Automatic Web Services Composition in Trust-aware Communities

Fahima Cheikh
Institut de Recherche en Informatique
de Toulouse, 118, route de Narbonne
31062 Toulouse, France
cheikh@irit.fr

Giuseppe De Giacomo
Massimo Mecella
Univ. Roma LA SAPIENZA
Dipartimento di Informatica e Sistemistica
Via Salaria 113 – 00198 Roma, Italy
{degiacomo,mecella}@dis.uniroma1.it

ABSTRACT

The promise of Web Service Computing is to utilize Web services as fundamental elements for realizing distributed applications/solutions. In particular, when no available service can satisfy client request, (parts of) available services can be composed and orchestrated in order to satisfy such a request. In this paper, we address the automatic composition when component services have access control & authorization constraints, and impose further reputation constraints on other component services. In particular, access & authorization control is based on credentials, component services may (or not) trust of credentials issued by other component services and the service behavior is modeled by the possible conversations the service can have with its clients. We propose an automatic composition synthesis technique, based on reduction to satisfiability in Propositional Dynamic Logic, that is sound, complete and decidable. Moreover, we will characterize the computational complexity of the problem.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: [Access controls]; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Authentication*; H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Web-based Services*

General Terms

Security, Verification

Keywords

Web services, Composition, Access control, Trust

1. INTRODUCTION

Web services (also called simply services) are self-describing, platform-agnostic computational elements that support rapid, low-cost and easy composition of loosely coupled distributed applications. From a technical standpoint, Web services are modular applications that can be described, published, located, invoked and composed over a variety of networks (including the Internet): any piece of code and any application component deployed on a system can be wrapped and transformed into a network-available service, by using standard (XML-based) languages and protocols (e.g., WSDL, SOAP, etc.) - see e.g., [2].

The promise of Web service is to enable the composition of new distributed applications/solutions: when no available service can satisfy a client request, (parts of) available services can be composed and orchestrated in order to satisfy such a request. Note that service composition involves two different issues [2]: the *synthesis*, in order to synthesize, either manually or automatically, a specification of how coordinating the component services to fulfill the client request, and the *orchestration*, i.e., how executing the previous obtained specification by suitably supervising and monitoring both the control flow and the data flow among the involved services.

As organizations increase their use of Web services and adopt them as the primary tool to build fairly complex distributed systems, security becomes crucial [27]. While security (and especially access control) has been widely studied in the literature and especially in database systems [12], only recently work on security for Web services has emerged as an important part of the Web service saga [30, 28, 3, 11, 21].

In this paper, we focus on automatic composition synthesis in presence on access control constraints by the component services, and in scenarios in which different component services may (or not) trust about others.

An access control model restricts the set of clients or subjects that can invoke Web service's operations. Since clients are not known a priori, we adopt credentials to enforce access control. Credentials are signed assertions describing properties of a subject that are used to establish trust between two unknown communicating parties before allowing access to information or services. Access control policies define rules stating that only subjects with certain credentials satisfying specific conditions can invoke a given operation of the Web service. In addiction, in many scenarios, the issuers of some credentials may not have a good reputation by other subjects (i.e., services), and therefore the automatic composition synthesis phase needs to consider all such constraints.

We consider the behavior of the available services as nondeterministic, and hence not fully controllable by the or-

SWS'06, November 3, 2006, Alexandria, Virginia, USA.

chestrator. The service behavior is modeled by the possible conversations the service can have with its clients. The presence of nondeterministic conversations stems naturally when modeling services in which the result of each interaction with its client on the state of the service can not be foreseen. Let us consider as an example, a service that allows buying items by credit card; after invoking the operation, the service can be in a state `payment_OK`, accepting the payment, or in a different state `payment_refused`, if the credit card is not valid, with not enough credit, etc. Note that the client of a nondeterministic service can invoke the operation but cannot control what is the result of it. In other words, the behavior of the service is partially controllable, and the orchestrator needs to cope with such partial controllability. Note also that if one observes the status in which the service is after an operation, then s/he understand which transition, among those nondeterministically possible in the previous state, has been undertaken by the service. We assume that the orchestrator can indeed observe states of the available services and take advantage of this in choosing how to continue a certain task [1].

Moreover, access control & authorization constraints can be represented as guard on such conversations; such guards express conditions on the credentials owned by the clients, and such credentials are issued by different subjects, including other services (that indeed may change and emit them during the conversations themselves). Therefore the guards on conversations may explicit assess that the issuer of a credential should be well considered/reputated/trusted by the service. Again, the automatically synthesized orchestrator needs to cope with these issues.

From a formal point of view, in this paper, we adhere to the setting proposed in [10, 7, 9] whose distinguished features can be summarized as follows.

- The available services are grouped together into a so-called *community*.
- Services in the community share a common set of actions Σ, the *actions of the community*. In other words, each available service in the community exports its behavior to the community itself in terms of the actions in Σ (the actions recognized be the community).
- Each action in Σ denotes a (possibly complex) interaction between the service and a client, and as result of such an interaction the client may acquire new information (not necessarily modeled explicitly) that may be of help in choosing the next action to perform.
- The behavior of each available service is described in terms of a *finite transition system* (aka finite state machine) that makes use of the actions in Σ. Since in this paper we assume that the behavior of the available services is nondeterministic, differently from [7, 10], such a transition system is nondeterministic in general. Moreover, as we assume that service providers impose constraints on credentials provided by clients, such a transition system, differently from [9], is guarded.
- The client request itself is expressed as a finite transition system that makes use of the actions in Σ. Such a transition system, called *target service*, is deterministic as in [10], since we assume that there is no uncertainty on the behavior that the client want to realize through composition of the available services. Again, it is also guarded.
- The orchestrator has the ability of scheduling services on a *step-by-step* basis. Hence the orchestrator has the ability of *controlling the interleaving* of multiple services executed concurrently.
- The *composition synthesis* consists on synthesizing a program for the orchestrator such that by suitably scheduling the available services it can provide the target service to the client.

