Generating Secure Service Compositions

Luca Pino[1(✉)], George Spanoudakis[1], Andreas Fuchs[2],
and Sigrid Gürgens[2]

[1] Department of Computer Science,
City University London, London, UK
{luca.pino.1,g.e.spanoudakis}@city.ac.uk
[2] Fraunhofer Institute for Secure Information Technology,
Darmstadt, Germany
{andreas.fuchs,sigrid.guergens}@sit.fraunhofer.de

Abstract. Ensuring that the compositions of services that constitute service-based systems satisfy given security properties is a key prerequisite for the adoption of the service oriented computing paradigm. In this paper, we address this issue using a novel approach that guarantees service composition security by virtue of the generation of compositions. Our approach generates service compositions that are guaranteed to satisfy security properties based on *secure service orchestration (SESO) patterns*. These patterns express primitive (e.g., sequential, parallel) service orchestrations, which are proven to have certain global security properties if the individual services participating in them have themselves other security properties. The paper shows how SESO patterns can be constructed and gives examples of proofs for such patterns. It also presents the process of using SESO patterns to generate secure service compositions and presents the results of an initial experimental evaluation of the approach.

Keywords: Software services · Secure service compositions · Security certificates

1 Introduction

The security of service based systems (SBS), i.e., systems that are composed of distributed software services, has been a critical concern for both the users and providers of such systems [3, 17, 24]. This is because the security of an SBS depends on the security of the individual services that it deploys, in complex ways that depend not only on the particular security properties of concern but also on the exact way in which these services are composed to form the SBS. Consider, for example, the case where the property required of an SBS is that the integrity of any data D, which are passed to it by an external client, will not be compromised by any of its constituent services that receive D. The assessment of this property requires knowledge of the exact services that constitute the SBS, the exact form of the composition of these services and the data flows between them, and a guarantee that each of the constituent services of SBS that receives D will preserve its integrity. Such assessments of security are required both during the design of an SBS and at

CLOSER 2014, CCIS 512, pp. 1–19, 2015. DOI:
10.1007/978-3-319-25414-2_6

runtime in cases where one of its constituent services S needs to be replaced and, due to the absence of any individual service matching it, a composition of services must be built to replace S. Whilst the construction of service compositions that satisfy functional and quality properties has been investigated in the literature (e.g., [1, 2, 27]), the construction of secure service compositions is not adequately supported by existing research.

In this paper, we present an approach for generating compositions of services, which are guaranteed to satisfy certain security properties. Our approach is based on the application of *SEcure Service Orchestration patterns* (SESO patterns). SESO patterns specify primitive service orchestrations, which are proven to have particular global (i.e., composition level) security properties, if their constituent services satisfy other service-level security properties. A SESO pattern specifies the order of the execution (e.g., sequential, parallel) of its constituent services and the data flows between them. It also specifies rules dictating the security properties that the constituent services of the pattern must have for the orchestration to satisfy a global security property. These rules express security property relations of the form *IF P THEN* $\wedge_{i=1,\ldots,n} P_i$ where P is a global security property required of the service orchestration and P_i are security properties, which must be satisfied by the services of the pattern for P to hold. These security property relations are formally proven. The constituent services of a SESO pattern are abstract "placeholder" services that need to be instantiated by concrete services when the pattern is instantiated.

When a constituent service S of an SBS needs to be replaced at runtime and no single alternative service S' satisfying exactly the same security properties as S can be found, SESO patterns can be applied to generate compositions of other services that have exactly the same security properties as S and could replace it within SBS. SESO patterns determine the criteria (security, interface and functional) that should be satisfied by the services that could instantiate them. These criteria are used to drive a discovery process through which the pattern can be instantiated. If this process is successful, i.e., different combinations of services that satisfy the required criteria and fit with the orchestration structure of the pattern are discovered, any combination (composition) of services that is built according to the pattern is guaranteed to have the required global security property by construction.

An earlier account of our approach has been given in [20, 22]. In this paper, we extend [22] by presenting the method that underpins the proof of security properties in SESO patterns, showing additional examples of concrete proofs of security properties for specific SESO patterns, and presenting the composition algorithm that generates secure service compositions that functionally relevant to the needed service.

The rest of this paper is organized as follows. Section 2 presents an overview of our approach. Section 3 discusses the validation of SESO patterns and provides examples of proofs of security properties for some patterns. Section 4 discusses the encoding of SESO patterns. Section 5 presents the SESO pattern driven service composition algorithm. Section 6 provides the results of an initial experimental evaluation of our approach. Section 7 overviews related work. Finally, Sect. 8 provides conclusions and directions for future work.

2 Overview

The service composition approach that we present in this paper extends a general framework developed at City University to support runtime service discovery [28]. This framework supports service discovery driven by queries expressed in an XML based query language, called *SerDiQueL*, which supports the specification of interface, behavioural and quality discovery criteria. The execution of queries can be reactive or proactive. In reactive execution, the SBS submits a query to the framework and gets back any services matching the query that are discovered by the framework. In proactive execution, the SBS submits to the framework queries that are executed in parallel, to find potential replacement services that could be used if needed, without the need to initiate and wait for the results of the discovery process at this point [28].

To take into account service security requirements as part of the service discovery process, we have extended the above framework in two ways. Firstly, we have extended *SerDiQueL* to enable the specification of the security properties that are required of individual services as querying conditions (the new language is called *A-SerDiQueL*). Secondly, we have developed a module supporting the generation of possible compositions of services that could replace a given service in an SBS in cases where a discovery query cannot find any single matching service. The generation of service compositions is based on the approach presented in this paper. In particular, this paper focuses on the process of searching for and constructing secure service compositions and the SESO patterns used in this process.

The key problems during the composition process are to ensure that the constructed composition of services: (a) provides the functionality of the service that it should replace, and (b) satisfies the security properties required of this service. To address (a), our approach uses abstract service workflows. These workflows express service coordination processes that realize known business processes through the use of software services with fixed interfaces. Such workflows are available for specific application domains such as telecom services (IBM BPM Industry Packs [13]), logistics (RosettaNet [25]), and are often available as part of SOA architecting and realization platforms (e.g., IBM WebSphere). Service workflows are encoded in an XML based language that represents the interfaces, and the control and data flows between the workflow activities.

To address (b), we are using SESO patterns. These patterns are based on primitive service orchestrations that have been proposed in the literature (e.g., sequential and parallel service execution) but augment them by specifying concrete security properties $P_1, \ldots, P_n$ that must be provided by the individual services that instantiate the pattern for the overall orchestration to satisfy a required security property P_0. The derivation of these security properties is based on rules that encode formally proven relations between the security properties of the individual placeholder services of the pattern and the security property required of the entire service orchestration represented by the pattern. Once derived through the application of rules, the security properties required of the individual partner services of the orchestration are expressed as queries in *A-SerDiQueL.* These queries are then executed to identify concrete services with the required security properties, which could instantiate the placeholder services of the pattern. If such services are found the pattern is instantiated. The pattern instantiation process is gradual and, if it is completed successfully, a new concrete and executable service composition that satisfies the overall security property guaranteed by the pattern is generated.

