Semantic Model Checking Security Requirements for Web Services

L. Boaro, E. Glorio, F. Pagliarecci, L. Spalazzi
Università Politecnica delle Marche,
I-60131 Ancona, Italy
{l.boaro, e.glorio}@diiga.univpm.it, {pagliarecci, spalazzi}@diiga.univpm.it

ABSTRACT

Model checking is a formal verification method widely accepted in the web service world because of its capability to reason about service behaviors, at their process-level. It has been used as basic tool in several scenarios as service selection, service validation, and service composition. Furthermore, it has been widely applied to problems of security verification. The importance of semantics is widely recognized, as well. Indeed, there exists several solutions to the problem of providing semantics to web services, most of them relies on some form of Description Logic. This paper presents an integration of model checking and semantic reasoning technologies in an efficient way. This can be considered the first step towards the usage of semantic model checking in problems of security verification. The approach relies on a representation of services at the process level that is based on semantically annotated state transition systems (ASTS) and a representation of specifications that is based on a semantically annotated version of the computation tree logic (AnCTL). The proposed approach permit to perform an efficient and yet useful, semantic reasoning at process-level about web services. Indeed, this paper proves that model checking for ASTS and AnCTL is sound and complete and it can be accomplished in polynomial time.

KEYWORDS: Semantic Web Service, State Transition System, Temporal Description Logic, Model Checking.

1. INTRODUCTION

Web service and semantic web service are an emerging area. Nevertheless, security concerns limit in some way their diffusion. On the other hand, it is possible to exploit the work on security applied to different domains in order to mitigate security problems for web services. In our opinion a great help can arrive from formal methods, specially model checking. Indeed, in the past, they have been successfully applied to security (e.g., see [1, 2]). As a consequence, this paper presents a formal verification method that can be applied to semantic web service. Our approach is based on the notion of semantic model checking that have been already applied to verification, selection, and composition of semantic web services [3, 4]. The key idea of the approach proposed in this paper is to keep the procedural description of web services separated from their ontological descriptions, and to link the two through semantic annotations. The main characteristics of this approach can be summarized as follows:

• The concepts that are needed to provide a semantic description of web services form an ontology and, thus, are formalized in a language that belongs to the Description Logic family [5]. As a consequence, the security concepts are represented in the ontology, as well. The opportunity of defining an appropriate security ontology has been investigated by several authors (e.g., see [6, 7, 8]).
• The procedural behavior of a web service is formally modeled as a State Transition System (STS) [9]. The semantics of exchanged data is described as state annotation according to the Description Logic. The resulting model is called annotated state transition system (ASTS) [3]. In our approach, we suppose the existence of a trusted third party (that according to Carminati et al. [6] we call Secure Capability Authority or SCA) that evaluates security capabilities and behaviors of a given web service generating signed semantic annotations certifying such capabilities and behaviors.
• The requirements on a web service, and thus the security requirements as well, can be classified in static and dynamic constraints [6]. Static constraints refer to requirements that can be evaluated without taking into account the behavior of a service. Dynamic constraints refer to requirements that can be evaluated only taking into account the behavior of the service. In our approach, both the kinds of requirements are expressed in a temporal logic (Computation Tree Logic or CTL) [10] enriched with semantic anno-
The resulting language is called annotated computation tree logic (AnCTL) and belongs to the Temporal Description Logic family [11].

- The semantic model checking algorithm works in two steps: The first step (called grounding) takes into account the semantic aspects, in other words it is produced a ground STS where each state contains all and only all the description logic assertions that are needed. The second step takes into account the behavioral aspects, in other words the ground STS is checked by a traditional propositional temporal model checking algorithm (e.g. [12]).

The paper is structured as follows: the related work is presented in Section 2. In Section 3 we introduce a semantic model for representing semantic web services and their security requirements. Section 4 briefly introduce a case study. Some conclusions are reported in Section 5.

2. RELATED WORK

There are several works on model checking security specifications of a system: a protocol [1], a software [2], or a web service [13, 14]. All the aforementioned works have proved that this approach is effective by means of discovering several flaws that made systems vulnerable. On the other hand, there are several works on web service security based on appropriate security ontologies [6, 7, 8]. Nevertheless, as far as we know, an ontology based model checking has not yet been proposed. The advantages of this approach rely on the fact that is possible to verify whether the behavior of a given web service satisfies given security requirements on the base of a semantic description of such behavior provided by a trusted third party (a sort of certification authority).

