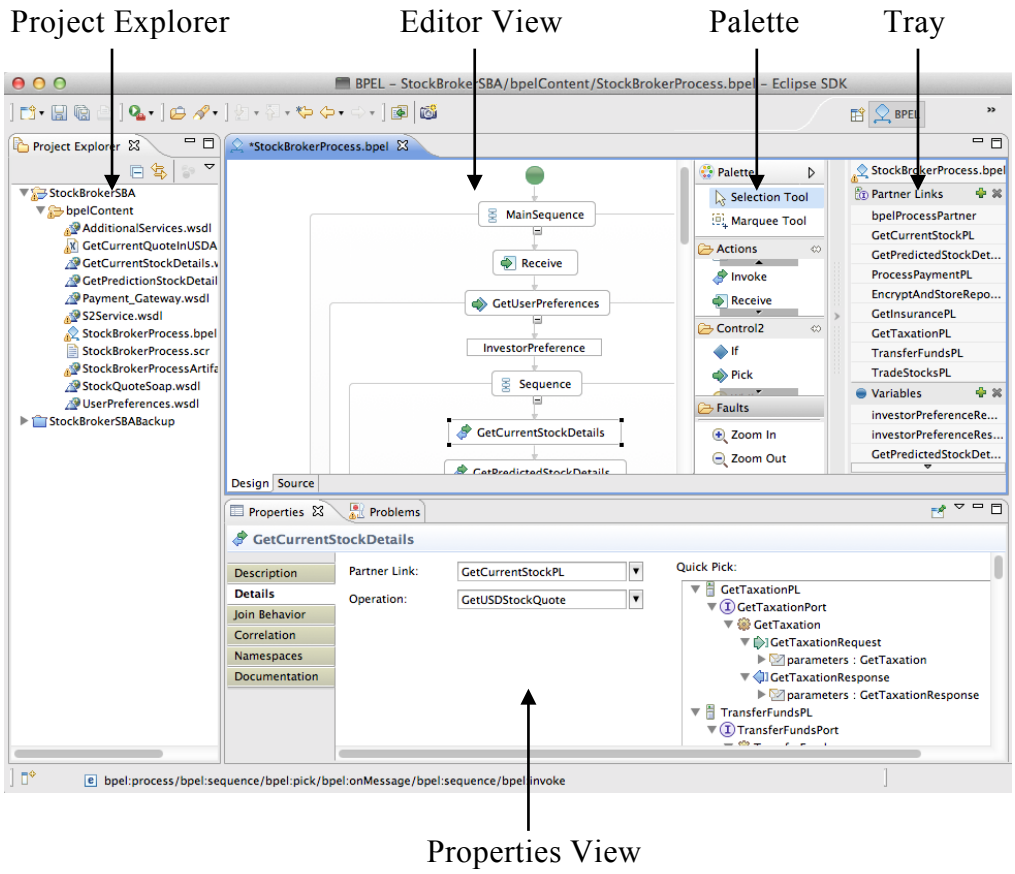


Figure 4.3: Screenshot of the BPEL Designer

Figure 4.3 shows a screenshot of the user interface of BPEL Designer. As shown in the figure, the user interface is divided in several parts, called Views. Starting from the top right we have the Project Explorer View, the Editor View (that contains two subparts, the Palette and the Tray) and the Properties View.

The Project Explorer lists the file resources part of a project, allowing opening, renaming, moving or deleting any resource.

Once a BPEL file is opened, the Editor View presents in the main part the graphical workflow representation of the BPEL process, allowing editing it. A Source tab on the bottom left of the view allows checking and editing the process code directly. The Palette contains the building blocks, i.e., BPEL activities that can be dragged and dropped in the main editing area in order to be added to the process. The Tray summarizes the BPEL process, listing all the elements part of the process, including the ones that do not have a graphical representation, allowing managing the entire process.

The Properties View provides detailed information of the selected element of the BPEL process, allowing editing them. For example, the Properties View of an `invoke` activity allows to select the partner link, port type, operation and variables of the operation of a partner service that should be invoked.

The BPEL Designer has been also integrated with Apache ODE in order to allow seamless deployment and execution of the produced BPEL processes in a BPEL execution engine.

Chapter 5

Secure Composition Patterns

5.1 Overview

The *secure composition patterns* are the part of the framework that is able to infer the needed security requirements. These are inferred for the parts of a composition and partner services involved in them. Inferences are driven by the security requirements on the whole composition. In other words, the inference process attempts to identify security requirements for the individual partner services which would be sufficient to guarantee the security requirements for the entire composition.

The secure composition patterns summarize some general security inferences on activity placeholders. Activity placeholders are instantiated by either other patterns or operations of individual partner services (when a pattern is instantiated to generate a executable service workflow).

More specifically a *secure composition pattern* contains three parts: (i) the *orchestration pattern* between activity placeholders representing the workflow on which the inferences apply (*WF* in the following), (ii) the security requirement requested for the composition (*RSP*) and (iii) the security requirements needed from the activity placeholders of the orchestration pattern to guarantee the security requirements for the whole

composition (*ASP*). Patterns may have a fourth optional part that expresses additional (boolean) conditions that need to hold in order for RSP to hold (*Conditions*), as in the case of the availabilty patterns in Section 5.3.4.

In order to avoid confusion between the orchestration pattern and the secure composition pattern concepts, in the context of this work we use the term *pattern* to indicate a secure composition pattern, and the term *orchestration* to indicate an orchestration pattern.

5.2 Orchestration Patterns

An *orchestration pattern* is a template specifying a service orchestration workflow with activity placeholders that can be bound to concrete service operations or to other orchestration patterns. These templates are based on the basic workflow patterns introduced by van Der Aalst, W.M. et al. [107] representing the control flow of orchestrations. The *orchestration patterns* augment the workflow patterns of [107] with a description of the data flow connecting the activities.

The same authors, in [90], have also defined the *workflow data patterns*, however these other patterns are not directly related with the control flow ones and are used to recognise the different mechanisms implemented by different workflows vendor to treat variables. In our approach we are only interested in which activities receive or send which data. The additional level of detail offered by the workflow data patterns (e.g., if the variables are sent/received in a pull or a push mode, if shared memory is used, …) was not required by the scope of the approach we are presenting, but it may be an interesting direction for future works as it may be helpful in order to explore and represent further security properties w.r.t. the ones in this thesis.

Our work focuses on a minimal set of workflow patterns that can be used to recursively build elaborate workflows. The concepts introduced by our approach, however, could be used on arbitrary workflow patterns (e.g. loops, handlers). Our initial set of workflow patterns include:

- the *sequential pattern*, which represents the execution of one activity after another one is completed. This can represent a set of BPEL `invoke` activities or further workflow patterns (i.e., BPEL non-atomic activities) connected by a BPEL `sequence` activity;

- the *choice pattern*, which represents the execution of one activity from a set of alternative activities based on some input value. This can represent a set of BPEL `invoke` activities or further workflow patterns (i.e., BPEL non-atomic activities) connected by a BPEL `pick` or `if-then-else` activity;

- the *parallel pattern* (or split-join), which represents the simultaneous execution of two or more activities. This can represent a set of BPEL `invoke` activities or further workflow patterns (i.e., BPEL non-atomic activities) connected by a BPEL `flow` activity.

The remaining BPEL atomic activities (i.e., `assign`, `receive` and `reply` activities) are used in our approach to encode the data flow.

As stated above, this set of workflow patterns can be used to recursively build elaborate workflows.

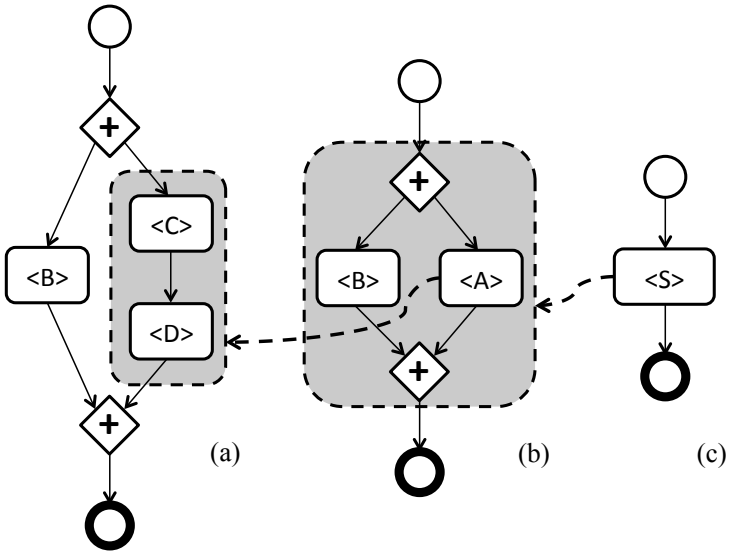


Figure 5.1: Example of workflow pattern recursion

An example of the recursion of workflow patterns is shown in Figure 5.1. Each workflow pattern contains a set of *activity placeholders* that can be either another pattern or an atomic activity. In the workflow shown in Figure 5.1(a), the sequential workflow pattern between C and D is highlighted. This pattern can be seen as a single activity instantiating the placeholder A in the parallel pattern between B and A shown in Figure 5.1(b). Likewise, the parallel pattern can be seen as a single activity instantiating the only activity in the workflow in Figure 5.1(c). This decomposition allows the secure composition patterns, which are based on the workflow patterns, to be used on arbitrary workflows.

The orchestration patterns used in this thesis are enriched versions of the basic workflow patterns described above. The orchestration patterns are enriched as they also describe the data flows between the activity placeholders that appear in a workflow pattern (this corresponds to the result of BPEL `assign`, `receive` and `reply` activities).

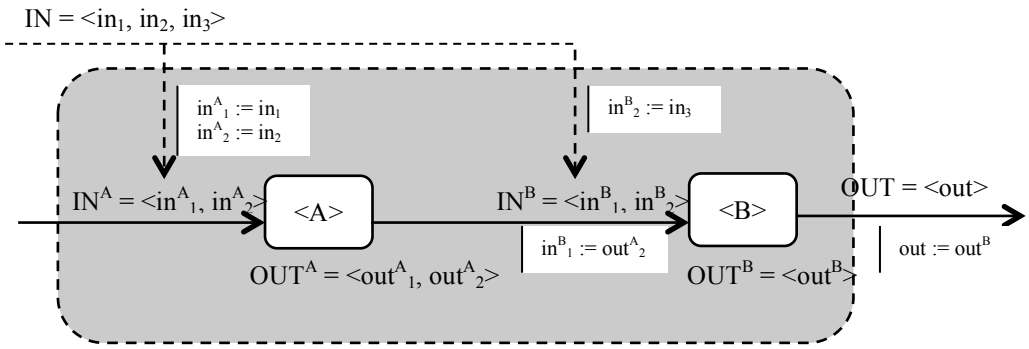


Figure 5.2: Example of a sequential orchestration pattern

An example of an orchestration pattern is provided in Figure 5.2, based on the sequential workflow pattern. The orchestration pattern shown in the figure represents an elementary control flow between two activity placeholders, i.e., A and B, that must be executed one after the other in the specific order shown in the figure (the order of execution of A and B is represented as a solid arrow in the figure). The data flow in this orchestration pattern is:

- An input message IN^A is passed to A that is part of the input message passed to the workflow IN. In particular the two parts in IN^A, in^A_1 and in^A_2, are taken from the first two parts of IN, in_1 and in_2.

- An input message IN^B is passed to B. IN^B comes partly from the input of the workflow IN and partly from the output of the first activity OUT^A. In particular the first part of IN^B, in^B_1, is taken from the second part of OUT^A, out^A_2, and the second part of IN^B, in^B_2, is taken from the third part of IN, in_3.

- The final output OUT is taken directly from the output of B, OUT^B.

Note that the data flow in the picture is just one of the possible data flows for this workflow; other ones can be obtained by changing the assignments. To represent an alternative data flow, a variant of the sequential orchestration pattern with the same control flow but different data flow is required, as shown in Figure 5.3.

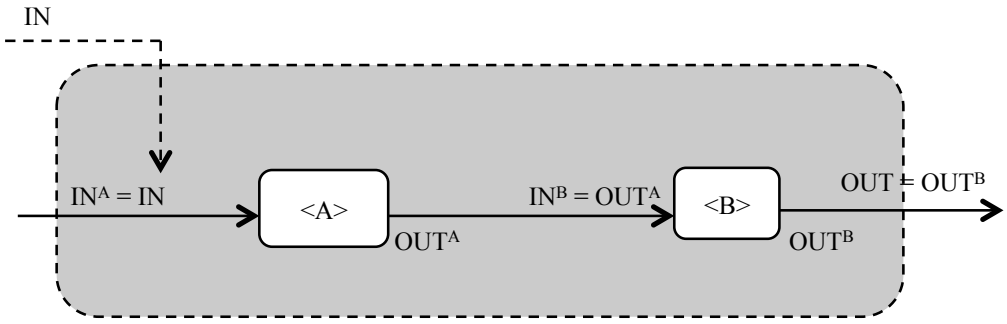


Figure 5.3: The Cascade orchestration pattern

More specifically Figure 5.3 shows a variant of the sequential orchestration pattern called Cascade. The Cascade orchestration pattern requires for all the inputs to the workflow (IN) to be consumed by the first activity (IN^A), for all the outputs of the first activity (OUT^A) to be consumed by the second activity (IN^B), and finally for all the outputs of the second activity (OUT^B), to be returned as output of the workflow (OUT).

The example in Figure 5.4, instead, is based on the parallel workflow pattern. This orchestration pattern specifies the simultaneous execution of just two activities, A and B, and where the data flow is set. In more detail, the data flow in the figure orchestration pattern states that:

- An input message IN^A is passed to A that is part of the input message passed to the workflow IN. In particular in^A is taken from the first part of IN, in_1.

- An input message IN^B is passed to B that is part of the input message passed to the workflow IN. In particular the two parts in IN^B, $in^B{}_1$ and $in^B{}_2$, are taken from the second and third parts of IN, in_2 and in_3.

- The final output OUT comes partly from the output message of A, OUT^A, and partly from the output message of B, OUT^B. In particular the first part of OUT, out_1, is taken from the second part of OUT^A, $out^A{}_2$, and the second part of OUT, out_2, is taken from the only part of OUT, out^B.

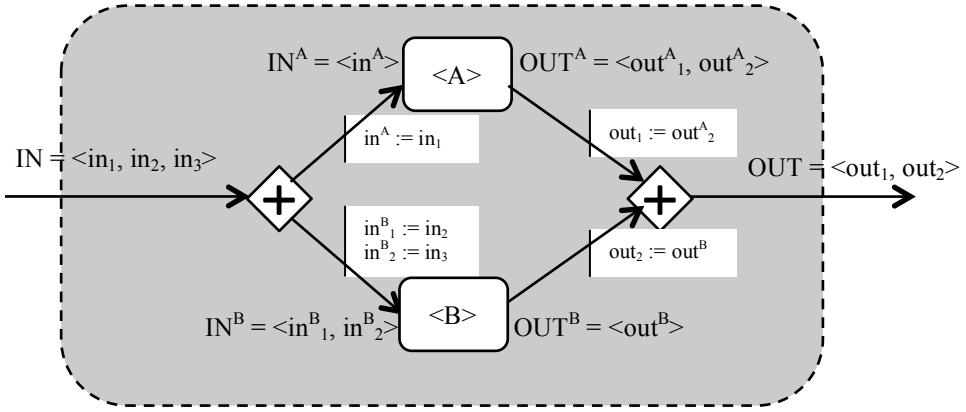


Figure 5.4: Example of a parallel orchestration pattern

5.3 Secure Composition Patterns

The orchestration patterns are used to describe cases on which exists some inference between security requirements at the composition and the security requirements for the individual activity of the orchestration. These inferences should be proved or verified through formal methods, to be able to respect the security requirement definition.

As an example, take a composition where a payment is handled by two different services based on if the payment card is a debit or a credit card. A

security requirement for such composition is to treat the information about the payment card (i.e., card number, expire date) with confidentiality. As this information is used by both branches of the choice orchestration between the two services, then two security requirements are generated, one for each service, asking to treat the data that they receive with confidentiality. Bear in mind that if a service in the workflow would not use such data (as for example a third branch to pay with PayPal), it should not be asked to respect security requirements w.r.t. data it does not use.

5.3.1 Representation of a Secure Composition Pattern

As mentioned in the overview, each *secure composition pattern* contains three parts: (i) the *orchestration pattern* representing the workflow on which the inferences apply, called *WF*, (ii) the security requirement requested for the composition, called *RSP,* and (iii) the security requirements needed from the activity placeholders of the orchestration pattern to guarantee RSP, called *ASP*. Patterns may have a fourth optional part that expresses additional (boolean) conditions that need to hold in order for RSP to hold (*Conditions*) and that can be verified only after a workflow has been fully instantiated with services.

Figure 5.5 shows an example of the graphical notation we use to describe a secure composition pattern. In the WF part of the table, the orchestration pattern P is shown, describing both the control and the data flow of the pattern between activity placeholders. ASP contains the security requirements that need to hold on each placeholder ($SecReq^X$) in order for the security requirement on P, described in the RSP part of the table, to hold.

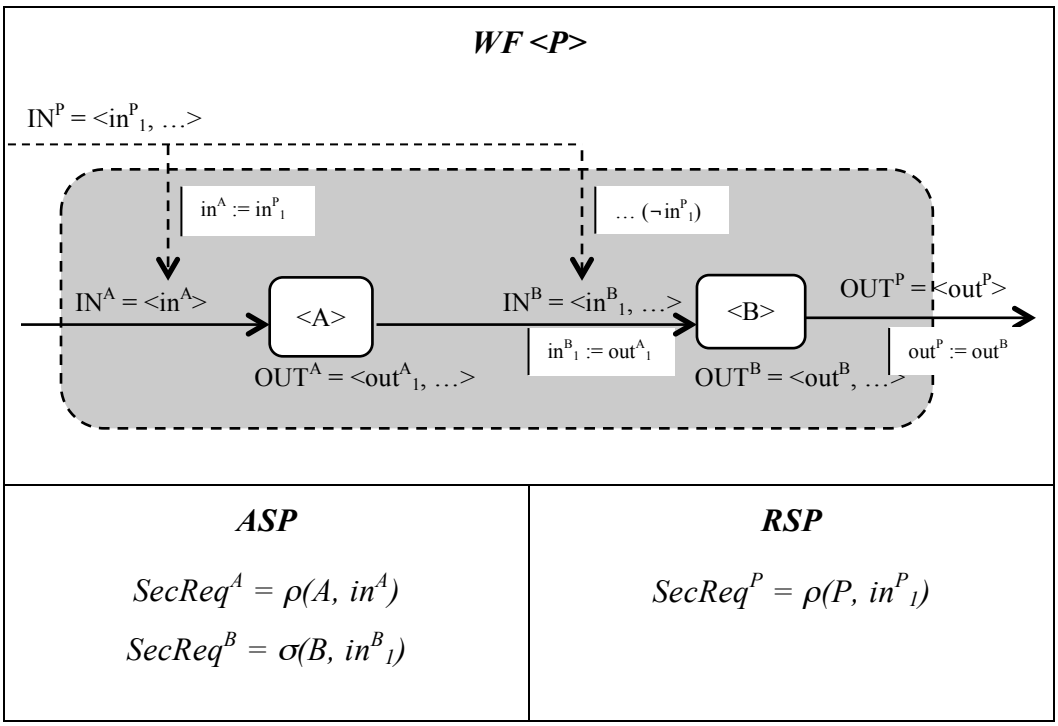


Figure 5.5: Example of a secure composition pattern

The example in Figure 5.5 shows a secure composition pattern where ρ and σ are security properties that hold on their first parameter. This pattern is about preserving property ρ in one of the variants of the sequential orchestration, as stated in the security requirement $SecReq^P$. In more detail, when property ρ is required on an input of the pattern in^P_1, and this input is used only as input of the first activity placeholder of the orchestration (in^A), the security requirements needed to guarantee $SecReq^P$ are ρ on in^A for A ($SecReq^A$) and σ on in^B_1 for B ($SecReq^B$). This second requirement could be necessary, for example, because information about in^A (and so about in^P_1) is part of A's output (i.e., out^A_1=in^B_1).

Generally speaking, we want for RSP to hold if all the conditions in ASP hold, meaning that we need a proof for the proposition:

$$ASP.SecReq_A \textbf{ and } ASP.SecReq_B \textbf{ and } \ldots \Rightarrow RSP.SecReq_P$$

So, in the case of Figure 5.5, we would need a proof for:

$$\rho(A, in^{A}) \textbf{ and } \sigma(B, in^{B}{}_{1}) \Rightarrow \rho(P, in^{P}{}_{1})$$

When a secure composition pattern is used to infer the security requirements for the pattern's placeholders, however, the logical implication is used in the "opposite" direction, i.e, we identify the security requirements for the individual pattern's placeholders using the following rule:

$$\text{RSP.SecReq}_{P} \Rightarrow \text{ASP.SecReq}_{A} \textbf{ and } \text{ASP.SecReq}_{B} \textbf{ and } \ldots$$

The rational for reversed ordering of the logical implication expressed in the formula above is that if ASP.SecReq_{A} and ASP.SecReq_{B} and … hold then RSP.SecReq_{P} will also hold, and therefore, the security aware service composition process has to check the requirements ASP.SecReq_{A} and ASP.SecReq_{B} and … are respected, as this would guarantee that their composition would satisfy RSP.SecReq_{P}. In other words, the security aware service composition process is driven by the verification of the sufficient conditions for a composition level security property to hold rather than the necessary conditions.

5.3.2 Integrity

The secure composition pattern described in this section is about preserving integrity on the Cascade orchestration pattern. In the scope of this research, we adopt the following data integrity definition, taken from RFC4949 [44]:

"The property that data has not been changed, destroyed, or lost in an unauthorized or accidental manner."

More specifically, the formal definition of integrity we used in the context of this work takes advantage of the *precede* property:

Definition 1: Given a set of traces of actions T, where each action can appear only once in each trace, the *precede* property between action a and action b ($precede(a, b)$) holds if and only if ∀ traces $\omega \in$ T with $b \in \omega$, $\omega = \langle \dots, a, \dots, b, \dots \rangle$ holds.

The traces used in Definition 1 are, in our case, traces of a single interaction between a client and a service, where each communication is split in *send* and *receive* actions.

Definition 2: Given a service *S*, with input *x* and output *y*, the *precede integrity* property $Integrity_{pr}(S, x, y)$ holds if and only if ∀ trace of communication between S and a client C, *precede(send(C, S, x), receive(C, S, y))* holds.

In other words, precede integrity holds if whenever a client receives *y* (i.e., *f(x)*) from service S, then the client has previously sent *x* to S.

5.3.2.1 Precede Integrity on Cascade Pattern

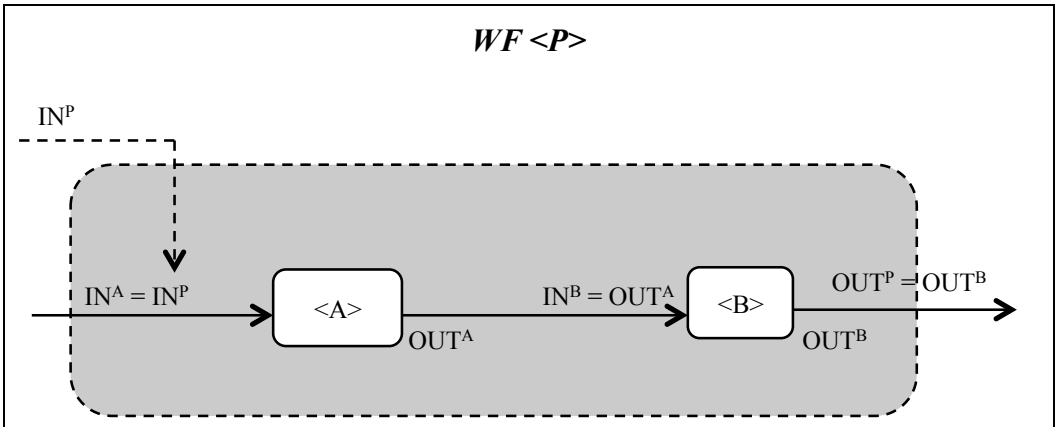
As shown in Figure 5.6, when the security requirement over the process portion encoded by the orchestration pattern P, called $SecReq^P$, requests to preserve the integrity of the orchestration's input and output data (IN^P and OUT^P) through Precede Integrity, given that the orchestration pattern is the Cascade orchestration, i.e.:

(i) the set of inputs of the first activity placeholder IN^A is equal to the set of inputs of the pattern IN^P

(ii) the set of inputs of the second activity placeholder IN^B is equal to the set of outputs of the first activity placeholder OUT^A, and

(iii) the set of outputs of the pattern OUT^P is equal to the set of outputs of the second activity placeholder OUT^B

Then the security requirements needed to guarantee $SecReq^P$ are: (a) Precede Integrity on A inputs and outputs ($SecReq^A$) and (b) Precede Integrity on B inputs and outputs ($SecReq^B$).



ASP	RSP
$SecReq^A = Integrity_{pr}(A, IN^A, OUT^A)$ $SecReq^B = Integrity_{pr}(B, IN^B, OUT^B)$	$SecReq^P = Integrity_{pr}(P, IN^P, OUT^P)$

Figure 5.6: Precede Integrity on Cascade Pattern

5.3.2.2 Orchestrator Requirements

In order to maintain data integrity, it is necessary to ensure that all the actors that handle the data do not tamper with the data.

The proof of the relation expressed by the pattern in this case (and in fact in all other cases) requires the introduction of one more concept, namely the concept of an *orchestrator*. An orchestrator in this context expresses the actual environment (e.g., middleware) that will execute the workflow expressed by a secure composition pattern, invoking individual services, passing data across them (i.e., receiving the outputs of one service S^1 that are to be received as inputs by another service S^2 and passing them

to S^2), and handling interactions with the external actors interacting with the workflow (i.e., receiving data from these actors and passing them to the service they were intended for, and receiving data from workflow services and passing them to the external actors they were intended for).

Given the presence of an orchestrator, the proof for the integrity pattern makes it also necessary to make a key assumption about the new component. This assumption is that the orchestrator itself is trustworthy, i.e., it does not tamper with the data that go through it. An orchestrator O will be defined to be trustworthy with respect to the Precede Integrity on Cascade Pattern if the following properties hold:

i. O sends to service A exactly the same data that it has received from a client for P,

ii. O sends to service B exactly the same data service A sent to it and

iii. O sends to the client exactly the same data it received from B for the client.

5.3.2.3 Proof

The security requirements about activity placeholder A and activity placeholder B (i.e., the hypothesis) can be translated into:

$$Integrity(A, IN^A, OUT^A) = precede(send(C^A, A, IN^A), receive(C^A, A, OUT^A))$$

$$Integrity(B, IN^B, OUT^B) = precede(send(C^B, B, IN^B), receive(C^B, B, OUT^B))$$

A and *B*, within the pattern, have as a client the orchestrator that execute the workflow. This means that $C^A = Orch$ and $C^B = Orch$, so we have:

$$precede(send(Orch, A, IN^A), receive(Orch, A, OUT^A)) \quad \text{(P1)}$$

precede(send(Orch, B, IN^B), receive(Orch, B, OUT^B)) (P2)

In addition, as mentioned in the previous section, the orchestrator is required to treat the data that it passes between endpoints with integrity, i.e., it must satisfy the following properties:

precede(send(C, Orch, Data), send(Orch, A, Data)) (P3)

precede(receive(Orch, A, Data), send(Orch, B, Data)) (P4)

precede(receive(Orch, B, Data), receive (C, Orch, Data)) (P5)

More specifically, (P3) corresponds to point i., (P4) to point ii. and (P5) to point iii. of the previous section.

Following Definition 1, it is possible to prove that *precede* is a transitive property, i.e.: $precede(a,b) \wedge precede(b,c) \Longrightarrow precede(a,c)$. In fact, from $precede(b,c)$ we know that $\forall\omega \in$ T with $c \in \omega$, but also that $b \in \omega$ as $\omega = \langle \dots, b, \dots, c, \dots \rangle$. Then, due to $precede(a,b)$ we know that $\omega = \langle \dots, a, \dots, b, \dots \rangle$, so we can conclude that $\omega = \langle \dots, a, \dots, b, \dots, c, \dots \rangle$.

Thanks to the transitivity of *precede*, from (P3), (P1), (P4), (P2), (P5) we know that the following property holds:

precede(send(C, Orch, IN^P), receive(C, Orch, OUT^P))

Since the orchestrator is just the executor of the pattern, this property corresponds to:

precede(send(C^P, P, IN^P), receive(C^P, P, OUT^P)) = Integrity(P, IN^P, OUT^P)

that is the security property required to hold for the orchestration pattern.

5.3.3 Confidentiality

One of the main topics studied in the information flow field is confidentiality as shown for example in the survey in [91].

In information flow, data activities are classified in *low*- and *high-level*. A data activity is regarded as low-level if the information about it is public. A data activity is regarded as a high-level if the information about it is secret.

Similarly, users are classified in *low-level security users* and *high-level security users*. Low-level security users are users who are able to access only public information, whilst high-level security users are users who can access both public and secret information. In information flow approaches there are several property definitions whose intent is to express the concept of confidentiality. The main difference between them is about what the low-level users are forbidden to know or discover about the high-level data activity (e.g. no information, just that an input activity happened or exactly which inputs were passed). In the following, we provide some of these definitions that we use in our secure information flow patterns.

Separability [63]: Separability is a form of confidentiality that requires complete independence between the high- and low-level sequences of activities. To achieve this it is necessary that all the high-level data activity can be interleaved in any position of the trace of the low-level activities, and that all values for the high-level data activity must be possible for any low-level trace. This means that there is absolutely no interaction between high-level and low-level data activities, i.e., the high-level and low-level data activities should be processed by separate system processes during the operation of a system without any communication whatsoever between them.

In most of the cases, separability is a too strong definition for confidentiality, as it does not allow high-level data outputs to depend on any low-level activity.

As an example, take a system where the low-level user activity (both inputs and outputs) is logged and sent as high-level output to the administrator (high-level user). This system is obviously secure, but do not respect the separability property.

To address this, other forms of confidentiality have been defined in literature.

Non-inference [71]: Non-inference (note that is different from non-interference) is a property stating that removing all high-level data activity from any trace results in another valid trace. This means that a low-level user, who is able to see just low-level data activity, cannot deduce the occurrence of any high-level data activity by just observing a trace.

This definition considers the previous example as secure, as traces with or without administrator checking the logs are valid. Non-inference, however, is too weak as argued in [63], as in this definition there is no check that high-level data inputs to a process are not revealed through low-level data outputs of it; in the absence of such checks there may be a leak of high-level security data.

As an example, take into consideration a system that may receive a confidential credit card number C_H as input, and that whenever the low-level user requests it, it receives from the system a string S_L that is either the credit card number C_H or a random string. This system does respect non-inference, as removing occurrences of C_H from a trace leads to a behaviour that is still possible, however the system cannot be said to be secure, as the low-level user can infer when the high-level input have occurred.

Perfect Security Property (PSP): According to [114], a system has "perfect security" if for any low-level trace observed the following two conditions hold: (i) all interleaving of high-level input in a trace are valid traces and (ii) high-level outputs can be inserted anywhere in a trace (if possible) and might depend on low-level activity, leading to valid traces.

PSP is a weaker version of separability as, due to condition (ii), it allows the high-level outputs to depend on low-level events. It is, however, stronger than non-inference as, due to condition (i), high-level inputs cannot be used to compose low-level outputs (as all the high-level input interleaving must be possible with the same low-level outputs).

From the above definitions, in fact, it is possible to prove the following, as described in [59]:

$$Separability \Rightarrow PSP \Rightarrow Non\text{-}inference \qquad (R1)$$

Furthermore, PSP can be proven to be the weakest property where the low-level user cannot determine anything about high-level activity. The interested reader might refer to [59] for proofs and comparisons with other properties.

In the following, we present secure composition pattern that can guarantee PSP in a service workflow. The patterns cover sequential and parallel orchestration.

5.3.3.1 Notation

In the following, let P be the composition of two activities, A and B; IN^X and OUT^X be the sets of inputs and outputs for an activity placeholder X, with $X \in \{ A, B, P \}$; $E^X = I^X \cup O^X$; and V^X and C^X be two subsets of E^X that partition it into its public/visible (i.e., low-level) V^X and confidential (i.e., high-level) C^X parts.

5.3.3.2 PSP on Cascade Pattern

As shown in Figure 5.7, when the security requirement $SecReq^P$ over the process portion encoded by a Cascade orchestration P (see Section 5.2 and 5.3.2.1 for a description of the Cascade orchestration) requests PSP confidentiality for a portion of input/output parameters C^P, then the security requirements $SecReq^X$ needed to guarantee $SecReq^P$, for $X \in \{ A, B \}$, are that PSP holds with: (a) the public actions of X are part of the public actions of P (i.e., $V^X \subseteq V^P$), and (b) the confidential actions of X do not include any public action of P (i.e., $C^A \cap V^P = \varnothing$). These conditions ensure that the low-level user cannot see any difference between the public trace of P and the public trace of the single service P is representing.