A key element of our approach is the formal validation of the relations between the security properties of the individual placeholder services of a SESO pattern and the security property of the entire composition expressed by the pattern. The validation of such relations is discussed in the next section.

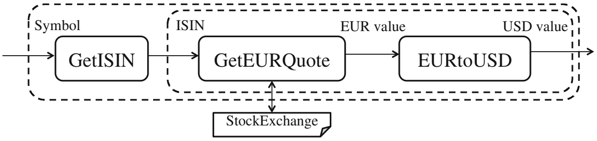

Fig. 1. Composition to replace *GetStockQuote*.

To illustrate our approach assume that a *Stock Broker* SBS that uses an operation *GetStockQuote* from a service *StockQuote* to obtain price quotations for a given stock. *GetStockQuote* takes as input a string *Symbol* identifying a stock and returns the current value of that stock in USD. If *StockQuote* becomes unavailable at runtime, then it becomes necessary to replace it with another service or a service composition (if no single replacement service can be discovered). A composition that may replace *StockQuote* is shown in Fig. 1. This composition contains three activities connected by two sequential patterns (indicated as dashed areas in workflow). The first placeholder of the outer sequence contains the activity *GetISIN*, which converts the *Symbol* identifying the Stock into the *ISIN* (another unique stock identifier). The second placeholder corresponds to the inner sequence. Within this inner sequence, the first placeholder is the activity *GetEURQuote* that involves *StockExchange* and returns the current stock value in EUR given the Stock *ISIN*. The second placeholder is the activity *EURtoUSD*, which converts a given amount from EUR to USD.

3 Validating Secure Service Compositions

In this section we introduce our approach for formally proving security properties of service compositions. This is based on generic models of service systems that take into account the different types of agents and actions that can be part of such systems. We then transform SESO patterns into different compositions of generic system models and show that such compositions satisfy specific security properties given that the individual system models satisfy some other security properties. In particular, we show that a sequential composition of two generic service models provides specific data integrity properties. Instantiating these service models with the concrete services of our example results in assurance that their sequential composition satisfies the respective concrete data integrity properties.

The task of formally validating the security of a service composition requires a three-step approach. It starts with a formal model of the service to be replaced and the formal models of the services to be composed. Firstly, the service composition is represented in terms of a formal model derived from the models of the individual services by applying a set of formal construction rules. These rules project the respective security properties of each of the composed services as well as the targeted property of the service to be replaced into the composed system. Secondly, additional properties are added to the composed system regarding the behaviour of the orchestrator, i.e., the primitive service orchestration pattern. Finally, the desired property is verified using the properties of the composed services and the orchestrator.

For the formal system representation and validation of security properties we utilize the Security Modeling Framework SeMF developed by Fraunhofer SIT [9–11]. In SeMF, a system specification is composed of a set $\mathbb{P}$ of agents and a set Σ of actions, $\sum_{/P}$ denoting the actions of agent P, and other system specifics that are not needed in this paper and are thus omitted. The behaviour B of a discrete system *Sys* can then be formally described by the set of its possible sequences of actions. Security properties are defined in terms of such a system specification. Relations between different formal models of systems are partially ordered with respect to different levels of abstraction. Formally, abstractions are described by so called alphabetic language homomorphisms that map action sequences of a finer abstraction level to action sequences of a more abstract level while respecting concatenation of actions. Language homomorphisms satisfying specific conditions are proven to preserve specific security properties, the conditions depending on the respective security property. Further information about SeMF can be found in [9–11].

Based on the representations of each of the service systems in the composition, we present a general construction rule using homomorphisms that map the service composition onto the individual services by preserving the individual services' security properties. This allows us to deduce the respective security properties to be satisfied by the composition. The different SESO patterns are translated into behaviour of the orchestrator regarding the invocation of the respective services. This includes functional and security related property statements. Based on this information it is possible to deduce the overall security properties of the composition system and validate whether they meet the expected results. In the next three sections, we illustrate our approach by exemplarily proving a specific data integrity property. The formal representation of services, composition and security properties is given in terms of generic agents and actions that are later instantiated by the SESO patterns towards concrete services and security properties. While our example focus on a single property on a specific set of orchestrations, our approach can handle various different orchestrations patterns, proving different instantiations of various security properties regarding integrity and confidentiality [21].

3.1 Formal Representation of Generic Service Composition

The formalization of a generic composition structure for service based systems is based on the following types of agents:

- *Clients C.* These are agents that use the service. They are thus specific to the service, and their actions are derived from the service's WSDL.

- *Service S.* This is the agent representing the service's communication interface (corresponding to its WSDL).
- *Backend Agents S-*.* These are service specific agents representing the implementation specifics, i.e. the internal functionality of a certain service (e.g. a backend storage used by the service).
- *Global agents G-**←representing third parties that are known to be identical for all services (e.g., some service providing external information).
- *R (the "rest of the word").* This is a default global external agent that is used to represent any agent other than those identified in *C, S-** and *G-**.

In the following, we denote (i) the system model of the service S^0 to be replaced by a composition by Sys^0, (ii) the system models of the services S^1 and S^2 to be composed by Sys^1 and Sys^2, respectively, and (iii) the composition system by Sys^c. The sets of agents and actions are denoted analogously (i.e. by $\mathbb{P}^i$, Σ^{i}, for $i = 0, 1, 2$). We mark service specific agents with the corresponding superscripts. We also mark global agents, even though being global, with superscripts in order to indicate the context of invocation (i.e. a global agent *G-A* being invoked by S^i is denoted by $G^i{-}A$. A generic system Sys^0 with service S^0, client C^0, a backend agent $S^0{-}A$ and a global agent $G^0{-}A$ can for example be instantiated by a service *StockbrokerService* using a backend storage service *StockbrokerService-Storage* for logging of client data and a global service *StockExchange* for actually retrieving the stock data.

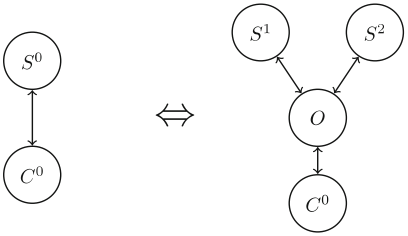

Fig. 2. Service Composition.

