Discovering Secure Service Compositions

Luca Pino[1], George Spanoudakis[1], Andreas Fuchs[2] and Sigrid Gürgens[2]
[1]*School of Informatics, City University London, London, U.K.*
[2]*Fraunhofer Institute for Secure Information Technology, Darmstadt, Germany*

Keywords: Software Services, Secure Service Compositions, Security Certificates.

Abstract: Security is an important concern for service based systems, i.e., systems that are composed of autonomous and distributed software services. This is because the overall security of such systems depends on the security of the individual services they deploy and, hence, it is difficult to assess especially in cases where the latter services must be discovered and composed dynamically. This paper presents a novel approach for discovering secure compositions of software services. This approach is based on secure service orchestration patterns, which have been proven to provide certain security properties and can, therefore, be used to generate service compositions that are guaranteed to satisfy these properties by construction. The paper lays the foundations of the secure service orchestration patterns, and presents an algorithm that uses the patterns to generate secure service compositions and a tool realising our entire approach.

1 INTRODUCTION

The security of service based systems (SBS), i.e., systems that make use of distributed and possibly dynamically assembled software services, has been a critical concern for both the users and providers of such systems (Raman et al., 2002; Majithia et al., 2004; Anisetti et al., 2013). 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 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 received considerable attention in the literature (e.g., Aggarwal et al., 2004; Dustdar et al., 2005; Tan et al. 2009; Alrifai et al., 2012)., the construction of secure service compositions is not adequately supported by existing research.

In this paper, we present an approach for discovering 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 security properties, if the constituent services of the orchestration satisfy other security properties. A SESO pattern specifies the order of the execution of its constituent services (e.g., sequential, parallel execution) and the data flows between them. It also specifies rules, which dictate the security properties that the constituent services of the orchestration must have for the orchestration to satisfy another security property as a whole. These rules express security property relations of the form *IF P THEN* $\wedge_{i=1,\dots,n} P_i$ where P is a security property that is required of the service orchestration as a whole and P_i are security properties of the constituent services, which must be satisfied for P to be guaranteed. The security property relations expressed by the rules are formally proven. The constituent services of a SESO

Pino L., Spanoudakis G., Fuchs A. and Gürgens S..
Discovering Secure Service Compositions.
DOI: 10.5220/0004855702420253
In *Proceedings of the 4th International Conference on Cloud Computing and Services Science* (CLOSER-2014), pages 242-253
ISBN: 978-989-758-019-2

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 discover 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 the orchestration specified by them. These criteria are used to drive a discovery process whose goal is to instantiate the pattern. If this discovery/pattern instantiation process is successful, i.e., different combinations of services that satisfy the required criteria and fit with the orchestration structure of the pattern can be discovered, any composition of services which is built from the pattern is guaranteed to have the required overall security property by-construction.

An earlier account of our approach has been given in (Pino and Spanoudakis 2012a; Pino and Spanoudakis 2012b). In this paper, we present the method that underpins the proof of security properties in SESO patterns, show examples of concrete proofs of security properties for specific SESO patterns, and present an amended version of the original composition algorithm that makes use of coarse-grained service workflows in the composition process in order to find service compositions that are not only secure but also functionally relevant to the service that is needed. In addition, we describe a tool that implements our approach.

The rest of this paper is organized as follows. Section 2 presents an overview of our approach. Section 3 discusses the validation of the security of primitive service orchestration patterns and provides examples of proofs of security properties for some of these patterns. Section 4 discusses the encoding of secure service orchestration patterns. Section 5 presents the new pattern driven secure service composition algorithm. Section 6 provides an overview of the tool that we have developed to implement our approach. Finally, Section 7 overviews related work and Section 8 provides conclusions and directions for future work.

2 OVERVIEW

The service composition approach that we present in this paper is part of a general framework developed at City University to support runtime service discovery (Zisman et al., 2013). 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 the latter can find. 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 (Zisman et al., 2013).