The contribution of this paper is to devise a formal technique to perform automatic composition synthesis, when available services are nondeterministic and impose security (specifically, *(i)* access control and authorization, and *(ii)* reputation) constraints. We will show that the technique proposed is sound, complete and terminating. Moreover we will characterize the computational complexity of the problem and show that the proposed technique is optimal wrt (worst-case) computational complexity.

Interestingly the technique proposed here is based on reduction to satisfiability in Propositional Dynamic Logic (PDL) [17] with a limited use of the reflexive-transitive-closure operator. Now, PDL satisfiability shares the same basic algorithms behind the success of the description logics-based reasoning systems used for OWL[2], such as FaCT[3], Racer[4], Pellet[5], and hence its applicability in the context of composition synthesis appears to be quite promising.

The rest of the paper is organized as follows. In Section 2 we first introduce the framework and the composition problem in formal details. In Section 3 we present an explanatory example, in order to highlight all the features of the approach. In Section 4 we develop the techniques to perform automatic composition, we show soundness and completeness and we characterize the complexity of both the techniques and the problem. In Section 5 we discuss some related work. Finally, in Section 6 we draw some conclusions.

2. FRAMEWORK

Community. A *community* $\mathcal{S}$ is formed by a finite set of available services $\{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$ that share the same set of *shared actions* $\mathcal{A}$. Such actions are the actions available to the agent that is client of the community. The client can use such actions to specify a behavior of its interest, the so-called *target service* $\mathcal{S}_0$. The community will try to realize the target service $\mathcal{S}_0$ by suitably *orchestrating* the available services $\mathcal{S}_1, \ldots, \mathcal{S}_n$, see [10, 9]. We also consider a reputation matrix *Rep* which has as rows available services and as columns available services and possibly third parties. The cell $Rep(i, j)$ represents the reputation level (set of all possible levels is finite) that the available service $\mathcal{S}_i$ has on the

[1]The reader should observe that also the standard proposal WSDL 2.0 has a similar point of view: the same operation can have multiple output messages (the `out message` and various `outfault messages`), and the client observe how the service behaved only after receiving a specific output message.

[2]`http://www.omg.org/uml/`

[3]`http://www.cs.man.ac.uk/~horrocks/FaCT/`

[4]`http://www.sts.tu-harburg.de/~r.f.moeller/racer/`

[5]`http://www.mindswap.org/2003/pellet/`

available service $\mathcal{S}_j$ or on the third party P_{j-n}. In addition, in this paper, a client has a set of *credentials* that allows him to execute various parts of an available service. The value of these credentials can be changed by an available service after the execution of an action.

Credentials. Credentials are the mean to establish trust between a client and the service provider. They are assertions about the client, issued by a given party. Formally, let $\mathcal{C} = \{c_1, \ldots, c_m\}$ be the set of credentials that are associated to clients. Each c_h is a pair of variables $(\mathcal{A}ttr, \mathcal{I}ssuer)$ where $\mathcal{A}ttr$ is the attribute variable of the credential, whose value characterizes the client (wrt c_h) and $\mathcal{I}ssuer$ is the issuer variable that contains the name of the entity that issued the value for the attribute variable[6]. The attribute variable assume values from a finite domain Δ. The possible values of the issuer are $\mathcal{I} = \{1, \ldots, n, n+1, \ldots, n+\ell\}$, where $1, \ldots, n$ are identifiers of available services and $n+1, \ldots, n+\ell$ are identifiers of third parties $P_1, \ldots P_\ell$. For convenience we assume that $\mathcal{I} \subseteq \Delta$.

Available Services. An available service is essentially a program for a client. However, such a program leaves the selection of the action to perform next to the client itself. More precisely, at each step the program presents to the client a choice of available actions; the client selects one of them; the action is executed; and so on.

Available services use credentials in order to decide which actions at each point of their execution are actually available to the client executing it (i.e., the client is authorized to execute the action). Moreover an available service may change the values of the credentials of the client in a dynamic way, i.e., while executing.

Conditions on a credential specify the security requirements of the available services. Let $\mathcal{S}_i$ be an available service and $c_h : (\mathcal{A}ttr, \mathcal{I}ssuer)$ be a credential, an atomic *credential condition* in $\mathcal{S}_i$ on c_h is an expression of the form T `op` v, where T is either $c_h.\mathcal{A}ttr$ or $Rep(i, c_h.\mathcal{I}ssuer)$, `op` is a comparison operator (interpreted on the obvious way), and v is a value in Δ. We denote by Ψ the set of closed FOL formulas build using as atoms the credential conditions. We also denote by Γ the set of (possibly partial) reassignments of values in Δ to the variables $\mathcal{I}ssuer$ and $\mathcal{A}ttr$ of credentials in $\mathcal{C}$. More precisely, let γ be a reassignment in Γ and let CA be the current assignment of all the variables of the form $c_h.\mathcal{I}ssuer$ or $c_h.\mathcal{A}ttr$ where $c_h \in \mathcal{C}$. Let $CA \circ \gamma$ be the assignment obtained from CA by reassigning the variables mentioned in γ according to γ itself. The reassignment γ must satisfy the following conditions:

- if $\gamma(c_h.\mathcal{I}ssuer) = v$ then v is either i (the service identifier itself) or the identifier of a third party.
- if there exists $c_h \in \mathcal{C}$, such that $CA \circ \gamma(c_h.\mathcal{A}ttr) \neq CA(c_h.\mathcal{A}ttr)$ then $\gamma(c_h.\mathcal{I}ssuer)$ has to be reassigned by γ (according to the rule above).

The first requirement states that an available service can assign to the issuer variable of a credential only the identifier of itself or that of a third party. Thus, an available service cannot assign to the issuer variable identifiers of other available services. The reason is that, in our framework, the available services are specified independently from each other and hence cannot delegate actions to other available services in the community. As for the assigning a third party, the intuition is that the available service delegates to a third party the guarantee of the attribute value of the credential and hence uses his identifier as value of the issuer variable. The second requirement states that if an available service reassigns the attribute variable of a credential then it must reassign also the issuer variable, since now it is taking charge of guaranteeing the new value.