Concerning the formal aspects of the approach propose din this paper, notice that there are several attempts of integrating logics for representing semantics and temporal logics. One of the most promising approaches consists of the so called temporal description logic, i.e. the integration in a unique framework of a description logic and a temporal logic [11]. Unfortunately, the result is a logic that is untractable or even undecidable [11]. On the other hand, model checking is a very efficient alternative to theorem proving when the process can be represented by a finite state machine [10]. This is becoming true for temporal description logic, as well. Even if this research line is still young, there are some works on this direction [15, 16, 17] that proposed very efficient algorithms for temporal description logic model checking.

Finally, concerning web service representation, there are several works focused on describing web services at the process-level. Consider, for instance, the standard languages for describing business processes, like BPEL [18], and the most popular standards for semantic web services, like OWL-S [19] and WSML [20]. In a process-level description, a web service is not simply represented as an “atomic” component - with its inputs, outputs, preconditions, and effects - that can be executed in a single step. Instead, the interface of the service describes its behavior, i.e., a process that interacts with other services in different steps, and which can have different control constructs, e.g., sequence, condition, and iteration. These information on the process model can be exploited to perform sophisticated kind of automated verification, discovery, selection, and composition of services and to express requirements at process level [21, 22, 23, 9, 15].

3. THE PROPOSED APPROACH

The approach proposed in this paper lies on the following four elements:

Security Ontology: it consists in providing all the terms that can be used in the other elements and are relevant for the application domain to be modeled. The ontology should be a unique commonly accepted formalization of the given domain. This assumption can be considered reasonable in the security domain for two reasons: First, international organizations are defining several standards in the security domain. SAML [24] is an example of language that can be used to semantically annotate a web service. Second, in our approach, web services must be semantically annotated by the SCA and thus they are annotated according to a unique ontology. Concerning the representation language, following the mainstream, in the proposed framework an ontology can be described by means of a language belonging to the Description Logic family [5]. Roughly speaking, according to the description logic, a knowledge base is formed by two components: TBox (or Terminology) and ABox. Intuitively a terminology (T) is a set of description logic axioms and describes the ontology. Notice that, in the approach proposed in this paper, the TBox does not vary over the time (e.g., see [11]). Concerning an ABox, intuitively it may represent the data of a knowledge base. In this work, an ABox represents the state of a process, i.e. a set of assertions that describe in which situation the process is. In other words, it is a temporal ABox (according to the definition in [11]), i.e. an ABox that varies over the time.

Web Service Representation A web service can be characterized in terms of its state and its behavior. The state can be represented by means of a set of description logic assertions (the so called ABox). The behavioral aspects of a service, i.e. how its state changes, can be represented by means of a state transition system. Sometimes, state transitions are labelled with the actions that make evolve the system; these are called labelled state transition systems. Combining all the above aspects, a service can be modeled by means of Annotated State Transition Systems (ASTS).
Roughly speaking, an annotated state transition system is a state transition system (STS) where each state is a set of description logic assertions, i.e., an ABox for each state. Lutz et al. [11] call this notion “temporal ABox”. In an ASTS, actions are distinguished in input actions, output actions, and $\tau$. Input actions represent the reception of messages, output actions represent messages sent to external services, and $\tau$ is a special action, called internal action. In other words, $\tau$ represents the fact that the state of the system can evolve without producing any output, and without consuming any input (this is a consequence of the fact that usually an external user does not have the possibility to know what are the internal actions, i.e. to her/him the internal actions are “opaque”). A transition relation describes how the state can evolve on the basis of inputs, outputs, or of the internal action $\tau$. Furthermore, each action can be viewed as a transition from a state consisting in an ABox in a different state consisting in a different ABox. In this work, it is assumed that infinite loops of $\tau$ transitions cannot appear in the system. Concerning finite loops, actions that must be iterated can be annotated with invariants, i.e., assertions that do not change throughout all the actions in the loop until the loop terminates.

**Definition 1 (Annotated State Transition System)** An annotated state transition system $\Sigma^A$ defined over a state transition system $\Sigma$ is a tuple $(\Sigma, T, A)$ where:

- $\Sigma = \langle S, S^0, I, O, R \rangle$ is the state transition system,
- $S$ is the finite set of states,
- $S^0 \subseteq S$ is the set of initial states,
- $I$ is the finite set of input actions,
- $O$ is the finite set of output actions,
- $R \subseteq S \times (I \cup O \cup \{\tau\}) \times S$ is the transition relation,
- $T$ is the terminology (TBox) of the annotation,
- $A : S \rightarrow 2^{Ax}$ is the annotation function, where $Ax$ is the set of all the concept assertions and role assertions defined over $T$.