The principal idea of substituting a service by a composition is depicted in Fig. 2. More specifically, we assume two services S^1 and S^2 to act independently of (i.e., not to invoke) each other and utilize an orchestration engine O for their composition that takes the roles of both the clients C^1 and C^2 of Sys^1 and Sys^2 respectively, as well as the role of the service S^0 in Sys^0 to be replaced. Any global agent of Sys^0 will be part of the composition system and will be invoked by either S^1 or S^2. Furthermore, backend agents of Sys^0 will be removed, their functionality will be provided by the services S^1 or S^2 or their backend agents which will be part of the composition as well. This gives rise to the following set of agents of the composition:

$$\mathbb{P}^c = \left(\mathbb{P}^0 \backslash \{S^0, S^0 - *, G^0 - *\right) \cup \left(\mathbb{P}^1 \backslash \{C^1\}\right) \cup \left(\mathbb{P}^2 \backslash \{C^2\}\right) \cup \{O\}$$

We then view the systems *Sys⁰*, *Sys¹* and *Sys²* as homomorphic images of the composed system *Sysᶜ*.

In order to determine the action set Σ^c of the composition we use a generic renaming function $r_{P\to Q}$:$\Sigma \to \Sigma_{r_{P\to Q}}$ that replaces all occurrences of agent P in an action by *Q*. Based on this function, we define functions r^i:$\Sigma^i \to \Sigma^c$ $(i = 0,1,2,\ j = 1,2)$ that formalizes the orchestrator taking the roles of S^0, $C^1 and\ C^2$ as follows:

$$\begin{aligned} r^0(a) &= r_{S^0\to O}(a) \\ r^{\mathrm{j}}(a) &= r_{C^{\mathrm{j}}\to O}(a) \end{aligned}$$

The resulting set Σ^c of actions of the composed system is then as follows:

$$\Sigma^c = r^0\left(\Sigma^0_{/C^0} \cup \Sigma^0_{/S^0}\right) \cup r^1\left(\Sigma^1\right) \cup r^2\left(\Sigma^2\right) \cup \Sigma^c_{/O}$$

$\Sigma^c_{/O}$ represents additional actions taken by the orchestration engine beyond the communication with client and services. These actions depend on the specific orchestration pattern used and will be discussed in the next section.

Now we need to assure that for all actions in Sys^0 exists an equivalent provided by either S^1 or S^2, i.e. the above set Σ^c of actions must satisfy the following:

$$\begin{aligned} \forall a \in \Sigma^0, \exists i \in \{1,2\}, \exists a' \in \Sigma^c:\quad & r_{G^0-*\to G^i-*}(a) &= a' \vee \\ & r_{S^0-*\to S^i-*}(a) &= a' \end{aligned}$$

Since the functions r^i are injective we can now use their inverse image in order to define the homomorphisms that map the composition system onto the abstract systems: each homomorphism h^i abstracts Σ^c to Σ^i. Regarding the actions corresponding to those in Σ^i, h^i is simply the inverse of r^i, and all other actions are mapped onto the empty word. Hence for $i = 0,1,2$ and $j = 1,2$ we define h^i:$\Sigma^c \to \Sigma^i$ as follows:

$$h^0(a) = \begin{cases} a' & if\ \exists a' \in \Sigma^0 : r^0\left(a'\right) = a \\ a'' & if\ \exists a'' \in \Sigma^0, \exists i \in \{1,2\} : r_{G^0-*\to G^i-*}\left(a''\right) = a \\ a''' & if\ \exists a''' \in \Sigma^0, \exists i \in \{1,2\} : r_{S^0-*\to S^i-*}\left(a'''\right) = a \\ \varepsilon & else \end{cases}$$

$$h^j(a) = \begin{cases} a' & if\ \exists a' \in \Sigma^j : r^j\left(a'\right) = a \\ \varepsilon & otherwise \end{cases}$$

These homomorphisms serve as a means to relate not only the models of the individual systems to the composition model but also to relate - under certain conditions - their security properties. A homomorphism that fulfills certain conditions "transports" a security property from an abstract system to the concrete one, i.e. if the conditions are satisfied and the property holds in the abstract system, the corresponding property will also hold in the concrete system. Thus, the homomorphism *preserves* the property. The conditions that must be satisfied depend on the property in question; see [9, 10] for example. We use this approach to prove specific security properties for a composition of services based on the security properties of these services.

3.2 Formally Representing Sequential Composition

Our methodology for service composition has been applied to various different patterns, proving different instantiations of various security properties (see [21] for more details). In the following, we will focus on a specific case for a sequential service composition that corresponds to the example introduced in Sect. 2.1 in order to illustrate our approach. We assume the original service S^0 to invoke a global agent $G{-}A$ (denoted by $G^0{-}A$). For its substitution, the pattern for sequential composition of services realizes the subsequent invocation of two services S^1 and S^2, where the output of S^1 serves as input for S^2. The global agent $G{-}A$ will be invoked by either S^1 or S^2 (denoted by $G^1{-}A$ and $G^2{-}A$, respectively).

The actions of the systems are constructed from the generic service operations op_0, op_1, and op_2 (that represent the operations of concrete services' WSDL) as prefix, followed by one of the suffixes *IS*, *IR*, *OS*, *OR* to represent *InputSend*, *InputReceive*, *OutputSend*, *OutputReceive*, respectively. This naming scheme corresponds to our method of transforming a service's WSDL into sets of agents and actions introduced in [12]. The actions of the global agents, not being part of the service's WSDL, do not follow this notation. This leads to the following specification of systems Sys^i:

$$\mathbb{P}^i \supseteq \left\{C^i, S^i\right\} \quad \Sigma^i \supseteq \left\{ \begin{array}{c} op_i{-}IS\left(C^i, S^i, data_i\right), \\ op_i{-}IR\left(S^i, C^i, data_i\right), \\ op_i{-}OS\left(S^i, C^i, f_i(data_i)\right), \\ op_i{-}OR\left(C^i, S^i, f_i(data_i)\right) \end{array} \right\}$$

$$\mathbb{P}^j \supseteq \{G^j{-}A\} \quad \Sigma^j \supseteq \{\mathrm{act}(\mathrm{G}^j{-}\mathrm{A}, \mathrm{in}_{\mathrm{G}^j-\mathrm{A}}, \mathrm{out}_{\mathrm{G}^j-\mathrm{A}})\}$$

with $i = 0,1,2,$ and $j = 0,1$ for Sys^1, and $j = 0,2$ for S^2.