To take into account service security requirements as part of the service discovery process, we have extended the above framework in two ways: (i) we have extended SerDiQueL to enable the specification of the security properties that are required of individual services, as querying conditions, and (ii) we have developed a composition module supporting the construction of possible compositions of services that could replace a given service in an SBS in cases where a query cannot find any single replacement service, based on the approach that we present in this paper. A detailed description of the extended version of SerDiQueL (called *A-SerDiQueL*) that is used for (i) and (ii) is beyond the scope of this paper and can be found in (Spanoudakis et al., 2011). In this paper, we focus on the process of searching for and constructing secure service compositions. 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), logistics (RosettaNet), 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 flow between the workflow's composing activities.

To address (b), we are using the 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, ..., 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.

3 VALIDATING SECURE SERVICE COMPOSITIONS

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 orchestration engine, 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 (Gürgens et al., 2005b). In SeMF, a system specification is composed of a set $\mathbb{P}$ of agents and a set $\sum$ 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. A detailed account of SeMF is beyond the scope of this paper can be found in (Fuchs and Gürgens, 2011; Fuchs et al., 2011; Gürgens et al., 2005a; Gürgens et al., 2002) for.

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 used by the SESO patterns for instantiation towards concrete services and security properties. While our example is a very simple one, our approach can handle more complex service models, e.g. involving global agents (unique to all services), or service specific agents (e.g. backend storage) as well as various different orchestrations patterns, proving different instantiations of various security properties regarding integrity and confidentiality (Pino et al., 2012).

3.1 Formal Representation of Generic Service Composition

In the following, we denote the system model of the service S^0 to be replaced by a composition by Sys^0, the system models of the services S^1 and S^2 to be composed by Sys^1 and Sys^2, respectively, and the composition system by Sys^c. The sets of agents and actions are denoted analogously (i.e. by $\mathbb{P}^i$, $\sum^i$, for i=0, 1, 2). We then view the systems Sys^0, Sys^1 and Sys^2 as homomorphic images of the composed system Sys^c.

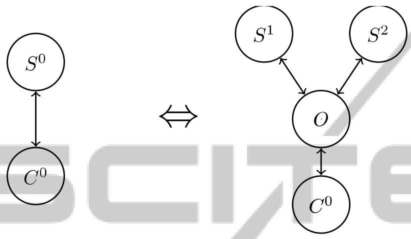


Figure 1: Service Composition.

The principal idea of substituting a service by a service composition is depicted in Figure 1: we assume services S^1 and S^2 to act independently of (i.e., not to invoke) each other. Thus we utilize an orchestration engine O for their composition that takes the role 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. We formalize this by using a generic renaming function $r_{P\to Q}: \sum \to \sum_{r_{P\to Q}}$ that replaces all occurrences of agent P in an action by Q. Based on this function, we define functions $r^i : \sum^i \to \sum^c$ (i = 0, 1, 2) as follows:

$$r^0(a) := r_{S^0\to O}(a) \quad \text{if } a \in \textstyle\sum^0_{/C^0} \cup \sum^0_{/S^0}$$

$$r^j(a) := r_{C^j\to O}(a) \quad \text{if } a \in \textstyle\sum^j_{/S^j} \cup \sum^j_{/C^j}$$

(j = 1, 2). The resulting set $\sum^c$ of actions of the composed system is then as follows:

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

$\sum^c_{/O}$ 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. 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 $\sum^c$ to $\sum^i$. Regarding the actions corresponding to those in $\sum^i$, 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, we define h^i : $\sum^c \to \sum^i$ as follows:

$$h^i(a) = \begin{cases} a' & \text{if } \exists a' \in \textstyle\sum^i : r^i(a') = a \\ \varepsilon & \text{else} \end{cases}$$

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 fulfils 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 (Gürgens et al., 2005a; Gürgens et al., 2002) 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

The actions of the systems are constructed from the service operations op_0, op_1, and op_2 as prefix, followed by one of the suffixes *IS*, *IR*, *OS*, *OR* to represent *InputSend*, *InputReceive*, *OutputSend*, *OutputReceive*, respectively. This results in the following agent and action sets:

$$\mathbb{P}^i \supseteq \{C^i, S^i\}, \textstyle\sum^i \supseteq \left\{\begin{matrix} op_i\text{–}IS(C^i, S^i, data_i), \\ op_i\text{–}IR(S^i, C^i, data_i), \\ op_i\text{–}OS(S^i, C^i, f_i(data_i)), \\ op_i\text{–}OR(C^i, S^i, f_i(data_i)) \end{matrix}\right\}$$