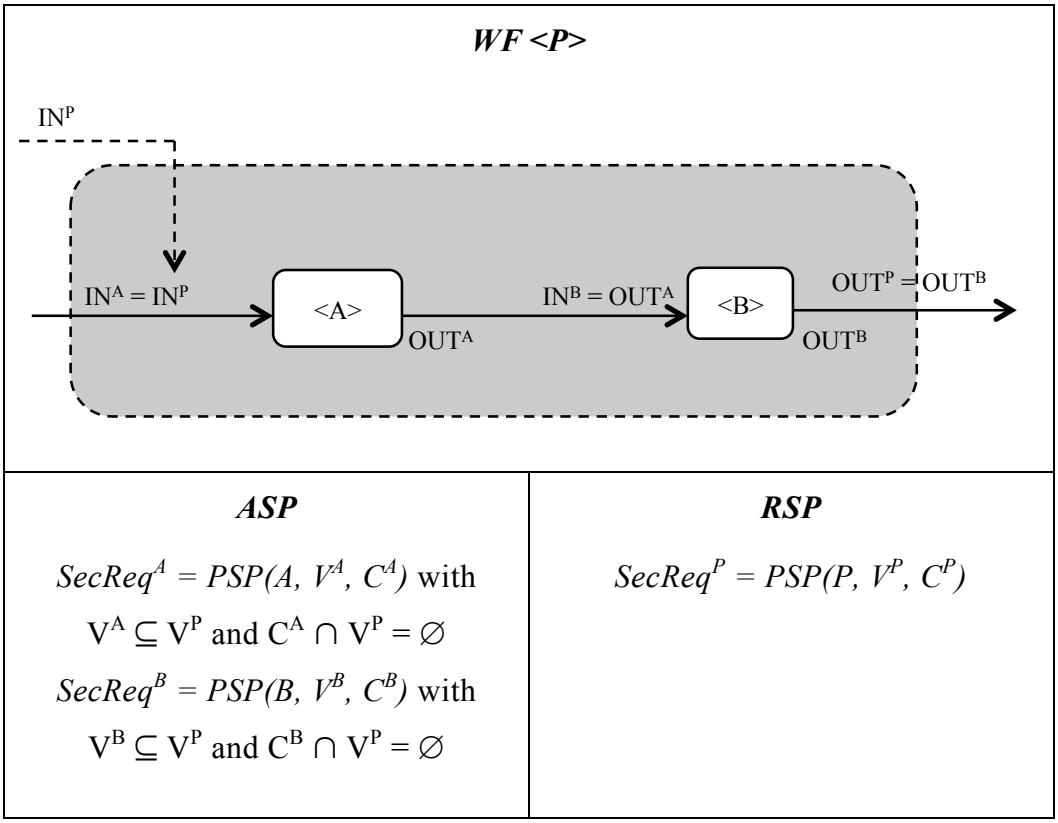


Figure 5.7: PSP on Cascade Pattern

5.3.3.3 PSP on Product Pattern

Similarly, as shown in Figure 5.8, when the security requirement over the process portion encoded by the orchestration pattern P, called $SecReq^P$, requests to preserve the confidentiality of a portion of the pattern's input and output data C^P through PSP, given that the orchestration pattern is the Product orchestration, i.e.:

(i) a parallel orchestration where the activity corresponding to placeholder A and the one corresponding to placeholder B are executed simultaneously,

(ii) the sets of inputs of the two activity placeholders, IN^A and IN^B, are a partition of the set of inputs of the pattern IN (i.e., $IN^A \subseteq IN^P$, $IN^B \subseteq IN^P$, and $IN^A \cap IN^B = \emptyset$), and

(iii) the sets of outputs of the two activity placeholders, OUT^A and OUT^B, are a partition of the set of outputs of the pattern OUT (i.e., $OUT^A \subseteq OUT^P$, $OUT^B \subseteq OUT^P$, and $OUT^A \cap OUT^B = \emptyset$).

Then the security requirements $SecReq^X$ needed to guarantee $SecReq^P$, for $X \in \{ A, B \}$, are that PSP holds with: (a) the public actions of X are part of the public actions of P (i.e., $V^X \subseteq V^P$), and (b) the confidential actions of X do not include any public action of P (i.e., $C^A \cap V^P = \emptyset$). These conditions ensure that the low-level user cannot see any difference between the public trace of P and the public trace of the single service P is representing.

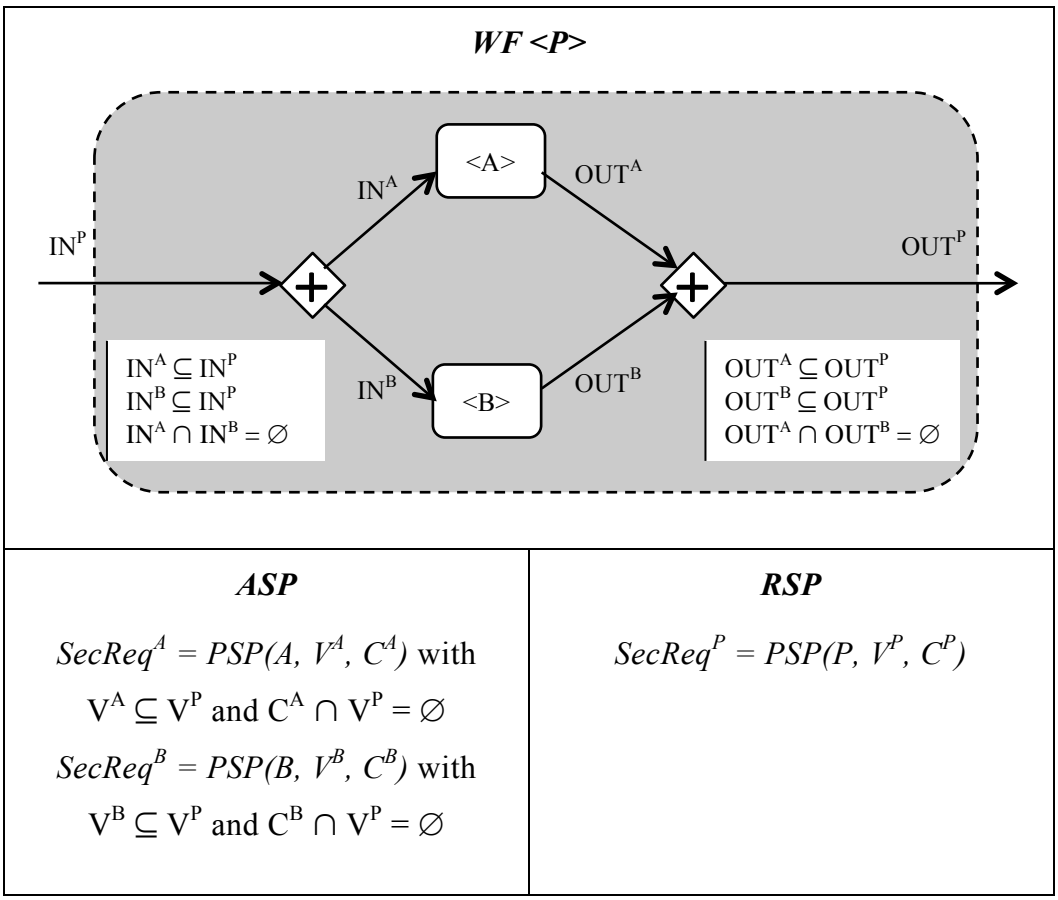


Figure 5.8: PSP on Product Pattern

5.3.3.4 Orchestrator Requirements

As for the Integrity, the orchestrator must be trustworthy in order for the security requirements to be preserved. In particular, for the given patterns of PSP, an assumption made in proving the pattern is that the orchestrator should not change the level of the data activities. This means that it should only pass the data without any modification (e.g., encrypting or decrypting the data), or passing the data to any entity other than the ones present in the pattern.

5.3.3.5 Proofs

The proofs for the patterns for PSP are based on the composition results in [59]. We report here the relevant conclusions (page 7, under First Class of Compositionality Results):

"Three main approaches to satisfy the first condition of Lemma 1 […] Following the second approach (security property ensures $N_1=\varnothing=N_2$), we obtain that *noninference (...), separability (...), and the perfect security property (...) are preserved under arbitrary compositions*"

More specifically, the compositions took into consideration in the cited work coincide with the Product Pattern and the Cascade Pattern. Furthermore non-inference and separability support the same kind of result, meaning that security composition patterns similar to the ones presented can be created also for these properties.

In order for Lemma 1 to hold, however, there are a set of assumptions, i.e., that (1) $V^P \cap E^X = V^X$ and (2) $C^P \cap E^X \subseteq C^X$, with $X \in \{A, B\}$.

In order for (1) to hold, we require the two conditions in SecReq^X, that are: (i) the public actions of X are part of the public actions of P ($V^X \subseteq V^P$), and (ii) the confidential actions of X do not include any public action of P ($C^X \cap V^P = \emptyset$). In fact, (i) is true if and only if $V^P \cap V^X = V^X$. Then, by (ii) we have $(V^P \cap C^X) \cup (V^P \cap V^X) = V^X$. By the distributive law $V^P \cap (C^X \cup V^X) = V^X$. Since $(C^X \cup V^X) = E^X$, then we can conclude that (1) $V^P \cap E^X = V^X$ holds.

Regarding (2) instead, only the condition (i) for SecReq^X, $V^X \subseteq V^P$, is required. In fact, from the fact that V^P and C^P partition E^P, we know that $C^P \cap V^P = \emptyset$. So by (i) we can say that $C^P \cap V^X = \emptyset$. Furthermore, we know that $C^P \cap C^X \subseteq C^X$ holds, as it is a basic property of the subset

definition. Obviously $\emptyset \cup (C^P \cap C^X) \subseteq C^X$ still holds, so we can substitute the empty set with the one we created earlier, obtaining $(C^P \cap V^X) \cup (C^P \cap C^X) \subseteq C^X$. By the distributive law we obtain $C^P \cap (V^X \cup C^X) \subseteq C^X$ that, since $(V^X \cup C^X) = E^X$, then we can conclude that (2) $C^P \cap E^X \subseteq C^X$ holds.

So, if these conditions are met, then as proven in [59], separability, PSP and non-inference are preserved under both *parallel* and *sequential* composition of activities.

Note also that, as a consequence of the relation (R1), the secure composition patterns for the PSP are valid also in cases where:

- $SecReq^P = Non\text{-}inference(P, V^P, C^P)$ as PSP ⇒ Non-inference
- For X = A or B, $SecReq^X = Separability(X, V^X, C^X)$ with $V^X \subseteq V^P$ and $C^X \cap V^P = \emptyset$, as Separability ⇒ PSP.

5.3.4 Availability

The next secure composition pattern is a pattern for *availability*. In the scope of this research, we adopt the following availability definition, taken from RFC4949 [44]:

"The property of a system or a system resource being accessible, or usable or operational upon demand, by an authorized system entity, according to performance specifications for the system; i.e., a system is available if it provides services according to the system design whenever users request them."

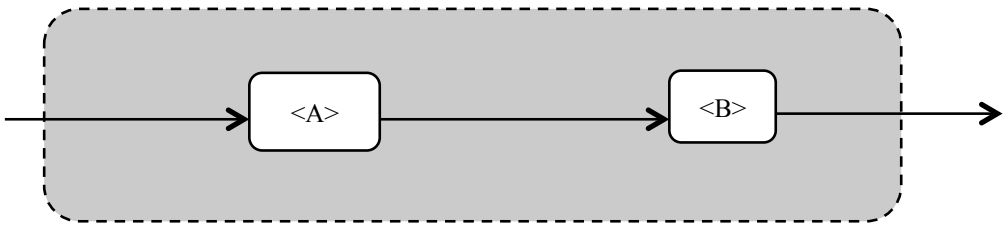
Availability conveys into the security area several concepts about quality of service (QoS), where the "performance specifications" are service level agreements (SLAs) that have been certified by an external authority.

Properties under the category of availability are different from the ones presented so far, as they involve the computation of one or more measure for the given service (e.g., execution time, throughput, uptime probability) and a constraint (boundary) for the values of these measures.

In the following, we give a set of secure composition patterns for availability using as a basis the measure of execution time. However, patterns for other measures of availability could in principle be defined in a similar way.

5.3.4.1 Maximum Execution Time on Generic Sequential Pattern

Figure 5.9 shows the Maximum Execution Time on Sequential Workflow Pattern, where the security requirement $SecReq^{P}$ over the Generic Sequential orchestration P (i.e., an orchestration based on the sequential workflow pattern, but where no data flow specification is given) requests that the maximum execution time is less than a given number x_P. In this case the security requirements needed to guarantee $SecReq^{P}$ are that the maximum execution times for the first activity placeholder A and the second activity placeholder B are x_A and x_B, and that the sum of x_A and x_B is equal or less than x_P.

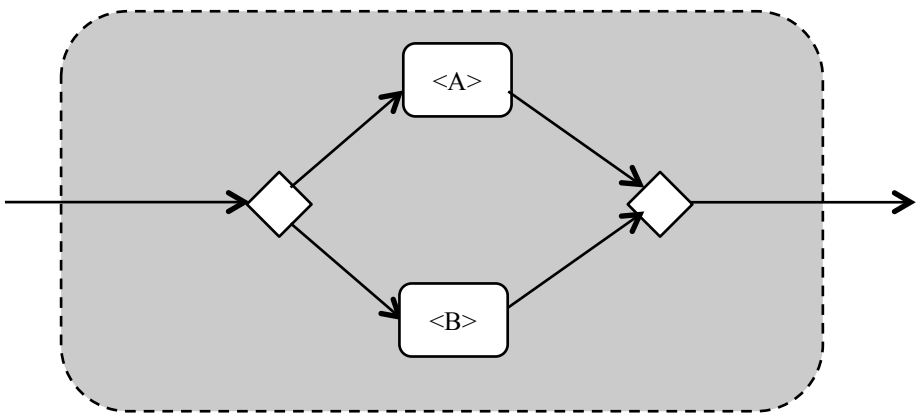


ASP	***RSP***
$SecReq^{A} = Avail(A, maxTime = x_A)$ $SecReq^{B} = Avail(B, maxTime = x_B)$	$SecReq^{P} = Avail(P, maxTime = x_P)$
Condition $x_P \geq x_A + x_B$	

Figure 5.9: Maximum Execution Time on Generic Sequential Pattern

5.3.4.2 Maximum Execution Time on Generic Choice Pattern

Figure 5.10 shows the Maximum Execution Time on Choice Workflow Pattern, where the security requirement $SecReq^{P}$ over the Generic Choice orchestration P (i.e., an orchestration based on the choice workflow pattern, but where no data flow specification is given) requests that the maximum execution time is less than a given number x_P. In this case the security requirements needed to guarantee $SecReq^{P}$ are that the maximum execution times for activity placeholders A, B are x_A, x_B and that the maximum value between x_A, x_B is equal or less than x_P.



ASP	***RSP***
$SecReq^{A} = Avail(A, maxTime = x_A)$ $SecReq^{B} = Avail(B, maxTime = x_B)$	$SecReq^{P} = Avail(P, maxTime = x_P)$
Condition	
$x_P \geq max(x_A, x_B)$	

Figure 5.10: Maximum Execution Time on Generic Choice Pattern

5.3.4.3 Orchestrator Requirements

The orchestrator influences almost all the availability dimensions. In this sense, the orchestrator impact may have to be part of the pattern. For the maximum execution time, for example, the time that will take to execute the composition code should be added to the computed time x_P.

5.3.4.4 Proofs

The proofs of the patterns described for availability are trivial, as based on the behaviour of the numerical dimension of QoS taken into consideration and on the orchestration, a different aggregation formula can

be defined. The interested reader may find a summary of aggregation formulas for QoS properties utilising workflow patterns in [46].

5.4 Security Inference Rules

As explained in Section 5.3, the formally proven secure composition patterns are encoded by production rules expressed in Drools (see Section 4.3). This makes it possible to apply the patterns in an automated manner in the composition process and derive dependencies between security requirements of services using rules, instead of trying to derive these relations from first principles (i.e., by re-constructing the proofs underpinning the patterns) at runtime. The latter would be a computationally expensive process.

5.4.1 Methodology to Encode the Rules

In this section we present some guidelines about how to encode the secure composition patterns into Drools production rules.

We distinguish between two kinds of rules: *inference rules* and *verification rules*. The inference rules can be used to generate security requirements for activity placeholders from a security requirement over a pattern, by using the *WF*, *ASP* and *RSP* parts of a pattern. Inference rules are used as part of the process of generating secure service compositions to replace unavailable services in operating service workflows at runtime or to generate possible secure service workflows during the design of service oriented systems. Verification rules are used to verify the *Conditions* of the patterns through the security properties that hold for the partner services that participate in a service workflow.

5.4.1.1 Inference Rules

The set of rules that generates security requirements for activity placeholders of an orchestration pattern (ASP) in order to guarantee a security requirement over the pattern (RSP) are called inference rules. These rules can be used either when the workflow is not yet instantiated with the partner services or when the workflow is instantiated, in order to generate the security requirements to request during the instantiation phase or to check if they are satisfied by the partner services.

The inference rules encode three parts of secure composition patterns: (i) the orchestration pattern on which the inferences apply (WF), (ii) the security requirement requested for the composition (RSP) and (iii) the security requirements for the activity placeholders that guarantee the requested requirement (ASP). The first two parts (WF and RSP) should be encoded as conditions for the rule to be applied (the `when` part of the rule). The ASP part of the pattern should be encoded as the consequence of the rule (the `then` part of the rule), since it determines which requirements should be added in order for the RSP requirement to hold. In fact, as a consequence of pattern proofs asserting that:

$$ASP \Rightarrow RSP$$

production rules are expressed as:

If RSP is required of WF **Then** require ASP

as if ASP holds then RSP would also hold as a consequence of it.

In order to support the encoding of secure composition patterns, we defined a set of classes that are used to represent orchestration patterns and security requirements. These classes are shown in Figure 5.11 and Figure

5.12 and classes represent the vocabulary used for the security production rules.

The *Placeholder* is the basic building block of a pattern, representing an activity placeholder that supports a set of input and output *Parameters* listed in its *parameters* field.

The *Placeholder* can be differentiated in three subtypes:

1. *UnassignedActivity* – a service invocation placeholder. It contains the structural description (i.e., the WSDL, in the *wsdl* field) that needs to be matched to properly instantiate the required functionality.

2. *PartnerLinkActivity* – a service invocation already bound to a partner service. It contains information about the service bound to the activity and an array listing the security properties certified for that service (in the *certifiedProperties* field).

3. *OrchestrationPattern* – control flow constructs handling further *Placeholder*s. In the context of this work, three main orchestration patterns have been defined (see also Section 5.2): the *Sequential*, the *Parallel* and the *Choice* patterns.

Through the usage of the classes described above, it is possible to encode the orchestration pattern part of a secure composition pattern.

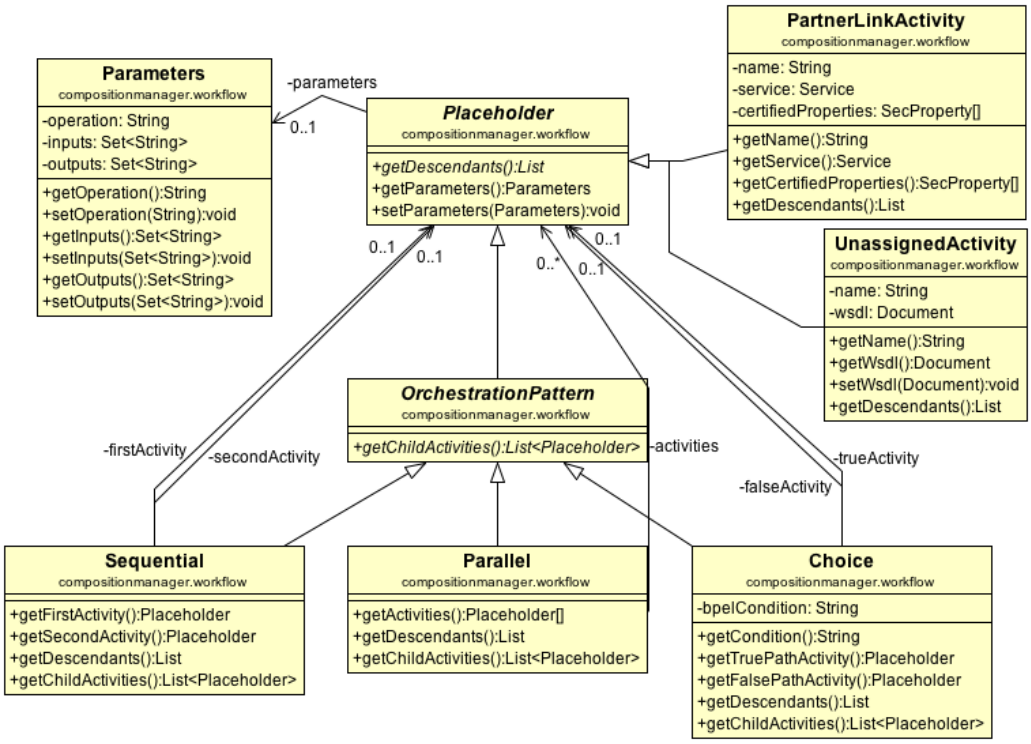


Figure 5.11: Class diagram of the activity and pattern classes available for the security production rules

As an example, we are going to encode the pattern shown in Figure 5.5 into a security production rule.

The first task is to encode the orchestration pattern (WF) into the rule, as shown in the snippet in Table 5.1. The snippet matches the sequential orchestrations (lines 12-14) that have, as a first activity, a Placeholder that takes as input an input of the orchestration (lines 5-6) and that has, as second activity an Placeholder that takes as input one of the outputs of the first activity, but not the input of the orchestration, and that outputs the output of the orchestration (lines 8-10).

```
$inP1 : String( )
$outA1 : String( )
$outP : String( )

$A : Placeholder( parameters.inputs contains $inP1,
             parameters.outputs contains $outA1 )

$B : Placeholder( parameters.inputs contains $outA1,
             parameters.inputs not contains $inP1,
             parameters.outputs contains $outP)

$WF : Sequential( firstActivity == $A, secondActivity == $B,
             parameters.inputs contains $inP1,
             parameters.outputs contains $outP)
```

Table 5.1: Snippet encoding an orchestration pattern into a Drools rule

The security requirements can be expressed, instead, through the classes in Figure 5.12. A *Requirement* represents a security requirement of a security property (*secProperty* field) for a certain *Placeholder* (*subject* field). It can optionally contain a set of *Parameters*, indicating on which inputs or outputs the security property should hold, and further requirements (in the *inferredReqs* field) that have been generated in order for this one to hold. The *satisfied* field keeps track if the security requirement has been checked and it is guaranteed by a certified security property.

The *SecProperty* class represents a security property, containing the security property name (*propertyName* field) and an optional set of attribute-value fields allowing expressing extra conditions over the property (*attributesMap* field).

The *SecPlan* class represents a set of security requirements that are requested to hold at the same time (conjunction). Each *SecPlan* object can hold a different set of inferred requirements for the same initial requirements, allowing expressing different options to guarantee a set of

requirements (disjunction); more details about how this is used can be found in Section 6.4.1.

The *SecPlan* class includes also the method *isAtomic()* allowing checking if the requirements contained have been successfully inferred to the workflow's leaves, i.e., the *UnassignedActivities* and *PartnerLinkActivities*.

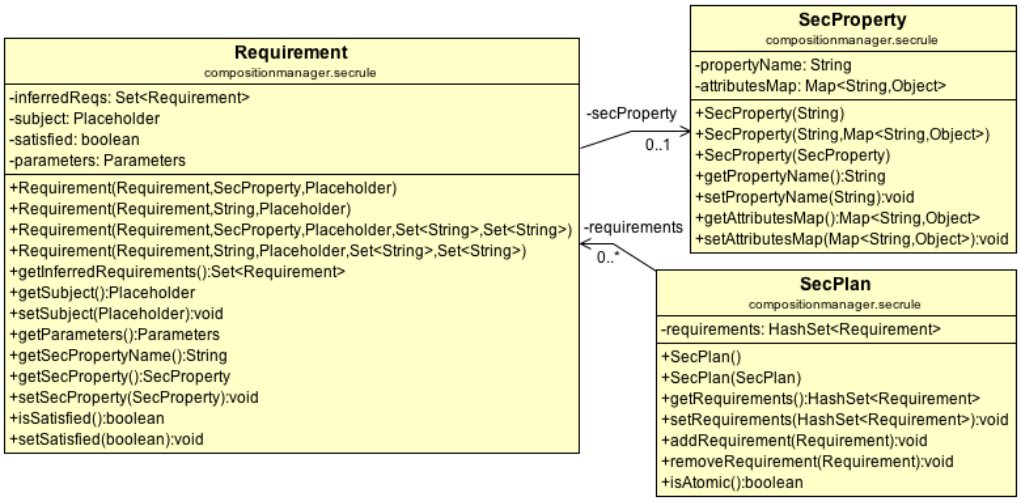


Figure 5.12: Class diagram of the requirement and security property classes available for the security production rules

Following the example above, Table 5.2 shows the encoding of the pattern security requirement (RSP) from Figure 5.5 into a security production rule.

The snippet matches the Requirement (lines 1-4) of the security property named "`rho`" for the input `$inP1` of the pattern `$WF` described in the previous snippet. As this is a rule to infer finer grained requirements from a more generic one, the Requirement should not be already satisfied and it must be part of a SecPlan (lines 6-7) that is not completely propagated (i.e., not atomic).

```
1  $RSP : Requirement( secProperty.propertyName == "rho",
2                 subject == $WF,
3                 parameters.inputs contains $inP1,
4                 satisfied == false )
5
6  $secPlan : SecPlan( requirements contains $RSP,
7                 atomic == false )
```

Table 5.2: Snippet encoding a security requirement into a Drools rule

The consequence part of the rule should encode the actions that generate the requirements over the activity placeholders (ASP) that guarantee that the requirement over the pattern holds.

Returning to our on-going example, Table 5.3 shows a snippet encoding the consequences part of the rule. The code generates the two new security requirements (lines 4-5 and 9-10), and inserts them into Drools Knowledge Base (lines 7 and 12).

Furthermore, a new SecPlan is generated using the old one as a basis (line 1), in order to be. The original requirement (RSP) is removed from the set (line 2), substituted now with the two new requirements (ASP) that guarantee it (line 6 and 11).

```
SecPlan newSecPlan = new SecPlan($secPlan);
newSecPlan.removeRequirement($origReq);

Requirement ASP_A = new Requirement(
             $RSP, "rho", $A, $inP1, null);
newSecPlan.getRequirements().add(ASP_A);
insert(ASP_A);

Requirement ASP_B = new Requirement(
             $RSP, "sigma", $B, $outA1, null);
newSecPlan.getRequirements().add(ASP_B);
insert(ASP_B);

insert(newSecPlan);
```

Table 5.3: Snippet encoding the creation of the inferred security requirements in a Drools rule

5.4.1.2 Verification Rules

In some cases the inference rules alone cannot guarantee a security requirement, as the secure composition pattern might need to check a condition over the actual certified security properties for each activity. This information is available only after the instantiation in the pattern of partner services for each activity placeholder.

The verification rules are used after the instantiation of the orchestration pattern (WF) in order to verify that the certified security properties guarantee the pattern's security requirement (RSP). These rules are needed in particular for those cases where the pattern's security requirement can be guaranteed not just by checking that the security requirements inferred by the inference rules are satisfied by the partner services through their certificates, but also by verifying additional conditions over the certified security properties of the partner services (Condition). An example is the availability security property, which can be

guaranteed by checking conditions over the numeric values in the certified security properties (the rule need the numeric values in the certificates in order to be computed).

The main difference between the inference and the verification rules is, then, that while the computation for inference rules flows from the security requirements of the pattern (RSP) to the ones for the single activity placeholders (ASP), the computation for the verification rules flows in the opposite direction.

More specifically, verification rules check if the certified security properties guarantee the security requirements and the conditions over them, as shown in the examples of Section 5.4.4. This process starts from the single instantiated activities (partner services) and goes over patterns where their activities are already checked and satisfy the conditions. Once an activity or a pattern is successfully checked, the process updates the *satisfied* field of the security requirements accordingly.

5.4.2 Integrity

The inference rule presented in Table 5.4 encodes the Precede Integrity on Cascade Pattern shown in Figure 5.6.

The rule states that if we have some data `$input` and `$output` used in an activity S_0 as inputs and outputs, and that S_0 guarantees Precede Integrity on them, then when we substitute S_0 with the sequence `$WF` = `$A` → `$B`, a requirement `$RSP` for the integrity of the data is formulated on `$WF`.

```
rule "Precede Integrity on Cascade"
  when
    $A : Placeholder( $input : parameters.inputs,
             $AtoBdata : parameters.outputs )
    $B : Placeholder( parameters.inputs == $AtoBdata,
             $output : parameters.outputs )
    $WF : Sequential( parameters.inputs == $inputs,
             parameters.outputs == $outputs,
             firstActivity == $A,
             secondActivity == $B )

    $RSP : Requirement(
             secProperty.propertyName == "integrity_pr",
             subject == $WF,
             parameters.inputs == $input,
             parameters.outputs == $output,
             satisfied == false )
    $secPlan : SecPlan( requirements contains $RSP,
             atomic == false )

  then
    SecPlan newSecPlan = new SecPlan($secPlan);
    newSecPlan.removeRequirement($RSP);

    Requirement ASP_A = new Requirement(
             $RSP, "integrity_pr", $A, $input, $AtoBdata);
    newSecPlan.getRequirements().add(ASP_A);
    insert(ASP_A);

    Requirement ASP_B = new Requirement(
             $RSP, "integrity_pr", $B, $AtoBdata, $output);
    newSecPlan.getRequirements().add(ASP_B);
    insert(ASP_B);

    insert(newSecPlan);
end
```

Table 5.4: Inference rule for Precede Integrity on Cascade Pattern

In more detail, the rule requires the following data flow conditions to encode the Cascade orchestration:

- `$A` input is the pattern input (`$input`),
- `$A` output, called `$AtoBdata`, is `$B` input, and
- `$B` output is the pattern output (`$output`).

If these conditions are met, then the two requirements `ASP_A` and `ASP_B`, with `ASP_A` = $Integrity_{pr}$(A, `$input`, `$AtoBdata`) and `ASP_B` = $Integrity_{pr}$(B, `$AtoBdata`, `$output`), guarantee the original requirement.

More specifically, lines 7-10 describe the control flow of the orchestration pattern, and lines 3-8 describe its data flow. Lines 12-19 encode the original security requirement RSP. Lines 25-33 encode the requirements that must hold on the single placeholders to guarantee the original requirement (ASP).

5.4.3 Confidentiality

The rules presented in this section encode the patterns shown in Section 5.3.3.1 about the Perfect Security Property (PSP).

Table 5.5 and Table 5.6 show the inference rules for PSP on, respectively, Cascade and Product orchestrations. These two rules are very similar, as they differ only in the orchestration pattern.

```
rule "PSP on Cascade"
  when
    $A : Placeholder( $input : parameters.inputs,
              $AtoBdata : parameters.outputs )
    $B : Placeholder( parameters.inputs == $AtoBdata,
              $output : parameters.outputs )
    $WF : Sequential( parameters.inputs == $inputs,
              parameters.outputs == $outputs,
              firstActivity == $A, secondActivity == $B )

    $RSP : Requirement( secProperty.propertyName == "PSP",
              subject == $WF, satisfied == false )
    $S : SecPlan( requirements contains $RSP, atomic == false )
  then
    SecPlan newSecPlan = new SecPlan($S);
    newSecPlan.removeRequirement($RSP);
    Set V_P = $RSP.getSecProperty().getAttributesMap().get("V");

    Requirement ASP_A = new Requirement($RSP, "PSP", $A);
    ASP_A.getSecProperty().getAttributesMap()
                    .put("V", new Operation("subset", V_P));
    ASP_A.getSecProperty().getAttributesMap()
                    .put("C", new Operation("subset",
                                new Operation("complement", V_P)));
    newSecPlan.getRequirements().add(ASP_A);
    insert(ASP_A);

    Requirement ASP_B = new Requirement($RSP, "PSP", $B);
    ASP_B.getSecProperty().getAttributesMap()
                    .put("V", new Operation("subset", V_P));
    ASP_B.getSecProperty().getAttributesMap()
                    .put("C", new Operation("subset",
                                new Operation("complement", V_P)));
    newSecPlan.getRequirements().add(ASP_B);
    insert(ASP_B);

    insert(newSecPlan);
end
```

Table 5.5: Inference rule for PSP on Cascade Pattern

```
rule "PSP on Product"
  when
    $paramsA : Parameters( )
    $paramsB : Parameters( inputs disjoint $paramsA.inputs
             outputs disjoint $paramsA.outputs )
    $paramsWF : Parameters( inputs containsall $paramsA.inputs
             inputs containsall $paramsB.inputs
             outputs containsall $paramsA.outputs
             outputs containsall $paramsB.outputs)

    $A : Placeholder( parameters == $paramsA )
    $B : Placeholder( parameters == $paramsB )
    $WF : Parallel( parameters == $paramsWF, $acts : activities )

    $RSP : Requirement( secProperty.propertyName == "PSP",
             subject == $WF, satisfied == false )
    $S : SecPlan( requirements contains $RSP, atomic == false )
  then
    SecPlan newSecPlan = new SecPlan($S);
    newSecPlan.removeRequirement($RSP);
    Set V_P = $RSP.getSecProperty().getAttributesMap().get("V");

    for(Placeholder currAct : $acts){
        Requirement currASP = new Requirement(
                    $RSP, "PSP", currAct);
        currASP.getSecProperty().getAttributesMap()
                    .put("V", new Operation("subset", V_P));
        currASP.getSecProperty().getAttributesMap()
                    .put("C", new Operation("subset",
                              new Operation("complement", V_P)));
        newSecPlan.getRequirements().add(currASP);
        insert(currASP);
    }

    insert(newSecPlan);
end
```

Table 5.6: Inference rule for PSP on Product Pattern

More specifically, after specifying on which orchestration the rule applies, i.e., Cascade or the Product orchestration (lines 3-9 for the Cascade rule, 3-13 for the Product rule), if PSP is requested over the orchestration (lines 11-12 for the Cascade rule, lines 15-16 for the Product rule), then a `Requirement` is generated for each activity placeholder in the orchestration, asking for the PSP (lines 19-35 for the Cascade rule, 24-32 for the Product rule). In particular, as two additional conditions are needed for the proof to hold, then these conditions are added to the security property (lines 20-24 and 29-33 for the Cascade rule, 26-30 for the Product rule).