Each available service $\mathcal{S}_i$ is defined in terms of a finite transition system of the form: $TS_i = (S_i, s_{i0}, G_i, \delta_i, F_i)$, where

- S_i is a finite set of states;
- $s_{i0} \in S_i$ is the single initial state of the service;
- G_i is a set of guards, which are formulas from Ψ;
- $\delta_i \subseteq S_i \times G_i \times \mathcal{A} \times \Gamma \times S_i$ is the service transition relation, where G_i is the set of guards of the service, $\mathcal{A}$ is the shared set of actions of community $\mathcal{S}$, and Γ is the set of possible reassignments for the credentials variables. Observe that $G_i \times \mathcal{A} \times \Gamma$ components of such tuple form the so-called *label of the transition*. We will drop the guard g to mean that $g = true$, and drop the reassignment if γ does not reassign any variable;
- $F_i \subseteq S_i$ is the set of final states of the service, that is, states in which the service may stop executing, but does not necessarily have to.

Observe that, in general, available services are *nondeterministic* in the sense that they may allow more than one transition with the same action a and two compatible guards evaluating to the same truth value. Such transitions differ either in the resulting state, in the resulting reassignment of the credentials or in both[7]. As a result, the client, when making its choice of which action to execute next, cannot be certain of which choices it will have later on, since that depends on what transition is actually executed: in other words, available services are only partially controllable

Target Services. The client specifies the service to be provided by the community, called target service, in terms of a finite transition system $TS_0 = (S_0, s_{00}, G_0, \delta_0, F_0)$, where transitions are labeled only by the guards and the actions (no reassignment). Guards can refer only to the attribute variable of the credentials, and not to the issuer variables. This captures the fact that the client can take its decision on what to do next based on the current value of its credential attributes, but not on who issued such values. We also require that the transitions are *deterministic*, in the sense that for every pair of transitions (s, g, a, t) and (s, g', a, t') the guards g and g' must be disjoint (i.e., $g \wedge g' = false$ for every possible assignment of the credentials in $\mathcal{C}$). This is because we want the client to be able to fully control the target service through its choices of actions.

To the client is also associated an *initial assignment* CA_{init} of the attribute and issuer variables of the credentials in $\mathcal{C}$, describing the value of credentials initially assigned to the agent executing the target service and who issued them.

[6]Credentials are usually implemented through the use of X.509 certificates or SAML assertions. Since the actual implementation format of credentials is not relevant to this paper, we omit further details on such an aspect.

[7]Note that this kind of nondeterminism is of a *devilish* nature, and captures the idea that through the choice of actions alone the agent cannot fully control the service.

Orchestrator Program. A *history* is an alternating sequence of the form $h = (s_1^0, \ldots, s_n^0, CA^0) \cdot a^1 \cdot (s_1^1, \ldots, s_n^1, CA^1) \cdots a^\ell \cdot (s_1^\ell, \ldots s_n^\ell, CA^\ell)$ such that the following constraints hold:

- $s_i^0 = s_{i0}$ for $i \in \{1, \ldots, n\}$, i.e., all services start in their initial state, and CA^0 is the assignment of all variables in $\mathcal{C}$;
- at each step k, *(i)* for one i we have that $(s_i^k, g, a^{k+1}, \gamma, s_i^{k+1}) \in \delta_i$ with $g = \mathit{true}$ in CA^k; *(ii)* $CA^{k+1} = CA^k \circ \gamma$;
  (iii) for all $j \neq i$ we have that $s_j^{k+1} = s_j^k$.

An *orchestrator program* is a function $OP : \mathcal{H} \times \mathcal{A} \rightarrow \{1, \ldots, n, u\}$ that, given a history $h \in \mathcal{H}$ (where $\mathcal{H}$ is the set of all histories defined as above) and an action $a \in \mathcal{A}$ to perform, returns the service (actually the service index) that will perform it, or the special value u (for "undefined").

Composition. Next we define when an orchestrator program is a composition that realizes the client request. Being the target service deterministic, its behavior is completely characterized by the set of its *traces*, defined by the set of infinite sequences of *guarded* actions that are obtained by following its transitions, and of finite sequences that in addition lead to a final state. Now, given a trace $t = (g^1, a_1) \cdot (g^2, a_2) \cdots$ of the target service, we say that an *orchestrator program OP realizes the trace t starting from an initial credential assignment CA_{init}* iff for all ℓ and for all histories $h \in \mathcal{H}_t^\ell$ such that $g^{\ell+1} = \mathit{true}$ in the last variable assignment CA_h^ℓ of h, we have that $P(h, a_{\ell+1}) \neq u$ and $\mathcal{H}_t^{\ell+1}$ is nonempty, where the sets $\mathcal{H}_t^\ell$ are inductively defined as follows:

- $\mathcal{H}_t^0 = \{(s_{10}, \ldots, s_{n0}, CA_{init})\}$
- $\mathcal{H}_t^{\ell+1}$ is the set of all histories such that if $(s_1^{\ell+1}, \ldots, s_n^{\ell+1}, CA^{\ell+1})$, with $s_i^{\ell+1} = s_i'$, $h \in \mathcal{H}_t^\ell$, and $P(h, a_{\ell+1}) = i$ (with $i \neq u$), then for all transitions $(s_i^\ell, g, a, \gamma, s_i') \in \delta_i$ with $g = \mathit{true}$ in the last variable assignment CA_h^ℓ, the history $h \cdot a_{\ell+1} \cdot (s_1^{\ell+1}, \ldots, s_n^{\ell+1}, CA^{\ell+1})$, with $s_i^{\ell+1} = s_i'$, $s_j^{\ell+1} = s_j^\ell$ for $j \neq i$, and $CA^{\ell+1} = CA_h^\ell \circ \gamma$ is in $\mathcal{H}_t^{\ell+1}$.