In our simple example of a sequential composition pattern, the orchestrator forwards $data_0$ received from C^0 to S^1 which returns $f_1(data_0)$. These data are then forwarded by the orchestrator to S^2 who returns $f_2(f_1(data_0))$ which the orchestrator finally returns to the client. 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-$IS(C^0, S^0, data_0)$ onto op_0-$IS(C^0, O, data_0)$, 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 $data_2 := f_1(data_1)$.

3.3 Validation of Integrity Preserving Compositions

Exemplarily, we will now prove that a specific data integrity property of S^0 is provided by the orchestration specified above. The definition of (data) integrity that we assume in our example is taken from RFC4949: "The property that data has not been changed, destroyed, or lost in an unauthorized or accidental manner." (Shirey, 2007). In SeMF, this property 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 (Fuchs and Gürgens, 2011).

Let us now assume that service S^0 provides the integrity property that whenever the client receives $f_0(data_0)$ from the service, the client has sent $data_0$ to this service before:

P1' $pre(op_0\text{-}IS(C^0, S^0, data_0), op_0\text{-}OR(C^0, S^0, f_0(data_0)))$

As explained above, precedence is preserved by h^0 (as constructed in Section 3.1). Hence the corresponding property of the composition is (assuming $f_0 = f_2 \circ f_1$):

P1 $pre(op_0\text{-}IS(C^0, O, data_0), op_0\text{-}OR(C^0, O, f_2(f_1(data_0))))$

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

P2' $pre(op_1\text{-}IS(C^1, S^1, data_1), op_1\text{-}OR(C^1, S^1, f_1(data_1)))$
P3' $pre(op_2\text{-}IS(C^2, S^2, data_2), op_2\text{-}OR(C^2, S^2, f_2(data_2)))$

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

P2 $pre(op_1\text{-}IS(O, S^1, data_0), op_1\text{-}OR(O, S^1, f_1(data_0)))$
P3 $pre(op_2\text{-}IS(O, S^2, f_1(data_0)), op_2\text{-}OR(O, S^2, f_2(f_1(data_0))))$

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

P4 $pre(op_0\text{-}IS(C^0, O, data), op_1\text{-}IS(O, S^1, data))$
P5 $pre(op_1\text{-}OR(O, S^1, data), op_2\text{-}IS(O, S^2, data))$
P6 $pre(op_2\text{-}OS(O, C^0, f_2(f_1(data_0))), op_2\text{-}OR(C^0, O, f_2(f_1(data_0))))$

Proof. By transitivity of precedence, from properties P2 to P6 we can conclude that property P1 holds.

The above proof is almost trivial but shows the principle of our approach. In (Pino et al., 2012) we have proven more complex integrity properties involving actions of global agents being invoked by either S^1 or S^2, as well as 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 service composition. It should also be noted that the proofs of security properties for specific SESO patterns need to be constructed offline and encoded in the patterns as rules, as we discuss in Sect. 4 below. At runtime, the rules encoded in specific pattern are used to deduce the security properties that need to be satisfied by the candidate services that can instantiate the pattern.

4 SECURE SERVICE ORCHESTRATION PATTERNS

Proofs of security properties, like the one that we discussed in Section 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

can be guaranteed if each S_i in P satisfies a set of security properties ρ_j ($j =1, \ldots, m_i$)".

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 (Drools). 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 actions of the rule. This execution can update the contents of the KB by adding or deleting facts in it. The reasoning process of Drools is based on the Rete algorithm a pattern-matching algorithm that is known to scale well for large sets of data facts and rules (Forgy, 1982);. The latter property of Drools is the main reason for selecting it to represent and reason with SESO patterns in our approach.