**Specification Representation** Security as well as selection, validation, and composition problems have in common the need of verifying certain conditions over the behavior and the semantics of a given service. Several approaches have been proposed to represent service specifications. Usually, they are based on the notion of preconditions and effects of a service or on the notion of security capabilities. The approach followed in this work is based on the possibility of representing properties while the service is in execution, in other words taking into account the (partially specified) behavior of a service. From a theoretical point of view, a partial representation of the behavior of a web service can be modeled as temporal properties (for example: “a credit card number can not be requested before the customer confirmation of the order”). Therefore, service specifications are represented by means of an annotated temporal logic: the Annotated Computation Tree Logic (Annotated CTL). Basically, an Annotated CTL formula is a Computation Tree Logic CTL expression [10] where one has description logic assertions instead of propositions. Let us start with the notion of conjunctive query over a description logic as defined in [25]. Intuitively, it can be considered as a (normalized) propositional condition where each proposition is a description logic assertion.

**Definition 2 (Conjunctive Query)** A conjunctive query $q$ over $(T, A(s))$ is a set of atoms $\{p_1(\bar{q}_1), \ldots, p_n(\bar{q}_n)\}$ where each $\bar{q}_i$ is either $x_i$ or the pair $x_{i,1}, x_{i,2}$, $x_i$, and $x_{i,2}$ are variables or individuals, and thus each $p_i(\bar{q}_i)$ is either $p_i(x_i)$ or $p_i(x_{i,1}, x_{i,2})$ and $\bar{q}_i$ is a tuple of variables or individuals:

$p_i(\bar{q}_i)$ denotes a concept or role assertion, i.e. $p_1(x_i) \equiv x_i : C_i$ and $p_i(x_{i,1}, x_{i,2}) \equiv x_{i,1} \sqsubseteq x_{i,2}$

$V(q)$ denotes the set of variables of $q$ and $C(q)$ denotes the set of individuals of $q$. Therefore, $VC(q) = V(q) \cup C(q)$ denotes the set of variables and individuals of $q$. When $V(q) = \emptyset$, $q$ is a ground conjunctive query, i.e. each
$x_1$, $x_{i,1}$, or $x_{i,2}$ is an individual. A concept assertion in a propositional condition intuitively denotes a typical description logic problem: the retrieval inference problem. Let $x : C$ be a concept assertion, for a given ABox $\Lambda(s)$, the retrieval inference problem is the problem of finding all the individuals mentioned in the ABox that are an instance of the concept $C$ w.r.t. the given TBox $T$. A non-optimized algorithm for a retrieval can be realized by testing for each individual $a$ occurring in the ABox whether it is an instance of the concept $C$ (i.e., $\langle T, \Lambda(s) \rangle \models a : C$). Once it has been retrieved a set of instances $\{a\}$ for the concept assertion $x : C$, it is possible to substitute $x$ in the propositional condition with the retrieved instances and check whether the condition holds. Therefore, a conjunctive query denotes in fact a set of tuples (i.e., a table as in relational databases). A ground conjunctive query returns either one tuple containing such individuals (i.e. a table with one row), when the propositional condition hold, or an empty set (i.e. an empty table), otherwise.

An Annotated Computation Tree Logic (AnCTL) formula is a CTL formula annotated with assertions containing individuals.

**Definition 3 (Annotated Computation Tree Logic)**

An Annotated Computation Tree Logic formula $\phi(p_1, \ldots, p_m)$ over $(\Sigma, T, \Lambda)$ is a formula defined over the set of concept and role assertions $\{p_1, \ldots, p_m\}$ as follows:

$\phi = p_i | \phi \land \phi | \phi \lor \phi | \neg \phi | \phi \rightarrow \phi | A\phi | AG\phi | EF\phi | EG\phi | AX\phi | EX\phi | A(\phi \land \phi) | E(\phi \land \phi) | A(\phi R \phi) | E(\phi R \phi)$

where either $p_i = a_i : C_i$ or $p_i = a_i . R_i = b_i$ with $a_i, b_i$ that are individuals.

A temporal specification for an ASTS is an AnCTL formula containing conjunctive queries, as defined in the following:

**Definition 4 (Temporal Specification)** A Temporal Specification over $(\Sigma, T, \Lambda)$ is an AnCTL formula $\phi(q_1, \ldots, q_m)$ where $\{q_1, \ldots, q_m\}$ is a set of conjunctive queries.