Moreover, if a trace is finite and ends after f actions, and all along all its guards are satisfied, we have that all histories in $\mathcal{H}_t^f$ end with all services in a final state. Finally, we say that an *orchestrator program OP realizes the target service* $\mathcal{S}_0$ if it realizes all its traces.

In order to understand the above definitions, let us observe that intuitively the orchestrator program realizes a trace if, as long as the guards in the trace are satisfied, it can choose at every step an available service to perform the requested action. If at a certain point a guard in the trace is not satisfied by the current credential assignment, then we may consider the trace finished (even if not in a final state). However, since when an available service executes an action it nondeterministically chooses what transition to actually perform, the orchestrator program has to require that for each of the possible resulting states of the activated available services and resulting credential assignment, the orchestrator is able to continue with the execution of the next action. Finally, the last requirement makes sure that available services are left in a final state, when a finite trace reaches its end with all guards satisfied all along.

3. CASE STUDY

Figure 1 shows a community of services for searching and listening MP3 files. The community includes several services:

- $\mathcal{S}_1$ allows the client to repeatedly *(i)* search a file by author (`sa`), if the client holds a credential assessing he is a `gold` customer and the issuer of such a credential is reputated higher than 5 by the service itself, and then, *(ii)* listen to the file (`l`).
- $\mathcal{S}_2$ allows the client to repeatedly *(i)* search a file by author (`sa`), if the client holds a credential assessing he is a `gold` customer and the issuer of such a credential is reputated higher than 2 by the service itself, and then, *(ii)* listen to the file (`l`). Or alternatively, it allows the client to *(i)* search a file by title (`st`), if the client holds a credential assessing he is registered, and the issuer of such a credential is reputated higher than 2 by the service itself, and then, *(ii)* listen to the file (`l`).
- $\mathcal{S}_3$ allows the client to repeatedly *(i)* search a file by title (`st`), if the client holds a credential assessing he is an inscribed one and the issuer of such a credential is reputated higher than 4 by the service itself, and then, *(ii)* listen to the file (`l`). Depending on the internal application logic of the service, sometimes the listen operation change the registration status of the client (thus forcing him to register again). The client can obtain the credential, issued by the service itself, assessing he is a registered client, through an inscription operation (`i`).
- $\mathcal{S}_4$ allows the client to obtain the credential, issued by the service itself, assessing he is a registered client, through an inscription operation (`i`).

Moreover, the community $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3\, \mathcal{S}_4\}$ is characterized by the following reputation matrix (in which P represents possible third parties, and the values for the reputation range from 0 to 5):

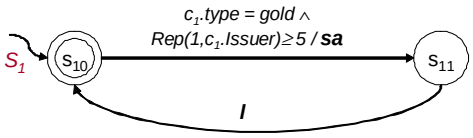
Rep	$\mathcal{S}_1$	$\mathcal{S}_2$	$\mathcal{S}_3$	$\mathcal{S}_4$	P
$\mathcal{S}_1$	5	0	0	0	5
$\mathcal{S}_2$	0	5	0	0	0
$\mathcal{S}_3$	0	0	5	5	5
$\mathcal{S}_4$	0	0	5	5	5

The set of credentials is $\mathcal{C} = \{c_1, c_2\}$ where $c_1 = (\mathit{type}, \mathcal{I}\mathit{ssuer})$ and $c_2 = (\mathit{inscribed}, \mathcal{I}\mathit{ssuer})$.

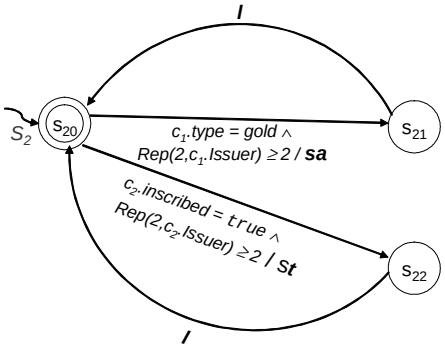
Figure 2 shows the target service $\mathcal{S}_0$: the client wants to (possibly) register himself, then to search by author or by title, and listen to the file.

The reader can notice that $\mathcal{S}_2$ and $\mathcal{S}_4$ would realize the target, but $\mathcal{S}_2$ doesn't trust too much of the other component services, in particular of $\mathcal{S}_4$, that may be the issuer for credential c_2. So an orchestrator program, given the previous reputation matrix, need to resort to $\mathcal{S}_1$ and $\mathcal{S}_3$ for realizing the target.

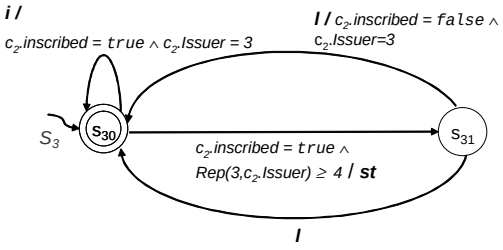
Figure 3 shows an orchestrator program P for available services $\mathcal{S}_1$, $\mathcal{S}_2$, $\mathcal{S}_3$ and $\mathcal{S}_4$, that realizes the target service $\mathcal{S}_0$, given the previous reputation matrix and the initial credential assignment $CA_{init}(c_1.\mathit{type}) = \mathit{gold}$, $CA_{init}(c_1.\mathcal{I}\mathit{ssuer}) = P$, $CA_{init}(c_2.\mathit{inscribded}) = \mathit{false}$ and $CA_{init}(c_2.\mathcal{I}\mathit{sssuer}) = P$. The reader should note that an