Table 1 shows the encoding of integrity in the sequential orchestration pattern that was presented in Section 3.3 as a Drools rule. In particular our rule uses the following definition of integrity:

Definition 2. *Integrity(S, x, y) = pre*(op_0*-IS*(C^0, *S*, *x*), op_0*-OR*(0^0, *S*, 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 3-5 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 `$S2`, is encountered. The rule defines the parameters of these activities: `$S1` has an input parameter `$d` and an output parameter `$f1d`, and `$S2` has an input parameter `$f1d` and an output parameter `$f2f1d`, as shown in Table 1. Line 6 describes the original security requirement requested on the composition pattern `$rhoP`, i.e. integrity on the pattern `$P` of its data `$d` and `$f2f1d`. This requirement is equivalent to the precedence property P1 presented in Section 3.3. Lines 8-9 (i.e., the `then` part of the rule) specify the security properties that are required of the activities of the pattern in order to guarantee `$rhoP`, namely: (i) integrity on the input (`$d`) and output (`$f1d`) of `$S1`, as stated by the precedence property P2, and (ii) integrity on the input (`$f1d`) and output (`$f2f1d`) of `$S2`, as required from P3. Additionally, we assume the framework executing the orchestration to satisfy properties P4–P6, 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 but cannot be discussed due to space limitations. 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.

Table 1: Integrity Rule for Sequential SESO Pattern.

```
 1:  rule "Integrity - Sequential Orchestration"
 2:   when
 3:     $S1 := Activity($d := inputs, $f1d := outputs)
 4:     $S2 := Activity($f1d := inputs, $f2f1d := outputs)
 5:     $P := Sequential($S1 := activ1, $S2 := activ2)

 6:     $rhoP : Integrity($P := subject, $d := inputs, $f2f1d := outputs)
 7:   then
 8:     insert(new Integrity($S1, $d, $f1d));
 9:     insert(new Integrity($S2, $f1d, $f2f1d));
10:     retract($rhoP);
11: end
```

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 its conditions.

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 (Carminati et al., 2006; Medjahed et al., 2003) and result from the generation of reference process models in specific domains as in (RosettaNet; IBM BPM Industry Packs). The activities of an abstract workflow are orchestrated through a process consisting of the primitive orchestrations that underpin the security patterns, as discussed in Section 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

Table 2: Service Composition Algorithm.

```
Require: $Q_S$ - query for the required service
Ensure: ResultSet - set of instantiated workflows
1:  procedure SERVICECOMPOSITION($Q_S$)
2:    for all abstract workflows AW in the repository do
3:      if STRUCTURALMATCH($Q_S$, AW) == true then
4:        Put a copy of AW in WStack
5:      end if
6:    end for
7:    while there are more workflows in WStack do
8:      Get the first workflow W in the WStack
9:      Pop the first unassigned activity A from W
10:     Extract the structural query $Q_A$ for A from W
11:     SecCond := SECURITYCONDITIONS($Q_S$, W)
12:     Add to $Q_A$ the security conditions SecCond
13:     Res := SERVICEDISCOVERY($Q_A$)
14:     for all services S* in Res do
15:       $W_{S*}$ := W[A/S*]              //i.e. substitute S* for A in W
16:       if exists an unassigned activity in $W_{S*}$ then
17:         Push $W_{S*}$ in WStack
18:       else
19:         Add $W_{S*}$ to ResultSet
20:       end if
21:     end for
22:   end while
23:   return ResultSet
24: end procedure
```

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 thanks to the fact that each workflow can be seen as a recursive application of primitive orchestrations.

Figure 2 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* (Figure 2 (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 Figure 2 (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* (Figure 2 (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.

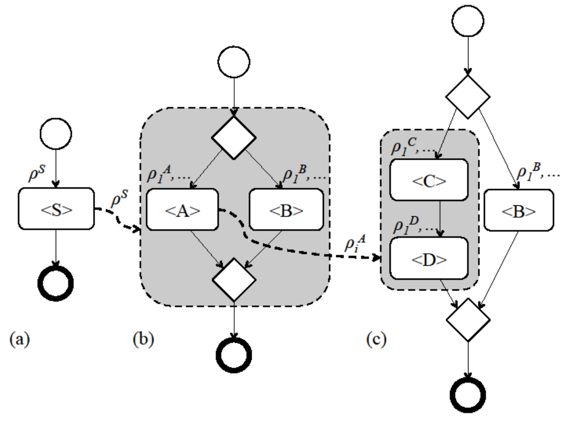


Figure 2: Recursive application of secure service orchestration patterns.