In more detail, the two additional conditions make usage of the `Operation` class offered in our implementation by the query language of the discovery tool and that is used to encode set operations that will be executed during the discovery process to find suitable matches. If no such a discovery feature is available it is possible to encode the secure composition pattern into a simpler inference rule with no such conditions, and encode the conditions in the verification rules, where the actual security properties offered by the services are available to be checked.

5.4.4 Availability

As mentioned earlier, availability is one case where the inference rules are not enough. Availability, in fact, comprises a set of numerical metrics: the composition of such metrics is supported by numerical functions that can be computed and checked only after the instantiation of the activity placeholders with partner link services. We introduce verification rules in order to check those conditions that required the pattern to be already instantiated.

```
1  rule "Availability inference"
2    when
3      $WF : OrchestrationPattern( $acts: childActivities )
4
5      $RSP : Requirement(
6                secProperty.propertyName == "availability",
7                $secProp : secProperty, subject == $WF )
8      $S : SecPlan( requirements contains $RSP, atomic == false )
9    then
10     SecPlan newSecPlan = new SecPlan($S);
11     newSecPlan.removeRequirement($RSP);
12
13     for(Placeholder currAct : $acts){
14         Requirement currASP = new Requirement(
15                  $RSP, new SecProperty($secProp), currAct);
16         newSecPlan.getRequirements().add(currASP);
17         insert(currASP);
18     }
19     insert(newSecPlan);
20 end
```

Table 5.7: Inference rule for Availability

Table 5.7 shows an inference rule for availability that simply replicates the security requirement RSP over the activity placeholders, in order to query partner services that are certified to take care of availability. This rule is applied on any orchestration (line 3) that is requested to maintain availability (lines 5-7) and generates availability requirements for all the activity placeholders in the pattern (lines 14-17).

In the following we specify the verification rules that encode the patterns about the Maximum Execution Time dimension of availability.

Furthermore, since the verification rules check that the certified properties guarantee the security requirements, an additional rule must be encoded for the base case, i.e., the activity bounded to a partner service `PartnerLinkActivity`.

The verification rule for the `PartnerLinkActivity` is shown in Table 5.8. This rule compares the maximum execution time `maxTime` in the certified security property (lines 3-5) of the `PartnerLinkActivity` (lines 7-8) with the one requested (lines 12-15). If the time in the Requirement is greater or equal than the certified one, then the requirement is updated with the certified time and is marked as satisfied.

```
1  rule "Verification of Max Execution Time on PartnerLinkActivity"
2    when
3      $certAttributes : Map( keySet contains "maxTime" )
4      $certProp : SecProperty( propertyName == "availability",
5                 attributesMap == $certAttributes )
6
7      $activity : PartnerLinkActivity(
8                certifiedProperties contains $certProp )
9
10     $attributes : Map( keySet contains "maxTime",
11               this["maxTime"] >= $certAttributes["maxTime"] )
12     $RSP : Requirement(
13               secProperty.propertyName == "availability",
14               secProperty.attributesMap == $attributes,
15               subject == $activity, satisfied == false )
16   then
17     modify($attributes){
18               put("maxTime", $certAttributes.get("maxTime"))};
19     modify($RSP){setSatisfied(true)};
20 end
```

Table 5.8: Verification rule for the Maximum Execution Time on Partner Link Activity

The verification rule for the Maximum Execution Time on Generic Sequential Pattern is shown in Table 5.9. This rule might seem a little bit complicated as it makes use of the `forall` condition and `accumulate` functions, however the application of these functions is quite standard, so new rules can be built by just following the following examples.

```
1  rule "Verification for Max Execution Time on Generic Sequential"
2    when
3      $WF : Sequential( $acts : childActivities )
4
5      forall( $currAct : Placeholder( ) from $acts
6           $attributes : Map( keySet contains "maxTime" )
7           Requirement( secProperty.propertyName == "availability",
8               secProperty.attributesMap == $attributes,
9               subject == $currAct, satisfied == true ) )
10
11     $totalTime : Number( ) from accumulate (
12          $attributes : Map( keySet contains "maxTime",
13              $maxTime : this["maxTime"] )
14          and
15          Requirement( secProperty.propertyName == "availability",
16              secProperty.attributesMap == $attributes,
17              $acts contains subject, satisfied == true ),
18          sum( $maxTime ) )
19
20     $attributes : Map( keySet contains "maxTime",
21              this["maxTime"] >= $totalTime )
22     $RSP : Requirement(
23              secProperty.propertyName == "availability",
24              secProperty.attributesMap == $attributes,
25              subject == $WF, satisfied == false )
26   then
27     modify($attributes){put("maxTime", $totalTime)};
28     modify($RSP){setSatisfied(true)};
29 end
```

Table 5.9: Verification rule for the Maximum Execution Time on Generic Sequential Pattern

The rule is applied on sequential orchestrations with no data flow specifications (line 3) and first checks that all the activities in pattern have a `Requirement` for availability, in particular for the maximum execution time dimension, and that all these `Requirements` are satisfied (the `forall` part, lines 5-9).

Then the `maxTime` of all the activities in the pattern are accumulated through the `sum` function (the `accumulate` part, lines 11-18), as specified by the Condition part of the secure composition pattern.

Finally the rule compares the accumulated `maxTime` with the one in the pattern's `Requirement` RSP. If the time in the `Requirement` is greater or equal than the accumulated one, then the requirement is updated with the computed time and is marked as satisfied, as specified by the Condition part of the secure composition pattern.

The verification rule for the Maximum Execution Time on Generic Choice Pattern, shown in Table 5.10, differs from the one on the Generic Sequential orchestration only for the orchestration pattern (line 3) and the accumulation function (line 18), i.e., the function returning the maximum (`max`) instead of the sum function.

```
rule "Verification for Max Execution Time on Generic Choice"
  when
    $WF : Choice( $acts : childActivities )

    forall( $currAct : Placeholder( ) from $acts
         $attributes : Map( keySet contains "maxTime" )
         Requirement( secProperty.propertyName == "availability",
             secProperty.attributesMap == $attributes,
             subject == $currAct, satisfied == true ) )

    $totalTime : Number( ) from accumulate (
         $attributes : Map( keySet contains "maxTime",
             $maxTime : this["maxTime"] )
         and
         Requirement( secProperty.propertyName == "availability",
             secProperty.attributesMap == $attributes,
             $acts contains subject, satisfied == true ),
         max( $maxTime ) )

    $attributes : Map( keySet contains "maxTime",
             this["maxTime"] >= $totalTime )
    $RSP : Requirement(
             secProperty.propertyName == "availability",
             secProperty.attributesMap == $attributes,
             subject == $WF, satisfied == false )
  then
    modify($attributes){put("maxTime", $totalTime)};
    modify($RSP){setSatisfied(true)};
end
```

Table 5.10: Verification rule for the Maximum Execution Time on the Choice Pattern

5.5 Summary

In this chapter we presented our approach to assess security over service orchestrations that makes use of secure composition patterns.

The secure composition patterns infer the security requirements needed from partner services part of a composition in order to guarantee a given security requirement on the entire composition.

The secure composition patterns summarize proven inferences between security requirements guaranteed by activity placeholders. These inferences are then encoded into business rules, called security production rules, in order to allow automated reasoning about a composition security.

Chapter 6

Security Aware Service Composition Process

6.1 Overview

Secure composition patterns can be used in different phases of the lifecycle of SBSs in order to guarantee the security of service compositions within such systems. More specifically the patterns allow (a) the generation of secure service compositions, and (b) the validation of the security of existing service compositions.

The former, (a), can be used either at design time, for designing entire or parts of workflows, and at runtime for substituting services that violate security requirements with service compositions generated to replace them. The latter, (b), is used at design time only, in order to ensure that a workflow or parts of it satisfy indeed a given security requirement.

6.2 Scenario

In this section we present a use case scenario for the security aware composition process that we are going to use to exemplify the approach throughout the document. The user in this scenario is a Business Analyst of

a Stock Broker organization, working on the design a SBS that automates the buying of stocks for their clients based on market data and client preferences. This SBS relies on a combination of internal and external services (e.g., Xignite, Amazon AWS) deployed on different clouds, making it an example of a hybrid cloud application.

The Business Analyst identifies a series of security threats (listed in Section 6.2.2) and introduces to the design of the SBS a set of security requirements to address these threats. During the design, the Business Analyst makes usage of the validation capabilities of the approach described in this work to assess if the security requirements are satisfied. If this is not the case, the Business Analyst can use the static discovery of secure services to substitute any service that has inadequate security. Finally, during the SBS implementation a service adaptation mechanism is included in order to support business continuity whenever one of the services used by the SBS becomes unavailable or no longer supports the functional/security requirements. The service adaptation mechanism uses the dynamic discovery of secure services to locate alternative services that support the requirements. In particular both the static and the dynamic discovery of secure service make usage of the generation of secure service composition described in this thesis when no atomic service can provide the requested functionality and security.

6.2.1 Stock Broker SBS

A simple version of the SBS workflow designed by the Business Analyst is presented in Figure 6.1. For the sake of presenting a reasonable example without dwelling into complex technicalities, we omit details of the process, like the ones about error handling and transactions support, as we assume that the Business Analyst is going to take care of them in a second design stage.

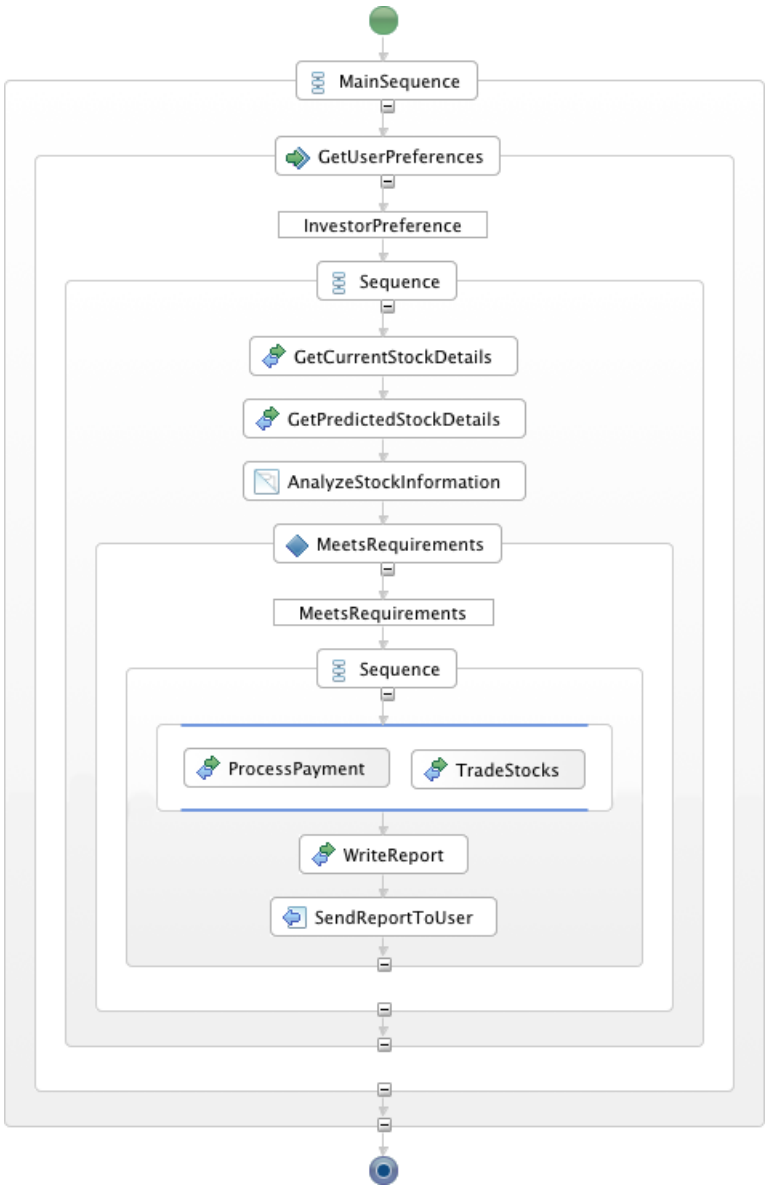


Figure 6.1: The Stock Broker SBS workflow

Upon receiving a set of user preferences, the SBS retrieves the current and the predicted stock values from a financial service (e.g., Xignite). Then it internally analyses all the information collected (activity AnalyzeStockInformation in the figure) in order to check if the user requirements are met by the stock values. If this is the case, then the stocks

are traded and the payment is processed by internal services that contact the bank/financial supplier through the SWIFT protocol [98][99]. Finally a report of the transaction is generated, stored in a storage service (e.g., Amazon S3), and finally sent back to the user.

6.2.2 Security Threats

This section presents a threat analysis of the given scenario following the STRIDE Threat Model [52], by identifying the threat categories and the resources at risk. STRIDE is the acronym of the following threat categories:

- Spoofing identity: when a malicious party successfully poses as an authorized user.
- Tampering with data: when the content of data or messages is altered without permission, either by a malicious party or by a malfunction.
- Repudiation: when a party denies authoring or initiating an action and no proof exists to contradict this.
- Information disclosure: when an unauthorized user accesses supposedly secure information.
- Denial of service: when a valid user access to a system or resource is limited or eliminated.
- Elevation of privilege: when a malicious user gains higher privileges than the ones that should be granted to that user.

Based on this model, the threats in the Stock Broker scenario can be identified as follows:

A. Spoofing the communications between the investor and the Stock Brokerage firm. A malicious user may pose as another investor and maliciously manage their portfolio. Also, a malicious user may pose as the Stock Brokerage SBS and obtain secret information from the investors. As the *investor - Stock Brokerage firm* relationship is not communicated to the underlying service providers, this threat does not require additional controls for the services used by the SBS and it should be taken care of with appropriate authentication at SBS level.

B. Spoofing the communications between the Stock Brokerage firm and the bank/financial supplier. A malicious user may pose as one of these parties and transfer funds, stocks or gaining secret information about this. As the SBS uses services internal to the Stock Brokerage organization to communicate with the bank/financial supplier, this threat should be addressed with appropriate authorization mechanisms in the implementation of such services.

C. Tampering of the investment plan, the trading account details or the trading report. A malicious user or a malfunction may alter this data, possibly changing the outcomes of a trading session. This threat requires assurance about the integrity of these data against both the SBS and the services used by it.

D. Repudiation of the investment plan from the user. A malicious user may deny that he/she has sent an investment plan to the SBS, and try to claim back the money for an unsuccessful investment. Again, as the *investor - Stock Brokerage firm* relationship is not communicated to the underlying service providers, this threat does not require additional controls for the services used by the SBS and it should be taken care of with appropriate non-repudiation mechanisms at the SBS level.

E. Repudiation of the successful trading from the bank/financial supplier. A malicious supplier may deny the trading from having taken place. This threat should be taken care of at a level of the internal services contacting the bank/financial supplier. In particular if SWIFT is used, non-repudiation is granted by the SWIFTNet protocol [99].

F. Repudiation of the successful trading from the SBS. In order to avoid the Stock Broker firm to potentially deny the result of a trading session, a report is sent to the user. This threat menaces the *investor - Stock Brokerage firm* relationship, and it should be taken care of with appropriate non-repudiation mechanisms at the SBS level (e.g., by signing the report).

G. Disclosure of the investment plan, the trading account details or the trading report. A malicious user may use secret information about investment strategies of competing investors as additional information for their investments, or use the obtained trading account details to maliciously manage the account of someone else. This threat requires assurance about the confidentiality of this data from both the SBS and from the services used by it.

H. Denial of service of the SBS. A malicious user or a malfunction may limit the access to the SBS or any of the services used by it, impeding or delaying a trading session. This threat requires assurance about the availability of both the SBS and of the services used by it.

The approach presented in this thesis addresses the threats introduced by the services used by the SBS. This is achieved by defining appropriate security requirements that reduce or mitigate fully the security threats and by checking that the security requirements are actually provided by the services involved in a composition. In particular, from the above list of

threats, the security requirements that should be introduced for the services part of this scenario are about integrity and confidentiality of the data exchanged (against threats C and G) and availability of the services (against threat H). The other security threats, directed to the SBS or the internal services implementation, can be addressed instead with alternative approaches present in literature, as the ones described in Section 2.4.

This analysis intentionally does not go into the details of identifying the actual sources for each threat, like vulnerabilities and attacks (e.g., SQL, OS, DNS or LDAP injection, cross-site scripting, SSL Heartbeat vulnerability), as this level of detail often depends on the implementation of the service and this implementation is usually unknown to the SBS designer. The vulnerabilities and attacks, however, have to be identified and addressed in order to release a security descriptor for each of the services used in a SBS, describing the security properties that the service guarantees. The security descriptors are used then to satisfy a security requirement introduced to minimize the security threats presented in this section. The interested reader may find a survey of service and cloud computing security issues in [104].

6.3 Workflows

In the following algorithms we assume that service composition workflows are represented as a composition of the orchestration patterns described above, as they represent abstract orchestrations that are not necessarily bound to services.

A workflow is defined by the outermost orchestration pattern that describes it and by the information about its functional applicability (in the current implementation this is achieved through a WSDL of the resulting composition). Each pattern contains a set of placeholders for activities. The

placeholders can contain the description of a single operation (in the current implementation this is also described through a WSDL) that should instantiate the activity or another pattern.

```
Workflow[orchestration=
  Sequential(
    PartnerLinkActivity(http://example.com/StockISIN.wsdl),
    Sequential(
        UnassignedActivity(GetQuoteActivity),
        UnassignedActivity(GetConversion) ) )]
```

Table 6.1: Example of a workflow

Each service placeholder in the workflow can be instantiated with a partner link service, as shown in the example in Table 6.1 (the `PartnerLinkActivity`). When all the service placeholders in a workflow are not instantiated (i.e., bounded to a concrete service), we call the workflow "abstract workflow".

An abstract workflow represents an activity that can be replaced by a set of other activities and contains: (a) information about the control flow between these activities, and (b) the set of orchestration patterns used to generate it.

Figure 6.2 shows the XML Schema used to represent an abstract workflow. The root of the schema is the `Workflow` element, which is required to have: (i) a `name`, (ii) a `WSDL` describing the abstract workflow interface (located either remotely or locally), and (iii) the outermost orchestration pattern of the workflow, described through the `OrchestrationType` XML Schema type.

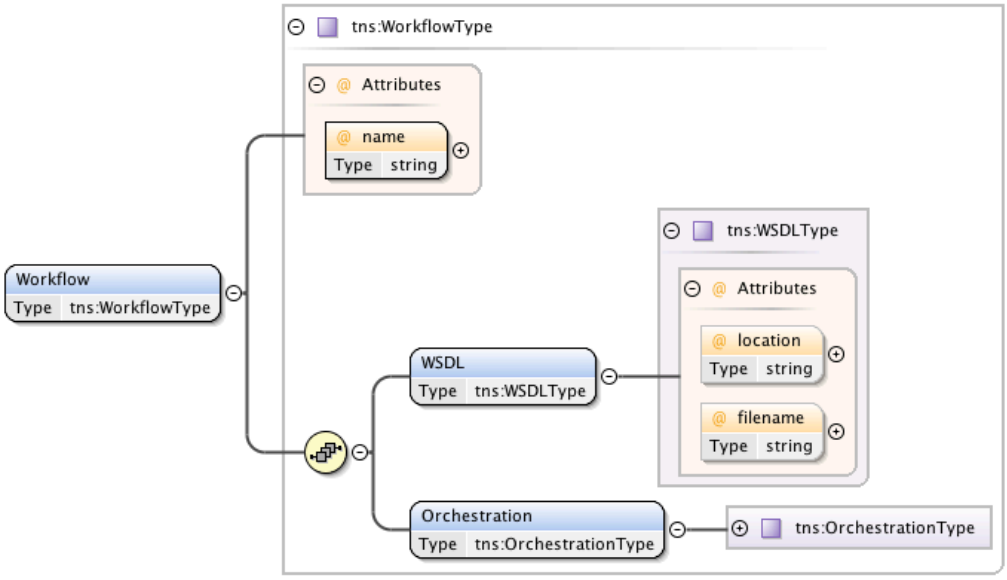


Figure 6.2: The Workflow element of the abstract workflow schema

The `OrchestrationType` contents are shown in Figure 6.3. The main element describing an orchestration pattern can be a `Sequential`, a `Choice`, or a `Parallel` element. The `Sequential` element is required to have two sub-elements describing the first and the second activity placeholders in the orchestration through the usage of the `PlaceholderType` type. The `Choice` element allows the specification of the condition over which the control flow is decided, an activity placeholder that is executed if the condition is true, and optionally an activity placeholder that is executed if the condition is false. The `Parallel` element contains one or more activity placeholders described through the `PlaceholderType` XML Schema type.

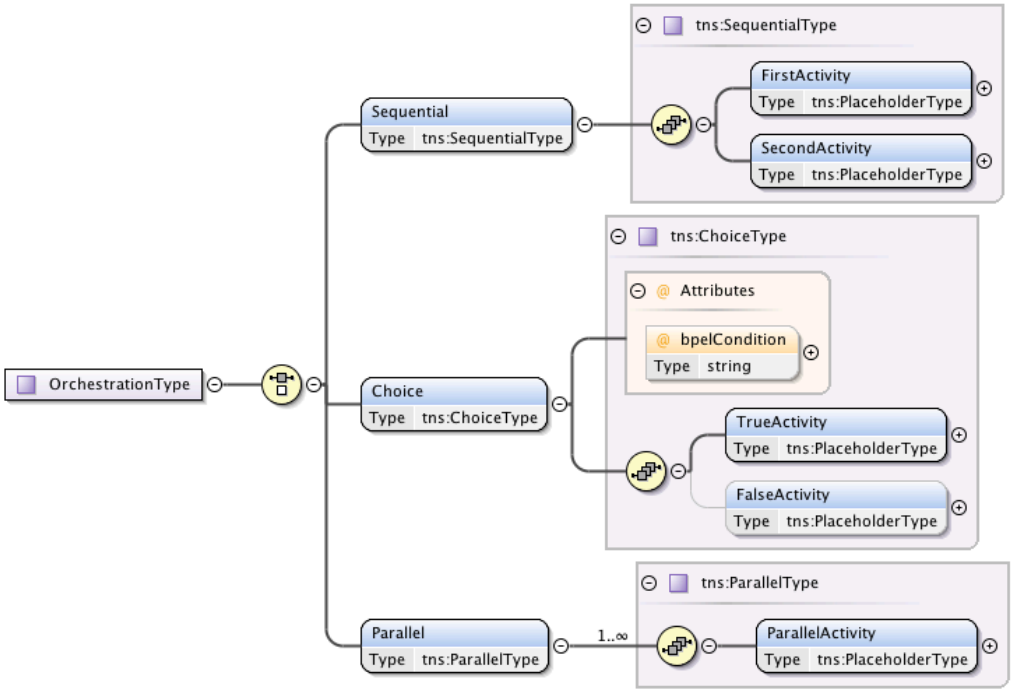


Figure 6.3: The OrchestrationType type of the abstract workflow schema

Figure 6.4 shows the `PlaceholderType` used to describe an activity placeholder. In particular an activity placeholder can represent either an orchestration pattern, or an atomic activity. The latter is encoded through the `Activity` element and is described through a `name` and a `WSDL` interface describing the functional requirements of the activity to perform. The WSDL is used to find and bound a service for the activity during the instantiation phase of the workflow.

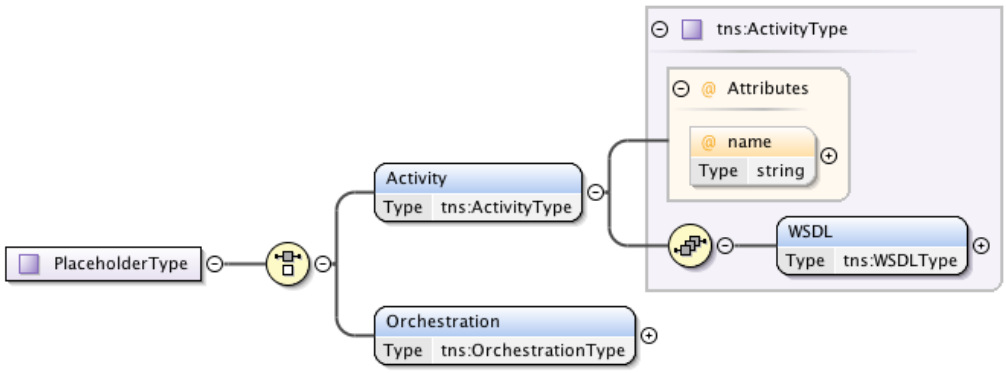


Figure 6.4: The PlaceholderType type of the abstract workflow schema

Table 6.2 shows an example of the XML encoding of an abstract workflow, based on the workflow in Table 6.1.

```
<Workflow name="GetCurrentQuoteInUSD">
    <WSDL location="http://localhost:8080/wfs/GCQ_USD.wsdl" />
    <Pattern>
        <Sequential>
            <FirstActivity>
                <Activity name="GetISIN">
                    <WSDL filename="GetISIN.wsdl" />
                </Activity>
            </FirstActivity>
            <SecondActivity>
                <Pattern>
                    <Sequential>
                        <FirstActivity>
                            <Activity name="GetQuoteActivity">
                                <WSDL filename="GetEURStock.wsdl" />
                            </Activity>
                        </FirstActivity>
                        <SecondActivity>
                            <Activity name="GetConversion">
                                <WSDL filename="Exchange.wsdl" />
                            </Activity>
                        </SecondActivity>
                    </Sequential>
                </Pattern>
            </SecondActivity>
        </Sequential>
    </Pattern>
</Workflow>
```

Table 6.2: Example of an abstract workflow

6.4 Algorithms

In the following we present the algorithms that we have developed to validate security requirements at design time and to build a secure service workflow. The first algorithm is fundamental to both approaches, as it describes how to use the secure composition patterns in order to infer security requirements for activities within a composition. In the case of validation, the inference of such requirements is used to check if the workflow services indeed satisfy them, and therefore the workflow can be confirmed to have the required overall security requirement. In the case of generation of secure service composition, the inference of the security requirements required of individual services is used in order to drive the search process that discovers if there are such services that could instantiate the placeholders of the workflows.

6.4.1 Inference of Security Requirements

The algorithm described in this section generates the security requirements to be requested to the (potential) partner services of a workflow. The algorithm is invoked having as an input the initial security requirement and a workflow. Based on these two inputs, the algorithm derives the security requirements that should be requested from the partner services that are (or will be) bound to each activity in the workflow. This may lead to find more than one combination of security requirements for the different activities. Each of these combinations is called a *Security Plan* in the context of this thesis.

The derivation of the security requirements is driven by the *inference rules* that express the security requirements that need to be satisfied by the individual partner services, which will instantiate the workflow for the latter to satisfy other requested security requirement as a whole. As discussed in

Chapter 5, the rules express dependencies between the security requirements of the individual activities of the workflow and the security requirements of the workflow as a whole. These dependencies have been established by the proofs of different *secure composition patterns*; in fact the *security production rules* can be considered encodings of the *secure composition patterns*.

6.4.1.1 Algorithm

The algorithm for making inferences about the security requirements of the placeholders of (and therefore the services that may be bound to) a workflow is shown in Table 6.3.

As shown in the table, given an input service workflow WF and a security requirement RSP, the algorithm tries to apply all the secure composition patterns that would be able to guarantee RSP. A pattern is applied if the workflow specification of the pattern (Pattern.WF) matches with WF. If a pattern matches the workflow, then the security plans computed up to that point are updated to replace the security requirement RSP with the security requirements for the matched placeholders in WF (these can be individual activities or sub-workflows) as determined by the pattern. If a matched placeholder PH of WF is an atomic activity, the process ends w.r.t it. If PH is a sub-workflow, the algorithm will continue trying to apply further patterns for it recursively.

More specifically, as mentioned in Section 5.4.1, the process of inferring security requirements can generate alternative plans. Each plan is represented by a set of security requirements, meaning that the enclosed requirements must hold at the same time. All the possible plans (i.e., the sets of security requirements) are collected in the list that is the output of the process.

```
Algorithm: INFERRECURSION(RSP, WF, InSecPlans)
Input:   WF – workflow
         RSP[WF] – requested security requirement for WF
         InSecPlans – list of security plans used for recursion
Output:  OutSecPlans – list of security plans of inferred requirements

1  For each pattern Patt such that Patt.CSP matches RSP do
2      If Patt.WF matches WF then
3          For each placeholder PH of WF do
4              SecReqs[PH] := security requirements identified by Patt.ASP
5          EndFor
6          For each security plan S in InSecPlans do
7              S' := replace RSP by SecReqs in S
8              Add S' to SecPlans_Patt
9          EndFor
10         For each placeholder PH in WF that is a sub-workflow do
11             SecPlans_Patt := INFERRECURSION
12                                  (PH, SecReqs[PH], SecPlans_Patt)
13         EndFor
14         Add all the plans SecPlans_Patt to OutSecPlans
15     EndIf
16 EndFor
17 Return OutSecPlans
```

```
Algorithm: INFERREQUIREMENT(WF, RSP[WF])
Input:   WF – workflow
         RSP[WF] – requested security requirement for WF
Output:  SecPlans – list of plans of inferred security requirements

1  Return INFERRECURSION(WF, RSP[WF], {RSP[WF]})
```

Table 6.3: Algorithm for the inference of Security Requirements

As discussed in Section 5.4, the security production rules are expressed in Drools production rules. The algorithm for the inference of security requirements has been implemented in Drools rule based reasoning.

6.4.1.2 Example

Consider the case of the *ProcessOrder (PO)* workflow shown in Figure 6.5, where a Stock is bought and paid for in parallel, and then a report of the action is written. We can use the algorithm above in order to derive the security requirements that should be required of the different activities in the process in order to guarantee the confidentiality of the Stock Investor current account. This security requirement can be expressed as *PSP*, with E_{PO}^{H} = {*currentAccount*} and E_{PO}^{L} = {*paymOrder*, *stocksOrder*, *tradingAccount*, *report*}.

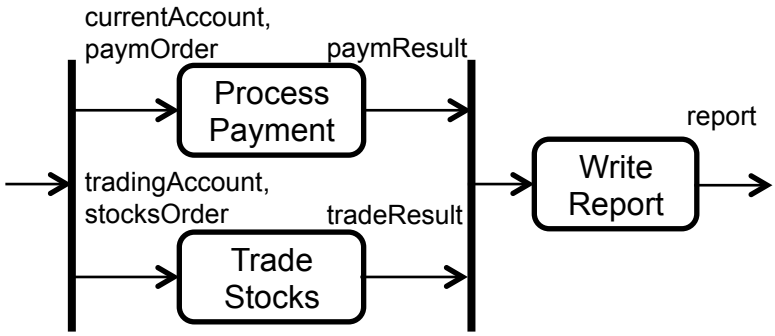


Figure 6.5: The ProcessOrder workflow

ProcessOrder can be seen as a sequential workflow consisting of a sub-workflow WF' and the atomic activity *WriteReport* that follows it (see Figure 6.6). WF' itself is a parallel workflow involving two atomic activities: *ProcessPayment* and *TradeStocks*.