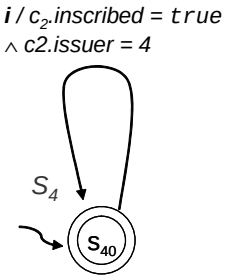
(a) S_1



(b) S_2



(c) S_3



(d) S_4

Figure 1: A Web service community

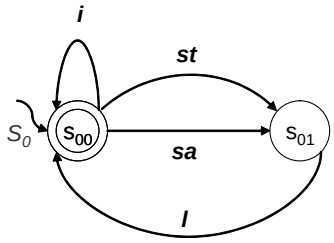


Figure 2: A target Web service

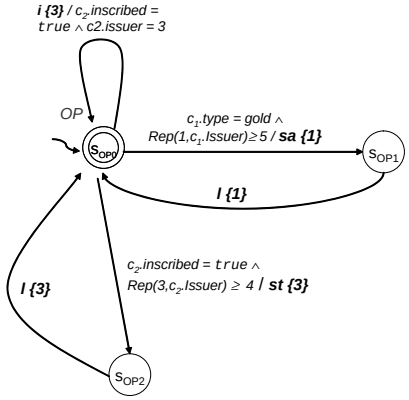


Figure 3: An orchestrator

initial credential assignment could produce different programs, possibly also no solutions (e.g., if $CA_{init}(c_1.type) = bad$ and $CA_{init}(c_1.\mathcal{I}ssuer) = P$).

4. AUTOMATIC COMPOSITION

We are now ready to investigate how to check for the existence of an orchestrator program that realizes the target service, by suitably coordinating the available ones, that takes into account the initial credentials and how these evolve over time.

Following [10, 9], we resort to a reduction to satisfiability in Propositional Dynamic Logic (PDL). PDL is a modal logic specifically developed for reasoning about computer programs [17]. Syntactically, PDL formulas are built from a set $\mathcal{P}$ of atomic propositions and a set $\mathcal{A}$ of atomic actions:

$$\begin{array}{rcl} \phi & \longrightarrow & P \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi \rightarrow \phi' \mid \\ & & \langle r\rangle\phi \mid [r]\phi \mid \texttt{true} \mid \texttt{false}, \\ r & \longrightarrow & a \mid r_1 \cup r_2 \mid r_1; r_2 \mid r^* \mid \phi?, \end{array}$$

where P is an atomic proposition in $\mathcal{P}$, r is a regular expression over the set of actions in $\mathcal{A}$, and a is an atomic action in $\mathcal{A}$. That is, PDL formulas are composed from atomic propositions by applying arbitrary propositional connectives, and modal operators $\langle r\rangle\phi$ and $[r]\phi$. Formula $\langle r\rangle\phi$ means that there exists an execution of r (i.e., a sequence of actions conforming to the regular expression r) reaching a state where ϕ holds; and formula $[r]\phi$ is intended to mean that all terminating executions of r reach a state where ϕ holds.

A PDL formula ϕ is satisfiable if there exists a model for ϕ, i.e., an interpretation where ϕ is true. Checking satisfiability of a PDL formula is EXPTIME-complete [17].

PDL enjoys two properties that are of particular interest for us [17]. The first is the *tree model property*, which says

that every model of a formula can be unwound to a (possibly infinite) tree-shaped model (considering domain elements as nodes and partial functions interpreting actions as edges). The second is the *small model property*, which says that every satisfiable formula admits a finite model whose size (in particular the number of domain elements) is at most exponential in the size of the formula itself.

Let $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_n)$ be a community of available services over the shared actions $\mathcal{A}$ and with credentials $\mathcal{C}$. To each available services $\mathcal{S}_i$ is associated its transition system $TS_i = (S_i, s_{i0}, G_i, \delta_i, F_i)$ defined as above. Let $TS_0 = (S_0, s_{00}, G_0, \delta_0, F_0)$ be the target service (over $\mathcal{A}$ and $\mathcal{C}$), and finally let CA_{init} be the initial assignment to the credentials $\mathcal{C}$. Then, we build a PDL formula Φ to check for satisfiability as follows.

As actions in Φ, we have the actions $\mathcal{A}$. As atomic propositions, we have:

- atomic propositions that enable us to encode credential assignments into propositions. For simplicity we denote by $\mathbf{CA}$ the propositional encoding of CA; we obviously assume that the encoding is faithful in the sense that $\boldsymbol{CA}$ is true in the current state of the model if CA is the assignment corresponding to the current state;
- one atomic proposition s for each $i \in \{0, 1, \ldots, n\}$ and each state s of TS_i, which intuitively denotes that TS_i is in state s;
- atomic propositions F_i, for $i \in \{0, 1, \ldots n\}$, denoting that TS_i is in a final state;
- atomic propositions $exec_{ia}$, for $i \in \{1, \ldots n\}$ and $a \in \mathcal{A}$, denoting that a will be executed next by the available service $\mathcal{S}_i$;
- one atomic proposition *undef* denoting that we are in an "illegal" situation, where the orchestrator program can be left undefined.

The formula Φ is built as follows. For representing the transitions of the target service $\mathcal{S}_0$, we construct a formula ϕ_0 as the conjunction

- $s \wedge \boldsymbol{CA} \rightarrow \langle a \rangle \mathtt{true} \wedge [a]s'$, for each transition $(s, g, a, s') \in \delta_0$ such that $CA \models g$ (i.e., g is true in CA). This encodes that the target service can do an a-transition, whose guard g is satisfied, by going from state s to state s'.
- $s \wedge \boldsymbol{CA} \rightarrow [a]\mathit{undef}$, for each a such that for no g and s' we have $(s, g, a, s') \in \delta_0$ with $CA \models g$. This takes into account that the target service cannot perform an a-transition.