The definition of $V$ is extended to temporal specifications as follows: $V(\phi(q_1, \ldots, q_m)) = V(q_1) \cup V(q_2) \cup \ldots \cup V(q_m)$. The definition of $C$ and $VC$ can be extended in a similar way. When a temporal specification does not have variables (i.e., $V(\phi(q_1, \ldots, q_m)) = \emptyset$) it is an AnCTL formula. The semantics of a temporal specification is defined in three steps:

- first, semantics of a conjunctive query [25];
- second, semantics of an AnCTL formula, extending the semantics of CTL [10];
- semantics of a temporal specification with variables.

For the sake of space, it is not possible to provide the formal definition of the such semantics; the reader can refer to [3, 16] for further details.

**Model checking** In principle, semantic annotations in a specification (i.e., in an AnCTL) may denote an infinite set of individuals. On the other hand, in general, model checking infinite state transition systems is undecidable. Nevertheless, in verification problems, it is not necessary to take into account all the individuals, but only the individuals relevant for the specification. As a consequence, it is needed a procedure by which the semantic annotations are “lowered” to a purely syntactic form; roughly speaking, concepts and role assertions are transformed into a finite set of Boolean propositions. This is the *grounding* procedure [16]. From a technical point of view: each annotated state in an ASTS is an ABox, i.e. a knowledge base modeling the assertions that hold in that state; each AnCTL is a set of conjunctive queries over such ABoxes [25]. The grounding algorithm is applied both to an ASTS and an AnCTL. Produces a finite STS and a set of propositional CTL specifications. Finally, the grounding algorithm is assumed to operate under the closed world assumption. From a practical point of view, this means that the semantic annotation must contain all it is needed to describe the effects of each service activity. From a technical point of view, this means that all the ground assertions that may be generated are mapped into Boolean propositions; for each state (say $s$), for each ground assertion that the algorithm for conjunctive query answering returns the corresponding Boolean variable (say $x$) must be set to true. All the other Boolean variables (say $y$) must be set to false. After that, a propositional temporal logic model checking algorithm can be applied to verify whether the finite STS produced by the grounder satisfies the set of propositional CTL specifications produced by the grounder as well. If not, a counter-example is provided.

The resulting semantic model checking algorithm can be proved to be sound and complete; this is a consequence of the closed world assumption and relies on the soundness and completeness of the conjunctive query answering, instance checking, and the model checking. The complexity of such algorithm is polynomial in the size and number of states, in the size of the temporal specification, and in the complexity of conjunctive query answering [16]. The complexity of conjunctive query answering ranges from polynomial to coNP-complete for the most common languages for ontologies [25]. This means that with a minimalistic approach as proposed in this paper (in Figure 1 we used DL-Lite as description logic), the semantic model checking is PTIME-complete. For the experiments proposed in this work, it has been used NuSMV[12], a well-known state-of-the-art model checker.

**4. THE CASE STUDY**

Let us consider as case study the problem of verifying a given security requirement over web services provided by
two e-commerce companies. 
At the two provider sites, both the companies deploy services that allow users to search for an item to buy. The two web services provide several search mechanisms for an item, but for space’s sake, here it is considered the keyword-based search only. In case the user found the item she/he was looking for at a convenient price, she/he may confirm the purchase order and proceed with a checkout phase upon authentication. The authentication and the checkout (that has a payment phase) can be accomplished in different ways. Concerning the web service of the first company (say Company A, see Figure at http://leibniz.diiga.univpm.it/ spalazzi/SWS/CompanyA.jpg): the authentication exploit the X.509 framework and can not fail more than three times; the payment must be performed by means of PayPal. Concerning the web service of the second company (say Company B, see Figure at http://leibniz.diiga.univpm.it/ spalazzi/SWS/CompanyB.jpg): the authentication exploit the X.509 framework as well, and can not fail more than four times; the payment can be performed by means of both PayPal and credit card. At the requester site, let us suppose a user that needs to select a service (e.g. for composing a new secure service) satisfying the following requirement: 

Requirement 1 The web service must require a payment only after a search has been successfully performed and the payment can be made only by means of an Internet based payment system\(^1\). The authentication must be performed by means of X.509 framework and can not fail more than three times.

