# Web services composition is hard but decidable

Ramy Ragab, Lhouari Nourine, Farouk Toumani

# Web services composition is hard but decidable

Ramy Ragab[1] Lhouari Nourine[2] Farouk Toumani[3]

Research Report LIMOS/RR-07-16

10 décembre 2007

[1]ragab@isima.fr
[2]nourine@isima.fr
[3]ftoumani@isima.fr

**Abstract**

We study the problem of automatic web service composition. We consider a formal framework where web service business protocols are described by means of Finite State Machines (FSM) and focus on the protocol synthesis problem. We show that this problem can be reduced to that of testing a simulation relation between an FSM and an (infinitely) iterated product of FSMs. While this later problem has never been investigated in the literature, existing results regarding close decision problems in the context of shuffle languages, an extension of regular languages with shuffle and shuffle closure operators, are rather negative and cannot be directly exploited in our context. In this paper, we develop a novel technique to prove the decidability of testing simulation in the case of interest in our setting. As a consequence, our results solve the problem of web service composition (synthesis) existence in presence of an unbounded number of instances, a problem left open in recent related works.

**Keywords:** Shuffle automata, Simulation relation, Web service composition, Web service protocol synthesis.

# 1 Introduction

Web services is an emerging computing paradigm that tends to become the dominant technology for interoperation among autonomous and distributed applications in the Internet environment [1]. Informally, a *service* is a self-contained and platform-independent application (i.e., program) that can be described, published, and invoked over the network by using standards network technologies. In a nutshell, web services provide a mean to wrap, and expose over a network, a functionality (e.g., a program that accesses a database) via self-describing standard-based interfaces, thereby facilitating interoperability between disparate information systems that, at the origin, were not developed with the intent to be easily integrated. One of the ultimate goals of the web service technology is to enable rapid low-cost development and easy composition of distributed applications, a goal that has a long history strewn with only partial successes. To achieve this goal, there has been recently numerous research work [8, 16, 6, 11, 7, 5, 22] on the challenges associated with web service composition. The research problems involved by service composition are varied in nature and depends mainly on the kind of the composition process, i.e., manual v.s. automatic, on the model used to describe the services and on the issues related to composition. A line of demarcation between existing works in this area lies in the nature of the composition process : manual v.s. automatic. The first category of work deal generally with low-level programming details and implementation issues (e.g., WS-BPEL [2]) while automatic service composition focuses on different issues such as composition verification [8, 23, 14], planning [25, 24, 21] or synthesis [7, 5, 22].

In this paper we investigate the problem of automatic web service composition. We consider more particularly the composition synthesis problem, i.e., how to generate automatically a new target service by reusing some existing ones. We consider this problem at the web service business protocol abstraction level. A web service business protocol (or simply, a service protocol) is used to describe the external behavior of a service. Recent works have drawn attention to the importance of the state machine based formalisms for modeling the external behaviors of web services [3, 4, 6]. Continuing with this line of research, we build our work upon a formal framework where web service business protocols are described by means of Finite State Machines (FSM) and we concentrate on the following protocol synthesis problem : *given a set of $n$ available web service protocols $P_1, \ldots, P_n$ and a new target protocol $P_T$, can the behavior described by $P_T$ be synthesized by combining (parts of) the behaviors described by the available protocols.* This problem has already been addressed in recent literature [6, 7, 5, 22] under the restriction that the num-

ber of instances of an existing protocol that can be used in a composition is bounded and fixed *a priori*. We call this restricted form of the composition problem *the instance bounded protocol (composition) synthesis problem*. The pioneer work of [6, 7] shows that the problem of composition existence in this context can be reduced to that of testing the satisfiability of a Propositional Dynamic Logic formula. In [5], the PDL-based framework proposed in [6, 7] is extended to deal with a more expressive protocol model. Interestingly, in [22] composition existence problem is reduced to the problem of deciding whether there exists a simulation relation between the target protocol and the available ones. The authors build upon this reduction to prove the EXPTIME completeness of the bounded instance protocol synthesis problem.

It should be noted that the restricted setting considered in existing works has severe practical limitations that may impede the usage of automatic service composition by organizations. Indeed, as illustrated in section 3 of this paper, some very simple cases of web service composition cannot be solved in such a restricted setting.

**Contributions.** In this paper, we concentrate on the general case of protocol synthesis problem by relaxing the restriction on the number of protocol instances that can be used in a given service composition (i.e., we consider the unbounded instances case). Up to our knowledge, the decision problem underlying composition existence in such an unrestricted setting has been left open in recent related work. We show that this problem can be reduced to that of testing a simulation relation between an FSM and an (infinitely) iterated product of FSMs. Up to our knowledge, this later problem has never been investigated in the literature.