After constructing Q_A, the query is executed by the runtime discovery framework in (Zisman et al., 2012) to identify a list of candidate services for Q_A. The candidate services in this list (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 process in order to explore fully the instantiation of a particular activity within a pattern before considering other activities, as this is expected to spot dead-ends sooner than a breadth-first search.

5.1 Example

As an example of applying the algorithm in Table 2, consider a *Stock Broker* SBS that uses an operation *GetStockQuote* from a service *StockQuote* to obtain price quotations for given stocks. *GetStockQuote* takes as input a string *Symbol* identifying a stock and returns the current value of that stock in USD.

Suppose that the Stock Broker SBS has a security requirement regarding integrity of the input and output data of this operation, and would consider replacement services that can offer the same operation only if they have certificates confirming the satisfaction of this particular security requirement by the service. 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 of 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 an abstract workflow W_{SQ} shown in Figure 3 that matches the query. W_{SQ} contains three activities connected by two sequential patterns (see two dashed areas of 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 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.

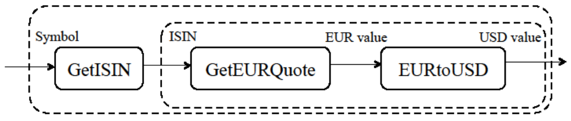


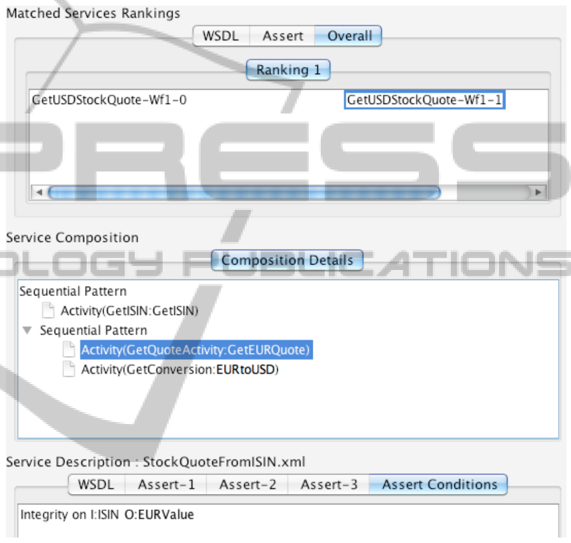
Figure 3: Abstract Workflow W_{SQ}.

The framework then infers the security properties required for each of the services that could instantiate the activities and uses them to query for such services. Initially, the rule shown in Table 1 is fired given the property required for the external sequential pattern, i.e. integrity on inputs and outputs of the workflow (i.e. *Symbol* and *USD value*). From the required security property, the rule derives two more properties: (1) integrity on inputs and outputs of *GetISIN* (i.e. *Symbol* and *ISIN*), and (2) integrity on inputs and outputs of the sequential inner pattern representing the second activity (i.e. *ISIN* and *USD value*). The second property fires again the rule and this propagates the requirement for integrity of the *ISIN* and *USD value*, resulting in the two properties: integrity on *GetEURQuote* of *ISIN* and *EUR value*, and integrity on *EURtoUSD* of *EUR value* and *USD value*.

After the application of the rules, we derive the required property for the first unassigned activity *GetISIN*, namely integrity of the input *Symbol* and the output *ISIN*. A query consisting of the interface and the security property required for *GetISIN* 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. In a similar way, a query specifying the required interface and security property of integrity is created for the second (*GetEURQuote*) and the last activity (*EURtoUSD*). Each query is 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 & 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 Section 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. Figure 4 shows the results of an execution in the case of the example in Section 5.1. These include two alternative service compositions; see *GetUSDStockQuote-Wf1-0* and *GetUSDStockQuote-Wf1-1* in the *Ranking-1* panel (the appearance of the two compositions in the same line in the panel indicates that there is no ranking between these two compositions). If one of these compositions is selected, details about the service operations that have instantiated the abstract workflow activities are shown in the *Composition Details* panel. In this case, the abstract workflow with the two nested sequences of activities has been instantiated by *sequential(GetISIN, sequential(GetEURQuote, EURtoUSD))*.