Notice that this requirement has static conditions (the payment method, and the authentication method) as well as dynamic conditions (the payment after the search, how many times an authentication can fail). First of all, how can dynamic conditions be expressed? A condition based on input, output, pre-conditions, and effects (i.e. the information that traditionally we have about a service) can not represent the requirement that the payment must have only one possibility after the search response or that the authentication can have only three attempts. Furthermore, a specification purely based on the behavior can not work without semantics. Indeed, how can the requirements on the kind of payment or authentication methods be expressed? Can this information be derived from data types? Generally speaking, the answer is no. In this example, in the web service of Company B, the same messages can be used for both PayPal and credit card payment methods, creating an ambiguity (e.g. see in Figure at http://leibniz.diiga.univpm.it/ spalazzi/SWS/CompanyB.jpg the message payMe). The usage of semantic annotations related to a given security ontology for e-commerce services allows user to clearly define this kind of information.

Ontology Representation. Part of the ontology for this case study is depicted in Figure 1. This ontology contains a part that is general for the secure service domain (Authentication, EncryptedMessage). It also contains other concepts that are specific of the virtual store domain that must be examined (e.g., Item, Search, Payment). Notice that, the definition of Search and Authentication contain the role result whose values are restricted to be instances of the concept Result. This role captures what is the result of a given operation. Indeed, when an authentication succeeds (or an item requested by a given user is available), the status assumes the value success; when the authentication failed (or the item is not available), the status assumes the value failure. The possible values (instances) for the concept Result are listed in the ABox of Figure 2. Furthermore, Authentication contains the role noOfAttempts that reports the number of attempts in the authentication process. Service Process Representation Their formal definition consists of two ASTSs, both of them with the terminology depicted in Figure 1 as \(T\).

For instance, in the definition of the web service of Company A, the set of states \(S\) is the following:

\[
S = \{ \\
\text{START}, \text{sequence}, \text{searchRequest}, \text{search}, \text{ifAct}, \text{sequence2}, \text{searchResult}, \text{checkout}, \text{checkout2}, \text{authenticationRequest}, \text{authentication}, \text{ifAct2}, \text{sequence7}, \text{authenticationResponse}, \text{authenticationRequest1}, \text{authentication2}, \text{ifAct3}, \text{sequence8}, \text{authenticationResponse1}, \text{authenticationRequest2}, \text{authentication3}, \text{ifAct4}, \text{sequence9}, \text{authenticationResponse2}, \text{authenticationRequest3}, \text{authentication4}, \text{ifAct5}, \text{sequence10}, \text{authenticationError}, \text{sequence11}, \text{authenticationSuccess}, \text{payMe}, \text{pick}, \text{sequence3}, \text{checkoutResponse}, \text{sequence4}, \text{checkoutResponse2}, \text{sequence5}, \text{searchError}, \text{END} \\
\}
\]

where \(S^0 = \{\text{START}\}\).

According to Definition 1, input actions \(I\) model all the incoming requests to the process and the information they bring whereas output actions \(O\) represent outgoing messages. The action \(r\) is used to model internal evolutions of the process, such as assignments and decision making. Such actions, in the web service of Company A, are the following:

\[
I = \{ \\
\text{search (string)}; \text{authenticationRequest (string)}; \text{checkout (abstract, string)}; \text{payMeFault (string, string)}; \text{payMeResponse (string)}; \\
\}
\]

\(^1\)An Internet-based payment service links by means of Internet a customer computer with a third party processor computer that is linked with a merchant computer, and indicates a request to pay for a purchase directly and not with a credit card.
\[
O = \{
\text{searchError (string, string);} \\
\text{searchResponse (string, string, string);} \\
\text{authenticationError (string);} \\
\text{authenticationResponse (string);} \\
\text{payMe (string, abstract);} \\
\text{checkout (abstract, string);} \\
\}
\]

For example, authenticationRequest is intuitively used for receiving a request of being authenticated and authenticationResponse for sending the result of such an authentication. The actions are used to describe the transition relation, in other words how the system evolves. For the sake of space, here it is reported only a fragment of the transition relation; namely the fragment concerning the payment from state authenticationRequest and the next ones:

\[
R = \{ \ldots \\
\text{authenticationRequest} \xrightarrow{\text{authenticationResponse(user)}} \text{authentication;} \\
\text{authentication} \xrightarrow{\text{ifAct2}} \text{authenticationRequest1;} \\
\text{ifAct2} \xrightarrow{\text{sequence11}} \text{authenticationSuccess;} \\
\text{authenticationSuccess} \xrightarrow{\text{authenticationResponse}} \text{payMe;} \\
\text{ifAct2} \xrightarrow{\text{sequence7}} \text{authenticationResponse;} \\
\text{sequence7} \xrightarrow{\text{authenticationResponse}} \text{authenticationRequest1;} \\
\ldots \}
\]

Notice that this fragment has \(\tau\) actions, an input action (authenticationRequest) and output actions (authenticationResponse and authenticationError).