This paper makes the following contributions :

- We formalize the composition existence problem as that of checking simulation between a FSM and a (iterated) product closure of a FSM.
- We provide a suitable model, called Product Closure Automata (PCA), to describe (iterated) product closure of a FSM as an infinite state machine.
- Building upon our formal framework, we develop a novel technique to prove the decidability of testing simulation between a FSM and a PCA. As a consequence, our results solve the problem of web service composition (synthesis) existence in presence of an unbounded number of instances, a problem left open in recent related works.

**Relationship with formal language theory.** Our work is strongly connected with existing research dealing with issues related to shuffle languages, an extension of regular languages with shuffle and shuffle closure operators [13, 20]. It is well known that the class of shuffle languages is a proper subset of context sensitive-languages and can be recognized using

Linearly Bounded Automata (LBA). More specifically, [19] introduces the notion of shuffle automata which accept shuffle languages and can be simulated by one way nondeterministic Turing machine in logarithmic space. The simulation problem has never been addressed in this context. However, close decidability problems are rather negative. For example, we know from [18] that the universe problem for shuffle languages is undecidable and hence it is the case of the inclusion problem of a regular language in a shuffle one. The PCA model, used in this paper, is a particular case of shuffle automata which recognizes a shuffle closure of a regular language. Note that the universe and the emptiness checking problems are decidable in the context of PCAs. Although not detailed here, the technique presented in this paper can be used to prove the decidability of the inclusion problem between a regular language and a shuffle closure of a regular language. Finally, it is worth mentioning the existing work related to the problem of shuffle decompositions of regular languages [17, 10]. Given a language $L$, a challenging question is to express $L$ as a shuffle of two languages such that neither one of them is the singleton language of the empty word. In our case, we deal with a kind of dual problem, i.e., how to compose a given language using existing ones.

**Paper outline.** The remainder of the paper is organized as follows. Section 2 provides some basic definitions. Section 3 defines the service composition problem dealt with in this paper and points out the main theoretical and practical limitations of current state of the art. Section 4 contains the main technical contribution. It formally introduces Product Closure Automata, an infinite state machines that are used to describe (infinitely) iterated product of FSMs, and propose an algorithm that decide simulation between a FSM and a PCA. We draw some conclusions in Section 5.

# 2 Preliminaries

In this section, we recall some basic notions that will be useful for the rest of this paper.

**Definition 1** ***(FSM)*** *A Finite State Machine (FSM) $M$ is tuple $M =< \Sigma_M, S_M, F_M, q^0_M, \delta_M >$, where :*

- *$\Sigma_M$ is a finite set of alphabet,*
- *$S_M$ is a finite set of states,*
- *$F_M \subseteq S_M$ is the set of final states,*
- *$q^0_M \in S_M$ is the initial state, and*
- *$\delta_M \subseteq S_M \times \Sigma_M \times S_M$ is the transition relation of the state machine.*

Let $M =< \Sigma_M, S_M, F_M, q^0_M, \delta_M >$ be a FSM and $q \in S_M$ a state in $M$. We note by $SP(q)$ the shortest path leading from $q$ to a final state in $F_M$.

We provide below a definition of the notion of simulation relation between FSMs.

**Definition 2** ***(simulation relation)***

*Let $M =< \Sigma_M, S_M, F_M, q^0_M, \delta_M >$ and $M' =< \Sigma_{M'}, S_{M'}, F_{M'}, q^0_{M'}, \delta_{M'} >$ be two FSMs. A state $q_1 \in S_M$ is simulated by a state $q'_1 \in S_{M'}$, noted $q_1 \preceq q'_1$, iff :*

*(i) $\forall a \in \Sigma_M$ and $\forall q_2 \in S_M$ s.t. $(q_1, a, q_2) \in \delta_M$ there is $(q'_1, a, q'_2) \in \delta_{M'}$ s.t. $q_2 \preceq q'_2$, and*

*(ii) if $q_1 \in F_M$ then $q'_1 \in F_{M'}$.*

*$M$ is simulated by $M'$, noted $M \preceq M'$, iff $q^0_M \preceq q^0_{M'}$.*

Let $M =< \Sigma_M, S_M, F_M, q^0_M, \delta_M >$ and $M' =< \Sigma_{M'}, S_{M'}, F_{M'}, q^0_{M'}, \delta_{M'} >$ be two FSMs. The asynchronous **product** (or simply, product) of $M$ and $M'$, denoted $M \times M'$, is a FSM $< \Sigma_M \cup \Sigma_{M'}, S_M \times S_{M'}, F_M \times F_{M'}, (q^0_M, q^0_{M'}), \lambda >$ where the transition function $\lambda = \{((q, q'), a, (q_1, {q_1}')) : (q, a, q_1) \in \delta_M \text{ or } (q', a, {q_1}') \in \delta_{M'}\}$.