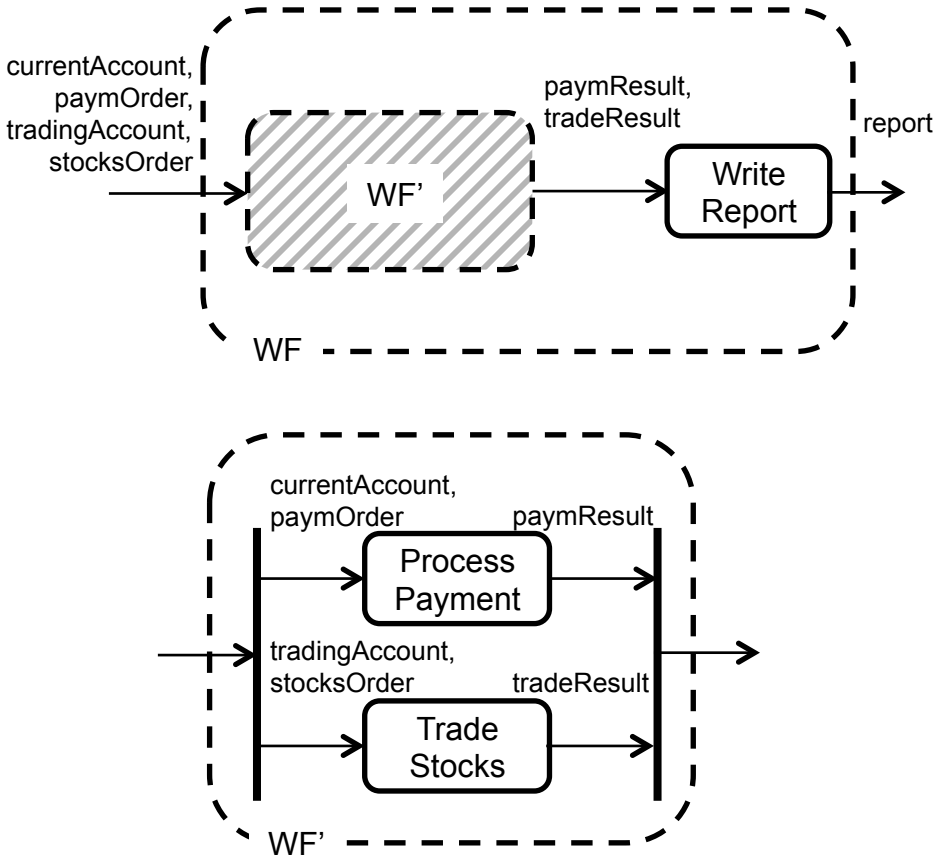


Figure 6.6: The Orchestration Patterns of the ProcessOrder workflow

Hence, when INFERREQUIREMENT is applied on to the workflow, in the first iteration the inference rule for PSP on the sequential flow presented in Section 5.4.3 can be applied on WF. This returns two security requirements: (1) PSP confidentiality for the inputs *currentAccount*, *paymResult* and the output *tradeResult* of WF', and (2) PSP-confidentiality for the input *paymResult* and output *tradeResult* of *WriteReport*.

The second iteration of the algorithm applies the inference rule for PSP on the parallel flow to WF'. The algorithm then creates and adds two security requirements to the final plan: one requiring PSP confidentiality for

currentAccount and *paymResult* of *ProcessPayment*, and another requiring PSP confidentiality for *tradeResult* of *TradeStocks*.

6.4.2 Verification of Security Requirements

The algorithm described in this section verifies the security requirements requested to the partner services of a workflow. The algorithm is invoked having as an input the fully instantiated workflow and the security requirements that have to be guaranteed. Based on these two inputs, the algorithm verifies that the security requirements are guaranteed by the appropriate partner services through their (certified) security property. The verification of these conditions is driven by the *verification rules*.

6.4.2.1 Algorithm

As shown in the table, given an input service workflow WF and a security requirement RSP, the algorithm tries to apply all the secure composition patterns that would be able to guarantee RSP. A pattern is applied if the workflow specification of the pattern (Pattern.WF) matches with WF. If a pattern matches the workflow, then the security plans computed up to that point are updated to replace the security requirement RSP with the security requirements for the matched placeholder in WF (these can be individual activities or sub-workflows). If a matched placeholder PH of WF is an atomic activity, the process ends w.r.t it. If PH is a sub-workflow, the algorithm is applied recursively for it.

```
   Algorithm: VERIFYREQUIREMENT(WF, SP)
   Input:   WF – workflow
            SP – security requirements that need to hold
   Output:  true or false – status of the verification

1  For each activity Act in WF do
2  |  If Act.Service.Certificates contains SP[Act] then
3  |  |  HoldsSP[Act] := true
4  |  EndIf
5  For each sub-workflow SW such that
6  |           for all the placeholders PH in SW HoldsSP[PH] do
7  |  Patt := pattern such that Patt.WF = SW and Patt.SP = SP[SW]
8  |  If EVALUATE(Patt.Condition, SW) then
9  |  |  HoldsSP[SW] := true
10 |  Endif
11 EndFor
12 Return HoldsSP[WF]
```

Table 6.4: Algorithm for the verification of Security Requirements

As in the case of the inference algorithms, the algorithm in Table 6.4 is realised through the application of Drools rules.

6.4.2.2 Example

Suppose we have a workflow that contains two partner services in a Generic Sequential Pattern. A security requirement on this workflow requires that the maximum execution time of the workflow is 500 milliseconds. The two partner services have a certificate each, stating that the maximum execution time of each of them is 300 milliseconds. The verification rules in Section 5.4.4 then cannot infer that the security requirement is satisfied, so the verification for this workflow fails.

6.4.3 Validation of Workflows

The algorithm described in this section is focused on checking if the fully instantiated workflow taken into consideration respects the requested security requirement. This algorithm is useful in order to check fully instantiated SBS workflows against security requirements.

Furthermore the following algorithm allows validation of workflow fragments, i.e., portions of a workflow delimited by a control flow activity (or sub-workflows). In case of BPEL workflows, a fragment consists of a *scope* or a control flow (i.e., *sequence*, *flow*, *while*, *forEach*, *repeatUntil*, *if-then-else* or *pick*) activity that can contain multiple service invocations (in the form of *invoke* activities) and further control flow activities.

6.4.3.1 Algorithm

Given a request to check whether a workflow (WF) satisfies a security requirement RSP, the algorithm INFERREQUIREMENT is applied to identify the list of alternative security plans (i.e., combinations of security requirements of the individual services in the fragment) that would guarantee RSP. As explained earlier INFERREQUIREMENT tries to apply different secure composition patterns in order to identify these alternative plans. If such plans exist, each of them is analysed further to check if the security requirements in the plan are provided by the services in the fragment.

```
Algorithm: ValidateWorkflow(WF, RSP)
Input:   WF – workflow
         RSP– security requirement that needs to be validated
Output:  validationStatus – true if WF satisfies the security requirement
         SecPlans – list of security plans for the activities in WF

1  SecPlans:= InferRequirement(WF, RSP)
2  For each plan SP in SecPlans do
3  |  validPlan := true
4  |  For each service S invoked in the fragment WF do
5  |  |  R := ServiceDiscovery(S, SP[S])
6  |  |  If service S is not contained in R then
7  |  |  |  validPlan:= false
8  |  |  Endif
9  |  EndFor
10 |  If validPlan = true and VerifyRequirements(WF, SP) then
11 |  |  Return true, SecPlans
12 |  Endif
13 EndFor
14 Return false, SecPlans
```

Table 6.5: Algorithm for the validation of workflows

To validate whether an individual service satisfies the security requirement by a security plan, we express the requirement as a service discovery query and then use the discovery tool described in Section 4.4 to match the specification of the individual service with the query and establish if it satisfies the query or not (see SERVICEDISCOVERY(*S*, *SP[S]*) invocation in the algorithm). In applying the service discovery process, we assume the existence of machine-readable security property descriptors for a service (e.g. certificates) indicating the security properties that a service S has. If the individual service validation succeeds for all the services of the fragment by even one of the identified security plan, and the patterns that

required it are verified, then the workflow is validated. Otherwise, if no security plan can be found, or if none of the found security plan can be satisfied by the services in the workflow, or if the satisfied security plans are not verified by the verification rules, the fragment is reported as not validated.

6.4.3.2 Example

Consider again the case of the *ProcessOrder (PO)* workflow previously explained. The workflow has been implemented into a SBS, bounding the activities to services as shown in Figure 6.7.

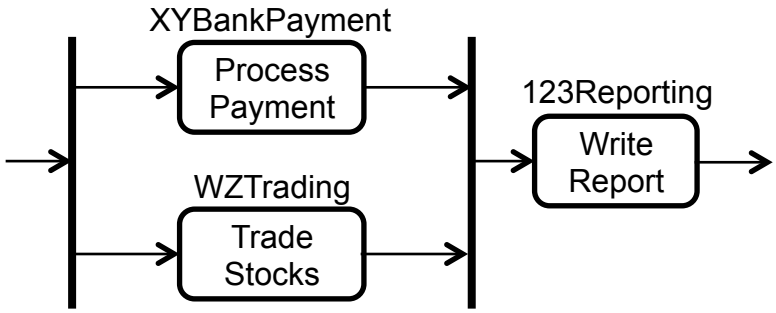


Figure 6.7: Services used by the ProcessOrder SBS

Assume that we have another rule to preserve the PSP can state that the confidentiality can be achieved by means of Separability. In this case we have 2 security plans, one requiring Separability for the three activities in the workflow, and the other requiring PSP. The algorithm considers one security plan at a time, so in a first instance it will check if the Separability plan is valid.

A first service discovery is performed for *ProcessPayment*, looking for services with the required interface and the Separability security property. The service *XYBankPayment* that is currently bound to the *ProcessPayment* activity can be found in the discovery results, so it appears to have the

requested Separability property. A second service discovery is performed for *TradeStocks*, however in this case the service *WZTrading* that is bound to it cannot be found in the discovery results. This means that the current security plan is not valid.

Going forward to the second security plan, the service discovery for all the workflow activities successfully return the bounded services, meaning that they all provide the security properties requested by the plan. Since there is at least one valid security plan, the algorithm returns true.

6.4.4 Discovery of Secure Workflows

The algorithm described in this section attempts to find appropriate workflows that can address the conditions expressed in a query (both the structural and the security conditions), in order to generate compositions.

The algorithm assumes the existence of a repository of abstract workflows that describe the control and data-flow of well-known procedures, in order to focus in the algorithm on the security requirement inference. The abstract workflows are assumed to be supplied by domain experts or generative composition algorithms. The practice of requiring a set of existent workflows is quite common, as described in Section 2.3. Alternative approaches can combine the security requirement inference of the given algorithm with a (existing) generative approach for the workflow, an overview of which is also present in the same section.

6.4.4.1 Algorithm

When the standard discovery doesn't find a single replacement service for a given service, then the query associated with the service to be replaced (Q) is sent to the algorithm that discovers secure workflows shown in Table 6.6.

```
   Algorithm: GETSECUREWORKFLOWS(Q)
   Input:   Q – query for required service
   Output:  WStack – stack of workflows to instantiate
            ReqMap – list of requirements for each workflow

1  For each abstract workflow AW in the repository do
2  |  If STRUCTURALMATCHING(Q, AW) == true then
3  |  |  RSP = GETSECURITYREQUIREMENTS(Q)
4  |  |  SecPlans = INFERREQUIREMENT(AW, RSP)
5  |  |  For each security plan S in SecPlans do
6  |  |  |  WF := Clone AW
7  |  |  |  Push WF in WStack
8  |  |  |  ReqMap[WF] := S
9  |  |  EndFor
10 |  EndIf
11 EndFor
12 Return WStack, ReqMap
```

Table 6.6: Algorithm for the discovery of secure workflows

Initially the algorithm identifies the abstract workflows that provide the requested functionality by calling the structural matching algorithm on the abstract workflow repository.

Then the algorithm described in Section 6.4.1 inferring the security requirements is called on each matching workflow, generating in this way a list of security requirements, representing different plans. For each plan, i.e., each set of security requirements, a copy of the workflow with the considered security requirement is created and to the outputs of the process.

6.4.4.2 Example

When a service providing Stock quotations for a given Symbol has to be substituted, the algorithm receives a query to find alternatives. By

matching the service structural description in the query, it finds in the abstract workflow repository a workflow with two activities in a sequence. The first activity converts a Symbol into an ISIN (different identifier format for the same Stock), while the second one uses the ISIN to provide Stock quotations.

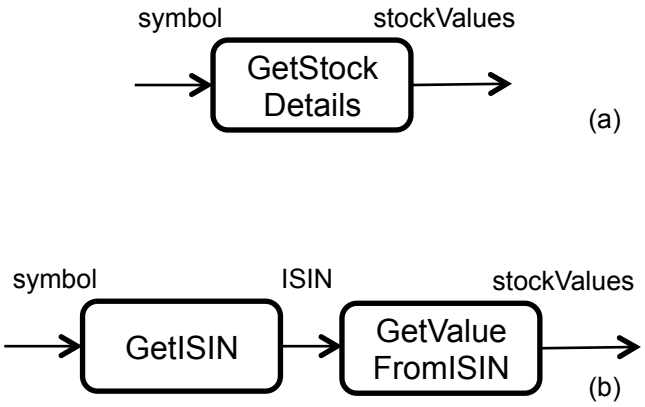


Figure 6.8: The GetStockDetails service (a) and the workflow that can replace the GetStockDetail service (b)

The query conditions about security require integrity on the data, making explicit references to the appropriate parts of the security property descriptor (e.g. certificate) and in a complex combination of expressions and operators, as shown in Table 6.7.

```
<AssertQuery name="A1" type="HARD" assertScope="SINGLE">
  <LogicalExpression><Condition relation="EQUAL-TO">
    <Operand1><AssertOperand facetType="Assert">
        //ASSERTCore/ToC/Assets/Asset[@Type='InputParameter']/Name
    </AssertOperand></Operand1>
    <Operand2><Function name="WSDLLookup"><Arguments>
        <Argument WSDLElementType="input-message"
                  WSDLElementName="symbol"/>
    </Arguments></Function></Operand2>
  </Condition>

  <LogicalOperator>AND</LogicalOperator>
  <LogicalExpression><Condition relation="EQUAL-TO">
    <Operand1><AssertOperand facetType="Assert">
        //ASSERTCore/ToC/Assets/Asset[@Type='OutputParameter']/Name
    </AssertOperand></Operand1>
    <Operand2><Function name="WSDLLookup"><Arguments>
        <Argument WSDLElementType="output-message"
                  WSDLElementName="stockValue"/>
    </Arguments></Function></Operand2>
  </Condition>

  <LogicalOperator>AND</LogicalOperator>
  <LogicalExpression><Condition relation="EQUAL-TO">
    <Operand1><AssertOperand facetType="Assert">
        //ASSERTCore/SecurityProperty/@PropertyAbstractCategory
    </AssertOperand></Operand1>
    <Operand2>
      <Constant type="STRING">Integrity</Constant>
    </Operand2>
  </Condition></LogicalExpression>
  </LogicalExpression>
  </LogicalExpression>
</AssertQuery>
```

Table 6.7: Example of query making explicit references to certificate parts through XPath

This complex expression is converted to a security requirement, containing a reference to the workflow and the security property “integrity” on the referenced parameters (`symbol` and `stockValue`).

The security requirements are derived through the inference algorithm previously explained, returning 2 plans, one requesting integrity from both activities and the other requesting authentication of the data. The process then returns two instances of the same workflow having different security requirements over the activities.

6.4.5 Workflow Instantiation

The algorithm described in this section instantiates the activities in the given workflows with services, by constructing ad hoc queries containing the structural conditions and the given security requirements. The workflows and the corresponding security requirements in input can be obtained through the algorithm described in the previous section.

6.4.5.1 Algorithm

The algorithm dealing with the instantiation of workflows is shown in Table 6.8. This algorithm makes use of a service discovery framework, supporting the discovery of security requirements.

```
Algorithm: WORKFLOWINSTANTIATION(WStack, ReqMap)
Input:   WStack – stack of workflows to instantiate
         ReqMap – list of requirements for each workflow
Output:  ResultSet – set of instantiated workflows

1  While there are more workflows in WStack do
2  |  Pop the first workflow W from the WStack
3  |  Get the first unassigned activity A from W
4  |  Services = SERVICEDISCOVERY(A, ReqMap[W])
5  |  For each service S in Services do
6  |  |  W_S := substitute A with S in W
7  |  |  If there is another unassigned activity in W_S then
8  |  |  |  Push W_S in WStack
9  |  |  Else
10 |  |  |  If VERIFYREQUIREMENTS(W_S, ReqMap[W_S])
11 |  |  |  |  Add W_S to ResultSet
12 |  |  |  EndIf
13 |  |  EndIf
14 |  EndFor
15 EndWhile
16 Return ResultSet
```

Table 6.8: Workflow Instantiation Algorithm

The activities of the workflows are instantiated progressively, by investigating each workflow W in a depth-first manner.

The algorithm takes the first unassigned activity A in W (in the control flow order) and queries the service discovery based on the structural requirements in the workflow information (e.g. the WSDL of A) and the security requirements.

The list of candidate service resulting from the discovery process is then used to instantiate A in W. In particular in the prototype we arbitrarily

limited the instantiation by taking just the first N (with N=10) services based on the structural distance, to avoid the generation of too many workflows. Each service is used to instantiate a new workflow W_S. This also means that if no service can be found to instantiate A, no more processing will happen on W (the workflow is discarded).

If W_S is not fully instantiated, then it is added to the working stack; otherwise the instantiation process for that workflow is ended, so if the security requirements are verified the workflow can be added to the list returned as result of the process.

6.4.5.2 Example

After the execution of the algorithm for discovery of secure workflows, we obtained two workflows that can be used to replace *GetStockDetails*. The requirements for the activities in the first workflow are integrity of the data, and for the second workflow data authentication is required. The instantiation algorithm takes into account a workflow at a time and tries to fully instantiate it by using the structural and security requirements for each activity.

Starting from the first workflow then, the algorithm tries to instantiate the *GetISIN* activity. The service discovery is performed, asking for services that respect *GetISIN* interface and that have the requested security requirement, i.e., integrity of data. The service discovery finds two compatible services, *Symbol2ISIN* and *ConvertStockIDs*. Two new workflows are then inserted in the stack in order to be taken in consideration, one instantiating *GetISIN* with *Symbol2ISIN* and the other with *ConvertStockIDs*. At this point the workflow containing *Symbol2ISIN* is taken into consideration, in order to get the second activity instantiated as well. If no services are found, the workflow is discarded, otherwise if the

workflow is fully instantiated is added to the list of results and other workflows are taken into consideration.

6.5 Summary

In this chapter we described the algorithms, which -through the usage od secure composition patterns- determine and assess the security requirements requested of the services that form a composition, in order to guarantee a security requirement across the entire composition.

The first two algorithms (INFERREQUIREMENT and VERIFYREQUIREMENT) are more generic, allowing the inference and verification of security requirements. The next algorithm (VALIDATEWORKFLOW) supports the design time check of validity of security of an SBS workflow that is being designed. The last two algorithms (GETSECUREWORKFLOWS and WORKFLOWINSTANTIATION) encode a secure service composition process, available both at design and runtime, that consists of discover alternative workflows that can satisfy functional and security requirements followed by instantiation of the workflows with services that respect the inferred security requirements.

Chapter 7

Implementation

7.1 Overview

Based on the approach described in the previous chapter, two tools have been implemented to support security aware composition respectively at runtime, during service discovery, and at design-time, while devising a SBS. These tools are called Security Aware Runtime Discovery Tool and Security aware BPEL Design Tool.

The discovery tool is based on an existing tool, RSDT (see Section 4.3); where most of the new features are performed by a new component called Composition Manager. This component is responsible for the creation of secure service compositions to meet queries in cases where the latter do not match with any single service. Furthermore, the component generates a virtual service pointer that can be used by SBSs to invoke the composition through the tool.

The Security aware BPEL design tool, instead, is an extension of the BPEL Designer plugin for Eclipse [25] that allows the security validation and adaptation of partner services (or service compositions) for a SBS being built as a service orchestration.

7.2 Security Aware Runtime Discovery Tool

The prototype described in this section is based on RSDT, the discovery tool described in Section 4.3, but extended to consider also security conditions, certificates and service composition.

The Security Aware Runtime Discovery Tool allows runtime service discovery based also on security requirements and security descriptors, describing which security properties hold for a specific service. The current implementation uses the concept of certificates as security descriptors, however the approach is compatible with any security descriptor. Furthermore the tool allows the generation of service compositions that may be used as alternative services and that respect the functional and security requirements that are requested. In particular service composition is triggered only after an attempt to find a single service that satisfies a query has failed. This failure may be because there is no service that satisfies the interface, behavioural, quality or security conditions expressed by the query and does not necessarily relate only to security or non-security conditions about of the required service.

7.2.1 Query language

The queries for this discovery tool are expressed in A-SerDiQueL, an extension of SerDiQueL (see Section 4.4.3) that we have developed to support the specification of security conditions as part of service discovery queries.

The specification of security conditions in A-SerDiQueL assumes that the security properties of services are described in certificates, as advocated by the ASSERT4SOA project [5][80]. In addition to specifying the relevant security property, certificates may include descriptions of the evidence justifying the certification of the property, the authority that has issued the

certificate, the validity period of the certificate and any other information related with the asserted property or the certification process.

Certificates are represented in XML according to a specific XML schema, and are published in service registries as service descriptions (facets). An example of an A-SerDiQueL condition regarding the integrity of an input parameter named `xyz` is shown in Table 7.1.

```
<AssertQuery name="A1" type="HARD" assertScope="SINGLE">
  <LogicalExpression><Condition relation="EQUAL-TO">
        <Operand1>
           <AssertOperand facetType="Assert">
         //ASSERTCore/ToC/Assets/Asset[@Type='InputParameter']/Name
           </AssertOperand>
        </Operand1>
        <Operand2>
           <Function name="WSDLLookup"><Arguments>
             <Argument WSDLElementType="input-message"
                   WSDLElementName="xyz" />
           </Arguments></Function>
        </Operand2>
     </Condition>
     <LogicalOperator>AND</LogicalOperator>
     <LogicalExpression><Condition relation="EQUAL-TO">
          <Operand1>
            <AssertOperand facetType="Assert">
       //ASSERTCore/SecurityProperty/@PropertyAbstractCategory
            </AssertOperand>
          </Operand1>
          <Operand2>
            <Constant type="STRING">Integrity
            </Constant>
          </Operand2>
     </Condition></LogicalExpression>
  </LogicalExpression>
</AssertQuery>
```

Table 7.1: Example of a security requirement expressed in A-SerDiQueL

7.2.2 Architecture

As shown in Figure 7.1, the prototype can be called by sending a discovery query to the Query Handler. This component parses the query and sends the parsed objects to the Discovery Manager, to find services that match the query. The Discovery Manager collects the list of candidate services and sends it together with the query to the Matchmaking Subsystem. This subsystem handles different slave matchmakers (MM) in order to match the different parts of the query against the list of services. In particular the structural interface of the service is matched by the Structural MM, while security conditions are matched by Security matchmakers.

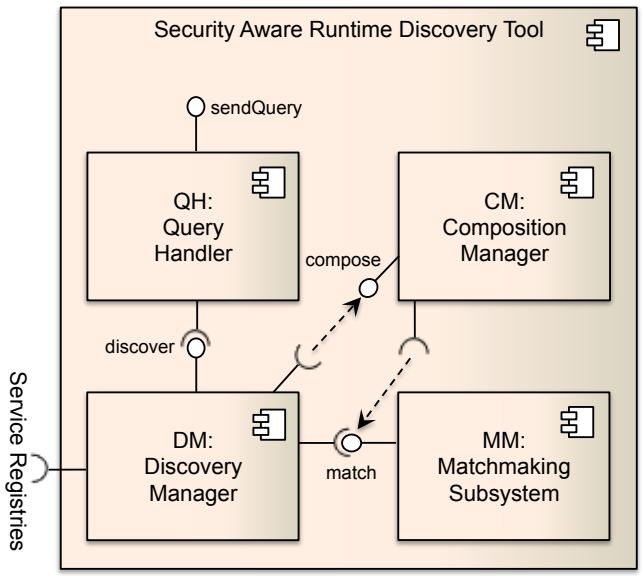


Figure 7.1: Architecture of the Security Aware Runtime Discovery Tool

Our tool supports the usage, through a plugin mechanism, of other matchmakers developed by third-parties, as the Ontological and Certificate Type Specific MMs developed in the context of the ASSERT4SOA project [5][80]. The Ontological Security MM is able to do matching on certificate attributes defined as concepts in an ontology, by comparing concepts,

instead of just comparing strings. The Certificate Type Specific MMs are based on the certificate type, allowing matching and comparison on the basis of characteristics of the type specific model used to certify the service (e.g. matching of a model described in the query with the one described in each security certificate).

If no services are found, the Discovery Manager calls the Composition Manager to try building a service composition that matches the query. The Composition Manager finds abstract workflows that match the searched functionality and potentially can satisfy the security requirements (following the GETSECUREWORKFLOWS algorithm in Section 6.4.4) and instantiates them with services that respect the inferred security requirements (using the WORKFLOWINSTANTIATION algorithm in Section 6.4.5). A BPEL process is then generated from each instantiated workflow, in order to make possible to execute the composition as a single service. Finally, the references to the discovered services are embedded in a response and sent by the Query Handler to the client.

7.2.3 Detailed Design of the Composition Manager

The purpose of the Composition Manager is to generate service compositions providing the queried functionality and security requirements on the overall, by discovering single services through the Discovery Manager.

7.2.3.1 Package compositionmanager

In this section we are going to describe the `compositionmanager` package through the usage of a Class Diagram and a table summarizing the purpose of each package and class.

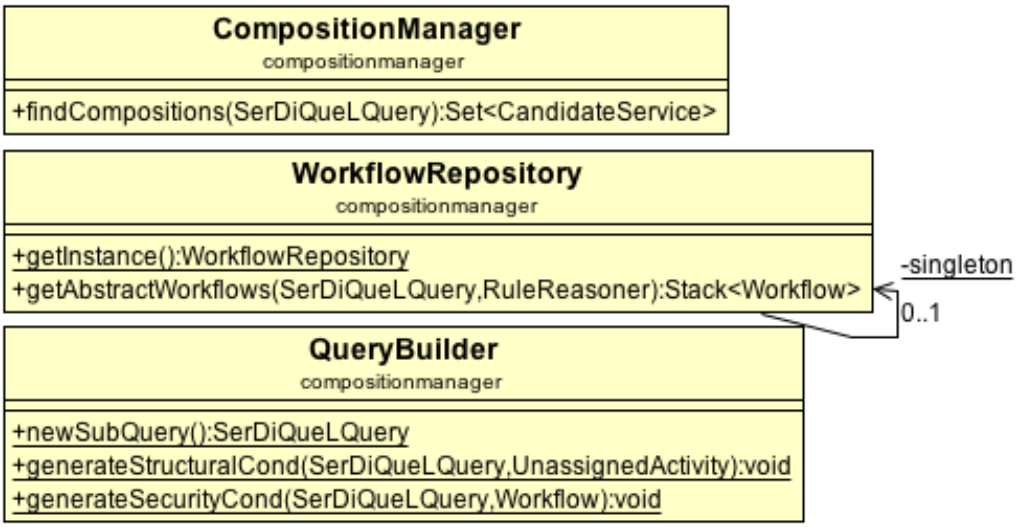


Figure 7.2: Class Diagram of the package compositionmanager

Package/Class	**Description**
composition manager	Provides the main functionalities of the composition manager
Composition Manager	The entry point of the component. Produces a list of service compositions from a discovery query, by implementing WORKFLOWINSTANTIATION algorithm in Section 6.4.5.
Workflow Repository	Retrieves a stack of abstract workflows from the repository, based on the structural and security part of the query, following the GETSECUREWORKFLOWS algorithm in Section 6.4.4.
QueryBuilder	Generates the service discovery queries used to instantiate the activities within a workflow.

Table 7.2: Description of the classes in the package compositionmanager

7.2.3.2 Package compositionmanager.secrule

In this section we are going to describe the `compositionmanager.secrule` package through the usage of a Class Diagram and a table summarizing the purpose of each package and class.

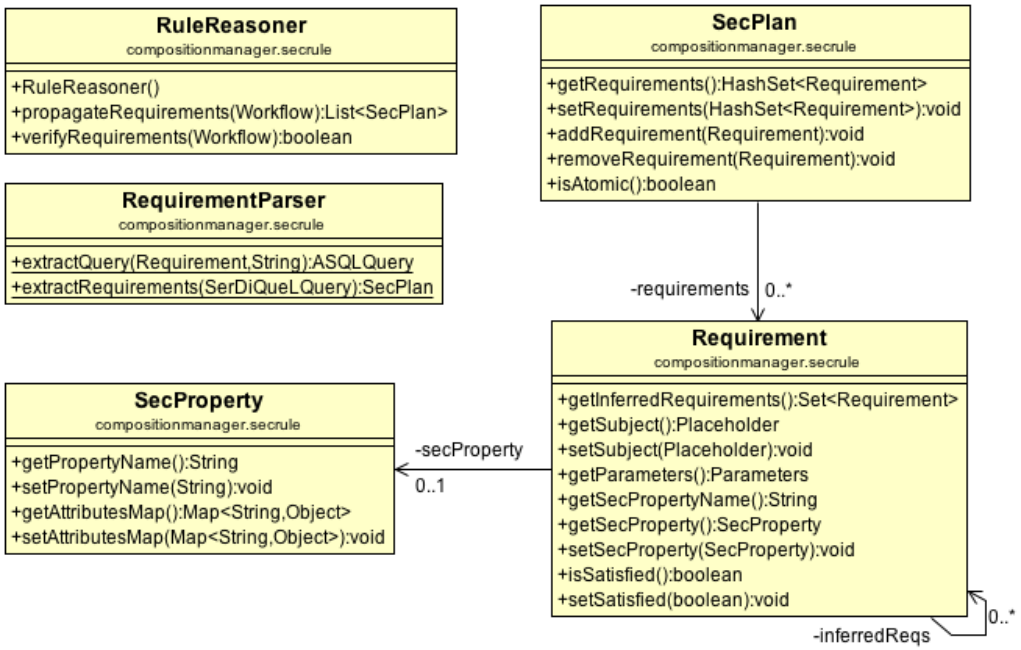


Figure 7.3: Class Diagram of the package compositionmanager.secrule

Package/Class	Description
composition manager .secrule	Provides the interconnection with the rule-based system by converting the query and the requirements and managing the Knowledge Base
RuleReasoner	Manages the Knowledge Base: it inserts the facts, fires the rules and retrieves the resulting facts. Through the rules it implements the INFERREQUIREMENT and VERIFYREQUIREMENT algorithms described in Section 6.4.1 and 6.4.2
RequirementParser	Converts the security conditions of the query into Requirement and vice versa.
SecPlan	Representation for the rule-based system of a collection of Requirement that compose a security plan.
Requirement	Representation for the rule-based system of the requirement of a security property upon an activity or a pattern of the workflow.
SecProperty	Security property description that allows a set of attributes to describe the specific instance of a property.

Table 7.3: Description of the classes in the package compositionmanager .secrule

7.2.3.3 Package compositionmanager.workflow

In this section we are going to describe the `compositionmanager.workflow` package through the usage of a Class Diagram and a table summarizing the purpose of each package and class.

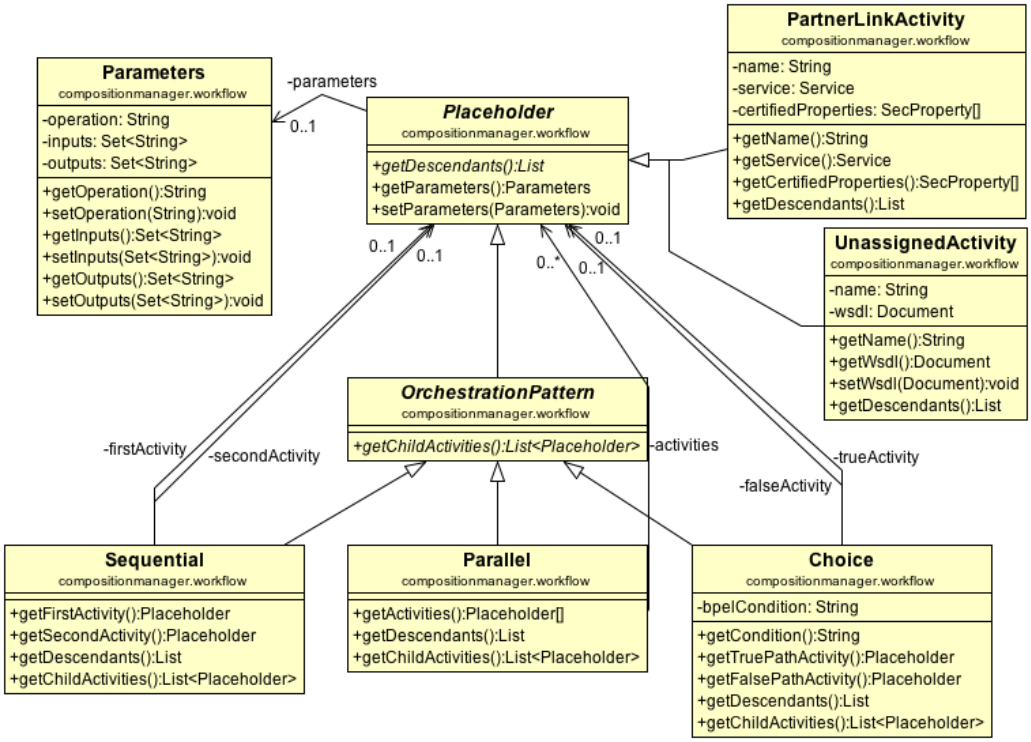


Figure 7.4: Class Diagram of the package compositionmanager .workflow

Package/Class	Description
composition manager .workflow	Contains the data structures representing the workflows: orchestration patterns and activities.
Workflow	Represents a workflow and contains the WSDL representation of the operation that it provides and a pattern that is the root of the workflow.
Placeholder	Represents a placeholder inside an orchestration pattern that can be fit by a service activity or another pattern.
OrchestrationP.	Represents a control flow pattern.
Sequential	Control flow pattern representing the sequential invocation of two placeholders.
Parallel	Control flow pattern representing the parallel invocation of two or more placeholders.
Choice	Control flow pattern representing the choice of which path of execution to take base on a condition (e.g. if-then-else).
Unassigned Activity	Represents an unassigned activity. Contains a WSDL representing the structure required from a service to instantiate it.
PartnerLink Activity	Represents an activity that has already been instantiated with a partner service. Contains also a list of the security properties guaranteed by the service through certificates.
Parameters	Contains the information about input and output parameters of the placeholder.