In the system Sys^0, when S^0 receives some data $data_0$ from the client, before forwarding it to the global agent it applies a function f_I^1. In our stockbroker example introduced in Sect. 2.1, this function could for instance remove the client's name or account). The global agent (*StockExchangeService* in the example) acts on receiving the input in_{G^0-A} and produces the output out_{G^0-A} (say, the stock value in Euros and the bill for the service provided). The global agent's input and output may or may not be functionally related. Such a relation is necessary in case an integrity property shall be expressed that involves the complete sequence of actions, starting with the client providing the input data and ending with the client receiving the final output. The case we investigate below considers only half of this sequence, starting with the global agent's output, thus a relation between global agent's input and output is not needed. Accordingly, $G^0{-}A$ returns out_{G^0-A} to S^0. The service then applies a function f_O^0 and sends the result to the client. The stockbroker service in our example could for instance remove the bill and just keep the stock value, and convert it to US dollar.

In the sequential composition pattern, the orchestrator forwards $data_0$ received from C^0 to S^1. In case it is S^1 that invokes the global agent, before doing so the service computes $f_I^1(data_0)$ (removes the client's name and account) and sends this to $G^1{-}A$. As in Sys^0, the global agent produces out_{G^1-A} and returns this to S^1. Note that input as well as output of the global agent are the same as in Sys^0 (otherwise the global agent would not be global). Now S^1 applies f_O^1 (removes the global agent's bill) and sends $f_O^1(out_{G^1-A})$ to the orchestrator. These data are then forwarded by the orchestrator to S^2 who applies f^2 (converts euro to dollar) and returns $f^2(f_O^1(out_{G^1-A}))$ which the orchestrator finally returns to the client. A similar sequence of actions occurs if the global agent is invoked by S^2. In a more complex scenario the orchestrator can for example alter (e.g., split) the client data and combine the output of S^1 with some data resulting from the client's input and send this to S^2. A proof for this more complex construction is achievable analogously to the one presented below.

The agent and action sets of the composition are constructed as specified in the previous section, using the functions r^0, r^1 and r^2. Function r^0 for example maps action $op_0\text{-}IS(C^0, S^0, data_0)$ onto $op_0\text{-}IS(C^0, O, data_0)$, hence $h^0(op_0\text{-}IS(C^0, O, data_0)) = op_0\text{-}IS(C^0, S^0, data_0)$, while $h^0(op_2\text{-}OR(O, S^2, f_2(data_2))) = h^0(r^2(op_2\text{-}OR(C^2, S^2, f_2(data_2)))) = \varepsilon$., with $data_1 := data_0$ and $data_2 = f_O^1(out_{G^1-A})$.

3.3 Validation of Integrity Preserving Compositions

Our approach of proving security properties of service compositions is generic and has already been applied to various integrity and confidentiality properties (see [21] for more details). As an example of such proofs, we will now present the proof regarding a specific data integrity property of S^0 being provided by the orchestration specified above. The definition of (data) integrity that we assume in our example is taken from RFC4949, i.e. "The property that data has not been changed, destroyed, or lost in an unauthorized or accidental manner." [26]. In SeMF, this property of data integrity is expressed by the concept of *precedence*: *pre(a, b)* holds if all sequences of actions $\omega \in B$ that contain action b also contain action a. Obviously, precedence is transitive (we omit the trivial proof). Further, precedence is preserved by any homomorphism if $h(B) \subseteq B'$ (see [11] for a proof). With $B^c = h^{1^{-1}}\left(B^1\right) \cap h^{2^{-1}}\left(B^2\right) \cap \{\omega \in \Sigma^c | \forall\, Prop \in prop_O{:}prop\}$ all precedence properties are preserved in the following, with $Prop^O$ denoting the orchestrator assumptions (see P4 and P5 below).

Out of the many properties related to integrity and sequential composition we now investigate one that is related to transmission of data between a global agent and a client which results into four different properties. On the one hand, we can investigate the integrity of data transmitted from the client to a global agent vs. the one transmitted from a global agent to the client. On the other hand, we can differentiate between the global agent being invoked by either S^1 or S^2. Exemplarily we use the case where S^1 invokes the global agent and assume S^0 to provide the following integrity property: Each time client C^0 receives data from service S^0, this data originates from the global agent that was properly manipulated by S^0. Formally:

$$\mathrm{P0{:}\ \forall data{:}op_0\text{-}OR}\left(\mathrm{C^0, S^0, data}\right) \in \mathrm{alph}\,(\omega) \rightarrow \mathrm{data} = \mathrm{f^0_O}\left(\mathrm{out_{G^0\text{-}A}}\right) \wedge$$
$$\mathrm{pre}\left(\mathrm{act}\left(\mathrm{G} - \mathrm{A}, \mathrm{in_{G^0\text{-}A}}, \mathrm{out_{G^0\text{-}A}}\right), \mathrm{op_0\text{-}OR}\left(\mathrm{C^0, S^0, f^0_O}\left(\mathrm{out_{G^0-A}}\right)\right)\right)$$

As explained above, precedence shall be preserved by h^0 (as constructed in Sect. 3.1). Since the global agent's action in the composition is identical to the one in Sys^0, it must receive the same input in order for the composition to achieve the same functionality, hence $f_I^0 = f_I^1$. Also, what C^0 receives in the composition must be identical to what it receives in Sys^0. This implies $f_2 \circ f_O^1 = f_O^0$ which results in the following property of the composition (corresponding to P1') that we want to prove:

$$P1{:}\quad \forall data{:}\ op_0{-}OR\left(C^0, O, data\right) \in alph\,(\omega) \rightarrow data = f^2\left(f_O^1\left(out_{G^1-A}\right)\right) \wedge$$
$$pre\left(act\left(G{-}A, in_{G^1-A}, out_{G^1-A}\right), op_0{-}OR\left(C^0, O, f^2\left(f_O^1\left(out_{G^1-A}\right)\right)\right)\right)$$

For our proof, we assume that the services Sys^1 and Sys^2 provide the properties:

$$P2'{:}\quad \forall data{:}op_1{-}OR\left(C^1, S^1, data\right) \in alph\,(\omega) \rightarrow data = f_O^1\left(out_{G^1-A}\right) \wedge$$
$$pre\left(act\left(G^1{-}A, in_{G^1-A}, out_{G^1-A}\right), op_1{-}OR\left(C^1, S^1, f_O^1\left(out_{G^1-A}\right)\right)\right)$$

$$P3'{:}\quad \forall data{:}op_2{-}OR\left(C^2, S^2, data\right) \in alph\,(\omega) \rightarrow data = f^2(data_2) \wedge$$
$$pre\left(op_2{-}IS\left(C^2, S^2, data_2\right), op_1{-}OR\left(C^1, S^2, f^2(data_2)\right)\right)$$