Let $\mathcal{R} = \{P_1, \ldots, P_n\}$ be a set of FSMs. We use $\odot(\mathcal{R})$ to denote the product of the FSMs in $\mathcal{R}$ (i.e., $\odot(\mathcal{R}) = P_1 \times \ldots \times P_n$). To make the presentation clearer, we assume that all the deterministic FSMs we are dealing in this paper recognize also the empty word. Note that, such an assumption does not impact the work presented here since any deterministic FSM $M$ can be transformed into another deterministic FSM $\tilde{M}$ which behaves exactly as $M$ and recognizes in addition the empty word. Now, with such an assumption at hand we have $\odot(\mathcal{R}) = \bigcup_{X \in 2^{\mathcal{R}}} \odot(X)$. In other words, $\odot(\mathcal{R})$ denotes the union of the asynchronous product of all the subsets of $\mathcal{R}$.

Let $k > 0$ be a positive integer. The $k$-iterated product of a state machine $M$ is defined by $M^{\otimes_k} = M^{\otimes_{k-1}} \times M$ with $M^{\otimes_1} = M$ .

A (iterated) **product closure** of an FSM $M$, noted $M^{\otimes}$, is an infinitely iterated product of $M$ (i.e, $M^{\otimes_{+\infty}}$). It is worth noting that for any finite positive integer $k$, the $k$-iterated product $M^{\otimes_k}$ is still an FSM. Unfortunately, this property does not hold in the case of product closures. As we will see later, a product closure of a FSM can be described by an infinite state machine.

# 3 Web services composition problem

In this section we define the service composition problem dealt with in this paper and we point out the main theoretical and practical limitations of

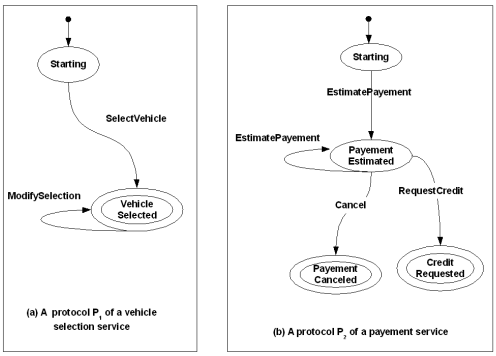


FIG. 1 – A repository of two business protocols.

current state of the art.

## Protocol model

We consider web services described by means of their protocols. A primary goal of a web service protocol is to describe the ordering constraints that govern message exchanges between a service and its clients (i.e., message choreography constraints). In this paper, we use the traditional state-machine formalism to represent message choreography constraints. States represent the different phases that a service may go through during its interaction with a requester. Transitions are triggered by messages sent by the requester to the provider or vice versa. Each transition is labeled with a message name. Usually the message names are followed by message polarity [27, 4] to denote whether the message is incoming (e.g., the plus sign) or outgoing (e.g., the minus sign). For simplicity reasons, and w.l.o.g., we do not consider message polarities in this paper (i.e., we do not make distinction between incoming and outgoing messages). Therefore, we obtain a web service protocol model very similar the so-called *Roman model* [6],i.e., a FSM where transitions are labeled by "abstract" activities. For instance, Figure 1(b) depicts the protocol of an hypothetical financing web service. The protocol specifies that the financing service is initially in the Start state, and that clients begin using the service by executing the activity estimate payment, upon which the service moves to the Payment Estimated state (transition EstimatePayement). In the figure, the initial state is indicated by an unlabeled entering arrow without source while final (accepting) states are double-circled.

As usual, and also w.l.o.g, we assume that protocols are deterministic FSMs. This is because non-determinism make protocols ambiguous in the

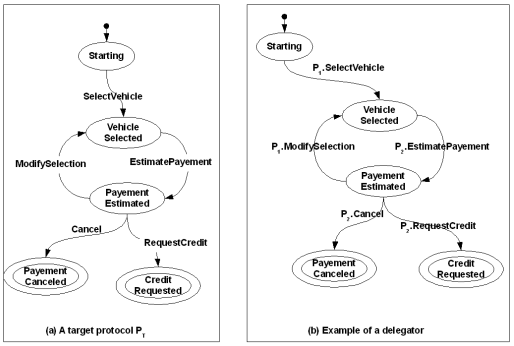


FIG. 2 – Example of protocol composition (synthesis).

sense that, for example, a service can move to a state that cannot be predicted by its client.