GetUSDStockQuote-Wf1-0 GetUSDStockQuote-Wf1-1

Sequential Pattern
- Activity(GetISIN:GetISIN)
- Sequential Pattern
  - Activity(GetQuoteActivity:GetEURQuote)
  - Activity(GetConversion:EURtoUSD)

Integrity on I:ISIN O:EURValue

Figure 4: Screenshot of Composition tool.

Then, by selecting an activity in the workflow, the details of the service instantiating the selected activity are shown. These can be the WSDL description, the required security properties that the patterns generated for the query that was used to identify the service, and the certificates that demonstrated the satisfaction of these properties during the composition process. The bottom part of Figure 4 shows the required security properties that were used in the query for the service *GetEURQuote*.

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 a discovery query. In the experiments, we used service registries of four sizes (150, 300, 600 and 1200), 25 abstract workflows and 3 patterns.

Table 3: Execution times (in milliseconds) w.r.t. service registry size and number of generated compositions.

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

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.

Whilst the benefits of the proactive approach have been shown in (Zisman et al., 2013) for the case of single service discovery, further experimentation is required to explore the same for the composition and assess the effect on performance of controls over the number of generated compositions.

7 RELATED WORK

The main focus of existing work in service composition is to address the problem of creating compositions that have certain functional and quality of service (QoS) property (Raman et al., 2002; Ponnekanti et al., 2002; Fujii et al., 2004; Majithia et al., 2004; Jaeger et al., 2004; Aggarwal et al., 2004; Dustdar et al., 2005; Tan et al. 2009; Alrifai et al., 2012). This work provides a foundation for functional and QoS properties but provides only basic support for addressing security properties in service composition, which is the main focus of our approach.

The problem of supporting security requirements (properties) in service composition has been a focus of work in the area of model based service composition. In this area, service compositions are modeled using formal languages and their required properties are expressed as properties on the model (Deubler et al., 2004; Dong et al., 2010; Bartoletti et al., 2005). Our approach to composition is also model based but uses model based property proofs to identify how overall security properties of compositions can be guaranteed through propagation to properties on the individual components (services) of the composition. Works in this field, however, provide proofs of additional security properties that could be used to extend the patterns used in our approach, even if they use different formalisms. An example of such proofs is given in (Mantel, 2002), which presents compositionality results related to information flows (e.g. non-interference) and that can be easily converted into SESO patterns and inference rules in our framework.

Another strand of work on automatic service composition focuses on discovering services that can guarantee given security properties (Carminati et al., 2006; Medjahed et al., 2003; Lelarge et al., 2006; Anisetti et al., 2013; Khan et al., 2012). Some of these approaches focus on specific types of security properties (Medjahed et al., 2003; Lelarge et al., 2006), whilst others (Carminati et al., 2006; Anisetti et al., 2013; Khan et al., 2012) focus on how to express and check security properties only for single partner services of a composition. In contrast, our approach can support arbitrary security properties and properties of entire service compositions.

The approaches in (Medjahed et al., 2003) and in (Khan et al., 2012) describe two ontology-based frameworks for automatic composition. The former work defines a set of metrics for selecting amongst different compositions but provides limited support for security. The latter work introduces hierarchies of security properties and mentions the possibility of using rules to reason about them but does not support the construction of secure service compositions. Lelarge et al., (2006) use planning techniques to build sequential compositions that guarantee the adoption of access control models. Carminati et al., (2006) introduce an approach to security aware service composition that matches security requirements with the external service properties. The approach presented in (Anisetti et al., 2013) focuses on the generation of test-based virtual security certificates for service compositions derived from the test-based security certificates of the external services part of the composition. The service compositions are based on templates that allow expressing security requirements on the external services. The ideas underlining this approach can be used to extend the one presented in this paper to support the generation of virtual certificates for compositions.