The homomorphisms h^1 and h^2 as constructed in Sect. 3.1 preserve these precedence properties. Accordingly, the corresponding properties in Sys^c are:

$$P2{:}\quad \forall data{:}op_1{-}OR\left(O, S^1, data\right) \in alph\,(\omega) \rightarrow data = f_O^1\left(out_{G^1-A}\right) \wedge$$
$$pre\left(act\left(G^1{-}A, in_{G^1-A}, out_{G^1-A}\right), op_1{-}OR\left(O, S^1, f_O^1\left(out_{G^1-A}\right)\right)\right)$$

$$P3{:}\quad \forall data{:}op_2{-}OR\left(\mathrm{O}, S^2, data\right) \in alph\,(\omega) \rightarrow data = f^2(data_2) \wedge$$
$$pre\left(op_2{-}IS\left(\mathrm{O}, S^2, data_2\right), op_1{-}OR\left(\mathrm{O}, S^2, f^2(data_2)\right)\right)$$

In addition, the orchestrator must act according to the pattern (as specified in Sect. 3.2), i.e., satisfy the following properties:

$$P4{:}\quad pre\left(op_1{-}OR\left(O, S^1, data\right), op_2{-}IS\left(O, S^2, data\right)\right)$$
$$P5{:}\quad pre\left(op_2{-}OR\left(O, S^2, data\right), op_0{-}OR\left(C^0, O, data\right)\right)$$

Proof. Assume there is $\omega \in \mathrm{B}$ with $\mathrm{op_0{-}OR}\left(C^0, \mathrm{O, data}\right) \in \mathrm{alph}(\omega)$. Property P5 implies that $\mathrm{op_2 - OR}\left(\mathrm{O, S^2, data}\right) \in \mathrm{alph}(\omega)$. By P3, $\mathrm{data} = \mathrm{f^2}\left(\mathrm{data_2}\right)$ and further $\mathrm{op_2{-}IS}\left(\mathrm{O, S^2, f^2}\left(\mathrm{data_2}\right)\right) \in \mathrm{alph}(\omega)$. In the next step, Property P4 implies that $\mathrm{op_1 - OR}\left(\mathrm{O, S^1, f^2}\left(\mathrm{data_2}\right)\right) \in \mathrm{alph}(\omega)$. By P2, we can deduce $\mathrm{data_2} = \mathrm{f^1_O(out_{G^1-A})}$, i.e. $\mathrm{data} = \mathrm{f^2}\left(\mathrm{f^1_O}\left(\mathrm{out_{G^1-A}}\right)\right)$, and $\mathrm{act}\left(\mathrm{G^1 - A, in_{G-A}, out_{G^1-A}}\right) \in \mathrm{alph}(\omega)$ which implies that property P1 holds.

Due to the simplicity of the precede property, the above proof is simple. In [21] we have proven other integrity properties (e.g. the global agent being invoked by S^2). We have also proven several confidentiality properties. All proofs use the approach presented in this paper: (i) deriving the formal model of the service composition from the formal models of the individual services, (ii) relating these models by using property preserving homomorphisms and thus representing the individual services' security properties in terms of the composition model, and (iii) using appropriate security properties to be satisfied by the orchestrator. Whilst we assume the orchestrator to behave correctly and hence to satisfy these additional properties, the security properties we assume for the individual services of the composition are translated into inference rules, which are then used in order to construct a concrete service compositions. The proofs of security properties for specific SESO patterns need to be constructed offline and encoded in the patterns as rules. At runtime, the rules encoding the patterns are used to deduce the security properties that must be satisfied by the candidate services that may instantiate the pattern.

4 Secure Service Orchestration Patterns

Proofs of security properties, like the one that we discussed in Sect. 3, form the basis of SESO patterns in our approach. More specifically, an SESO pattern encodes: (a) a primitive orchestration describing the order of the execution and the data flow between placeholder services, and (b) the implications between the security properties of these services and the security property of the whole orchestration. The placeholder services within a primitive orchestration can be atomic activities (i.e., abstract partner services) or other patterns. The implications in (b) are of the form:

> “IF P is a primitive orchestration with placeholders S_1, …, S_n and ρ^P is a security property required for P THEN ρ^P is guaranteed if each S_i in P satisfies the security properties ρ_j ($j = 1$, …, m_i)”.

These implications reflect proofs of security properties, developed based on the approach discussed in Sect. 3. They are encoded as inference rules and used during the composition process to infer the security properties that would be required of the placeholders of a pattern P for it to satisfy ρ^P. The benefit of encoding proven implications as inference rules is that there is no need to reason from first-principles when attempting to construct compositions of services, based on SESO patterns.

To be more specific, SESO patterns and implications of the above form are encoded as Drools production rules [8]. Drools is a rule-based reasoning system supporting reasoning driven by production rules. Production rules in Drools are used to derive information from data facts stored in a Knowledge Base (KB). A production rule in Drools has the general form: *when <conditions> then <actions>*. When a rule is applied, the rule engine of Drools checks, through pattern matching, whether the conditions of the rule match with the facts in the KB and, if they do, it executes the actions of the rule. This execution can update the contents of the KB. Table 1 shows the encoding of integrity in the sequential orchestration pattern that was presented in Sect. 3.3 as a Drools rule. This rule uses the following definitions of integrity:

Table 1. Integrity Rule for Sequential SESO Pattern.

```
 1:  rule "Integrity on Sequential. Case GA at S1 to C"
 2:   when
 3:     $outGA : Parameter()
 4:     $f1-outGA : Parameter(functionOf == $outGA)
 5:     $f2-f1-outGA : Parameter(functionOf == $f1-outGA)
 6:     $GA1 : GlobalAgent(parameter == $outGA)
 7:     $S1 : Activity(globalAgents contains $GA1,
                         outputs contains $f1-outGA)
 8:     $S2 : Activity(inputs contains $f1-outGA,
                         outputs contains $f2-f1-outGA)
 9:     $P : Sequential(activity1 == $S1, activity2 == $S2)
 10:    $ rhoP : IntegrityGA2C(subject == $P, GA == $GA1,
                         data == $f2-f1-outGA)
 11:  then
 12:    insert(new IntegrityGA2C($S1, $GA1, $f1-outGA));
 13:    insert(new IntegrityE2E($S2, $f1-outGA, $f2-f1-outGA));
 14:    retract($rhoP);
 15: end
```

Definition 1. The integrity of data X generated by a global agent GA and sent to the client by S^1: *IntegrityGA2C*(S^i, GA, f^i(X)) = *pre*(act(GA, _, X), op_i-OR(C^i, S^i, f^i(X)))

Definition 2. The end-to-end integrity of the data, from input to output (i.e. the property investigated in a former version of this work [22]): *IntegrityE2E*(S^i, X, Y) = *pre*(op_i-IS(C^i, S^i, X), op_i-OR(C^i, S^i, Y)).