Table 7.4: Description of the classes in the package compositionmanager .workflow

7.2.3.4 Package compositionmanager.bpel

In this section we are going to describe the `compositionmanager.bpel` package through the usage of a Class Diagram and a table summarizing the purpose of each package and class.

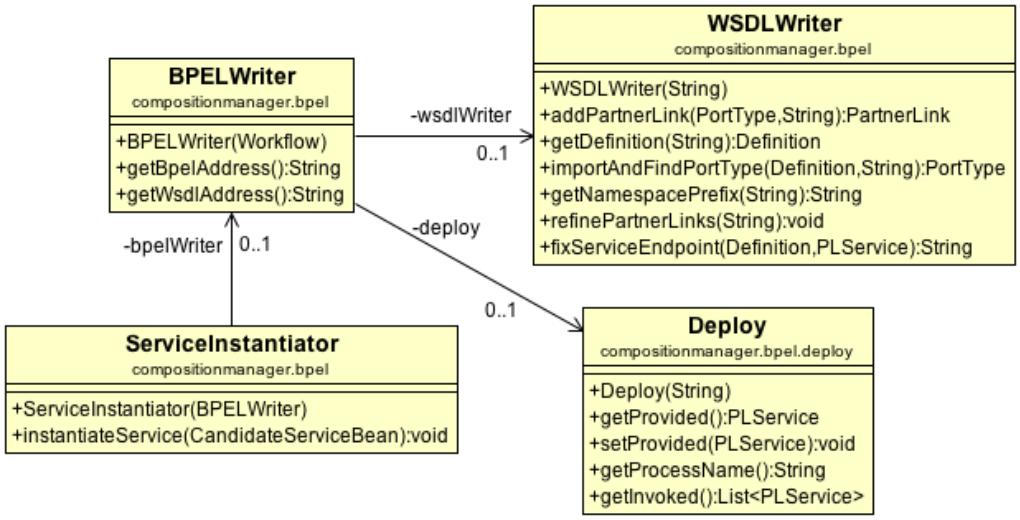


Figure 7.5: Class Diagram of the package compositionmanager.bpel

Package/Class	Description
composition manager .bpel	Provides the means to handle BPEL, WSDL and deployment descriptors for the generated compositions, in order to allow publishing the compositions and execute them straight away.
BPELWriter	Given an instantiated workflow (i.e., a composition), it produces a BPEL referring to the correct partner links.
WSDLWriter	Produces the WSDL of the composition, importing the WSDLs of the partner services.
Deploy	Representation of the deployment descriptor file used by Apache-ODE to run a BPEL file.
ServiceInstantiator	Makes the correct edits to the BPEL and WSDL in order to substitute the placeholder with the service that instantiate the composition.

Table 7.5: Description of the classes in the package compositionmanager.bpel

7.2.4 Example

The example used to demonstrate the approach is based on the Stock Brokerage scenario described in Section 6.2 and focuses on a Stock Service providing the current value of the given stock. In particular the Stock Service provides an operation called *Get Stock Value Details*. This operation takes as input the stock symbol of a given stock and returns the current stock value in USD dollars.

The query that requires the composition expresses a security requirement regarding integrity of data on the input and the output of the

activity realised by the service in the scenario. The service discovery query Q to replace the service providing the described operation, is shown in Table 7.6.

```
<?xml version="1.0" encoding="utf-8"?>
<tns:ServiceQuery xmlns:... name="QueryStockQuote"
   queryID="UUID:550e8400-e29b-41d4-a716-446655440060">
  <par:Parameters>
    <par:Mode value="PULL" />
    <par:Threshold value="0.1" />
    <par:Composition value="true" />
  </par:Parameters>

  <!-- Structural sub-query -->
  <tns:StructuralQuery>
    <wsdl:definitions xmlns:...>
       <wsdl:types>
         <xsd:schema elementFormDefault="qualified"
           targetNamespace="http://www.webserviceX.NET/">
           <xsd:element name="Quote">
             <xsd:complexType><xsd:sequence>
               <xsd:element minOccurs="0" maxOccurs="unbounded"
                  name="symbol" type="xsd:string" />
             </xsd:sequence></xsd:complexType>
           </xsd:element>
           <xsd:element name="QuoteResponse">
             <xsd:complexType><xsd:sequence>
               <xsd:element minOccurs="0" maxOccurs="unbounded"
                  name="USDValue" type="xsd:string" />
              </xsd:sequence></xsd:complexType>
           </xsd:element>
         </xsd:schema>
       </wsdl:types>
       <wsdl:message name="GetQuoteSoapIn">
          <wsdl:part name="parameters" element="tns:Quote" />
       </wsdl:message>
       <wsdl:message name="GetQuoteSoapOut">
          <wsdl:part name="parameters" element="tns:QuoteResponse" />
       </wsdl:message>
```

```
      <wsdl:portType name="StockQuoteSoap">
         <wsdl:operation name="GetUSDStockQuote">
            <wsdl:input message="tns:GetQuoteSoapIn" />
            <wsdl:output message="tns:GetQuoteSoapOut" />
         </wsdl:operation>
      </wsdl:portType>
   </wsdl:definitions>
</tns:StructuralQuery>

<AssertQuery name="A1" type="HARD" assertScope="SINGLE">
   <LogicalExpression>
      <Condition relation="EQUAL-TO">
         <Operand1>
            <AssertOperand facetType="Assert">
          //ASSERTCore/ToC/Assets/Asset[@Type='InputParameter']/Name
            </AssertOperand>
         </Operand1>
         <Operand2>
            <Function name="WSDLLookup"><Arguments>
                <Argument WSDLElementType="input-message"
                      WSDLElementName="symbol" />
             </Arguments></Function>
         </Operand2>
      </Condition>
      <LogicalOperator>AND</LogicalOperator>
      <LogicalExpression>
        <Condition relation="EQUAL-TO">
          <Operand1>
             <AssertOperand facetType="Assert">
            //ASSERTCore/SecurityProperty/@PropertyAbstractCategory
             </AssertOperand>
          </Operand1>
          <Operand2>
             <Constant type="STRING">Integrity</Constant>
          </Operand2>
         </Condition>
      </LogicalExpression>
   </LogicalExpression>
</AssertQuery>
```

```
  <AssertQuery name="A2" type="HARD" assertScope="SINGLE">
    <LogicalExpression>
       <Condition relation="EQUALS-TO">
          <Operand1>
             <AssertOperand facetType="Assert">
          //ASSERTCore/ToC/Assets/Asset[@Type='OutputParameter']/Name
             </AssertOperand>
          </Operand1>
          <Operand2>
             <Function name="WSDLLookup"><Arguments>
               <Argument WSDLElementType="input-message"
                     WSDLElementName="USDValue" />
             </Arguments></Function>
          </Operand2>
       </Condition>
       <LogicalOperator>AND</LogicalOperator>
       <LogicalExpression>
          <Condition relation="EQUIVALENT-CLASS">
            <Operand1>
              <AssertOperand facetType="Assert">
              //ASSERTCore/SecurityProperty/@PropertyAbstractCategory
              </AssertOperand>
            </Operand1>
            <Operand2>
              <Constant type="STRING">Integrity</Constant>
            </Operand2>
          </Condition>
       </LogicalExpression>
    </LogicalExpression>
  </AssertQuery>

</tns:ServiceQuery>
```

Table 7.6: Stock Service replacement query

When Stock Service becomes unavailable, the discovery query Q is executed. If the single service discovery doesn't find any service, service composition is performed.

The first task is to discover workflows that can support the queried security requirements; this is accomplished by the GETSECUREWORKFLOWS(Q) algorithm. The algorithm discovers an abstract workflow W matching the structural constraints, containing three activities connected by two Cascade Patterns. The first activity of the outer sequence is *Get ISIN*, which converts the symbol identifying the stock into the ISIN (that is another identifier of the stock). The second activity is instantiated with the other Cascade Pattern.

```
Workflow[orchestration=
  Sequential(
    UnassignedActivity(GetISIN),
    Sequential(
        UnassignedActivity(GetQuoteActivity),
        UnassignedActivity(GetConversion) ) )]
```

Figure 7.6: Representation of the abstract workflow extracted from the tests execution

The first activity of this inner sequence is *Get Stock Quote*, which returns the current stock value in EUR given the stock ISIN. The second activity is *Get Currency Converter*, which converts a given amount from EUR to USD.

To infer the security requirements for each of the services that are going to instantiate the activities, the INFERREQUIREMENT(RSP, W) algorithm is called. In particular the requirement about the integrity is propagated by the pattern described in Section 5.3.2.1, generating three new requirements about integrity for the activity placeholders.

At first the rule representing said pattern is fired for the security requirement on the external sequential orchestration. As a consequence of this rule, the requirement of integrity on the inputs and outputs is inferred to

both the activities in the pattern: *Get ISIN* and the other (internal) sequential orchestration.

The new security requirement on the internal sequential orchestration then fires again the rule above. This rule then splits the requirement into two different integrity requirements: one for *Get Stock Quote* and one for *Currency Converter*. Other abstract workflows with appropriate security requirements are discovered in the same way and added to the stack in output.

The workflows and their security requirements are then passed to the WORKFLOWINSTANTIATION(WStack, ReqMap) algorithm in order to discover appropriate partner services for the workflows. The first unassigned activity of the first workflow in the stack has to be instantiated, in this case *Get ISIN* of W. The query for *Get ISIN* is then built from the structural specifications in the workflow and the security requirements generated in the previous step, namely integrity for its inputs and outputs, as shown in Table 7.7.

```
<tns:ServiceQuery xmlns:... ...>
  <par:Parameters>
    <par:Mode value="PULL" />
    <par:Threshold value="0.1" />
  </par:Parameters>

  <tns:StructuralQuery>
    <wsdl:definitions xmlns:...>
      <wsdl:types>
        <xsd:schema elementFormDefault="qualified" ...>
          <xsd:element name="GetISINReq">
            <xsd:complexType><xsd:sequence>
                <xsd:element minOccurs="0" maxOccurs="unbounded"
                    name="symbol" type="xsd:string" />
            </xsd:sequence></xsd:complexType>
          </xsd:element>
          <xsd:element name="GetISINRes">
            <xsd:complexType><xsd:sequence>
                <xsd:element minOccurs="0" maxOccurs="unbounded"
                    name="ISIN" type="xsd:string" />
            </xsd:sequence></xsd:complexType>
          </xsd:element>
        </xsd:schema>
      </wsdl:types>
      <wsdl:message name="GetISINSoapIn">
        <wsdl:part name="parameters" element="tns:GetISINReq" />
      </wsdl:message>
      <wsdl:message name="GetISINSoapOut">
        <wsdl:part name="parameters" element="tns:GetISINRes" />
      </wsdl:message>
      <wsdl:portType name="StockISINSoap">
        <wsdl:operation name="GetISIN">
          <wsdl:input message="tns:GetISINSoapIn" />
          <wsdl:output message="tns:GetISINSoapOut" />
        </wsdl:operation>
      </wsdl:portType>
    </wsdl:definitions>
  </tns:StructuralQuery>

  <AssertQuery name="A1" type="HARD" assertScope="SINGLE">
```

```
41      <LogicalExpression>
42         <Condition relation="EQUAL-TO">
43            <Operand1>
44               <AssertOperand facetType="Assert">
45          //ASSERTCore/ToC/Assets/Asset[@Type='InputParameter']/Name
46               </AssertOperand>
47            </Operand1>
48            <Operand2>
49               <Function name="WSDLLookup"><Arguments>
50                 <Argument WSDLElementType="input-message"
51                       WSDLElementName="symbol" />
52               </Arguments></Function>
53            </Operand2>
54         </Condition>
55         <LogicalOperator>AND</LogicalOperator>
56         <LogicalExpression>
57            <Condition relation="EQUAL-TO">
58              <Operand1>
59                <AssertOperand facetType="Assert">
60            //ASSERTCore/SecurityProperty/@PropertyAbstractCategory
61                </AssertOperand>
62              </Operand1>
63              <Operand2>
64                <Constant type="STRING">Integrity
65                </Constant>
66              </Operand2>
67            </Condition>
68         </LogicalExpression>
69      </LogicalExpression>
70   </AssertQuery>
71
72   <AssertQuery name="A2" type="HARD" assertScope="SINGLE">
73      <LogicalExpression>
74         <Condition relation="EQUALS-TO">
75            <Operand1>
76               <AssertOperand facetType="Assert">
77        //ASSERTCore/ToC/Assets/Asset[@Type='OutputParameter']/Name
78               </AssertOperand>
79            </Operand1>
80            <Operand2>
```

```
81              <Function name="WSDLLookup"><Arguments>
82                <Argument WSDLElementType="input-message"
83                      WSDLElementName="ISIN" />
84              </Arguments></Function>
85           </Operand2>
86        </Condition>
87        <LogicalOperator>AND</LogicalOperator>
88        <LogicalExpression>
89           <Condition relation="EQUIVALENT-CLASS">
90             <Operand1>
91               <AssertOperand facetType="Assert">
92          //ASSERTCore/SecurityProperty/@PropertyAbstractCategory
93               </AssertOperand>
94             </Operand1>
95             <Operand2>
96               <Constant type="STRING">Integrity
97               </Constant>
98             </Operand2>
99           </Condition>
100       </LogicalExpression>
101    </LogicalExpression>
102  </AssertQuery>
103
104 </tns:ServiceQuery>
```

Table 7.7: Get ISIN discovery query

More specifically, lines 1 – 5 contain generic query parameters, indicating the type of query execution (i.e., PULL), and the minimum distance value for accepting results. Lines 7 – 38 show the structural conditions defining the required interface of the service using a WSDL specification.

Lines 40 – 70 contain a first set of security conditions (called `A1`) that must be evaluated against a single certificate (as their scope is `SINGLE`) and satisfied by all candidate services (as their type is `HARD`). The conditions specify that the input Symbol must be in the list of assets on which the

certification of the security property is applied (lines 41 – 54) and that certified security property provided by the service is called Integrity (lines 56 – 68).

A second set of security conditions is shown in lines 72 – 102 and can be evaluated against different certificates then the first one. In particular even these conditions must be evaluated against a single certificate (as their scope is `SINGLE`) and satisfied by all candidate services (as their type is `HARD`). The conditions specify that the output ISIN must be in the list of assets on which the certification of the security property is applied (lines 73 – 86) and that the certified security property provided by the service is called Integrity (lines 88 – 100).

The service discovery executes the query to find in the registry a set of services that match the structural and security conditions. Each service is used to instantiate a new workflow copy of W.

```
Workflow[orchestration=
  Sequential(
    PartnerLinkActivity(http://example.com/StockISIN.wsdl),
    Sequential(
        UnassignedActivity(GetQuoteActivity),
        UnassignedActivity(GetConversion) ) )]
```

Figure 7.7: Representation of the abstract workflow after the first instantiation

The activity is now bounded to a partner service, represented by its WSDL address in Figure 7.7, and it contains the security properties extracted from the certificates associated to the service. All the instantiated workflows are added to the stack, as they still need partner services for the other activity placeholders.

Similarly a query for the second activity is created and executed, and the workflow gets instantiated again:

```
Workflow[orchestration=
  Sequential(
    PartnerLinkActivity(http://example.com/StockISIN.wsdl),
    Sequential(
        PartnerLinkActivity(http://example.com/QuoteFromISIN.wsdl),
        UnassignedActivity(GetConversion) ) )]
```

Figure 7.8: Representation of the abstract workflow after the second instantiation

And finally the third one, completing the workflow:

```
Workflow[orchestration=
  Sequential(
    PartnerLinkActivity(http://example.com/StockISIN.wsdl),
    Sequential(
        PartnerLinkActivity(http://example.com/QuoteFromISIN.wsdl),
        PartnerLinkActivity(http://example.com/CurrConvert.wsdl) ) )]
```

Figure 7.9: Representation of the final workflow

Figure 7.10 shows the final results of the discovery from a GUI. On the left side is possible to select a partner service to replace, after selecting one, the discovery query for the service is shown on the bottom left part of the window. After executing the query, is possible to see the results on the right side. The top part shows all the alternatives that were found, whilst the middle part shows the workflow of the selected composition.

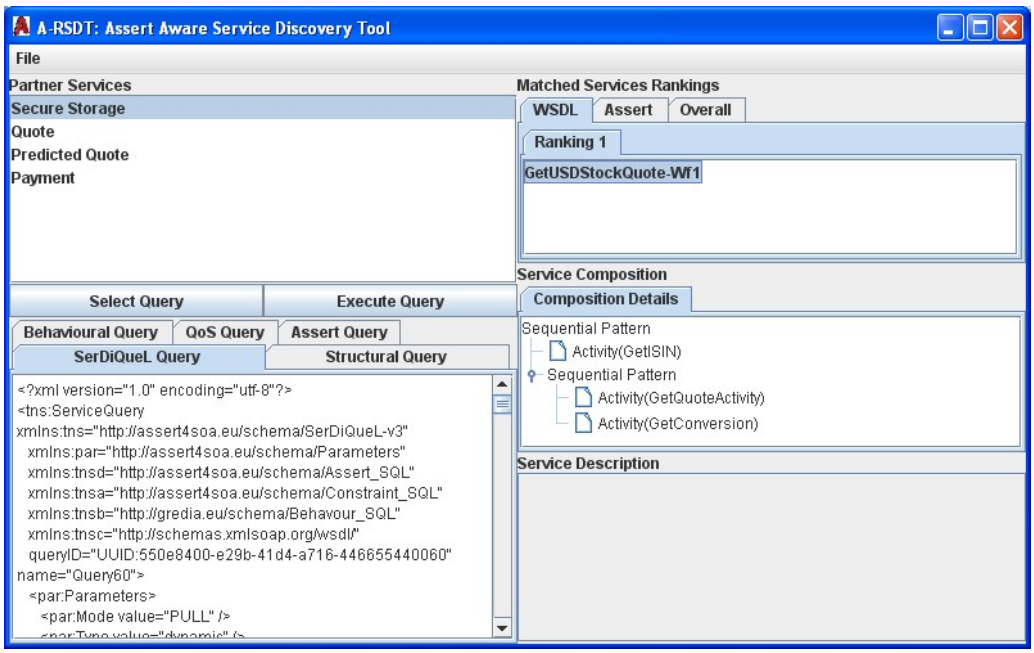


Figure 7.10: Execution result of the composition example

7.3 Security Aware BPEL Design Tool

The aim of the Security Aware BPEL Design Tool is to facilitate the tasks of an SBS Designer by allowing the specification of security requirements during the designing of a BPEL process. The information about security requirements provide the means to timely validate or substitute services, so that the services are appropriate for the application being developed.

The specification of security requirements is not possible in the common BPEL design tools, but it is rather important part of the model of a process. The Security Aware BPEL Design Tool allows defining security requirements to an *invoke* activity or to the control flow constructs that contain other activities (e.g. *scope*, *sequence*, *flow*, *pick*, *if-then-else*, …). After such definition, the SBS Designer can ask the system to validate the security requirements, i.e., check that the service operations associated with

the activities subject to a security requirement guarantee the requirements, without having to investigate this with external tools. If a service doesn't guarantee the requirements, the SBS Designer can use the tool to explore and select an alternative service that complies with the requirements or, if no atomic service is found, an alternative workflow that can be substituted with the activity to satisfy the requirements. After selection of an alternative service or service composition, the BPEL process is automatically adapted in order to use the new service or to incorporate the composition workflow. This addition allows the SBS Designer to fix BPEL processes that do not respect security requirements without having to (i) manually check the satisfaction of the requirements, (ii) write a query to search for alternative services that respect the security requirements, (iii) submit the query to the discovery engine and (iv) change manually the BPEL process with a suitable service suggested by the discovery engine.

7.3.1 Architecture

The overall architecture of the Security Aware BPEL Design Tool is shown in Figure 7.11. As shown in the figure, the tool is based on the BPEL Designer plugin of Eclipse and on the Security Aware Runtime Discovery Tool that is used as plugin for the Eclipse IDE platform. The new component we introduced is a Security Extension package for the BPEL Designer, were all the new functionalities are deployed.

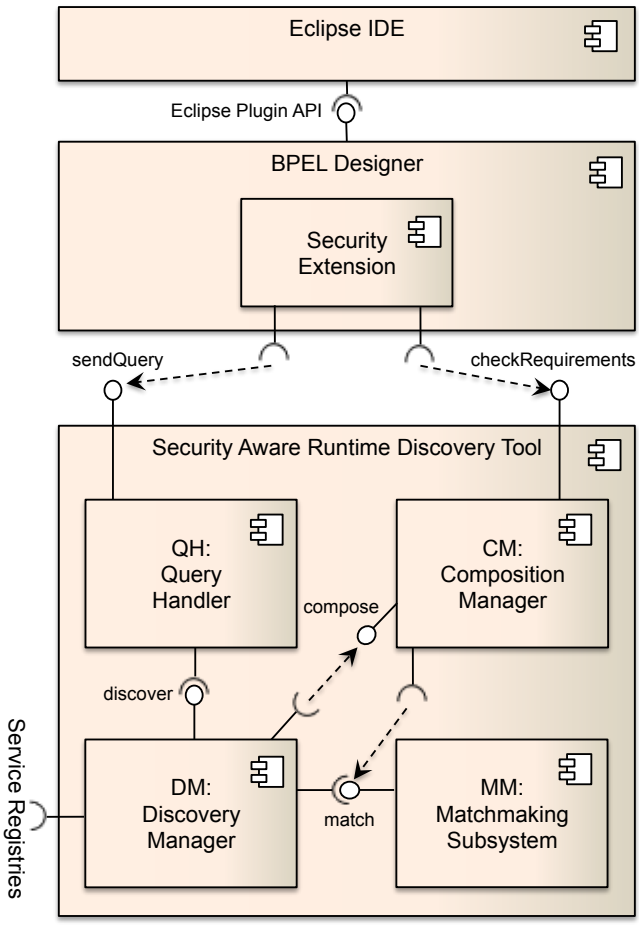


Figure 7.11: Architecture of the Security Aware BPEL Design Tool

BPEL Designer is an Eclipse plugin that offers comprehensive support for the definition, authoring, editing, deploying, testing and debugging of WS-BPEL 2.0 processes through Eclipse IDE. In the development of the Security Aware BPEL Design Tool, we have extended the plugin with the Security Extension component that: (i) allows the specification of security requirements for the *invoke* and control flow activities in a BPEL process, (ii) introduces a new button to request the validation of the security requirements, (iii) shows the results of the validation and eventual alternative services or service compositions that satisfy the requirements, and (iv) allows the adaptation of the BPEL process by replacing an existing

service linked with an *invoke* activity with an alternative service or replacing the *invoke* activity altogether with a service composition. In doing (iv), we also guarantee that under certain conditions the modified BPEL process can be executed.

7.3.2 Detailed Design of the Security Extension

The purpose of the Security Extension is to introduce the definition, validation and adaptation based on security requirements to the BPEL Designer plugin of Eclipse. This component is also the intermediary to the Security Aware Runtime Discovery Tool at design time.

7.3.2.1 Package securityextension

In this section we are going to describe the `securityextension` package through the usage of a Class Diagram and a table summarizing the purpose of each package and class.

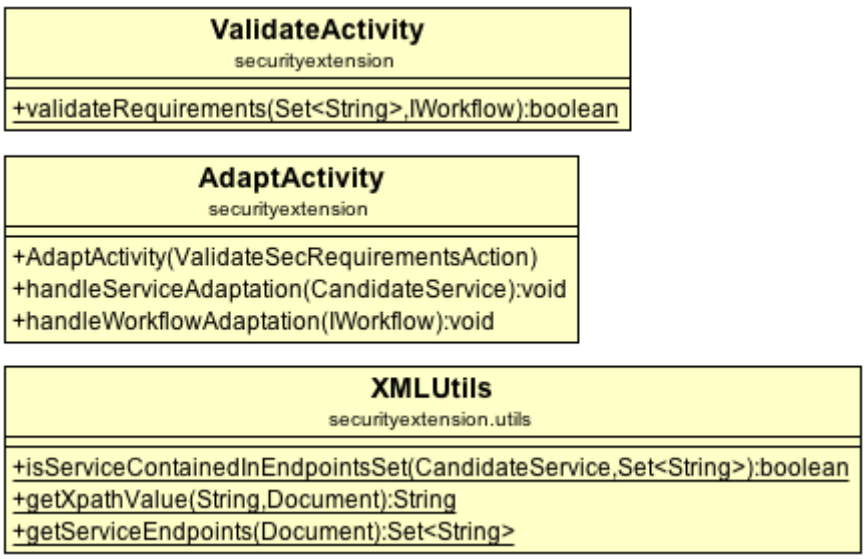


Figure 7.12: Class Diagram of the package securityextension

Package/Class	**Description**
securityextension	Provides the main functionalities of the security extension of the BPEL Designer
ValidateActivity	Checks the validity of the security requirements by contacting the appropriate functions in the Composition Manager and the Query Handler. It implements the VALIDATEWORKFLOW algorithm described in Section 6.4.3.
AdaptActivity	Adapts the BPEL process by replacing a service with another one in an invoke activity or by replacing the invoke activity altogether with the workflow of a service composition.
utils.XMLUtils	Utility class to handle XML documents.

Table 7.8: Description of the classes in the package securityextension

7.3.2.2 Package securityextension.securitymodel

In this section we are going to describe the `securityextension .securitymodel` package through the usage of a Class Diagram and a table summarizing the purpose of each package and class.

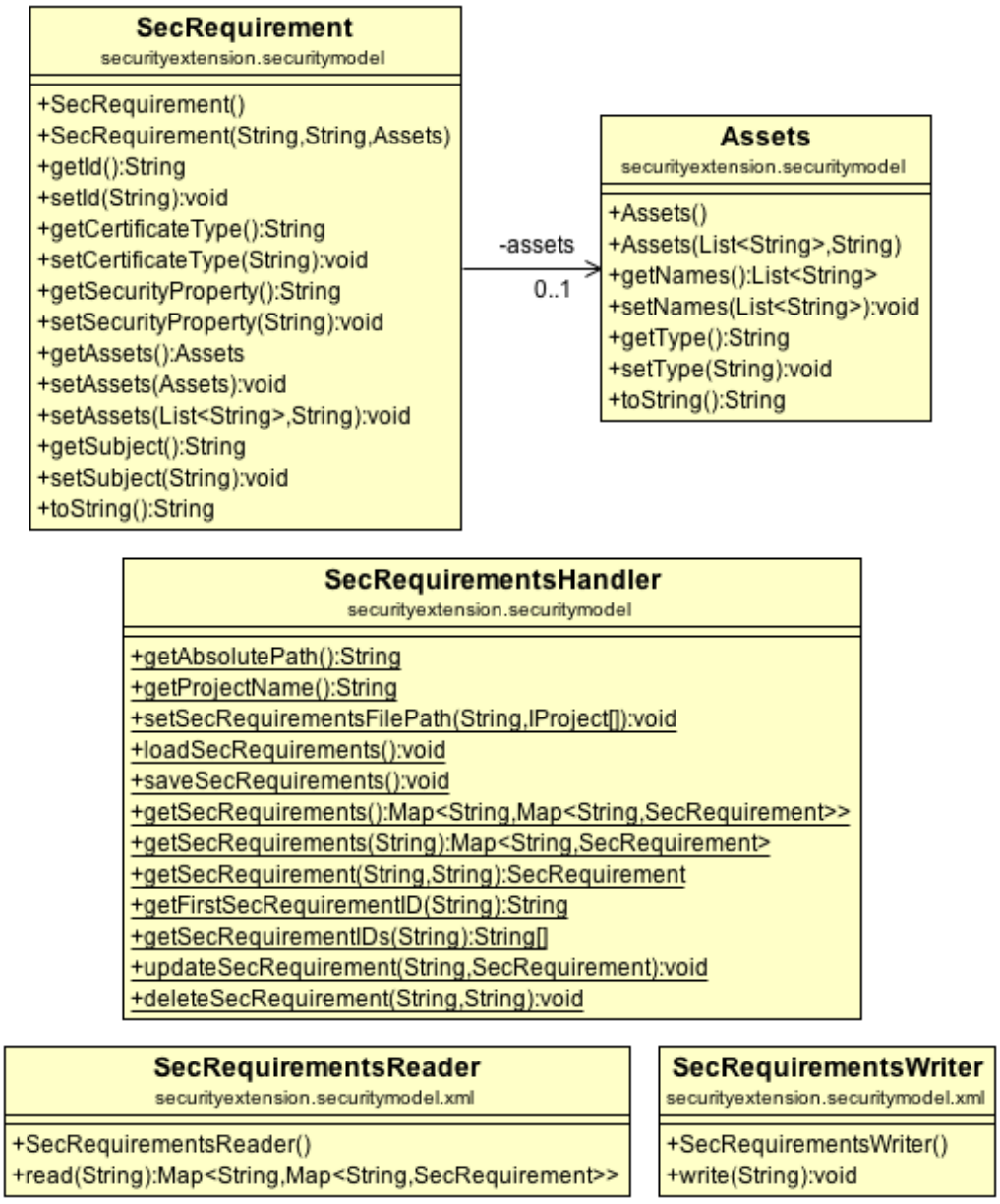


Figure 7.13: Class Diagram of the package securityextension .securitymodel

Package/Class	Description
securityextension .securitymodel	Provides the representation and the utilities to handle the Security Requirements linked with a BPEL process.
SecRequirement	Representation of a security requirement for an activity of a BPEL process. It allows the specification of an identifier, an activity that is subjected to the requirement, a security property and other details about the certificate and the assets for which the security property should hold.
Assets	Representation of the assets for which a property should hold. Each asset has a type and a name.
SecRequirements Handler	Provides the management facilities for the security requirements of a BPEL process, allowing to load and save them into files and to access to them.
SecRequirements Reader	Parses the XML file representation of a security requirement to the object representation for the SecRequirementsHandler.
SecRequirements Writer	Writes the XML file representation of a security requirement from the object representation for the SecRequirementsHandler.

Table 7.9: Description of the classes in the package securityextension .securitymodel

7.3.2.3 Package securityextension.gui

In this section we are going to describe the `securityextension.gui` package through the usage of a Class Diagram and a table summarizing the purpose of each package and class.

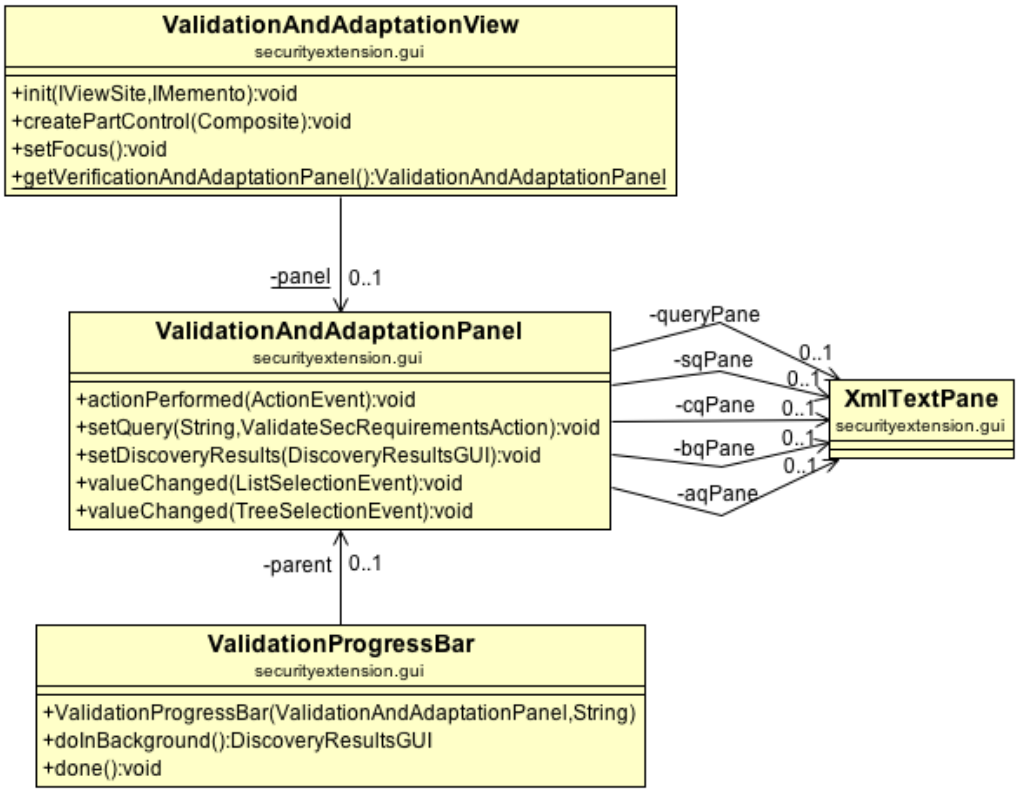


Figure 7.14: Class Diagram of the package securityextension.gui

Package/Class	Description
securityextension .gui	Provides the graphical user interface additions of the Security Extension.
ValidationAnd AdaptationView	Defines a new View for the Eclipse Workbench, comprising a tab where the information for the Validation and Adaptation are shown.
ValidationAnd AdaptationPanel	Defines the interface of the Validation and Adaptation panel. It shows the information about the queries executed for the validation process, the result of the validation and a list of alternative services or service compositions that satisfy the requirements.
XmlTextPane	Formats the XML of the queries in order to highlight the different parts of the XML encoding.
Validation ProgressBar	Progress bar that is shown during the validation process.