For representing the transitions of each available services $\mathcal{S}_i$, we construct a formula ϕ_i as the conjunction of:

- $s \wedge \boldsymbol{CA} \wedge exec_{ia} \rightarrow \bigwedge_{(s',\gamma)\in\mathcal{E}} (\langle a \rangle (s' \wedge \boldsymbol{CA} \circ \boldsymbol{\gamma})) \wedge [a](\bigvee_{(s',\gamma)\in\mathcal{E}} (s' \wedge \boldsymbol{CA} \circ \boldsymbol{\gamma}))$,

  where $\mathcal{E} = \{(s', \gamma) \mid (s, g, a, \gamma, s') \in \delta_i,\ CA \models g\}$, for each credential assignment CA, each s of $\mathcal{S}_i$, and each $a \in \mathcal{A}$. These assertions encode that if the current credential assignment is CA and the available service $\mathcal{S}_i$ is in state s and is selected for the execution of an action a (i.e., $exec_{ia}$ is true), then for each possible a-transition of $\mathcal{S}_i$ with its guard true in CA, we have a possible a-successor in the models of Φ.
- $s \wedge \boldsymbol{CA} \wedge exec_{ia} \rightarrow [a]\mathtt{false}$, for each credential assignment CA, and each state s of $\mathcal{S}_i$ such that for no g, s', and γ, we have that $(s, g, a, \gamma, s') \in \delta_i$ with $CA \models g$. This states that if the current credential assignment is CA and the available service $\mathcal{S}_i$, whose current state is s, is selected for the execution of a, but a cannot be executed by $\mathcal{S}_i$ under the credentials CA, then there is no a-successor in the models of Φ.
- $s \wedge \neg exec_{ia} \rightarrow [a]s$, for each state s of $\mathcal{S}_i$ and each action a. This assertion encodes that if available service $\mathcal{S}_i$ is in state s and is not selected for the execution of a, then if a is performed (by some other available service), $\mathcal{S}_i$ does not change state.

In addition, we have the formula ϕ_{add} obtained as the conjunction of:

- $s \rightarrow \neg s'$, for all pairs of states s, s' of $\mathcal{S}_i$, and for $i \in \{0, 1, , \ldots, n\}$; these say that propositions representing different states of $\mathcal{S}_i$ are disjoint.
- $F_i \leftrightarrow \bigvee_{s\in F_i} s$, for $i \in \{0, 1, , \ldots, n\}$; this highlights the final states of $\mathcal{S}_i$.
- $\mathit{undef} \rightarrow [a]\mathit{undef}$, for each action $a \in \mathcal{A}$; these say that once a situation is reached where *undef* holds, then *undef* holds also in all successor situations.
- $\neg \mathit{undef} \wedge \langle a \rangle \mathtt{true} \rightarrow \bigvee_{i\in\{1,\ldots,n\}} exec_{ia}$, for each $a \in \mathcal{A}$, denoting that, unless *undef* is true, if a is performed, then at least one of the available services must be selected for the execution of a.
- $exec_{ia} \rightarrow \neg exec_{ja}$ for each $i, j \in \{1, \ldots, n\}$, $i \neq j$, and each $a \in \mathcal{A}$, stating that only one available service is selected for the execution of a.
- $F_0 \rightarrow \bigwedge_{i\in\{1,\ldots,n\}} F_i$, stating that when the target service is in a final state, so are all the available services.

Observe that by the nature of the propositional encoding we also have that $\boldsymbol{CA} \rightarrow \neg \boldsymbol{CA'}$ for each distinct pairs of credential assignments CA and CA'.

Finally, we define Φ as

$$\mathit{Init} \wedge [\mathbf{u}](\phi_0 \wedge \bigwedge_{i\in\{1,\ldots,n\}} \phi_i \wedge \phi_{add}),$$

where *Init* stands for $\boldsymbol{CA}_0 \wedge s_{00} \wedge s_{10} \wedge \cdots \wedge s_{n0}$, and represents the initial credential assignment and the initial state of all services $\mathcal{S}_i$ (including the target), and $\mathbf{u} = (\bigcup_{a\in\mathcal{A}} a)^*$, which acts as the *master modality* [17], is used to force $\phi_0 \wedge \bigwedge_{i\in\{1,\ldots,n\}} \phi_i \wedge \phi_{add}$ to be true in every point of the model. Note that $\mathbf{u}$ is the only complex program that appears in the PDL formula Φ. We can now state our main result.

THEOREM 4.1. *The PDL formula Φ, constructed as above, is satisfiable if and only if there exists an orchestrator program for the available services $\mathcal{S}_1, \ldots, \mathcal{S}_n$ that realizes the target service $\mathcal{S}_0$ starting from the initial credential assignment CA_{init}.*

By the finite-model property of PDL (i.e., if a formula is satisfiable, it is satisfiable in a model that is at most exponential in the size of the formula), we can build a systematic procedure for synthesizing the composition.

THEOREM 4.2. *If there exists an orchestrator program for the available services $\mathcal{S}_1, \ldots, \mathcal{S}_n$ that realizes the target service $\mathcal{S}_0$ starting from the initial credential assignment CA_{init}, then there exists one that requires a finite number of states. Moreover such a finite state program can be extracted from a finite model of Φ.*

As for computational complexity, we observe that Φ is polynomial in the size of the target service $\mathcal{S}_0$, in the size of the available services $\mathcal{S}_1, \ldots, \mathcal{S}_n$, and in the number of possible credential assignment (and hence exponential in the number of credential in $\mathcal{C}$). So, considering that satisfiability in PDL is EXPTIME-complete, we can state the following theorem.

THEOREM 4.3. *Checking the existence of an orchestrator program for the available services $\mathcal{S}_1, \ldots, \mathcal{S}_n$ that realizes the target service $\mathcal{S}_0$ starting from the initial credential assignment CA_{init} can be done in time at most exponential in the size of $\mathcal{S}_1, \ldots, \mathcal{S}_n$ and $\mathcal{S}_0$ and doubly exponential in the number of credentials in $\mathcal{C}$.*

Notice that the problem of composition is already EXPTIME-hard in the case considered in [10] where all available services are deterministic and we have no credentials requirements [23]. So, the result above is optimal wrt the bounds on the size of the services (including the target). As for the credential assignments, we observe that a clever propositional encoding can reduce substantially this computational cost, although this issue is out of the scope of this paper.