Using such more abstract security properties in the rules avoids the need to encode in the rule the formalism that the proof is based on. This makes it also possible to use SESO patterns proven through different formalisms in our approach.

Returning to the rule in Table 1, lines 3–9 describe the primitive orchestration that the security property refers to. More specifically, the rule can be applied when a sequential pattern (`$P`) with two placeholders, i.e., activity `$S1` followed by activity `$S2`, is encountered. Activity `$S1` interacts with a global agent `$GA1` that generates output `$outGA`. The rule defines the parameters of these activities: `$S1` has an output parameter `$f1-outGA`, that is a function of `$outGA`, and `$S2` uses the input parameter `$f1-outGA` in order to generate the output parameter `$f1-f2-outGA`, as shown in Table 1. Line 10 describes the original security requirement requested on the composition pattern `$rhoP`, i.e. integrity on the pattern `$P` of the data `$f2-f1-outGA` originally generated by `$GA1`. This requirement is equivalent to the property P1 presented in Sect. 3.3. Lines 12–14 (i.e., the `then` part of the rule) specify the security properties that are required of the activities of the pattern in order to guarantee `$rhoP`, namely: (i) integrity on the output (`$f1-outGA`) of `$S1` generated by `$GA1`, as stated by the precedence property P2, and (ii) end-to-end integrity on the input (`$f1-outGA`) and output (`$f2-f1-outGA`) of `$S2`, as required from P3. Additionally, we assume the framework executing the orchestration to satisfy properties P4–P5, hence these need not be mentioned in the rule.

Finally, according to the rule, once the original requirement $rhoP is guaranteed by the new ones, it can be removed from the KB.

Similar encodings of other SESO patterns have been expressed using this approach. SESO pattern encoding rules, like the one presented above, are used during the composition process to infer the security properties that are required of the concrete services that may instantiate the placeholder services in a workflow. This process is discussed next.

5 SESO Pattern Driven Service Composition

The service composition process is carried out according to the algorithm shown in Table 2. This algorithm is invoked when an SBS service needs to be replaced but the service discovery query specified for it cannot identify any single service matching it.

In such cases, the structural part of the query, which defines the operations that a service should have and the data types of the parameters of these operations, is used to retrieve from the repository of the discovery framework abstract workflows that can provide the required service functionality. An abstract workflow represents a coarse grained orchestration of activities, which collectively offer a specific functionality, and is exposed as a composite service. Such workflows are fairly common (e.g., [5, 19]) and result from reference process models in specific domains [13, 25]. The activities of an abstract workflow are orchestrated through a process consisting of the primitive orchestrations that underpin the security patterns, as discussed in Sect. 4. If such workflows are found the generation of a service composition is attempted by trying to instantiate each abstract workflow.

As shown in Table 2, initially, the algorithm identifies the abstract workflows that could be potentially used to generate a composition that can provide the operations of the required service (see STRUCTURALMATCH function in line 3). This is based on the execution of the query associated with the service to be replaced (Q_S). If such workflows are found, the algorithm continues by starting a process of instantiating the activities of each of the found workflows with services. The activities of the workflows are instantiated progressively, by investigating each workflow W in a depth-first manner. More specifically, the algorithm takes the first unassigned activity A in W (in the control flow order) and builds a query Q_A based on the workflow specification and the discovery query Q_S. In particular, the structural part of Q_A is taken from the description of A in the abstract workflow. The security conditions in Q_A are generated through the procedure SECURITYCONDITIONS(Q_S, W).

This procedure infers the security conditions for A based on the Drools rules that encode the SESO patterns detected within the current workflow. More specifically, all the information about the workflow, its patterns, activities, security properties and requirements are put into the KB. Then the rules that represent the detected SESO patterns are fired (i.e. applied), propagating the requirements through the workflow. The generated requirements for the unassigned activity are then retrieved and converted to query conditions. The propagation of security requirements is possible as each workflow can be seen as a recursive application of primitive orchestrations.

Table 2. Service Composition Algorithm.

```
Require: $Q_S$ - query for the required service
Ensure: ResultSet - set of instantiated workflows
procedure ServiceComposition($Q_S$)
  for all abstract workflows AW in the repository do
    if structuralMatch($Q_S$, AW) == true then
      Put a copy of AW in WStack
    end if
  end for
  while there are more workflows in WStack do
    Get the first workflow W in the WStack
    Pop the first unassigned activity A from W
    Extract the structural query $Q_A$ for A from W
    SecCond := SecurityConditions($Q_S$, W)
    Add to $Q_A$ the security conditions SecCond
    Res := ServiceDiscovery($Q_A$)
    for all services S* in Res do
      $W_{S*}$ := W[A/S*]                    //i.e. substitute S* for A in W
      if exists an unassigned activity in $W_{S*}$ then
        Push $W_{S*}$ in WStack
      else
        Add $W_{S*}$ to ResultSet
      end if
    end for
  end while
  return ResultSet
end procedure
```

Figure 3 shows the order of propagation through the use of the rules, on a workflow shown in (c). A security requirement ρ^S is initially given for a service *S* (Fig. 3(a)). The first rule that will be fired by Drools is the one for the outermost pattern of the workflow: a choice pattern (i.e., the *if-then-else* primitive orchestration in Fig. 3(b)). The security requirement is then propagated by the relevant rule (if such a rule exists) to the placeholders *A* and *B* returning the requirements ρ^{A1}, …, ρ^{An} and ρ^{B1}, …, ρ^{Bm} (with $n, m \geq 0$ and $n + m \geq 1$). For each security requirement ρ^{Ai} (with $i = 1, \ldots, n$), a rule is fired to propagate the requirement to the sequential pattern that instantiates *A* (Fig. 3(c)). This process generates the security requirements for placeholders *C* and *D*.

If a security requirement cannot be propagated to the atomic activity level (e.g., no rules are defined for the given pattern or security property) then Drools returns an error state to point out that a security requirement cannot be guaranteed by the existing set of rules. This ensures that no security requirements are ignored.

After constructing Q_A, the query is executed by the runtime discovery framework in [28] to identify a list of candidate services for Q_A. The candidate services (if any) are then used to instantiate the activity A in W. Note that the composition algorithm implements a depth-first search in the composition generation process in order to explore fully the instantiation of a particular activity within a pattern before considering other activities. This spots dead-ends sooner than a breadth-first search.