Table 7.10: Description of the classes in the package securityextension .gui

7.3.2.4 Other Packages

The Security Extension introduces also several classes that conceptually should be part of the existing packages of the BPEL Designer. In particular all the classes should be part of the packages children of the `org.eclipse.bpel.ui` package. In order to isolate our additions from the original code of the BPEL Designer, these classes have been placed in the `securityextension` package, but retaining the part of the package name that should have been placed after `org.eclipse.bpel.ui`.

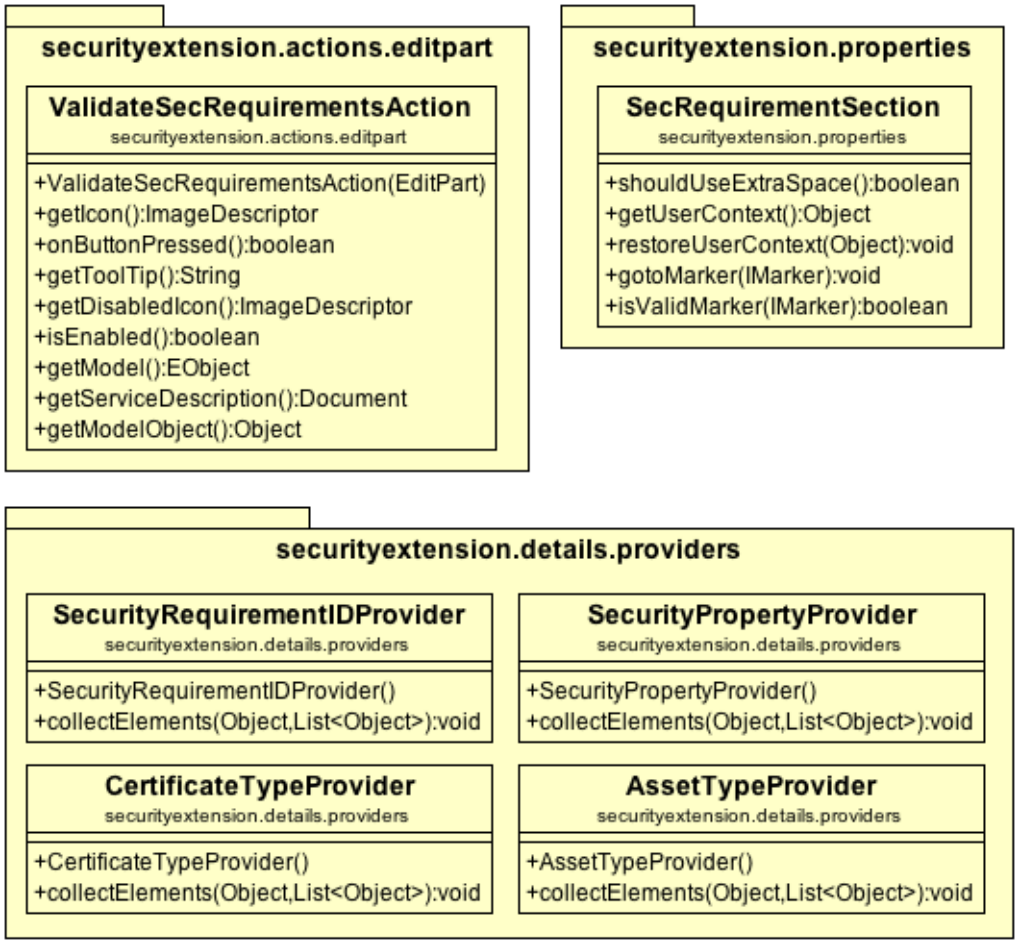


Figure 7.15: Class Diagram of other packages part of the extension

Class	Description
actions.editpart. Validate SecRequirements	Defines a new button and the action to perform when pressed. The button allows requesting the validation of security requirements.
properties. SecRequirements Section	Defines a new property section for a given BPEL activity where it is possible to specify the security requirements for the activity.
details.providers.*	Retrieve the information to populate the combo-boxes in the SecRequirementsSection.

Table 7.11: Description of the classes in the other packages part of the extension

7.3.3 Example

The specification of the security requirements for an activity of the BPEL process is performed under the Properties view of the activity. To open the Properties view, right click on the activity that you wish to check and select “Show in Properties” option in the pop up menu. Then in the Properties tab select “Security Specification” and use it to specify the requirements.

As shown in Figure 7.16, in the security specification tab the designer can select different parameters to define a security requirement. More specifically, the designer needs to specify an ID for the security requirement, and select the *category* of the security property. The selection of the security property can be made from a list of possible categories. Optionally, the designer may also select the *certificate type* and the *asset name* and *type* that should guarantee the selected security property, where the assets can be selected from the input or output parameters. After

completion, pushing the "Add" button adds the security requirement to the list; in this way it is possible to define multiple security requirements for the same activity (to navigate through them, use the "Security Properties" dropdown menu).

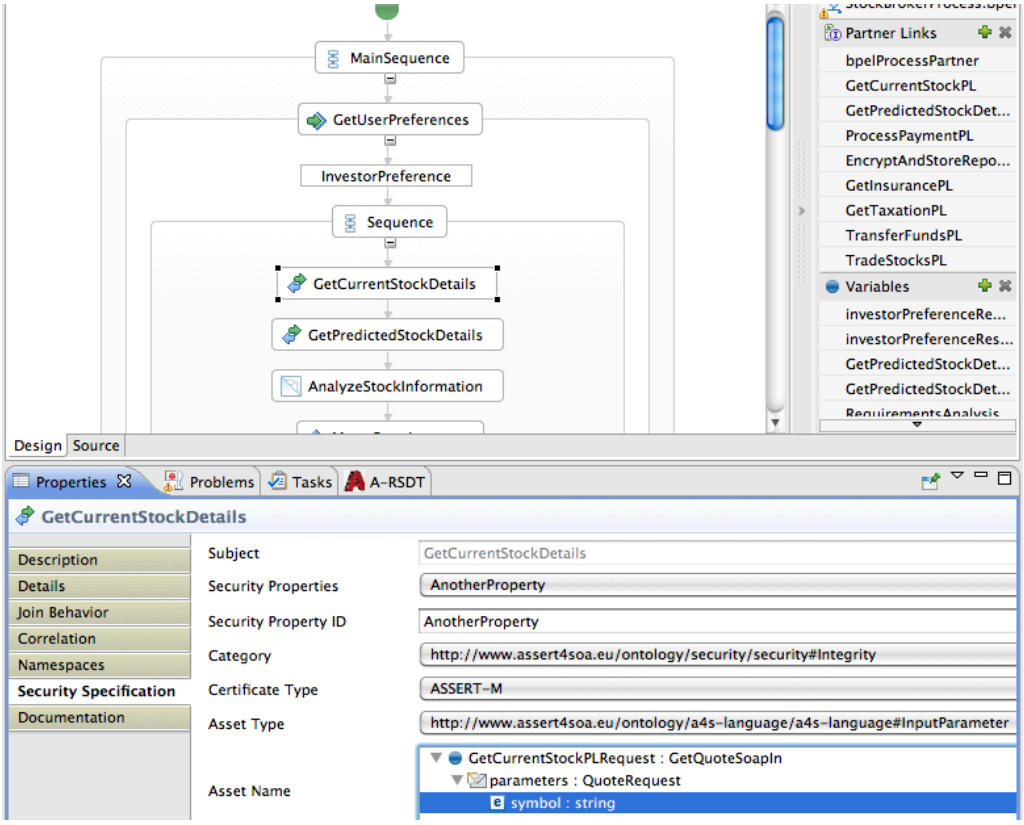


Figure 7.16: Security Specification for the GetCurrentStockDetails activity

In the figure a security requirement is specified for the activity *GetCurrentStockDetails* of the Stock Brokerage scenario (described in Section 6.2), requiring that the Integrity of the input *symbol* is preserved.

Once the security requirement is specified, the designer can select the "Verify Security Properties" option from the activity's contextual menu in order to check that the services used as a partner links for the activity

satisfy the security requirements. This is obtained by right clicking on the activity to obtain the menu shown in Figure 7.17, and by clicking on the appropriate option.

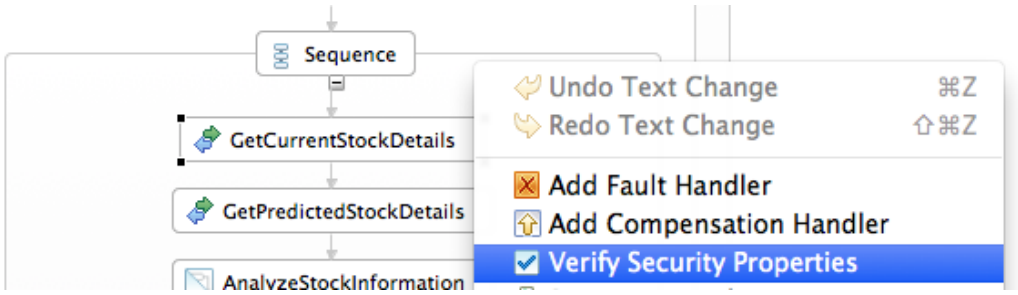


Figure 7.17: The activity contextual menu showing the "Verify Security Property" option

After selecting "Verify Security Properties", the tool opens the validation and adaptation view and a progress bar notifies that the validation process is ongoing. Once the validation is done, the results are displayed in the new view, as shown in Figure 7.18.

In particular, the left side of the view shows the query that was used to validate the security requirements. Sub tabs in this part allow the selection of different parts of this query, namely the structural, behavioural, QoS and security related query part. On the right the validation result is shown (under *Security Property Verification Status*) and a list of other services or service compositions that satisfy the same requirements, and could be used as a replacement.

In our example the partner service used for the *GetCurrentStockDetails* activity does not satisfy the requirements: this is shown by the display of the status message "Service does not satisfy security requirements" in red. Note that if the requirements were satisfied the status would be displayed by a green message saying: "Service satisfies security requirements".

Furthermore, in this case, no single service satisfying the security requirements is found. Hence, the tool identified a service composition that could substitute the activity and guarantee the security requirements. By selecting the composition it is possible to investigate the workflow of the composition, as shown in the figure (see the panel *Composition Details* in the bottom right part of the figure). The discovered composition has been based on the sequential orchestration and is explained in the example section of the Security Aware Runtime Discovery Tool.

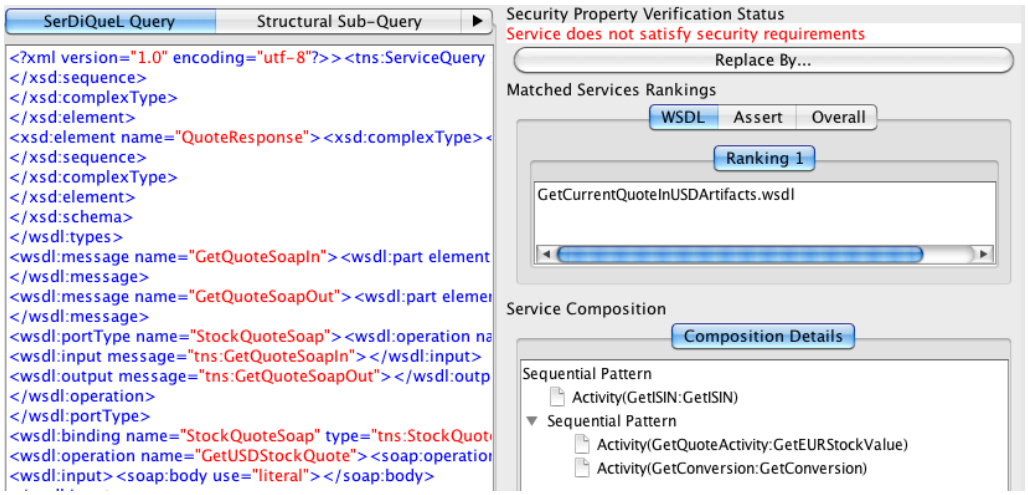


Figure 7.18: The Validation and Adaptation view

If the security requirements are not satisfied by the service specified as a partner link for an *invoke* activity, it is possible to select a replacement service or service composition and click the “Replace By…” button. This will cause the adaptation of the BPEL process with the selected service or service composition.

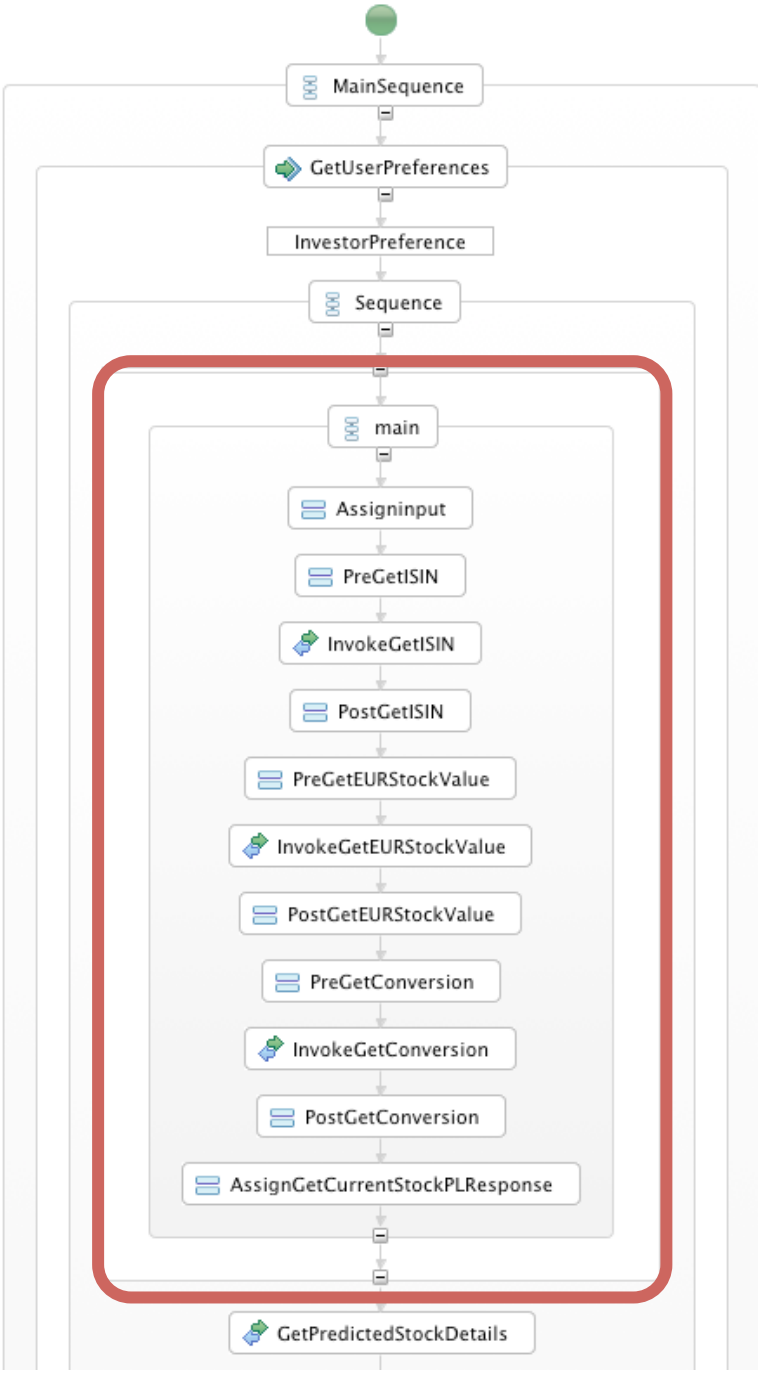


Figure 7.19: BPEL process after the adaptation of a service composition in place of the *GetCurrentStockDetails* activity

In this example, when the designer selects the alternative composition and selects the “Replace By…” button, the *GetCurrentStockDetails* activity is replaced with the activities of the composition, within a *scope*, as shown in Figure 7.19 (the part highlighted in red is the composition that replaced the *GetCurrentStockDetails* activity).

7.4 Summary

In this chapter we described two tools that have been implemented and that make usage of the pattern-based Security Aware Service Composition approach described in Chapters 5 and 6.

The first tool, Security Aware Runtime Discovery Tool, can be used to find services based on discovery queries that can include security requirements. The innovative feature of the tool is the support to generation and discovery of service compositions that respect the security requirements. The service compositions are automatically deployed in a server, ready to be executed, allowing seamless runtime adaptation of a SBS with service compositions.

The second tool is the Security Aware BPEL Design Tool. This tool is an extension of the BPEL Designer plugin for Eclipse that allows the introduction of security requirements and the security validation and adaptation for a SBS being built as a service orchestration. The graphical user interface offers seamless integration of the validation algorithm, the discovery platform and the service adaptation. The latter is performed on user request in order to replace a service that does not respect security requirements with a service or a service composition that does so. In particular the service orchestration is automatically updated in order to include new activities that call the new partner services.

The two tools described allow taking advantage of the Security Aware Service Composition approach in the different stages of SBSs lifecycle.

Chapter 8

Evaluation

8.1 Overview

This chapter describes the performance study performed on the discovery tool. The approach adopted for the test session is described in Section 8.2. Results are reported and analysed in Section 8.2.3. Finally, Section 8.4 summarizes the results and considerations on the evaluation and application of the framework.

8.2 Evaluation Setup

8.2.1 Scenario

The performance study has been based on services and security requirements part of the Stock Brokerage scenario described in Section 6.2. The scenario focuses on the discovery and integration of services in a SBS that automates the stock purchase using some stock investor preferences. The scenario devises a set of services and security requirements that are needed by the SBS; we give here, as an example, a short description of the service providing *Get Current Stock Detail*. *Get Current Stock Details* provides information about the current value of specific stocks. It takes as input the company code in string format (`symbol`) and returns the current

dollar value of the stock in string format (`USDvalue`). The SBS introduces the following security requirements for *Get Current Stock Details*:

- Confidentiality of Data
- Integrity of Data
- Availability

8.2.1.1 Service Registry

Several incarnations of all the services in the scenario have been added to a registry. The registry has been augmented also with some additional services not part of the scenario to simulate a real world scenario, where the services that one might be interested in have to be discovered from a broader set of services. The WSDLs of the additional services come from the QWS Dataset [2][3], i.e., a collection of real web services available on the public web that is offered as a basis for tests and researches.

The registry has also been populated with security certificates and each service has been associated with a variable number of certificates. In particular an initial set of security certificates have been composed manually to fulfil the security requirements introduced above. Additional security certificates have been randomly generated through the support of an automated tool.

In more details, the registry contains 1200 service WSDLs, 44 of them are part of the stock brokerage scenario, while the others come from the QWS dataset. The registry has a very disparate composition, as in a real world scenario, with services offering an average of five operations (with a range from 1 to 264 operations per service) and transmitting input and output messages with an average of five data types per interaction (with a

range from 0 to 111 data types per operation). Such a disparate composition is important for this kind of evaluation, as the performances of the structural matching are strictly linked with the complexity of the operation and data graphs of the WSDLs in the registry.

On the security side, the registry contains 3663 security certificates; meaning that each service has as an average three security certificates. In particular the registry contains services associated with a minimum of one and a maximum of twelve security certificates.

A total of 91 security certificates have been manually generated to cover other possible security requirements for the stock brokerage scenario. The produced security certificates cover different security properties such as Confidentiality, Authenticity, Integrity, Privacy, Availability and Non-Repudiation, with each property being in at least four different security certificates.

Finally, to properly support the composition scenario, the repository containing abstract workflows (i.e., service coordination processes that realize known business processes through the definition of fixed interfaces for potential participating services, as described in Section 6.2) has been populated with 20 business processes.

8.2.1.2 Service Discovery Queries

A-SerDiQueL queries are used, for the sake of this study, as an instrument to investigate the behaviour of the discovery tool. A set of three queries has been formulated to test the different rules in the system used by the composition process.

The queries differ mainly for the security conditions that affect the matching of services based on their security certificates. The structural

conditions in the queries instead, i.e., the ones that affect matching based on the service interface (WSDL), target all the same interface (i.e., one that provides *Get Current Stock Details*), as structural matching is a basic aspect of service discovery that was not part of the investigation (the interested reader can find some performance considerations on the traditional service discovery in [117]). The security conditions for the queries are:

Q1. Security Property: Integrity AND
Assets contains: `symbol` (input) AND
Assets contains: `USDvalue` (output)

Q2. Security Property: Perfect Security Property

Q3. Security Property: Availability AND
Maximum execution time ≤ 1000 millisec

Take for example query Q1. The query requires a specific Security Property (i.e., Integrity). The other conditions that must hold to include a service in the results is that the certificate matched by the previous condition of this query has the `symbol` input element and the `USDvalue` output element (as specified in the service WSDL) specified in the Target of Certification's Assets.

8.2.2 Configuration

8.2.2.1 Prototype

The main enhancement in the prototype used for the actual testing is the introduction of a caching mechanism in the Discovery Manager. This feature is very important for the performances of a service discovery system.

The caching mechanism is used to maintain in memory the set of WSDL and certificate objects representing the services that are in the registry. Parsing the XML artefacts to generate this kind of object is a very time-consuming activity, so caching them through an offline initialisation allows achieving better online performances. The caching mechanism relies on the fact that whenever there is a change in the registry (e.g. a service is no longer supported, a security certificate is revoked, and so on), either (i) the registry sends notifications of the change in order to maintain the cache up to date, or (ii) the cache is synchronized periodically with the contents of the registries through a polling process. The prototype has been updated also to annotate the execution time of each component and return all of them to the client.

8.2.2.2 Test Execution

The performance for each query was evaluated incrementally for registries containing 150, 300, 600 and 1200 services in order to analyse the scalability of the solution. The execution time for each query using each registry capacity was calculated as average across 30 executions, to avoid distorted data.

A simple client has been written to execute sequentially all the queries against the prototype and save the results in CSV files (i.e., comma-separated values, a file format to represent tables with plain text files). The prototype and the client have been executed four times to support all the service registry sizes, by manually changing the service registry configuration file.

The tests have been executed on a load-free iMac system with an Intel Core i3 CPU (3.06 GHz) and 4 GB RAM (DDR3, 1333 MHz) running Mac OS X 10.6.8.

8.2.3 Threats to validity

Different factors may impact and bias the results of performance tests. In our study we tried to minimize such threats, however some factors may still have affected the results.

The QWS dataset has been chosen to populate the registry with WSDLs from real web services available on the web, in order to allow generalization of our results to the industry. This solution relies on the range and quality of the dataset; please refer to [2][3] for a discussion on the characteristics of the dataset.

The certificates in the registry have been produced ad-hoc for the experiments, as such artefacts are a rather new concept and they are not currently available in any repository. The distribution of security properties and other parameters over the certificate population has been uniformly randomized, however we expect that this will not be the case for real services. In particular one can expect certain properties to be more represented than others, based on industry interest and diffusion of mechanisms to achieve the required security. The impact of this difference in the distribution is only marginal, as the performances to match a certificate do not change with the change of security property. The number of services that provide a property, however, may change the number of workflows that are generated during a composition, and by so, it may influence the execution time.

In order to avoid the performance data to be biased by the effect of concurrency, the tests have been performed on a load-free system. Furthermore, in order to remove also the effects of background processes and of the garbage collector that may be asynchronously executed, we performed each test 30 different times.

Finally, to avoid introducing human errors in the data collection phase, all the test results have been collected and analysed automatically, through the usage of CSV files and Excel formulas.

8.3 Evaluation Results

This section presents an analysis of the test results through a set of tables summarising the results. A more complete set of results can be found in Appendix A.

		Avail.	Integr.	PSP	
Receive and parse query	Mean	233.40	167.80	189.50	Single Discovery
	SD	70.02	45.05	11.87	
Retrieval of service descr.	Mean	7.43	7.50	7.23	
	SD	1.02	0.92	0.80	
Matching	Mean	466.93	467.27	464.73	
	SD	28.26	34.16	9.25	
Abstract WF Matching	Mean	4.93	7.17	4.97	Composition
	SD	0.73	10.18	0.71	
Inference rules	Mean	46.40	38.17	45.63	
	SD	5.79	8.29	5.27	
Composition Algorithm	Mean	1644.47	532.17	1159.97	
	SD	711.79	78.53	123.75	
Sub-queries time	Mean	21187.10	11439.73	19268.63	
	SD	195.03	256.74	159.70	
BPEL Generation	Mean	5353.33	2505.50	7007.43	
	SD	222.94	385.05	195.70	
# Generated Compositions		79 (120)	40	104	
# Generated Sub-queries		51	25	46	

Table 8.1: Execution times by operations, with 1200 services in the registry

Table 8.1 shows the execution times for each operation that is part of the composition process when using the registry containing 1200 services, with respect to the three queries. The top section shows the operations of the single service discovery that are performed in the first instance when the query is submitted, i.e., receive and parse query, retrieval of services from the cache, structural and security matching. Since no single service can be found to match the query, the Composition Manager is called, and performs the operations in the middle of the table, i.e., abstract workflow (WF) matching, inference rules, discover services for the WF activities (through the generation of sub-queries), other operations of the composition algorithm and, finally, generation of a BPEL for each composition.

In particular, the first two composition operations are part of the GETSECUREWORKFLOWS algorithm described in Section 6.4.4, and the third and fourth composition operations are part of the WORKFLOWINSTANTIATION algorithm described in Section 6.4.5. The fifth operation, i.e., the BPEL generation for each composition, is not part of the algorithms described in previous sections, however it is needed in order to be able to automatically use the produced composition.

The bottom section of the table shows the number of generated activity sub-queries and the number of the compositions returned by the algorithm.

The first element to note, as highlighted in Figure 8.1, is how the different queries have very similar execution times not only for the single service discovery part, but also for the abstract workflow matching and for the inference rules. In particular the rules fired for each query are quite different, but this doesn't seem to be reflected in a difference of timings.

Figure 8.1: Comparison of the single service discovery, inference rules, and abstract WF matching execution times over the different queries

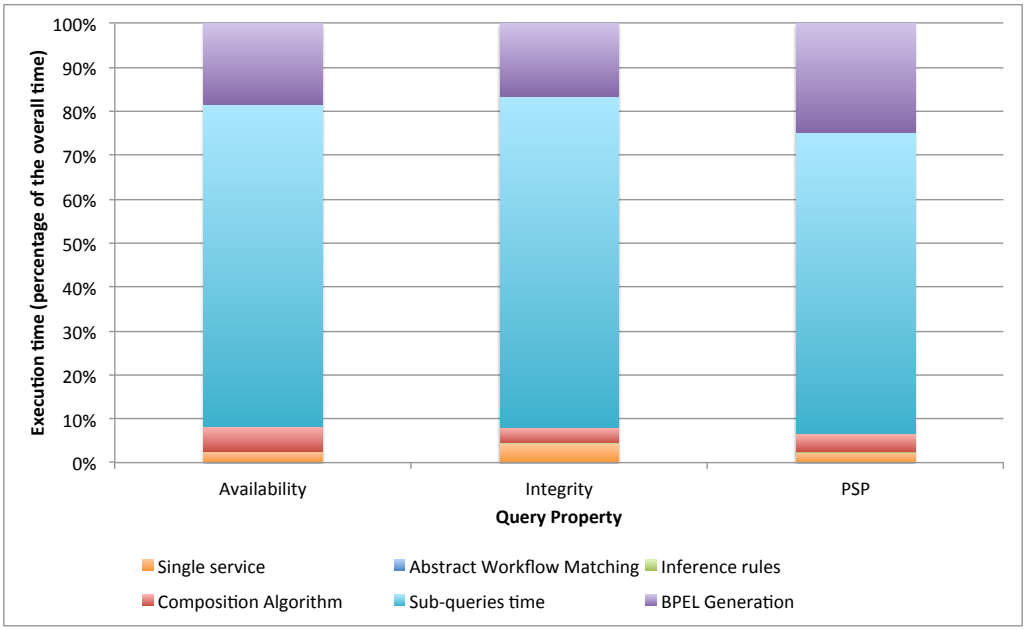


Figure 8.2: Proportion of the execution time spent for each composition operation over the different queries

The main differences that were observed were related to the execution time of the composition algorithm, the execution of sub-queries and the BPEL generation, that also constitute the most time expensive operations in the composition times as shown by Figure 8.2. These differences depend on the number of sub-queries generated by the process as shown in Figure 8.3.

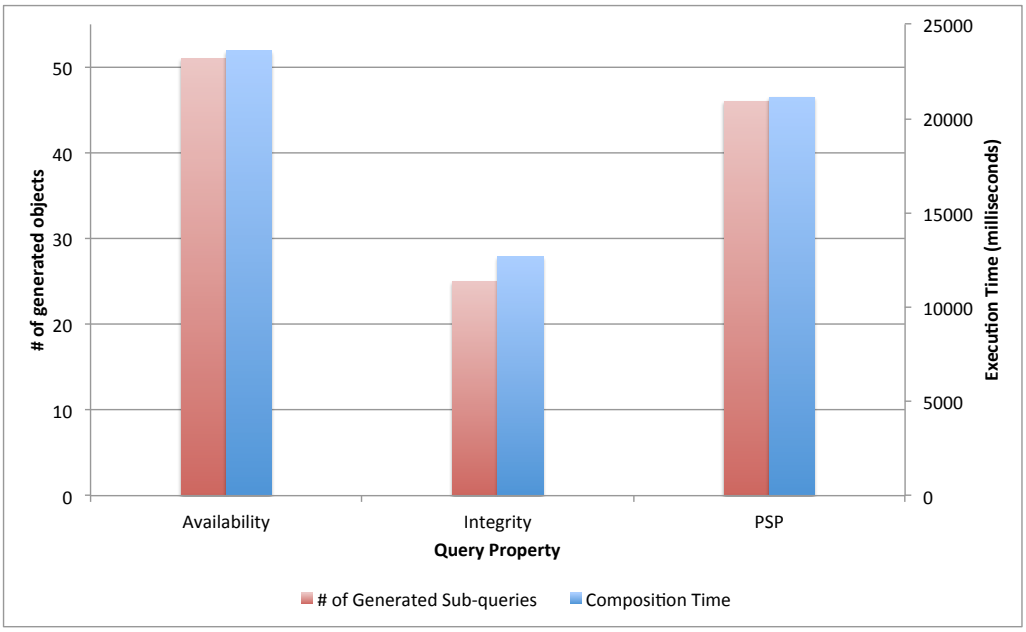


Figure 8.3: Correlation between the number of generated sub-queries and the composition time over the different queries

More specifically, the current prototype includes only a single rule for integrity on the sequential orchestration; so abstract workflows including other orchestrations are not taken into consideration when an integrity requirement must be satisfied. This means that the abstract workflows that can guarantee the security requirements are fewer for the query about integrity than for the other two properties; this is reflected by the number of generated sub-queries and so, as explained above, by the composition time, as less composition are generated.

The availability rules, instead, make use of verification post-instantiation (i.e., the verification rules described in Section 5.4.4). Table 8.1 reports both the number of verified generated workflows (79) and the number of all the generated workflows in the parenthesis (120). The former conditions only the time for the BPEL generation while the latter influences the composition algorithm and the sub-queries times.

		150	300	600	1200	
Receive and parse query	Mean	131.43	138.10	150.40	196.90	Single Discovery
	SD	68.14	28.25	29.41	55.70	
Retrieval of service descr.	Mean	1.28	2.32	4.38	7.39	
	SD	0.75	0.61	0.89	0.93	
Matching	Mean	63.34	144.78	239.33	466.31	
	SD	7.07	6.98	23.15	26.17	
Abstract WF Matching	Mean	4.98	4.98	4.70	5.69	Composition
	SD	0.98	1.07	0.78	6.00	
Inference rules	Mean	44.40	45.40	44.01	43.40	
	SD	9.59	22.99	21.69	7.56	
Composition Algorithm	Mean	184.18	476.71	696.21	1112.20	
	SD	129.47	222.39	307.25	619.18	
Sub-queries time	Mean	815.14	3105.22	7114.00	17298.49	
	SD	256.30	1021.81	2037.06	4221.27	
BPEL Generation	Mean	700.08	2036.29	3024.07	4955.42	
	SD	295.04	865.28	1166.30	1880.38	
# Generated Compositions		10	32	55	88	
# Generated Sub-queries		10	19	30	40	

Table 8.2: Summary of the results for each registry size, in milliseconds

Table 8.2 shows the results for each registry size, regardless of the query, to analyse the scalability of the approach. It is quite clear from the table that the global composition time becomes more and more time consuming with the increase of services in the registry.