5. RELATED WORK

Web services represent the core element for building complex services and applications provided either by single organizations or by a set of cooperating companies. Web services are built on top of two major standards: SOAP and WSDL; SOAP defines the protocol to exchange XML messages with a Web service, and WSDL describes the interface of a Web service as a set of operations and it provides information on how to locate and invoke them. Recent papers [6, 10, 24, 4] have argued that a Web service is more than a set of independent operations. In fact, during a Web service's invocation, a client interacts with the service performing a sequence of operations in a particular order. Such a sequence is called *conversation*. Specifically, [6, 10, 4] adopt a model based on finite transition systems (aka finite state machines) for representing all possible conversations. The approach of [24] is based on the combined use of two Web service languages, WS-Conversation (WSCL) and WS-Agreement, that allows one to specify non-trivial conversations in which several messages have to be exchanged before the service is completed and/or the conversation may evolve in different ways depending on the state and the needs of the requesting agents and of the service provider.

As far as security issues in Web services are concerned, a fair amount of related research in this area comes from the industry. Two major standards have emerged, namely Security Assertion Markup Language (SAML) and eXtensible Access Control Markup Language (XACML). SAML defines an XML framework for exchanging authentication and authorization information for securing Web services. XACML is an XML framework for specifying access control policies for Web-based resources. Recently it has been extended to specify access control policies for Web services. Other emerging specifications include WS-Security and WS-Policy. WS-Security is a specification for securing SOAP messages using XML Encryption and XML Signature standards and attaching security credentials thereto. WS-Policy is used to describe the security policies in terms of their characteristics and supported features (such as required security tokens, encryption algorithms, privacy rules, etc.).

These proposals do not address the issue of enforcing access control, and do not consider how to include it during the composition/synthesis phase. Several approaches [30, 28, 3, 11] suggest some preliminary ideas, but none of them provide a comprehensive solution. [30] proposes two RBAC (Role Based Access Control) models, SWS-RBAC, for single Web services, and CWS-RBAC, for composite Web services. In both models, a service has a few access modes and a role is associated with a list of services which clients, who are assigned that role, have permission to execute. In CWS-RBAC model, the role to which a client is assigned to access a composite service, must be a global role, which is mapped onto local roles of the service providers of the component Web services. [28] proposes an approach for specifying and enforcing security policies. These are specified using a language called WebGuard based on temporal logic and are processed by an enforcement engine to yield site and platform-specific access control. This code is integrated with a Web server and platform specific libraries to enforce the specified policies on a given Web service. [11] presents WS-AC1, an access control model with flexible granularity in protecting objects and negotiation capabilities. WS-AC1 is based on the specification of policies stating conditions on the values of the identity attributes and service parameters that a client must provide to invoke the service. Conditions may also be specified against context parameters, such as time. Further, it is possible to define fine-grained policies by associating them with a specific service as well as coarse-grained policies, to be applied to a class of services. The negotiation capabilities of WS-AC1 are related to both identity attributes and service parameters. Through a negotiation process, the client is driven toward an access request compliant with the service description and policies.

The Semantic Web Service initiative has focused on richer formalisms and description/specification languages for policies, based on specific ontologies for "security" [15], in order to be able to match service and client requirements [19, 1] during the discovery phase. Moreover, the concept of *semantic firewall* has been introduced [5] as a component operating on top of a traditional firewall, and able to reason about the acceptability of messages on the basis of semantics of policies.

All the previous proposals basically describe access control models: they are based on the enforcement of access control policies stating the requirements to be satisfied by a client to be granted access to a Web service. Since a Web service can be invoked potentially by anyone, the requirements are expressed as conditions on the digital credentials owned by a client. But none of them address the issue of considering such conditions during the automatic synthesis

of composite Web services. On the other hand, a lot of research has been conducted on *automatic* service composition, but up to know by considering only functional aspects (i.e., operations and service behaviors). In order to compare different approaches, in [8] a conceptual framework has been introduced, by proposing a classification into two general approaches: Service-tailored, the oldest one and considered less flexible, and the Client-tailored, more recent but technically more difficult from a computational point of view..

In fact, much of the research on automatic service composition has adopted, up to now, a service-tailored approach. For example, the works based on Classical Planning in AI (e.g., [31], [13]) and the work of McIlraith et al. [20]. The work by Hull et al. [14, 18] describes a setting where services are expressed in terms of atomic communications that they can perform, and channels that link them with other services. The aim of the composition is to refine the behavior of each service so that the conversations realized by the overall system satisfy a given goal (dynamic property) expressed as a formula in linear time logic. Although possibly more on choreography synthesis than on composition synthesis, we can consider it a service-tailored approach, since there is no effort in hiding the service details from the client that specifies the goal formula.

Much less research has been done following a client-tailored approach, but some remarkable exceptions should be mentioned: the work of Knoblock at al. [22], the work of Traverso et al. [29, 26, 25], and the line of research taken in [7, 10], but also in [16], and in the present paper.

But, as already discussed, none of them considers security, specifically access control and reputation, during the automatic synthesis of composite Web services.

6. CONCLUSION

In this paper we studied how to synthesize a composition to realize a client service request expressed as a target service, in the case where *(i)* available services are only partially controllable (modeled as devilish nondeterminism) but fully observable by the orchestrator, and *(ii)* access control & authorization constraints, and reputation are taken into account. We have shown that the problem is EXPTIME-complete and we have given effective techniques to address the problem based on PDL satisfiability. Our results extend those in [10, 7], where only deterministic available services where considered, and no security aspects were addressed.

We would like to remark the importance of considering mutual reputation among services, as in dynamic environments, as those one targeted by Web service technologies, not all available services may trust each others, nevertheless the composition should be possible, by exploiting all available services in order to satisfy clients.

In future work, we will consider the issue of identifying the minimal set of credentials that a client should hold in order to find an orchestrator for its target service. This is an important issue in real scenarios, as automatic composition should not impose unnecessary constraints on clients, especially regarding security aspects. Moreover, on the basis of an ongoing implementation of a composition engine[8], we plan to implement the proposed techniques into a prototype.