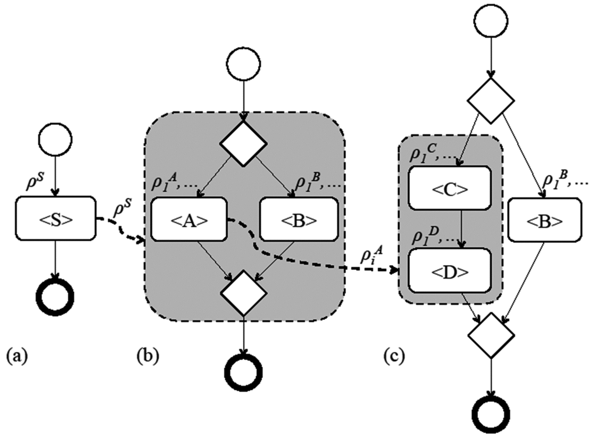

Fig. 3. Recursive applications of secure service orchestration patterns.

As an example of applying the algorithm in Table 2, consider the Stock Broker example introduced in Sect. 2.1. Suppose that the Stock Broker SBS has a security requirement regarding integrity of the output data of its *StockQuote* service, and would consider replacement services that can offer the same operation only if they satisfy this particular security requirement. To deal with potential problems with *StockQuote* at runtime (e.g., unavailability), *Stock Broker* can subscribe a service discovery query Q_{SQ} for replacing *StockQuote* to the discovery framework and request its execution in proactive mode. Q_{SQ} should specify the functional and security properties that the potential replacement services of *StockQuote* must have. If the execution of Q_{SQ} results in discovering no single service matching it (i.e., when single service discovery fails), the service composition process is carried out. At this stage, according to the algorithm of Table 2, the framework will query the abstract workflow repository to locate workflows matching Q_{SQ}.

Suppose that this identifies the abstract workflow W_{SQ} shown in Fig. 1 that matches the query. This workflow contains a sequence of three activities: *GetISIN*, *GetEURQuote* and *EURtoUSD*. The framework then infers the security properties required for each of the services that could instantiate the activities of W_{SQ} and uses them to query for such services. Initially, a rule for integrity of data D on a sequential pattern with the global agent generating D in the second activity is fired on the external sequential pattern. This rule and the related proof are given in [21] (Sect. 3.3.3, case 2). The rule is applied because the property required for the external sequential pattern is that the output of the workflow (i.e., *USD value*) must have been computed from the value returned by *StockExchange*. From the required security property, the rule derives only one property about the integrity of *USD value* (again, from the value coming from *StockExchange*) for the inner sequential pattern. This newly generated property fires the rule shown in Table 1 resulting in two security properties: (1) integrity from the global agent *StockExchange* to the client for *GetEURQuote* output *EUR value*, and (2) end-to-end integrity on inputs and outputs of *EURtoUSD*.

After the application of the rules, we derive the required property for the first unassigned activity *GetISIN*. In this instance, no security properties are requested from the first activity. This means that the query used to instantiate the workflow consists only of the interface required for *GetISIN*. In a similar way, a query specifying the required interface is created for the second activity, *GetEURQuote*. This query, however, will include also the security property required for the activity i.e., integrity of *EUR value* that is passed from *StockExchange* to the client. The query is then executed and the discovered services are used to instantiate the workflow. Note that in the discovery process, services are considered to satisfy the required security properties only if they have appropriate certificates asserting these properties. Similarly for the last activity, *EURtoUSD*, a query is generated from the service interface and the required security properties and then executed, and the workflow gets instantiated by the results. After the replacement service is fully composed, the service composition is published in a BPEL execution engine and its WSDL is sent to the *Stock Broker* SBS in order to update its bindings.

6 Tool Support and Experiments

To implement and test our approach, we have developed a prototype realizing the composition process and integrated it with the runtime service discovery tool described in Sect. 2. The prototype gives the possibility to select a service discovery query and execute it to find potential candidate services and service compositions. If alternative service compositions can be built, the alternatives are presented to the user who can select and explore the services in each of them.

Early performance tests of our approach have been carried out using service registries of different sizes. Table 3 shows average execution times for single service and service composition discovery obtained from using our tool on an Intel Core i3 CPU (3.06 GHz) with 4 GB RAM. The reported times are average times taken over 30 executions of each discovery query. In the experiments, we used service registries of four sizes (150, 300, 600 and 1200), 25 abstract workflows and 3 patterns.

As shown in the table, the time required for building service compositions is considerably higher than the time required for single service discovery. The main part of this cost comes from the process of discovering the individual services to instantiate the partner links of the composition. Although the overall composition time is high, its impact is not as significant, since as we discussed in Sect. 2 our framework can apply discovery and service composition in a proactive manner, i.e., discover possible service compositions in parallel with the operation of an SBS and use them when a service needs to be replaced. Furthermore, the cost of compositions can be reduced or kept under a given threshold by controlling the number of alternative compositions that the algorithm in Table 2 builds. In [28], the authors have shown the benefits of a proactive execution of the service discovery process used in our approach. Hence, we believe that the proactive generation of compositions could also reduce execution time but this would need to be confirmed experimentally.

Table 3. Execution times (in msecs).

Registry size	150	300	600	1200
Single service discovery time	194	275	355	642
Composition discovery time	777	2214	4943	12660
No. of generated compositions	4	12	24	40

7 Related Work

Existing work in service composition has focused on creating compositions that have certain functional and quality of service properties (e.g., [1, 2, 14, 17, 23, 24, 27]) and provides only basic support for addressing security properties in service composition, which is the main focus of our approach.

The creation of service compositions that satisfy given security properties has been a focus of work on model based service composition (e.g., [4, 6, 7]). In this area, service compositions are modeled using formal languages and their required properties are expressed as properties on the model. Our approach to composition is also model based. However, it uses model based property proofs to identify how overall security properties of compositions can be guaranteed through the security properties of the individual components (services) of the composition. Existing work on model based service composition could provide proofs of additional security properties, which could be used to extend the patterns used in our approach, even if they use different formalisms. The compositionality results for information flows discussed in [18], for example, can be easily converted into SESO patterns.

Other work on service composition focuses on discovering services that have given security properties (e.g., [3, 5, 15, 16, 19]). Some of these approaches focus on specific types of security properties (e.g., [16, 19]) whilst other focus on how to express and check security properties but only for single partner services of a composition (e.g., [3, 5, 15]). In contrast, our approach can support arbitrary security properties and properties of entire service compositions.

Two ontology-based frameworks for automatic composition are described in [15] and [19]. The first framework defines a set of metrics for selecting amongst different compositions but provides limited support for security. The second framework introduces hierarchies of security properties but does not support the construction of secure service compositions. In [16] planning techniques have been used to build sequential compositions guarantying access control models, and [5] introduces an approach to security aware service composition that matches security requirements with external service properties. The focus of [3] is on generating test-based virtual security certificates for service compositions, derived from the test-based security certificates of the external services that form the composition.