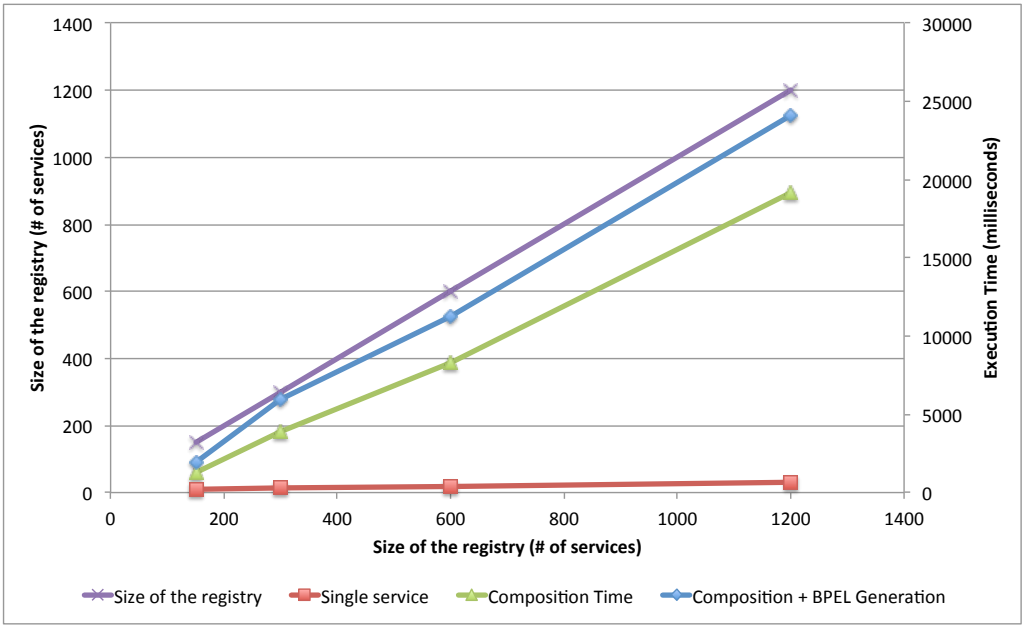


Figure 8.4: Comparison of the single service and service composition discovery times over the different sizes of the registry

Figure 8.4 shows a comparison between the single service discovery and the composition discovery times: both the single service discovery and the composition discovery time increases proportionally with the increase of services in the registry. More specifically, the abstract WF matching and the firing of the inference rules are very fast operations whose duration does not increase depending on the number of services in the registry, as shown in Figure 8.5. These operations, in fact, depend only on the size of the abstract WF repository and the number of rules. Both these numbers, however, are not expected to become much bigger than the ones used for tests.

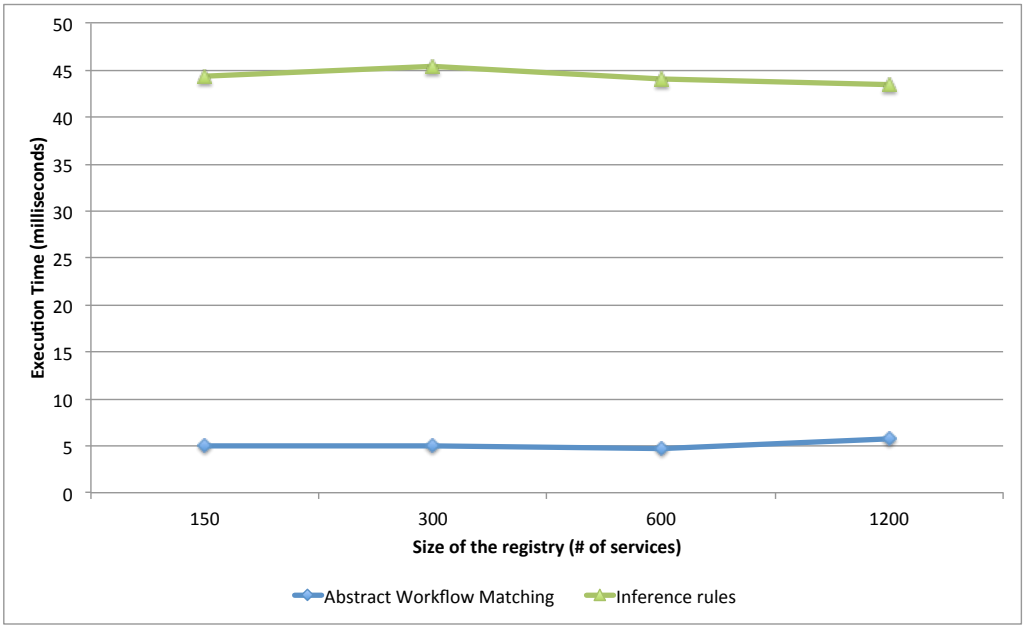


Figure 8.5: Comparison of the abstract WF matching and the inference rule times over the different sizes of the registry

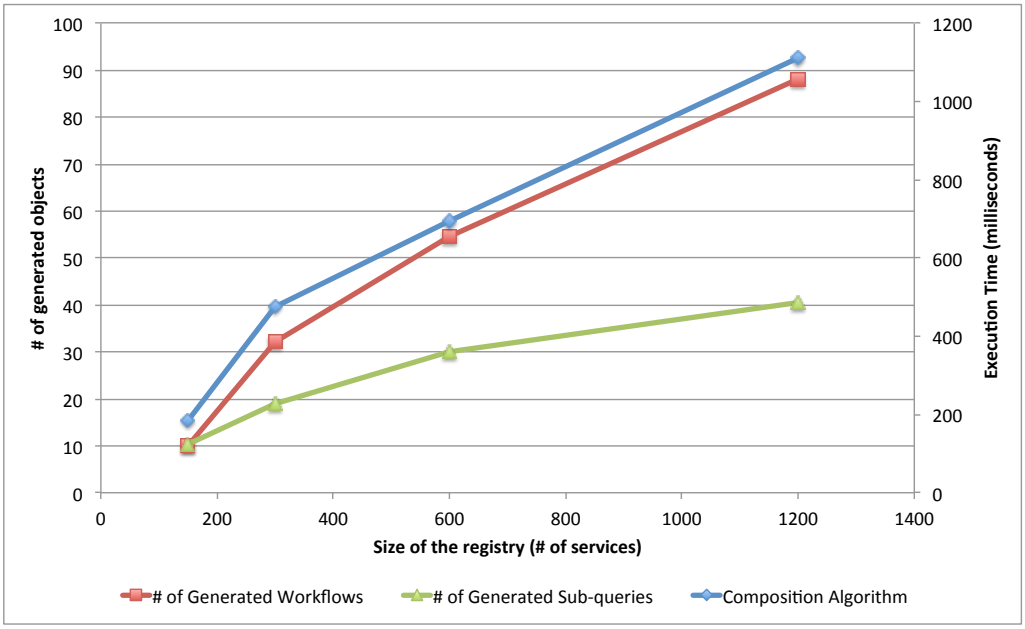


Figure 8.6: Correlation between the composition algorithm time and the number of generated workflows and sub-queries over the registry sizes

The discovery of services for the WF activities, instead, is the operation that results in the biggest increase of the execution time, as previously described for Figure 8.2. This can be explained by two factors, the first being that service discovery naturally becomes more expensive the more services are in the registry, the second being that for each service that matches an activity in a WF, the algorithm generates a new WF with the activity instantiated by the service. This means the more matching services; the more WFs are generated and need to be completely instantiated. So an increase in the number of matched services corresponds to an increase in the number of sub-queries for the different WFs, and therefore to an increase of the time required to execute the composition process. These observations are also summarised in Figure 8.6, where the composition algorithm time is plotted against with the number of generated sub-queries and compositions.

A possible optimisation that might produce drastic improvements on the composition time is to avoid separated executions of similar sub-queries, as these are the most computationally expensive operations. In particular, the services that match the structural conditions of an activity in a WF might be pre-computed, stored and maintained in a cache, so that part of the discovery process might be skipped at runtime. Furthermore other similarities might be found also in the dynamically generated security conditions of sub-queries for the same activity but in different WFs: the algorithm can then be changed to process the same sub-query just once per process and keep a temporary buffer of the results.

The last operation computed by the Composition Manager is the conversion of the generated compositions into BPEL files. This operation is depends on the number of generated compositions. Furthermore, the actual implementation uses the BPEL data model of Eclipse BPEL Designer [25]: this simplifies the generation of BPEL files that are structurally correct, but

introduces an overhead. An alternative to this is to treat the BPEL files as strings and just edit the strings using some predefined placeholders.

Overall the composition process in the presented prototype is very time consuming, as expected. However, it is not an impossible task, with times that oscillate between 1 and 20 seconds. In particular, by using the proactive facilities provided by the discovery engine, the composition discovery can be obtained in a timely fashion for applications requiring it at runtime. As explained for the single service discovery, by subscribing a proactive query, the discovery engine maintains a buffer of discovered compositions in background, to be able to offer immediate responses when the need for replacements arises.

8.4 Summary

The approach has been evaluated through testing the performances of the Security Aware Runtime Discovery Tool, and checking the overhead introduced by the security aware service composition approach.

The performances of the single service discovery are generally in the order of seconds, allowing timely responses to service discovery queries, even at runtime.

Service composition increases the discovery time, based on the number of services in the registry. This might be an issue for SBSs that require timely responses; however the proactive approach offered by RSDT addresses this exact problem. By subscribing a query to the discovery engine, it is possible to maintain an up-to-date buffer of results in the background and, when the need arises, the SBS can obtain an immediate response.

Even in this scenario, it is important to avoid the waste of resources. To improve the Composition Manager performance the composition algorithm might be changed to avoid the repetition of similar queries for the same activity in similar workflows.

Chapter 9

Conclusions

9.1 Overview

In this thesis we have described a framework that allows inferring security requirements expressed for a security composition to requirements for the single activities of the composition and checking security requirements over security service descriptors. The framework introduces the concept of secure composition patterns, modelling proven causal relations of security requirements within an orchestration pattern. Furthermore, prototypes using the composition process have been implemented and tested extensively.

9.2 Contributions

The presented research provides the means to infer and validate security requirements over service compositions. This approach allows to: (i) generate secure service compositions at design and runtime, and (ii) validate the security requirements over a service workflow. To support the above we have developed:

- *Definition of the concept and of an initial set of secure composition patterns and production rules*

We introduced the concept of secure composition patterns, i.e., models describing abstract dependencies between the service composition security requirements and the component service security requirements. The dependencies must be formally proven in order to ensure the same level of security of the original requirements.

The patterns can be applied in different steps of a composition lifetime, to discover services guaranteeing the security or to validate the security of an existing composition.

The secure composition patterns are a new concept, different from the Security Patterns present in literature (e.g., [4][23]) that usually represent the best practices and the mechanisms that can be used in order to comply to a security requirement.

We gave the definition and the rule-based encoding of an initial set of secure composition patterns, comprising patterns for integrity, confidentiality and availability. This set, whether not complete, gives an idea of the different ways in which the approach can be used and of how the production rules can be encoded.

- *Development of security aware service composition algorithms*

  A set of algorithms is given that allows: (i) the inference of security requirements from the service composition layer to the single composing services, (ii) the verification of security requirements on the service composition layer from the security descriptors at the service layer, (iii) the validation of security requirements over instantiated service orchestrations (i.e., workflows) and (iv) the generation of service compositions that respect security requirements. All these algorithms represent a novelty as they are

based and make usage of the new concept of secure composition patterns.

The existing works in literature, as examined in Section 2.4.4.1, are limited to the verification of security properties (usually specified at design time) through formal methods, requiring conversions of the services, compositions and properties into models. Our approach does not require any conversion and allows SBS designer to check the security of a composition without the need to know formal models or specialized languages. Furthermore, the works in the literature quite often require knowing the service internals or the exact mechanisms that are in place to guarantee a property. The proposed approach is more general, offering a framework for any security property that does not require to know the specific implementation (or a model of it) of the services involved, but requires information just about the workflow of a service composition.

- *Development of a Security Aware Runtime Discovery Tool*

  The discovery tool allows finding services that provide given structural and security requirements. The tool allows the creation of service compositions during the discovery of a service, and guarantees that the service compositions have the requested level of security, by using the algorithms listed in the previous point.

  The works in the literature, as discussed in Sections 2.4.3 and 2.4.4, are usually specialised to the discovery of specific security properties and offer specification and matching of properties only against single services, instead of entire service compositions. Our approach is generic w.r.t. the security properties that can be used it

with, and allows the expression and inference of security requirements over entire service compositions, so our work is an improvement in both these directions.

Furthermore our approach is based on a tool that allows service discovery either at design or runtime, thanks to the proactive capabilities of the discovery tool described in Section 4.4.

- *Development of a Security Aware BPEL Design Tool*

  The BPEL Design tool allows the description and the validation of security requirements during the design phase of a SBS. Service adaptation is also offered in order to replace services with alternative services or service compositions that comply with the functionality and the security requested.

  Existing approaches allow only the expression of security requirements on single activities and to bound services that respect the requirements, as we point out in Section 2.4.5. The tool presented in this thesis, instead, allows also: (i) the expression of security requirements over workflow fragments, (ii) the inference and validation of security requirements over the activities part of the workflow, and (iii) to discover and automatically adapt alternative service or service compositions that satisfy the requirements.

- *Evaluation of the approach*

  The feasibility and scalability of the approach have been tested, giving results in the order of seconds. While such result is already encouraging, the proactive approach offered by the discovery tool offers an answer to the SBSs where the timely availability of discovery results is critical. Furthermore, some improvement directions were given.

9.3 Approach Implications

The presented approach is compatible with technologies and languages already available in the market, however it implies additional efforts on the security aspects from Service Providers and SBS Designers.

Our approach requires some security descriptors of each service to be available in service registries. While some languages are available to describe security aspects (see Section 2.4.1), these are not used to describe services in publicly available service registries, to the best of our knowledge. While the exact mechanisms that implement a security property might need to be confidential, the security properties provided by a service can be a very important aspect for service selection (and so, for service provisioning).

Our approach goes in this direction and requires a level of transparency about the security of services, encouraging trust in service-based solutions. It is important to notice that this assumption we made requires Service Providers to handle additional tasks like creating the security descriptors and submit them to (compatible) service registry, so not all Service Providers might consider this in the immediate future. Some Service Providers, however, may consider facing the additional costs to handle this task, since security may be advertised by providers as an additional feature that outclass competitors and since security has been one of the critical concerns in the SOC field [55]; kick-starting the market in this direction.

A step that can improve the level of transparency and trust, and that can be reflected also by our approach, can be offered by security certificates signed by third parties. As explained in Section 2.4.1 certification processes and Certification Authorities are already used by some software companies, probably the most important and known being the Common Criteria

certification, however these solutions are not taking advantage of the SOC paradigm. A new certification approach for services, releasing digital certificates that could be used as security descriptors by our approach, would however require Certification Authorities and Laboratories to adopt new standards and change their processes (for more details, please refer to the ASSERT4SOA [5] and the CUMULUS [21] projects).

Finally our approach puts the service users and SBS designers in control of the security level they can require, so a minimal training on security may be needed in order to be able to use the features presented in this work.

9.4 Future Work

This work presented a new approach that allows for Security Aware Service Composition, however the ideas presented can be used as a basis for different tracks for future works. In this section we describe some directions for research that originate from this work:

- *Development of additional secure composition patterns.*

  The set of secure composition pattern presented in this work shows the feasibility of the idea in different cases, however it may lack of completeness and generality. The topic of proving secure composition patterns is a very big area for further research that can address with different formalisms. Some fields already have works researching in this direction, e.g. the proofs for composability patterns in the Information Theory field [59][63], however further research is needed to have a more complete set of patterns.

- *Semi-Automated Proofs for secure composition patterns.*

  One of the possible drawbacks for the presented approach is the fact that the secure composition patterns need to be formally proven. The definition of additional secure composition patterns, then, would require quite some time and efforts before being proven, limiting the results of our approach.

  An interesting line of research is to make usage of existing theorem proving or automated reasoning approaches (e.g., Coq, Isabelle) in order to help obtaining proofs for the secure composition patterns.

- *Variations of the composition algorithms.*

  The composition algorithms presented in Section 6.4 can be seen as a basis for further works, some changes that we envision are: (i) recursive instantiation of abstract workflows with other abstract workflows, in order to enable a richer offer of workflows during functional matching; (ii) usage of algorithms that define service workflows for the functional part of the composition, instead of relying on abstract workflows (as the ones described in Section 2.3.1); (iii) adding some optimizations to prune and cache the sub-queries used during the composition process, as sub-queries are one of the most time consuming aspects of the instantiation algorithm, but also they are often quite similar to each other; and (iv) to allow comparison and sorting of the generated service compositions, by defining appropriate metrics.

Bibliography

[1] Aggarwal, R., Verma, K., Miller, J., and Milnor, W. (2004, September). Constraint driven web service composition in METEOR-S. In *IEEE International Conference on Services Computing, 2004 (SCC 2004). Proceedings*. pp. 23-30. IEEE. DOI: 10.1109/SCC.2004.1357986

[2] Al-Masri, E., and Mahmoud, Q. H. (2007, August). Qos-based discovery and ranking of web services. In *Proceedings of 16th International Conference on Computer Communications and Networks, 2007 (ICCCN'07)*, pp. 529-534. IEEE. DOI: 10.1109/ICCCN.2007.4317873

[3] Al-Masri, E., and Mahmoud, Q. H. (2007, May). Discovering the best web service. In *Proceedings of the 16th international conference on World Wide Web (WWW'07)*, pp. 1257-1258. ACM. DOI: 10.1145/1242572.1242795

[4] Aniketos Consortium (2012). Run-time Secure Composition and Adaptation Realisation *Techniques. Aniketos project Deliverable 3.3*. Available at:

http://www.aniketos.eu/

[5] ASSERT4SOA Consortium. ASSERT4SOA project website. Available at:

http://www.assert4soa.eu/

[6] Bartoletti, M., Degano, P., and Ferrari, G. L. (2005, June). Enforcing secure service composition. In *18th IEEE Computer Security Foundations Workshop, 2005 (CSFW'05)*. pp. 211-223. IEEE. DOI: 10.1109/CSFW.2005.17

[7] Bartoletti, M., Degano, P., and Ferrari, G. L. (2006, July). Types and effects for secure service orchestration. In *19th IEEE Computer Security Foundations Workshop, 2006 (CSFW'06)*. pp. 57-69. IEEE. DOI: 10.1109/CSFW.2006.31

[8] Baryannis, G., and Plexousakis, D. (2010). Automated Web Service Composition: State of the Art and Research Challenges. *Technical Report ICS-FORTH/TR-409*. Foundation for Research and Technology, Hellas. Available at:

https://www.ics.forth.gr/tech-reports/2010/2010.TR409_Automated_Web_Service_Composition.pdf

[9] Beeri, C., Eyal, A., Kamenkovich, S., and Milo, T. (2006, September). Querying business processes. In *Proceedings of the 32nd international conference on Very large data bases (VLDB'06)*, pp. 343-354. VLDB Endowment. Available at:

http://www.vldb.org/conf/2006/p343-beeri.pdf

[10] Berry, M. (2008). IBM Survey: SOA a Top Business Priority in 2008. *IBM Media Relations Websphere & SOA*. Available at:

https://www-03.ibm.com/press/us/en/pressrelease/24436.wss

[11] Bormann, F., Flake, S., Tacken, J., and Zoth, C. (2005, June). Towards context-aware service discovery: A case study for a new

advice of charge service. In *14th IST Mobile and Wireless Communications Summit*. Available at:

http://www.eurasip.org/Proceedings/Ext/IST05/papers/502.pdf

[12] CA Wily (2008). CA Wily TechWeb Study Results. Available at:

http://www.ca.com/~/media/Files/SupportingPieces/cmp-global-survey_196383.pdf

[13] Carminati, B., Ferrari, E., and Hung, P. C. (2006, September). Security conscious web service composition. In *IEEE International Conference on Web Services, 2006 (ICWS'06)*. pp. 489-496. IEEE. DOI: 10.1109/ICWS.2006.115

[14] Casati, F., Ilnicki, S., Jin, L., Krishnamoorthy, V., and Shan, M. C. (2000). Adaptive and Dynamic Service Composition in eFlow. In *Advanced Information Systems Engineering: 12th International Conference on Advanced Information Systems Engineering (CAiSE'00), Stockholm, Sweden, June 5-9, 2000. Proceedings*. Vol. 12, p. 13. Springer. DOI: 10.1007/3-540-45140-4_3

[15] Chan, C. Y., Garofalakis, M., and Rastogi, R. (2003). Re-tree: an efficient index structure for regular expressions. *The VLDB Journal*, 12(2), 102-119. DOI: 10.1007/s00778-003-0094-0

[16] Charfi, A., and Mezini, M. (2005, July). Using aspects for security engineering of web service compositions. In *IEEE International Conference on Web Services, 2005 (ICWS'05). Proceedings*. pp. 59-66. IEEE. DOI: 10.1109/ICWS.2005.126

[17] Clark, C. (2008). Why traditional security doesn't work for SOA. *InfoWorld*. Available at:

http://www.infoworld.com/article/2653660/application-security/why-traditional-security-doesn-t-work-for-soa.html

[18] Common Criteria Project Sponsoring Organisations (2012). Common Criteria for Information Technology Security Evaluation. Version 3.1, Revision 4. Available at:

https://www.commoncriteriaportal.org/cc/

[19] Common Criteria Project Sponsoring Organizations. Certified Products. Available at:

https://www.commoncriteriaportal.org/products/

[20] Cuddy, S., Katchabaw, M., and Lutfiyya, H. (2005, August). Context-aware service selection based on dynamic and static service attributes. In *IEEE International Conference on Wireless And Mobile Computing, Networking And Communications, 2005 (WiMob'05)*, Vol. 4, pp. 13-20. IEEE. DOI: 10.1109/WIMOB.2005.1512944

[21] CUMULUS Consortium. CUMULUS project website. Available at: http://www.cumulus-project.eu/

[22] Deubler, M., Grünbauer, J., Jürjens, J., and Wimmel, G. (2004, November). Sound development of secure service-based systems. In *Proceedings of the 2nd international conference on Service oriented computing (ICSOC'04)*, pp. 115-124. ACM. DOI: 10.1145/1035167.1035185

[23] Dong, J., Peng, T., and Zhao, Y. (2010). Automated verification of security pattern compositions. *Information and Software Technology*, 52(3), 274-295. DOI: 10.1016/j.infsof.2009.10.001

[24] Dornan, A. (2003). XML: The End of Security Through Obscurity?. *Network Magazine*, 18(4), 36-41.

[25] Eclipse BPEL Project. Eclipse BPEL Designer. Available at:

http://www.eclipse.org/bpel/

[26] Eclipse Foundation. Discovering and importing a Web service. In *Eclipse Documentation*. Available at:

http://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.jst.ws.consumption.ui.doc.user%2Ftasks%2Ftdiscov.html&cp=63_3_1_4

[27] Fischer, K. P., Bleimann, U., Fuhrmann, W., and Furnell, S. M. (2007, April). Security policy enforcement in BPEL-defined collaborative business processes. In *IEEE 23rd International Conference on Data Engineering Workshop, 2007 (ICDEW'07)*, pp. 685-694. IEEE. DOI: 10.1109/ICDEW.2007.4401056

[28] Forgy, C. L. (1982). Rete: A fast algorithm for the many pattern / many object pattern match problem. *Artificial intelligence*, 19(1), 17-37. DOI: 10.1016/0004-3702(82)90020-0

[29] Frankova, G., Massacci, F., and Seguran, M. (2007, July). From Early Requirements Analysis towards Secure Workflows. In *Trust Management: Proceedings of IFIP Joint ITrust and PST Conferences on Privacy, Trust Management and Security (IFIPTM'07), July 30-August 2, 2007, New Brunswick, Canada*. Vol. 238, p. 407. Springer. DOI: 10.1007/978-0-387-73655-6_28

[30] Fujii, K., and Suda, T. (2004, November). Dynamic service composition using semantic information. In *Proceedings of the 2nd*

international conference on Service oriented computing (ICSOC'04), pp. 39-48. ACM. DOI: 10.1145/1035167.1035174

[31] Fujii, K., and Suda, T. (2006). Semantics-based dynamic web service composition. *International Journal of Cooperative Information Systems*, 15(03), 293-324. DOI: 10.1142/S0218843006001372

[32] Grigori, D., Corrales, J. C., and Bouzeghoub, M. (2006, September). Behavioral matchmaking for service retrieval. In *Proceedings of the IEEE International Conference on Web Services (ICWS'06)*, pp. 145-152. IEEE Computer Society. DOI: 10.1109/ICWS.2006.37

[33] Grundy, J., and Ding, G. (2002). Automatic validation of deployed J2EE components using aspects. In *17th IEEE International Conference on Automated Software Engineering, 2002 (ASE'02). Proceedings*. pp. 47-56. IEEE. DOI: 10.1109/ASE.2002.1114993

[34] GS1 US. RosettaNet website. Available at:

https://members.gs1us.org/RosettaNet

[35] Gu, X., and Nahrstedt, K. (2006). Distributed multimedia service composition with statistical QoS assurances. *IEEE Transactions on Multimedia*, 8(1), 141-151. DOI: 10.1109/TMM.2005.861284

[36] Gu, X., Nahrstedt, K., and Yu, B. (2004, June). SpiderNet: An integrated peer-to-peer service composition framework. In *13th IEEE International Symposium on High performance Distributed Computing, 2004 (HPDC'04). Proceedings*. pp. 110-119. IEEE. DOI: 10.1109/HPDC.2004.1323507

[37] Gürgens, S., Ochsenschläger, P., and Rudolph, C. (2005). On a formal framework for security properties. *Computer Standards & Interfaces*, 27(5), 457-466. DOI: 10.1016/j.csi.2005.01.004

[38] Gutiérrez, C., Fernández-Medina, E., and Piattini, M. (2006). Towards a process for web services security. *Journal of Research and Practice in Information Technology*, 38(1), 57-68. ISSN: 1443-458X

[39] Hafner, M., Breu, R., Agreiter, B., and Nowak, A. (2006). SECTET: an extensible framework for the realization of secure inter-organizational workflows. *Internet Research*, 16(5), 491-506. DOI: 10.1108/10662240610710978

[40] Health Level Seven International. HL7 website. Available at:

http://www.hl7.org/

[41] Heffner, R. (2010). Adoption of SOA: Still strong, even in hard times. *Forrester Research*. Available at:

http://www.forrester.com/Adoption+Of+SOA+Still+Strong+Even+In+Hard+Times/fulltext/-/E-RES56874

[42] Heffner, R., Leganza, G., and Ranade, K. (2008). Soa adoption: Many firms got started in 2007. *Forrester Research*. Available at:

http://www.forrester.com/SOA+Adoption+Many+Firms+Got+Started+In+2007/fulltext/-/E-RES43832

[43] IBM. IBM BPM Industry Packs. Available at:

http://www-03.ibm.com/software/products/en/business-process-manager-industry-packs

[44] IETF Network Working Group (2007). Internet Security Glossary, Version 2. *Request For Comments 4949 (RFC 4949, Informational)*. Available at:

http://www.ietf.org/rfc/rfc4949.txt

[45] ISO/IEC - Information Technology - Security Techniques (2009). Evaluation criteria for IT security. *ISO/IEC International Standard 15408-1, 3rd Edition*. Available at:

https://www.iso.org/obp/ui/#iso:std:iso-iec:15408:-1:en

[46] Jaeger, M. C., Rojec-Goldmann, G., and Muhl, G. (2004, September). Qos aggregation for web service composition using workflow patterns. In *Eighth IEEE International Enterprise distributed object computing conference, 2004 (EDOC'04). Proceedings*. pp. 149-159. IEEE. DOI: 10.1109/EDOC.2004.1342512

[47] JBoss Drools Team. Drools. Available at:

http://www.drools.org/

[48] Kagal, L., Finin, T., Paolucci, M., Srinivasan, N., Sycara, K., and Denker, G. (2004). Authorization and privacy for semantic web services. *IEEE Intelligent Systems*, 19(4), 50-56. DOI: 10.1109/MIS.2004.23

[49] Keller, U., Lara, R., Lausen, H., Polleres, A., and Fensel, D. (2005). Automatic location of services. In *The Semantic Web: Research and Applications*, pp. 1-16. Springer Berlin Heidelberg. DOI: 10.1007/11431053_1

[50] Klusch, M., Fries, B., and Sycara, K. (2006, May). Automated semantic web service discovery with OWLS-MX. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems (AAMAS'06)*, pp. 915-922. ACM. DOI: 10.1145/1160633.1160796

[51] Korhonen, J., Pajunen, L., and Puustjarvi, J. (2003, October). Automatic composition of web service workflows using a semantic agent. In *IEEE/WIC International Conference on Web Intelligence, 2003 (WI'03). Proceedings*. pp. 566-569. IEEE. DOI: 10.1109/WI.2003.1241269

[52] LeBlanc, D., and Howard, M. (2002). *Writing secure code (2nd Edition)*. Pearson Education. ISBN: 9780735617223

[53] Lécué, F., Silva, E., and Pires, L. F. (2008). A framework for dynamic web services composition. In *Emerging Web Services Technology, Volume II (ECOWS'08 workshop)*, pp. 59-75. Birkhäuser Basel. DOI: 10.1007/978-3-7643-8864-5_5

[54] Lelarge, M., Liu, Z., and Riabov, A. V. (2006, September). Automatic composition of secure workflows. In *Proceedings of the 3rd Int. conf. on Autonomic and Trusted Computing (ATC'06)*, pp. 322-331. Springer-Verlag. DOI: 10.1007/11839569_31

[55] Lewis, G. A., Smith, D. B., and Kontogiannis, K. (2010). A research agenda for service-oriented architecture (SOA): Maintenance and evolution of service-oriented systems. *Technical Report CMU/SEI-2010-TN-003*. Carnegie Mellon University. Available at:

http://www.sei.cmu.edu/reports/10tn003.pdf

[56] Li, L., and Horrocks, I. (2004). A software framework for matchmaking based on semantic web technology. *International Journal of Electronic Commerce*, 8(4), 39-60. ISSN: 1086-4415

[57] LSDIS. METEOR-S v0.8 Demonstration. METEOR-S: Semantic Web Services and Processes. Available at:

http://lsdis.cs.uga.edu/projects/meteor-s/index.php?page=3

[58] Majithia, S., Walker, D. W., and Gray, W. A. (2004, April). A Framework for Automated Service Composition in Service-Oriented Architectures. In *The Semantic Web: Research and Applications: First European Semantic Web Symposium, ESWS 2004, Heraklion, Crete, Greece, May 10-12, 2004, Proceedings*, Vol. 1, p. 269. Springer. DOI: 10.1007/978-3-540-25956-5_19

[59] Mantel, H. (2002). On the composition of secure systems. In *IEEE Symposium on Security and Privacy, 2002 (SP'02). Proceedings*. pp. 88-101. IEEE. DOI: 10.1109/SECPRI.2002.1004364

[60] Mao, Z. M., Katz, R. H., and Brewer, E. A. (2001). Fault-tolerant, scalable, wide-area internet service composition. *Technical Report UCB/CSD-01-1129*, University of California, Berkeley. Available at:

http://www.eecs.berkeley.edu/Pubs/TechRpts/2001/CSD-01-1129.pdf

[61] Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., et al. (2004). OWL-S: Semantic markup for web services. *W3C member submission*. Available at:

http://www.ai.sri.com/daml/services/owl-s/1.2/overview/

[62] Mccarthy, J., and Hayes, P. J. (1969). Some Philosophical Problems from the Standpoint of the Artificial Intelligence. *Machine Intelligence*, 4, 463-502. Available at:

http://www-formal.stanford.edu/jmc/mcchay69.pdf

[63] McLean, J. (1994, May). A general theory of composition for trace sets closed under selective interleaving functions. In *IEEE Computer Society Symposium on Research in Security and Privacy, 1994. Proceedings*. pp. 79-93. IEEE. DOI: 10.1109/RISP.1994.296590

[64] Medjahed, B., Bouguettaya, A., and Elmagarmid, A. K. (2003). Composing web services on the semantic web. *The VLDB Journal*, 12(4), 333-351. DOI: 10.1007/s00778-003-0101-5

[65] Mell, P., and Grance, T. (2011). The NIST Definition of Cloud Computing. *NIST Special Publication*, 800-145. NIST. Available at:

http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[66] Menzel, M., Warschofsky, R., and Meinel, C. (2010, July). A pattern-driven generation of security policies for service-oriented architectures. In *IEEE International Conference on Web Services, 2010 (ICWS'10)*. pp. 243-250. IEEE. DOI: 10.1109/ICWS.2010.25

[67] Mikhaiel, R., and Stroulia, E. (2006, December). Examining usage protocols for service discovery. In *Proceedings of the 4th international conference on Service-Oriented Computing (ICSOC'06)*, pp. 496-502. Springer-Verlag. DOI: 10.1007/11948148_46

[68] Milner, R. (1989). *Communication and concurrency*. Prentice-Hall, Inc. ISBN:0-13-115007-3

[69] Moschetta, E., Antunes, R. S., and Barcellos, M. P. (2010). Flexible and secure service discovery in ubiquitous computing. *Journal of Network and Computer Applications*, 33(2), 128-140. DOI: 10.1016/j.jnca.2009.11.001

[70] Mukhopadhyay, T., Kekre, S., and Kalathur, S. (1995). Business value of information technology: a study of electronic data interchange. *MIS Quarterly*, 19(2), 137-156. DOI: 10.2307/249685

[71] O'Halloran, C. (1990). A Calculus of Information Flow. In *European Symposium on Research in Computer Security (ESORICS'90)*, pp. 147–159.