The secure orchestration patterns that we use in our framework are similar to the workflow patterns in (Van Der Aalst et al., 2003), as they specify elementary workflows used to build compositions. Our patterns, however, include information not only about the control flow within the pattern but also about the data flow. They also extend these patterns with information regarding security properties to hold for the individual services in order to guarantee that their composition satisfies a required security property.

8 CONCLUSIONS

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 implications between the security properties of the individual placeholders and the security property of the orchestration as a whole. The formal proofs (and patterns) achieved so far cover different integrity and confidentiality properties for various forms of primitive orchestrations. The extension of our approach to cover other security properties (e.g., availability) is subject of ongoing work. During the composition process, the proven implications are used to deduce the actual properties that should be required of the individual services that may instantiate an orchestration for the orchestration as a whole to satisfy specific security properties.

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 has been described in (Zisman et al., 2012). We are currently investigating the validity of our approach through a series of focus group evaluations. We are also conducting further performance and scalability analysis of our prototype, focusing on exploring the effect of a proactive composition generation approach and setting heuristic controls over the number of compositions generated by the algorithm.

ACKNOWLEDGEMENTS

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

REFERENCES

Aggarwal, R., Verma, K., Miller, J., and Milnor, W., 2004. Constraint driven web service composition in METEOR-S. In *Proc. of the IEEE International Conference on Services Computing, (SCC 2004)*, pp. 23-30.

Alrifai, M., Risse, T., and Nejdl, W., 2012. A hybrid approach for efficient Web service composition with end-to-end QoS constraints. In *ACM Transactions on the Web (TWEB)*, vol. 6, no. 2, Article 7.

Anisetti, M., Ardagna, C., and Damiani, E., 2013. Security Certification of Composite Services: A Test-Based Approach. In *Proc. of the IEEE 20th International Conference on Web Services (ICWS)*, pp. 475-482.

Bartoletti, M., Degano, P. and Ferrari, G. L., 2005. Enforcing secure service composition. In *Proc. 18th Comp. Sec. Found. Workshop (CSFW)*. IEEE Comp. Soc., pp. 211-223.

Carminati, B., Ferrari, E. and Hung, P. C. K., 2006. Security conscious web service composition. In *Proc. of the Int. Conf. on Web Serv. (ICWS)*. IEEE Comp. Soc., 489-496.

Deubler, M., Grünbauer, J., Jürjens, J. and Wimmel, G., 2004. Sound development of secure service-based systems. In *Proc. of 2nd International Conference on Service Oriented Computing (ICSOC)*. ACM, pp. 115-124.

Dong, J., Peng, T. and Zhao, Y., 2010. Automated verification of security pattern compositions. *Inf. Softw. Technol.*, vol. 52, no. 3, pp. 274-295.

Drools. [Online]. Available: http://www.jboss.org/drools/

Dustdar, S., and Schreiner, W., 2005. A survey on web services composition. *International Journal of Web and Grid Services*, vol. 1, no. 1, pp. 1-30.

Forgy, C., 1982. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligences*, vol. 19, no. 1, pp. 17-37.

Fuchs, A. and Gürgens, S., 2011. D05.1 Formal Models and Model Composition. ASSERT4SOA Project, Tech. Rep. [Online]. Available: http:// assert4soa.eu/public-deliverables/

Fuchs, A., Gürgens, S. and Rudolph, C., 2011. Formal Notions of Trust and Confidentiality - Enabling Reasoning about System Security. *Journal of Information Processing*, vol. 19, pp. 274-291.

Fujii, K., and Suda, T., 2004. Dynamic service composition using semantic information. In *Proc. of the 2nd international conference on Service oriented computing (ICSOC)*, pp. 39-48. ACM.

Gürgens, S., Ochsenschläger, P. and Rudolph, C., 2002. Authenticity and provability - a formal framework. In *Infrastr. Sec. Conf. (InfraSec)*. LNCS, vol. 2437, SV, pp. 227–245.

Gürgens, S., Ochsenschläger, P. and Rudolph, C., 2005a. Abstractions preserving parameter confidentiality. In *Europ. Symp. On Research in Computer Security (ESORICS)*. 418–437.