8 Conclusion

In this paper, we have presented an approach supporting the discovery of secure service compositions. Our approach is based on secure service orchestration (SESO) patterns. These patterns comprise specifications of primitive orchestrations describing the order of the execution and the data flow between placeholder services, and rules reflecting formally proven relations between the security properties of the individual placeholders and the security property of the whole orchestration. The formal proofs (and patterns) developed so far cover different integrity and confidentiality properties for different forms of primitive orchestrations. During the composition process, the proven relations between security properties are used to deduce the actual properties that should be required of the individual services that may instantiate an orchestration for the orchestration to satisfy specific security properties as a whole. In order to facilitate reasoning, SESO patterns are encoded as Drools rules. This enables the use of the Drools rule based system for inferring the required service security properties when trying to generate a service composition.

Our approach has been implemented and integrated with a generic framework supporting runtime service discovery that is described in [28]. We are currently investigating the validity of our approach through a series of focus group evaluations. We are also investigating further SESO patterns (e.g., for availability), and conducting further performance and scalability analysis of our prototype. We are also exploring the use of heuristic controls over the number of compositions generated by the algorithm to speed up the processing.

Acknowledgements. The work reported in this paper has been partially funded by the EU F7 project ASSERT4SOA (grant no.257351).

References

1. Aggarwal, R., et al.,: Constraint driven web service composition in METEOR-S. In: Proceedings of the IEEE International Conference on Services Computing (SCC 2004), pp. 23–30 (2004)
2. Alrifai, M., Risse, T., Nejdl, W.: A hybrid approach for efficient Web service composition with end-to-end QoS constraints. ACM Trans. Web **6**(2), 7:1–7:31 (2012)
3. Anisetti, M., Ardagna, C., Damiani, E.: Security certification of composite services: a test-based approach. In: Proceedings of the IEEE 20th International Conference on Web Services, pp. 475–482 (2013)
4. Bartoletti, M., Degano, P., Ferrari, G.L.: Enforcing secure service composition. In: Proceedings of the 18th Computer Security Foundations Workshop (CSFW), pp. 211–223. IEEE Computer Society (2005)
5. Carminati, B., Ferrari, E., Hung, P.C.K.:. Security conscious web service composition. In: Proceedings of the International Conference on Web Services (ICWS), pp. 489–496. IEEE Computer Society (2006)
6. Deubler, M., et al.: Sound development of secure service-based systems. In: Proceedings of 2nd International Conference on Service Oriented Computing, pp. 115–124 (2004)
7. Dong, J., Peng, T., Zhao, Y.: Automated verification of security pattern compositions. Inf. Softw. Technol. **52**(3), 274–295 (2010)

8. Drools. http://www.jboss.org/drools/
9. Gürgens, S., Rudolph, C., Ochsenschläger, P.: Authenticity and provability - a formal framework. In: Rees, O., Frankel, Y., Davida, G.I. (eds.) InfraSec 2002. LNCS, vol. 2437, pp. 227–245. Springer, Heidelberg (2002)
10. Gürgens, S., Ochsenschläger, P., Rudolph, C.: Abstractions preserving parameter confidentiality. In: di Vimercati, S., Gollmann, D., Syverson, P.F. (eds.) ESORICS 2005. LNCS, vol. 3679, pp. 418–437. Springer, Heidelberg (2005)
11. Gürgens, S., et al.: D05.1 Formal Models and Model Composition. ASSERT4SOA Project, Technical report (2011). http://assert4soa.eu/public-deliverables/
12. Gürgens, S., et al.: D05.3 Model Based Certification Artefacts. ASSERT4SOA Project, Technical report (2013). http://assert4soa.eu/public-deliverables/
13. IBM BPM industry packs. http://www.ibm.com/software/products/us/en/business-process-manager-industry-packs/
14. Jaeger, M.C., Rojec-Goldmann, G., Muhl, G.: QoS aggregation for web service composition using workflow patterns. In: Proceedings of the 8th IEEE International Enterprise Distributed Object Computing Conference, pp. 149–159 (2004)
15. Khan, K.M., Erradi, A., Alhazbi, S., Han, J.: Security oriented service composition: A framework. In: Proceedings of International Conference on Innovations in Information Technology (IIT), pp. 48–53 (2012)
16. Riabov, A.V., Liu, Z., Lelarge, M.: Automatic composition of secure workflows. In: Ungerer, T., Yang, L.T., Jin, H., Ma, J. (eds.) ATC 2006. LNCS, vol. 4158, pp. 322–331. Springer, Heidelberg (2006)
17. Majithia, S., Walker, D.W., Gray, W.A.: A framework for automated service composition in service-oriented architectures. In: Bussler, C.J., Davies, J., Fensel, D., Studer, R. (eds.) ESWS 2004. LNCS, vol. 3053, pp. 269–283. Springer, Heidelberg (2004)
18. Mantel, H.: On the composition of secure systems. In: Proceedings of the 2002 IEEE Symposium on Security and Privacy (SP2002). IEEE Computer Society (2002)
19. Medjahed, B., Bouguettaya, A., Elmagarmid, A.K.: Composing web services on the semantic web. VLDB J. **12**(4), 333–351 (2003)
20. Pino, L., Spanoudakis, G.: Constructing secure service compositions with patterns. In: Proceedings of 2012 IEEE 8th World Congress on Services, pp. 184–191 (2012)
21. Pino, L., et al.: D02.2 ASSERT aware service orchestration patterns. ASSERT4SOA Project, Technical report (2012). http://assert4soa.eu/public-deliverables/
22. Pino, L., Spanoudakis, G., Gürgens, S., Fuchs, A.: Discovering secure service compositions. In: Proceedings of the International Conference on Cloud Computing and Services Science (2014)
23. Ponnekanti, S.R., Fox, A.: Sword: a developer toolkit for web service composition. In: Proceedings of the 11th World Wide Web Conference, pp. 7–11 (2002)
24. Raman, B., et al.: The SAHARA model for service composition across multiple providers. In: Mattern, F., Naghshineh, M. (eds.) PERVASIVE 2002. LNCS, vol. 2414, pp. 1–14. Springer, Heidelberg (2002)
25. RosettaNet. Available: http://www.rosettanet.org/
26. Shirey, R.: Internet Security Glossary, Version 2. RFC 4949 (Informational), IETF (2007). Available: http://www.ietf.org/rfc/rfc4949.txt
27. Tan, W., Fan, Y., Zhou, M.: A petri net-based method for compatibility analysis and composition of web services in business process execution language. IEEE Trans. Autom. Sci. Eng. **6**(1), 94–106 (2009)
28. Zisman, A., Spanoudakis, G., Dooley, J., Siveroni, I.: Proactive and reactive runtime service discovery: A framework and its evaluation. IEEE Trans. Softw. Eng. **39**(7), 954–974 (2013)