[72] OASIS Security Services Technical Committee. (2005). SAML V2.0. *OASIS Standard*. Available at:

http://saml.xml.org/saml-specifications

[73] OASIS Web Services Federation Technical Committee (2009). WS-Federation 1.2. *OASIS Standard*. Available at:

http://docs.oasis-open.org/wsfed/federation/v1.2/ws-federation.html

[74] OASIS Web Services Secure Exchange Technical Committee (2007). WS-SecureConversation 1.3. *OASIS Standard*. Available at:

http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/ws-secureconversation-1.3-os.html

[75] OASIS Web Services Secure Exchange Technical Committee (2007). WS-SecurityPolicy 1.2. *OASIS Standard*. Available at:

http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.html

[76] OASIS Web Services Secure Exchange Technical Committee (2009). WS-Trust 1.4. *OASIS Standard*. Available at:

http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/os/ws-trust-1.4-spec-os.html

[77] OASIS Web Services Security Technical Committee (2004). Web services security: SOAP message security 1.1 (WS-Security 2004). *Oasis Standard Specification*. Available at:

https://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

[78] Papazoglou, M. (2008). *Web services: principles and technology*. Pearson Education. ISBN: 9780321155559

[79] Pawar, P. and Tokmakoff, A. (2006). Ontology-Based Context-Aware Service Discovery for Pervasive Environments. In *1st IEEE International Workshop on Services Integration in Pervasive Environments (SIPE workshop, colocated with ICPS'06), 29 June 2006, Lyon, France*. Available at:

http://eprints.eemcs.utwente.nl/8117/

[80] Pazzaglia, J. C., Lotz, V., Cerda, V. C., Damiani, E., Ardagna, C., Gürgens, S., et al. (2011). Advanced security service certificate for soa: Certified services go digital. In *Information Security Solutions Europe (ISSE'10) 2010 Securing Electronic Business Processes*, pp. 151-160. Vieweg+ Teubner. DOI: 10.1007/978-3-8348-9788-6_15

[81] Pino, L., and Spanoudakis, G. (2012, June). Constructing secure service compositions with patterns. In *IEEE Eighth World Congress on Services, 2012 (SERVICES'12)*, pp. 184-191. IEEE. DOI: 10.1109/SERVICES.2012.61

[82] Pino, L., and Spanoudakis, G. (2012, May). Finding secure compositions of software services: Towards a pattern based approach. In *5th IFIP International Conference on New Technologies, Mobility and Security, 2012 (NTMS'12)*, pp. 1-5. IEEE. DOI: 10.1109/NTMS.2012.6208741

[83] Pino, L., Mahbub, K., and Spanoudakis, G. (2014, November). Designing Secure Service Workflows in BPEL. In *Proceedings of the international conference on Service-Oriented Computing (ICSOC'14)*, pp. 551-559. Springer Berlin Heidelberg. DOI: 10.1007/978-3-662-45391-9_48

[84] Pino, L., Spanoudakis, G., Fuchs, A., and Gürgens, S. (2014, April). Discovering Secure Service Compositions. In *4th International Conference on Cloud Computing and Services Sciences (CLOSER'14), Barcelona, Spain*. DOI: 10.5220/0004855702420253

[85] Pino, L., Spanoudakis, G., Fuchs, A., and Gürgens, S. (to appear). Generating Secure Service Compositions. In *Cloud Computing and Services Science: Fourth International Conference, CLOSER 2014, Barcelona, Spain, April 3-4, 2014, Revised Selected Papers*. Springer.

[86] Ponnekanti, S. R., and Fox, A. (2002, May). Sword: A developer toolkit for web service composition. In *Proc. of the Eleventh*

International World Wide Web Conference (WWW'02), Honolulu, HI, Vol. 45. Available at:

http://www2002.org/CDROM/alternate/786/

[87] Raman, B., Agarwal, S., Chen, Y., Caesar, M., Cui, W., Johansson, P., et al. (2002). The SAHARA model for service composition across multiple providers. In *Pervasive Computing*, pp. 1-14. Springer Berlin Heidelberg. DOI: 10.1007/3-540-45866-2_1

[88] Rao, J. (2004). Semantic Web Service Composition via Logic-based Program Synthesis. *Doctoral dissertation, Norwegian University of Science and Technology*. Available at:

http://www.cs.cmu.edu/~jinghai/papers/thesis.pdf

[89] Rao, J., Kungas, P., and Matskin, M. (2004, July). Logic-based Web services composition: from service description to process model. In *IEEE International Conference on Web Services, 2004 (ICWS'04). Proceedings*. pp. 446-453. IEEE. DOI: 10.1109/ICWS.2004.1314769

[90] Russell, N., Ter Hofstede, A. H., Edmond, D., and van der Aalst, W. M. (2005). Workflow data patterns: Identification, representation and tool support. In *Conceptual Modeling–ER 2005*, pp. 353-368. Springer Berlin Heidelberg. DOI: 10.1007/11568322_23

[91] Sabelfeld, A., and Myers, A. C. (2003). Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1), 5-19. DOI: 10.1109/JSAC.2002.806121

[92] Shen, Z., and Su, J. (2005, July). Web service discovery based on behavior signatures. In *IEEE International Conference on Services Computing, 2005 (SCC'05)*, Vol. 1, pp. 279-286. IEEE. DOI: 10.1109/SCC.2005.107

[93] Sholler, D. (2008). SOA user survey: adoption trends and characteristics. *Gartner Group*. Available at:

https://www.gartner.com/doc/765720/-soa-user-survey-adoption

[94] Silva, E., Pires, L. F., and Van Sinderen, M. (2009). Supporting dynamic service composition at runtime based on end-user requirements. In *Proc. 1st International Workshop on User-generated Services (CEUR workshop, colocated with ICSOC'09)*. ISSN: 1613-0073

[95] Singh, S., Grundy, J., and Hosking, J. (2004, April). Developing .NET Web Service-based Applications with Aspect-Oriented Component Engineering. In *The Fifth Australasian Workshop on Software and System Architectures (AWSA'04)*. Vol. 7, p. 19. Available at:

https://www.cs.auckland.ac.nz/~john-g/papers/asaw2004.pdf

[96] Singh, S., Grundy, J., Hosking, J., and Sun, J. (2005, November). An architecture for developing aspect-oriented web services. In *Third IEEE European Conference on Web Services, 2005 (ECOWS'05)*. pp. 11. IEEE. DOI: 10.1109/ECOWS.2005.7

[97] Sivashanmugam, K., Miller, J. A., Sheth, A. P., and Verma, K. (2005). Framework for semantic web process

composition. *International Journal of Electronic Commerce*, 9(2), 71-106. ISSN: 1086-4415

[98] Society for Worldwide Interbank Financial Telecommunication. SWIFT website. Available at:

http://www.swift.com/

[99] Society for Worldwide Interbank Financial Telecommunication (2011). SWIFT messaging services. *Factsheet*. Available at:

http://www.swift.com/dsp/resources/documents/factsheet_messaging_services.pdf

[100] Souza, A. R., Silva, B. L., Lins, F. A., Damasceno, J. C., Rosa, N. S., Maciel, P. R., et al. (2009, November). Incorporating Security Requirements into Service Composition: From Modelling to Execution. In *Proceedings of the 7th International Joint Conference on Service-Oriented Computing (ICSOC-ServiceWave '09)*, pp. 373-388. Springer-Verlag. DOI: 10.1007/978-3-642-10383-4_27

[101] Spanoudakis, G., and Zisman, A. (2010). Designing and Adapting Service-based Systems: A Service Discovery Framework. *Service Engineering: European Research Results*, pp. 261-297. Springer-Verlag. DOI: 10.1007/978-3-7091-0415-6_10

[102] Stallings, W. (2013). *Cryptography and Network Security: Principles and Practice (6th Edition)*. Prentice Hall. ISBN: 9780133354690

[103] Stam, K.T., O'Ree, A. (2014). Apache jUDDI Client and GUI Guide. Available at:

http://juddi.apache.org/docs/3.2/juddi-client-guide/pdf/jUDDI_Guide.pdf

[104] Subashini, S., and Kavitha, V. (2011). A survey on security issues in service delivery models of cloud computing. *Journal of network and computer applications*, 34(1), 1-11. DOI: 10.1016/j.jnca.2010.07.006

[105] Tavakolan, M., Zarreh, M., and Azgomi, M. A. (2009). Web service discovery based on privacy preferences. *International Journal of Web Services Practices*, 4(1), 28-35. ISSN: 1738-6535

[106] Trabelsi, S., Gomez, L., and Roudier, Y. (2007, May). Context-aware security policy for the service discovery. In *21st International Conference on Advanced Information Networking and Applications Workshops, 2007 (AINAW'07)*. Vol. 1, pp. 477-482. IEEE. DOI: 10.1109/AINAW.2007.132

[107] van Der Aalst, W. M., Ter Hofstede, A. H., Kiepuszewski, B., and Barros, A. P. (2003). Workflow patterns. *Distributed and parallel databases*, 14(1), 5-51. DOI: 10.1023/A:1022883727209

[108] W3C Web Service Description Working Group (2007). Web services description language (WSDL) version 2.0 part 0: Primer. *W3C Recommendation*. Available at:

http://www.w3.org/TR/wsdl20-primer/

[109] W3C Web Service Policy Working Group (2007). Web Services Policy 1.5 - Framework (WS-Policy). *W3C Recommendation*. Available at:

http://www.w3.org/TR/ws-policy

[110] W3C Web Services Architecture Working Group (2004). Web services glossary. *W3C Working Group Note*. Available at:

http://www.w3.org/TR/ws-gloss/

[111] Vijaykumar, W. WebserviceX.NET website. Available at:

http://www.webservicex.net/

[112] Xignite. Xignite website. Available at:

http://www.xignite.com/Products/

[113] Ye, Y., and Fischer, G. (2001, November). Context-aware browsing of large component repositories. In *16th Annual International Conference on Automated Software Engineering, 2001 (ASE'01). Proceedings*. pp. 99-106. IEEE. DOI: 10.1109/ASE.2001.989795

[114] Zakinthinos, A., and Lee, E. S. (1997, May). A general theory of security properties. In *IEEE Symposium on Security and Privacy, 1997. Proceedings*. pp. 94-102. IEEE. DOI: 10.1109/SECPRI.1997.601322

[115] Zisman, A., Spanoudakis, G., and Dooley, J. (2008, September). A framework for dynamic service discovery. In *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE'08)*, pp. 158-167. IEEE Computer Society. DOI: 10.1109/ASE.2008.26

[116] Zisman, A., Spanoudakis, G., and Dooley, J. (2009). A Query Language for Service Discovery. In *Proceedings of 4th International Conference on Software and Data Technologies (ICSOFT'09)*, pp. 55-65. DOI: 10.5220/0002260400550065

[117] Zisman, A., Spanoudakis, G., Dooley, J., and Siveroni, I. (2013). Proactive and reactive runtime service discovery: a framework and its evaluation. *IEEE Transactions on Software Engineering*, 39(7), 954-974. DOI: 10.1109/TSE.2012.84

Appendix A: Performance Test Results

A.1 Overview

In this appendix we report all the raw results from the performance tests. In particular each section presents the test results for a given registry size and security property in a table where each row represents one of the 30 executions of the test. The legend for the columns is the following:

A. Receive and parse query

B. Retrieval of service descriptions from the registry

C. Structural Matching

D. Security Matching

E. Abstract WF Matching

F. Inference rules

G. BPEL Generation

H. Number of generated workflows

I. Number of verified workflows

J. Composition Algorithm

K. Sub-queries time

A.2 Registry size: 150 services

A.2.1 Security Property: Availability

A	B	C	D	E	F	G	H	I	J	K
173	1	64	0	6	57	930	12	8	362	940
167	2	62	0	4	61	1206	12	8	331	944
172	1	84	0	6	70	808	12	8	450	989
149	2	63	0	5	61	902	12	8	350	1034
152	1	57	1	4	62	786	12	8	311	952
138	1	64	0	5	59	784	12	8	313	945
132	1	67	0	4	54	841	12	8	280	913
134	2	88	0	6	82	927	12	8	293	965
133	1	67	0	4	57	804	12	8	308	1010
131	1	55	1	4	48	801	12	8	277	935
135	2	58	1	4	49	936	12	8	304	909
126	1	61	0	6	51	942	12	8	262	932
125	1	61	1	6	50	795	12	8	264	960
117	2	64	0	6	48	800	12	8	247	908
111	1	58	0	4	42	720	12	8	228	875
136	1	64	1	4	46	692	12	8	231	836
113	1	60	1	4	51	682	12	8	237	888
124	1	56	1	5	49	759	12	8	249	897
115	1	64	0	4	44	743	12	8	226	892
129	1	64	1	5	49	706	12	8	230	919
119	1	66	0	5	51	731	12	8	213	884
124	1	65	0	4	46	662	12	8	230	886
123	1	53	0	5	41	662	12	8	240	890
131	2	59	0	4	45	850	12	8	228	916
118	1	56	0	5	43	634	12	8	238	930
189	1	62	0	5	44	655	12	8	212	837
141	1	61	0	5	48	683	12	8	233	952
116	1	60	0	5	44	734	12	8	224	900
117	1	58	0	5	38	646	12	8	219	895
106	1	65	0	4	42	655	12	8	487	853

A.2.2 Security Property: Integrity

A	B	C	D	E	F	G	H	I	J	K
128	1	63	0	4	35	328	4	4	76	465
106	2	82	0	5	59	316	4	4	72	450
130	2	58	1	5	36	286	4	4	68	442
115	1	64	0	5	34	294	4	4	73	433
144	2	64	0	5	35	313	4	4	64	450
139	1	65	0	5	42	287	4	4	66	457
122	1	65	0	4	34	293	4	4	65	438
116	1	59	1	4	38	318	4	4	73	434
116	1	59	0	7	35	296	4	4	69	417
125	1	61	0	6	36	285	4	4	67	423
126	1	78	0	6	34	287	4	4	64	433
127	1	63	0	5	32	311	4	4	66	459
142	1	68	0	5	40	315	4	4	76	480
146	2	75	0	5	48	376	4	4	89	491
103	1	63	1	6	30	536	4	4	78	461
107	1	61	1	5	37	311	4	4	65	448
121	2	75	1	5	41	401	4	4	91	537
140	1	70	0	4	46	336	4	4	75	471
123	1	68	0	5	38	293	4	4	84	522
124	1	60	0	6	39	324	4	4	84	546
110	1	68	0	5	29	339	4	4	77	458
121	1	54	0	8	32	282	4	4	78	447
94	1	61	0	6	38	308	4	4	66	471
105	1	69	0	4	38	286	4	4	67	445
114	1	59	1	4	33	261	4	4	60	423
154	1	54	0	5	28	309	4	4	67	449
305	1	60	1	4	36	278	4	4	74	446
122	1	62	0	5	36	323	4	4	74	453
107	1	53	0	4	34	475	4	4	100	545
133	2	71	0	6	62	452	4	4	90	592

A.2.3 Security Property: PSP

A	B	C	D	E	F	G	H	I	J	K
132	1	85	0	5	53	983	14	12	243	1138
118	2	62	0	5	42	970	14	12	213	1091
106	1	61	0	4	48	972	14	12	188	1097
104	1	56	0	7	39	929	14	12	241	1065
115	2	54	1	5	36	1238	14	12	203	1060
118	1	65	0	5	49	947	14	12	214	1041
128	1	65	1	5	45	1094	14	12	220	1049
108	2	59	0	6	42	938	14	12	188	1058
130	1	59	0	6	49	950	14	12	217	1024
106	1	65	0	4	42	1310	14	12	216	1059
114	1	57	0	4	34	872	14	12	190	1030
101	1	56	0	6	46	1136	14	12	194	1020
102	1	62	0	4	34	940	14	12	188	1072
124	1	78	0	8	69	1027	14	12	225	1121
111	2	62	0	8	40	934	14	12	197	1053
110	2	58	0	5	47	1052	14	12	189	1052
102	1	60	1	5	43	923	14	12	195	1057
116	0	63	0	5	39	933	14	12	202	1072
119	1	69	0	4	49	905	14	12	213	1049
99	1	66	0	5	41	922	14	12	203	1058
725	1	63	0	4	41	938	14	12	186	1066
105	2	57	0	7	39	1262	14	12	192	1053
96	2	59	0	5	47	886	14	12	186	1020
105	2	63	0	5	44	947	14	12	182	1025
152	1	59	0	5	36	925	14	12	195	1031
109	1	77	0	4	52	914	14	12	209	1022
106	1	55	0	4	46	935	14	12	182	1122
90	1	61	0	4	44	1101	14	12	208	1069
114	7	56	0	4	40	894	14	12	196	1064
103	1	71	0	4	53	935	14	12	206	1053

A.3 Registry size: 300 services

A.3.1 Security Property: Availability

A	B	C	D	E	F	G	H	I	J	K
176	2	141	0	10	49	2550	39	28	773	3615
186	3	133	0	6	48	2520	39	28	1173	3701
165	2	139	1	5	48	2485	39	28	694	3604
156	2	154	0	5	46	2752	39	28	690	3615
159	3	144	0	6	49	2418	39	28	667	3642
149	3	146	1	5	50	2291	39	28	835	3476
158	2	134	0	6	53	2330	39	28	664	3678
155	2	152	0	7	51	2522	39	28	656	3770
141	3	146	1	4	49	2263	39	28	656	3657
138	3	147	0	5	45	2393	39	28	642	3589
134	2	147	1	4	47	2165	39	28	615	3582
141	3	145	0	4	47	2366	39	28	622	3670
160	3	134	0	5	47	2135	39	28	603	3564
134	2	143	0	4	45	2185	39	28	621	3600
142	2	145	0	4	44	2314	39	28	659	3535
153	2	148	0	5	47	2253	39	28	627	3556
140	3	145	0	5	46	2469	39	28	612	3623
141	2	152	0	5	43	2337	39	28	594	3563
150	2	143	0	5	46	2172	39	28	752	3486
152	3	133	0	4	51	2254	39	28	615	3596
148	2	146	0	6	43	2388	39	28	623	3655
149	3	154	0	5	45	2123	39	28	621	3674
129	2	140	1	4	46	2436	39	28	579	3503
132	2	143	1	4	41	2116	39	28	574	3481
148	3	147	0	5	44	2435	39	28	602	3523
155	3	140	1	5	45	2286	39	28	570	3461
168	3	141	0	5	53	2202	39	28	590	3593
132	2	143	1	5	46	2136	39	28	893	3586
136	2	147	0	4	44	2311	39	28	574	3487
146	2	139	0	5	41	2325	39	28	612	3561

A.3.2 Security Property: Integrity

A	B	C	D	E	F	G	H	I	J	K
143	2	142	0	5	35	830	12	12	185	1696
131	2	150	0	5	37	827	12	12	202	1744
122	2	133	0	4	31	917	12	12	168	1662
141	3	141	1	4	32	834	12	12	194	1684
136	3	142	0	5	253	905	12	12	198	1729
145	3	145	1	4	39	862	12	12	175	1727
133	3	164	0	5	41	864	12	12	174	1663
144	3	152	0	6	44	853	12	12	255	2215
135	3	139	1	4	32	984	12	12	183	1822
108	2	142	0	8	36	853	12	12	177	1608
126	3	145	1	5	32	1029	12	12	167	1704
133	2	147	0	4	31	919	12	12	267	1871
109	3	137	0	4	30	884	12	12	196	1701
146	2	144	0	5	56	841	12	12	351	1696
124	2	148	0	4	42	827	12	12	177	1688
116	2	142	0	4	34	864	12	12	170	1660
111	2	134	0	5	37	874	12	12	172	1654
131	1	147	0	5	35	837	12	12	187	1678
317	2	137	0	4	32	856	12	12	178	1668
119	1	140	0	5	45	1020	12	12	178	1641
111	3	142	0	5	37	784	12	12	175	1629
98	2	137	1	5	30	852	12	12	176	1618
118	2	130	0	4	31	852	12	12	173	1638
106	3	144	0	5	35	855	12	12	255	1629
104	3	150	0	4	35	814	12	12	169	1612
113	2	147	0	5	29	876	12	12	178	1617
117	3	151	0	5	36	1052	12	12	174	1698
113	2	148	0	5	34	926	12	12	184	1672
103	3	150	0	8	34	1066	12	12	181	1647
129	2	140	0	4	37	928	12	12	181	1655

A.3.3 Security Property: PSP

A	B	C	D	E	F	G	H	I	J	K
128	2	142	1	5	59	3366	45	39	614	4173
143	2	137	0	5	44	2997	45	39	905	4218
134	1	141	0	5	47	3364	45	39	586	4236
151	2	152	0	6	54	3274	45	39	673	4251
144	2	142	0	5	49	3460	45	39	605	4206
125	2	147	0	4	48	3023	45	39	579	4157
127	2	151	0	5	42	3056	45	39	632	4483
140	3	150	0	6	45	3236	45	39	589	4317
198	3	171	0	6	48	3040	45	39	647	4294
128	2	151	1	4	43	3297	45	39	621	4335
129	3	144	0	5	45	3119	45	39	730	4326
128	2	143	0	4	47	3147	45	39	612	4324
111	1	147	0	6	45	2520	45	39	514	3817
128	2	136	0	5	43	2716	45	39	520	3859
142	3	150	1	4	48	2606	45	39	521	3800
111	1	125	1	4	45	3064	45	39	505	3904
140	3	141	1	4	43	2581	45	39	505	3832
107	3	154	0	6	47	2748	45	39	537	3825
130	2	148	0	6	43	2487	45	39	499	3853
124	3	145	0	5	51	2994	45	39	510	3887
130	2	145	0	5	44	2487	45	39	522	3781
133	1	152	1	4	44	2733	45	39	518	3933
151	3	157	0	5	59	2493	45	39	518	3886
126	3	156	1	5	44	2808	45	39	544	3873
235	2	144	0	4	45	2509	45	39	500	3890
147	3	157	0	5	48	2710	45	39	522	3883
131	2	148	0	5	43	2567	45	39	520	3887
111	2	145	0	5	42	2939	45	39	513	3910
130	2	143	1	9	46	2567	45	39	536	3850
112	1	145	0	4	44	2741	45	39	499	3908

A.4 Registry size: 600 services

A.4.1 Security Property: Availability

A	B	C	D	E	F	G	H	I	J	K
145	3	203	0	9	43	3474	72	50	983	8116
174	5	235	0	5	42	3749	72	50	1223	8536
174	5	248	0	5	44	3661	72	50	1171	8735
162	4	242	1	4	45	3815	72	50	922	8323
168	5	226	1	4	45	3509	72	50	991	8298
181	5	257	0	5	53	3357	72	50	920	8319
186	5	242	0	5	48	3424	72	50	937	8604
166	5	265	1	5	52	3404	72	50	933	8507
150	4	244	0	6	45	3636	72	50	900	8404
164	4	243	0	5	52	3454	72	50	1099	8352
175	6	236	0	5	44	3322	72	50	900	8365
194	5	299	0	5	59	3054	72	50	1004	8448
158	5	242	0	4	44	3686	72	50	936	8612
150	4	219	0	4	45	3575	72	50	937	8556
160	4	246	0	5	49	3641	72	50	909	8428
169	4	252	0	4	45	3612	72	50	904	8358
156	5	249	0	5	43	3056	72	50	897	8400
155	4	245	1	4	47	3333	72	50	925	8674
156	5	242	1	5	48	3376	72	50	989	8558
163	3	248	0	5	46	3697	72	50	926	8416
163	4	255	0	5	44	3720	72	50	931	8437
178	5	246	0	5	48	3348	72	50	911	8415
195	5	298	0	6	62	3360	72	50	920	8522
155	4	239	0	5	47	3438	72	50	938	8695
160	5	241	0	5	47	3350	72	50	934	8613
170	4	243	0	5	228	3477	72	50	917	8506
171	6	254	1	5	50	3675	72	50	912	8498
168	4	251	1	5	47	3201	72	50	1054	8506
197	6	297	1	5	57	3253	72	50	936	8573
165	5	248	1	5	41	3142	72	50	1033	8708

A.4.2 Security Property: Integrity

A	B	C	D	E	F	G	H	I	J	K
140	3	253	0	5	35	1966	24	24	306	4323
145	4	228	1	5	35	1384	24	24	303	4298
125	4	245	0	4	35	1464	24	24	300	4311
127	5	213	0	4	27	1378	24	24	333	4415
128	6	251	0	4	36	1375	24	24	307	4274
121	4	207	0	4	28	1432	24	24	366	4682
138	5	210	0	4	26	1607	24	24	303	4244
131	5	211	0	4	34	1490	24	24	297	4313
122	4	206	0	4	28	1390	24	24	442	4257
129	4	206	0	4	35	1404	24	24	291	4299
143	5	214	0	5	29	1381	24	24	303	4259
121	3	211	1	4	25	1628	24	24	310	4294
152	3	236	0	4	29	1703	24	24	306	4215
136	3	211	1	5	37	1439	24	24	313	4344
117	4	207	1	4	25	1386	24	24	304	4253
127	4	208	1	4	26	1689	24	24	334	4378
152	3	235	0	4	27	1377	24	24	312	4180
126	5	209	0	4	35	1415	24	24	306	4294
338	3	233	0	6	36	1348	24	24	276	4043
130	3	238	0	4	26	1385	24	24	288	4117
113	6	217	1	5	27	1544	24	24	292	4147
109	3	202	0	4	27	1385	24	24	296	4133
105	3	208	0	4	37	1403	24	24	408	4159
138	5	229	0	4	24	1438	24	24	283	4117
101	3	207	0	4	32	1388	24	24	292	4132
109	3	203	0	3	27	1533	24	24	296	4139
114	5	226	1	4	50	1385	24	24	294	4137
132	4	237	0	4	33	1358	24	24	303	4118
99	4	206	1	4	33	1408	24	24	295	4143
109	5	204	0	4	26	1435	24	24	302	4181

A.4.3 Security Property: PSP

A	B	C	D	E	F	G	H	I	J	K
116	3	204	1	5	50	4220	68	60	724	8432
150	4	243	0	4	41	4205	68	60	732	8635
158	5	250	0	4	42	4436	68	60	734	8515
161	3	304	0	5	46	4095	68	60	728	8471
156	5	249	1	5	46	4298	68	60	977	8522
143	4	241	0	6	59	3936	68	60	829	8553
147	4	258	0	5	43	3874	68	60	730	8528
148	4	246	1	5	46	3688	68	60	731	8738
134	4	239	1	4	50	4158	68	60	744	8710
147	5	251	0	4	44	4076	68	60	848	8555
155	4	245	0	5	44	4346	68	60	726	8480
142	4	244	0	5	44	4259	68	60	715	8489
165	5	243	0	5	47	4115	68	60	725	8505
179	6	292	1	6	59	4096	68	60	728	8549
144	5	238	0	5	42	3859	68	60	750	8715
152	3	250	0	5	42	3979	68	60	757	8736
144	5	244	0	4	49	4094	68	60	749	8584
150	5	250	1	5	42	4374	68	60	740	8550
146	4	246	0	5	44	3981	68	60	736	8547
152	5	248	0	5	45	4092	68	60	751	8583
186	6	302	0	6	55	3670	68	60	807	8595
148	6	245	0	4	52	5789	68	60	813	8929
144	5	248	0	5	64	3692	68	60	1864	8871
161	4	241	0	5	42	4258	68	60	774	8799
150	5	229	0	4	44	4019	68	60	917	8681
169	5	251	0	5	41	4514	68	60	746	8608
139	4	250	1	5	57	4084	68	60	831	8574
149	5	255	0	5	44	4109	68	60	997	8641
147	5	254	0	6	46	4193	68	60	746	8612
149	3	254	1	5	51	3940	68	60	757	8872

A.5 Registry size: 1200 services

A.5.1 Security Property: Availability

A	B	C	D	E	F	G	H	I	J	K
169	6	392	0	4	35	5324	120	79	1683	20534
221	8	454	0	5	45	5255	120	79	1523	21006
196	8	483	0	6	41	5181	120	79	1497	20964
382	8	470	1	5	46	5273	120	79	1458	21036
190	6	450	0	5	44	5271	120	79	1573	21031
197	8	465	0	5	45	5918	120	79	1482	21125
379	7	567	0	4	52	5429	120	79	1508	21072
183	9	463	0	5	44	5571	120	79	1450	21078
196	8	479	0	6	52	5319	120	79	1476	21161
206	8	455	1	5	45	5117	120	79	1470	21138
353	7	470	0	6	44	5457	120	79	1472	21195
193	8	432	0	4	50	5174	120	79	1521	21082
328	8	459	0	4	45	5338	120	79	1449	21189
175	9	462	0	5	47	5336	120	79	1488	21168
180	8	471	0	6	41	5443	120	79	1482	21171
193	7	465	0	5	44	5172	120	79	1454	21117
333	6	458	1	4	52	5180	120	79	1464	21142
332	6	457	0	6	46	5221	120	79	1515	21196
185	7	475	1	4	40	5273	120	79	1568	21152
190	6	467	0	4	46	5139	120	79	1532	21373
364	6	472	0	5	46	5325	120	79	1491	21276
177	9	466	1	5	52	5150	120	79	1492	21211
196	9	470	1	4	46	5524	120	79	1507	21290
178	6	462	0	5	46	5330	120	79	1473	21371
185	7	433	0	4	42	5280	120	79	1565	21344
224	7	480	0	6	44	6159	120	79	5464	21642
235	9	527	0	5	70	5177	120	79	1707	21197
202	7	470	1	5	45	5402	120	79	1510	21401
178	8	486	0	6	46	5269	120	79	1570	21448
282	7	448	0	5	51	5593	120	79	1490	21503

A.5.2 Security Property: Integrity

A	B	C	D	E	F	G	H	I	J	K
171	8	487	1	5	35	2180	40	40	510	11827
162	7	513	1	6	40	2291	40	40	777	11865
147	7	461	0	5	65	2625	40	40	522	11639
189	7	472	1	5	34	2459	40	40	516	11954
182	9	529	1	5	38	2815	40	40	498	11544
187	10	449	0	4	35	2431	40	40	531	11525
145	8	486	1	5	36	3065	40	40	482	11579
168	8	473	0	5	36	2497	40	40	599	11559
161	7	489	1	4	34	2731	40	40	495	11559
167	7	484	0	5	36	2529	40	40	486	11607
147	7	491	0	5	39	2460	40	40	500	11566
324	6	491	0	6	36	2437	40	40	489	11624
179	7	443	0	15	33	2450	40	40	576	11663
178	8	470	0	4	49	2155	40	40	505	11634
182	9	563	0	6	60	2404	40	40	566	11683
171	8	462	0	5	50	2351	40	40	533	11695
180	7	470	0	5	33	2593	40	40	463	11112
133	7	447	0	61	39	2440	40	40	525	11225
130	6	444	0	4	34	2400	40	40	458	11092
316	7	463	0	6	35	2360	40	40	454	11109
151	8	463	0	5	45	2421	40	40	458	11138
151	9	468	0	6	32	2452	40	40	649	11147
118	8	461	0	6	34	2376	40	40	537	11148
161	8	449	0	5	46	2037	40	40	458	11190
128	7	434	1	4	32	2175	40	40	486	11395
157	6	390	0	4	33	2309	40	40	628	11346
144	8	444	0	6	32	4273	40	40	736	11236
124	7	382	0	4	26	2312	40	40	592	11275
136	7	471	0	4	33	2708	40	40	463	11177
145	7	469	0	5	35	2429	40	40	473	11079

A.5.3 Security Property: PSP

A	B	C	D	E	F	G	H	I	J	K
187	7	448	0	5	43	6911	104	104	1036	18701
182	7	473	0	5	41	6936	104	104	1340	19161
174	6	470	0	5	43	7035	104	104	1060	19120
180	7	469	0	4	43	7169	104	104	1084	19068
177	8	454	1	4	54	7202	104	104	1050	19163
172	7	463	0	6	44	6963	104	104	1060	19156
182	7	463	1	4	42	6751	104	104	1147	19140
196	7	463	0	5	47	6805	104	104	1081	19217
179	6	440	0	5	44	6760	104	104	1181	19195
191	8	468	0	4	45	6991	104	104	1100	19208
184	7	458	0	5	52	6884	104	104	1562	19340
170	7	454	0	5	45	6863	104	104	1065	19244
188	7	462	1	5	44	6943	104	104	1327	19186
202	8	468	0	6	48	6868	104	104	1172	19229
188	7	465	0	6	44	6884	104	104	1099	19245
179	7	481	0	6	56	7298	104	104	1081	19229
194	7	464	1	4	43	7561	104	104	1111	19225
198	9	460	1	4	65	7159	104	104	1078	19293
183	8	462	0	5	41	7324	104	104	1097	19310
191	7	461	0	6	41	6939	104	104	1088	19332
202	7	462	0	5	43	7015	104	104	1080	19378
192	6	474	0	4	43	6887	104	104	1140	19360
179	6	465	0	5	44	7020	104	104	1109	19410
204	7	463	0	6	53	6939	104	104	1277	19392
216	7	473	0	5	42	6882	104	104	1262	19413
186	9	474	0	4	43	7505	104	104	1106	19431
212	9	462	0	5	42	6926	104	104	1193	19377
185	8	459	0	5	47	6936	104	104	1088	19500
200	7	477	0	6	43	6917	104	104	1286	19521
212	7	487	0	5	44	6950	104	104	1439	19515