Gürgens, S., Ochsenschläger, P. and Rudolph, C., 2005b. On a formal framework for security properties. *International Comp. Standards & Interface Journal (CSI), Special issue on formal methods, techniques and tools for secure and reliable app.* 27(5) 457–466.

IBM BPM industry packs. [Online]. Available: http:// www-03.ibm.com/software/products/us/en/ business-process-manager-industry-packs/

Jaeger, M. C., Rojec-Goldmann, G., and Muhl, G., 2004. QoS aggregation for web service composition using workflow patterns. In *Proc. of the 8th IEEE International Enterprise distributed object computing conference, (EDOC 2004)*, pp. 149-159.

Khan, K. M., Erradi, A., Alhazbi, S. and Han, J., 2012. Security oriented service composition: A framework. In *Proc. of International Conference on Innovations in Information Technology (IIT)*, pp. 48-53.

Lelarge, M., Liu, Z. and Riabov, A.V., 2006. Automatic composition of secure workflows. In *Proc. of the Third international conference on Autonomic and Trusted Computing, (ATC)*. Berlin, SV, pp. 322-331.

Majithia, S., Walker, D. W., and Gray, W. A., 2004. A framework for automated service composition in service-oriented architectures. In *Proc. of the 1st European Semantic Web Symposium*, Lecture Notes in Computer Science, vol. 3053, pp. 269-283.

Mantel, H., 2002. On the Composition of Secure Systems. In *Proc. of the 2002 IEEE Symposium on Security and Privacy (SP2002)*. IEEE Computer Society, Washington, DC, USA, 88-.

Medjahed, B., Bouguettaya, A. and Elmagarmid, A.K., 2003. Composing web services on the semantic web. *The VLDB Journal*, vol. 12, no. 4, pp. 333-351.

Pino, L. and Spanoudakis, G., 2012a. Constructing secure service compositions with patterns. In *Services (SERVICES), 2012 IEEE Eighth World Congress on*. IEEE, pp. 184-191.

Pino, L. and Spanoudakis, G., 2012b. Finding secure compositions of software services: Towards a pattern based approach. In *5th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. IEEE, pp. 1-5.

Pino, L., Spanoudakis, G., Gürgens, S., Fuchs, A. and Mahbub, K., 2012. D02.2 ASSERT aware service orchestration patterns. ASSERT4SOA Project, Tech. Rep. [Online]. Available: http://assert4soa.eu/public-deliverables/

Ponnekanti, S. R., and Fox, A., 2002. Sword: A developer toolkit for web service composition. In *Proc. of the 11th World Wide Web Conference (Web Engineering Track)*, pp. 7-11.

Raman, B., Agarwal, S., Chen, Y., Caesar, M., Cui, W., Johansson, P., ... and Stoica, I., 2002. The SAHARA model for service composition across multiple providers. In *Proceedings of the First International Conference on Pervasive Computing*, Lecture Notes in Computer Science, vol. 2414, pp. 1-14.

RosettaNet. [Online]. Available: http:// www.rosettanet.org/

Shirey, R., 2007. Internet Security Glossary, Version 2. RFC 4949 (Informational), IETF. [Online]. Available: http://www.ietf.org/rfc/rfc4949.txt.

Spanoudakis, G., Mahbub, K., Pino, L., Foster, H., Maña, A. and Pujol, G., 2011. D02.1 ASSERTs aware service query language and discovery engine. ASSERT4SOA Project, Tech. Rep. [Online]. Available: http://assert4soa.eu/public-deliverables/

Tan, W., Fan, Y., and Zhou, M., 2009. A Petri Net-Based Method for Compatibility Analysis and Composition of Web Services in Business Process Execution Language. In *IEEE Transactions on Automation Science and Engineering*, vol.6, no.1, pp.94-106.

Van Der Aalst, W. M. P., Ter Hofstede, A. H. M., Kiepuszewski, B. and Barros, A.P., 2003. Workflow patterns. *Distrib. Parallel Databases*, vol. 14, no. 1, pp. 5-51.

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*, http:// doi.ieeecomputersociety.org/10.1109/TSE.2012.84, Dec 2012.