The \(\Lambda\) function contains, for each state, the set of global assertions (included the concept assertions declared in the ontology) and the assertions, if any, associated to that state. For the sake of space, here it is reported the \(\Lambda\) function for only two annotated states, namely authenticationSuccess and authenticationResponse.

\[
\Lambda(\text{authenticationSuccess}) = \{
\}
\]

\[
\Lambda(\text{authenticationResponse}) = \{
\}
\]

The last two assertions in authenticationSuccess and the last three assertions in authenticationResponse have been added during the annotation phase, whereas the other ones derive directly from the ontology and are present in every state, since they are global assertions (see Figure 2). Notice that, the semantic annotations that have to be added to this purpose are very limited if compared to processes defined in languages such as OWL-S. Nevertheless, they are sufficient for effective automated verification.

Selection Specification Representation Let us consider Requirement 1. This requirement has dynamic constraints that require not to try the authentication process more than three times. This means having the notion of time or sequence of actions to express the concept that something must happens in the future “after that” something else occurred. Second, it presents a static constraints that requires the usage of a given authentication framework. The use of semantics eliminates possible misunderstanding about the terms used. As long as one refers to the ontology used by a given Secure Capability Authority, the meaning of “Search”, “X509”, and “InternetPayment” is clear. Requirement 1 can thus be expressed by the following two annotated temporal specifications:

\[
\text{AG } (x.\text{searchResult=success} \rightarrow (1)) \\
\text{AF } (y:x509 \rightarrow \text{AF}(z:InternetPayment)) (2)
\]

Semantic Model Checking The experimental evaluation has been conducted using the so-called minimalistic approach: DL-Lite as description logic and CTL as temporal logic. The semantic model checking algorithm [16] has been implemented in Java and it is based on Pellet [26], for the conjunctive query answering, and on NuSMV[12], for the propositional temporal logic model checking. The experiments have been conducted on a laptop with an Intel Core2Duo at 2.40 GHz as processor with 4GB RAM and Windows Vista Premium (32bit) as operating systems. The results are reported in Table 1. The two Specifications (1) and (2) are satisfied by the web service of Company A, but they are not satisfied by the web service of Company B. Indeed, that web service has a path that involves an alternative payment method and a path that involves four authentication attempts.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we present an approach based on semantic model checking for the verification of security requirements of semantic web services and evaluate it on a case study. The approach proposed in this paper is based on the
Table 1. Experimental evaluation.

<table>
<thead>
<tr>
<th>Process Name</th>
<th>states</th>
<th>transitions</th>
<th>annotations</th>
<th>Specification</th>
<th>annotations</th>
<th>Grounding (msec.)</th>
<th>Model Checking (msec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company A</td>
<td>55</td>
<td>60</td>
<td>571</td>
<td>4</td>
<td>4</td>
<td>3447.0</td>
<td>49.57</td>
</tr>
<tr>
<td>Company B</td>
<td>107</td>
<td>118</td>
<td>1116</td>
<td>4</td>
<td>4</td>
<td>6320.2</td>
<td>166.89</td>
</tr>
</tbody>
</table>

idea of representing a service as a state transition system and annotating it by means of description logic assertions. The verification algorithm works in two steps: the first step, called grounding, aims at reasoning at an ontological level and it is based on conjunctive query answering, the second step aims at behavior verification at a process level and it is based on propositional temporal logic model checking. The result is a model checking algorithm for semantically annotated processes. The algorithm has been proved to be sound and complete and its complexity to be polynomial time. The experiments reported in Table 1 point out that in spite of the minimalistic approach, it is possible to represent problems that are still interesting from a practical point of view. Indeed, the web services that have been used are inspired by real web services. Furthermore, even if the experimented web services have a quite large number of annotations, the verification requires around 1 sec. with a non-engineered and non-optimized software. As future work, there is the task of implementing an engineered and optimized software and investigating how this impacts on the performances.

Finally, as future work there is also the task of investigating how this approach can be applied to the so called Dolev-Yao model [27]. This model verifies the security of a system by means of an explicit representation of an attacker that can overhear, intercept, and synthesize any message and is only limited by the constraints of the cryptographic methods used.

REFERENCES