[8]cfr. the PARIDE (Process-based frAmewoRk for composItion and orchestration of Dinamyc *E*-Services) Open Source Project: `http://sourceforge.net/projects/paride/`.

7. REFERENCES

[1] S. Agarwal, B. Sprick, and S. Wortmann. Credential Based Access Control for Semantic Web Services. In *AAAI Spring Symposium Series*, 2004.

[2] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services. Concepts, Architectures and Applications*. Springer, 2004.

[3] C. Ardagna, E. Damiani, S. De Capitani di Vimercati, and P. Samarati. A Web Service Architecture for Enforcing Access Control Policies. In *Proceedings of 1st International Workshop on Views on Designing Complex Architectures*, 2004.

[4] R. Ashri, G. Denker, D. Marvin, M. Surridge, and T. Payne. Semantic Web Service Interaction Protocols: An Ontological Approach. In *Proc. 3rd Internation Semantic Web Conference (ISWC 2004)*, 2004.

[5] R. Ashri, T. Payne, D. Marvin, M. Surridge, and S. Taylor. Towards a Semantic Web Security Infrastructure. In *Proc. Semantic Web Services 2004 Spring Symposium Series*, 2004.

[6] B. Benatallah, F. Casati, and F. Toumani. Web Service Conversation Modeling: A Cornerstone for e-Business Automation. *IEEE Internet Computing, 8(1):46 – 54*, 2004.

[7] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Synthesis of Underspecified Composite e-Services based on Automated Reasoning. In *Proc. of ICSOC 2004*.

[8] D. Berardi, D. Calvanese, G. De Giacomo, and M. Mecella. Automatic Web Service Composition: Service-tailored vs. Client-tailored Approaches. In *Proc. AISC 2006, International Workshop jointly with ECAI 2006*.

[9] D. Berardi, D. Calvanese, G. De Giacomo, and M. Mecella. Automatic Composition of Web Services with Nondeterministic Behavior. Technical Report TR-05-2006, Univ. Roma LA SAPIENZA, Dipartimento di Informatica e Sistemistica, 2006. Extended abstracts/short papers in Proc. ICSOC 2005 and in Proc. ICWS 2006.

[10] D. Berardi, D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Mecella. Automatic Service Composition based on Behavioral Descriptions. *International Journal of Cooperative Information Systems, 14(4):333 – 376*, 2005.

[11] E. Bertino, L. Martino, F. Paci, and A. Squicciarini. An Adaptive Access Control Model for Web Services. *To appear on International Journal of Web Services Research (JWSR)*, 2006.

[12] E. Bertino and R. Sandhu. Database Security – Concepts, Approaches and Challenges. *IEEE*, 2005.

[13] J. Blythe and J. Ambite, editors. *Proc. ICAPS 2004 Workshop on Planning and Scheduling for Web and Grid Services*, 2004.

[14] T. Bultan, X. Fu, R. Hull, and J. Su. Conversation Specification: A New Approach to Design and Analysis of E-Service Composition. In *Proc. of WWW 2003*.

[15] G. Denker, L. Kagal, T. Finin, M. Paolucci, and K. Sycara. Security for DAML Web Services: Annotation and Matchmaking. In *Proc. of the 2nd*

Internationl Semantic Web Conference (ISWC 2003), 2003.

[16] C. Gerede, R. Hull, O. H. Ibarra, and J. Su. Automated Composition of E-Services: Lookaheads. In *Proc. ICSOC 2004*.

[17] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. The MIT Press, 2000.

[18] R. Hull, M. Benedikt, V. Christophides, and J. Su. E-Services: a Look Behind the Curtain. In *Proc. of PODS 2003*, pages 1–14, 2003.

[19] L. Kagal, M. Paolucci, N. Srinivasan, G. Denker, T. Finin, and K. Sycara. Authorization and Privacy for Semantic Web Services. *IEEE Intelligent Systems, July/August*, 2004.

[20] S. McIlraith and T. Son. Adapting Golog for Composition of Semantic Web Services. In *Proc. KR 2002*.

[21] M. Mecella, M. Ouzzani, F. Paci, and E. Bertino. Access Control Enforcement for Conversation-based Web Services. In *Proc. of the 15th International World Wide Web Conference, Edinburgh, UK*, May 2006.

[22] M. Michalowski, J. Ambite, S. Thakkar, R. Tuchinda, C. Knoblock, and S. Minton. Retrieving and Semantically Integrating Heterogeneous Data from the Web. *IEEE Intelligent Systems*, 19(3):72–79, 2004.

[23] A. Muscholl and I. Walukiewicz. A Lower Bound on Web Services Composition. Submitted, 2006.

[24] S. Paurobally and N. R. Jennings. Protocol Engineering for Web Services Conversations. *Engineering Applications of Artificial Intelligence*, 18, 2005.

[25] M. Pistore, A. Marconi, P. Bertoli, and P. Traverso. Automated Composition of Web Services by Planning at the Knowledge Level. In *Proc. of IJCAI 2005*, 2005.

[26] M. Pistore, P. Traverso, P. Bertoli, and A. Marconi. Automated Synthesis of Composite BPEL4WS Web Services. In *Proc. of ICWS 2005*, 2005.

[27] K. E. Seamons, M. Winslett, and T. Yu. Limiting the Disclosure of Access Control Policies during Automated Trust Negotiation. In *Proceedings of the Network and Distributed System Security Symposium, San Diego, California, USA*, 2001.

[28] E. Sirer and K. Wang. An Access Control Language for Web Services. In *Proc. ACM SACMAT*, 2002.

[29] P. Traverso and M. Pistore. Automated Composition of Semantic Web Services into Executable Processes. In *Proc. ISWC 2004*.

[30] R. Wonohoesodo and Z. Tari. A Role based Access Control for Web Services. In *Proc. SCC 2004*, 2004.

[31] D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-S Web Services Composition using SHOP2. In *Proc. ISWC 2003*.