## The protocol synthesis problem

Let us now turn our attention to the web service composition problem. We first illustrate this problem on an example. We assume a repository of two available services $S_1$ and $S_2$, respectively, described by their protocols $P_1$ and $P_2$ depicted at Figure 1. We consider the development of a new web service $S_T$ whose protocol $P_T$, called a target protocol, is depicted at Figure 2(a). An interesting question is to see whether or not it is possible to implement the service $S_T$ by combining the functionality provided by the available services $S_1$ and $S_2$. Dealing with this composition problem at the business protocol abstraction level, leads to the following question : is it possible to generate the protocol $P_T$ by combining (parts of) the available protocols $P_1$ and $P_2$. In our illustrative case the answer is yes and an example of the composition of the target protocol $P_T$ using the protocols $P_1$ and $P_2$ is depicted at Figure 2(b). In this case, $P_T$ is called the target protocol while $P_1$ and $P_2$ are called the component protocols. Informally speaking, the service composition (or *protocol synthesis*) problem is defined in [6, 7] as the problem of generating a *delegator* of a target service using available services. A delegator is a FSM where the activities are annotated with suitable *delegations* in order to specify to which component each activity of the target service is delegated. Continuing with our example, Figure 2(b) shows a *delegator* that enables to *compose* the protocol $P_T$ using the available protocols $P_1$ and $P_2$ of Figure 1. For instance, this delagator specifies that the activity selectVehicle of the target protocol is delegated to the protocol $P_1$ while the activity estimatePayment is delegated to the protocol $P_2$.

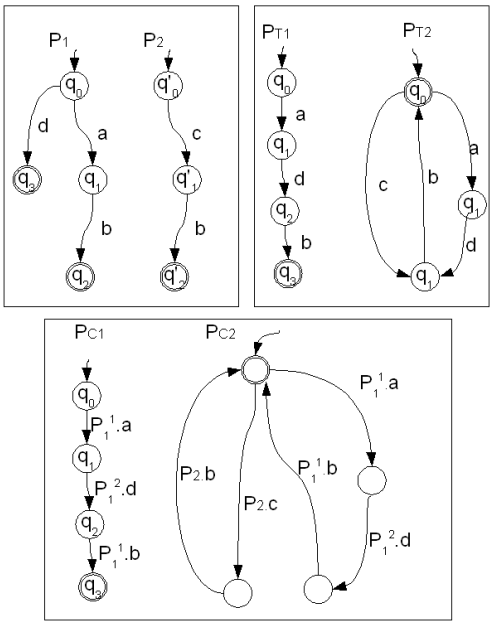


FIG. 3 – Instance Bounded Limitations

The notion of a delegator is defined formally in [6, 7] and the composition synthesis problem is expressed as the problem of finding a *"correct"* delegator for a given target protocol using a set of available protocols. A crucial question regarding this problem lies in the number of instances of the available services that can be used in a composition (i.e., to build a delegator). Figure 3 shows two examples of delegators, namely $P_{C1}$ and $P_{C2}$, that use several instances of available services to respectively compose target protocols. More precisely, the delegator $P_{C1}$ uses two instances of the protocol $P_1$, namely $P_1^1$ and $P_1^2$, to compose the target protocol $P_{T1}$. The delegator $P_{C2}$ uses however (infinitely) many instances of the protocols $P_1$ and $P_2$ to compose the protocol $P_{T2}$. Indeed, each execution of the loop $a.d.b$ (respectively, $c.b$) of the target protocol $P_{T2}$ is realized by two new instances of the available protocol $P_1$ (respectively, one new instance of $P_2$).

We provide below a definition of a generic protocol synthesis problem that makes explicit the number of instances of protocols allowed in a composition. Let $\mathcal{R}$ be a repository of services protocols, i.e., $\mathcal{R} = \{P_i,\ i \in [1, n]\}$, where each $P_i =< \Sigma_i, S_i, F_i, s_i^0, \delta_i >$ is a protocol. For each $P_i \in \mathcal{R}$, we denote by $P_i^j$ the $j^{th}$ copy of the protocol $P_i$. Given a protocol repository $\mathcal{R}$, we note by $\mathcal{R}^m = \bigcup_{i=1}^{n} \{P_i^1, \ldots, P_i^m\}$, with $m \in \mathbb{N}$.

**Definition 3** ***generic protocol composition problem*** *Let $\mathcal{R}$ be a set of*

*available service protocols and $P_T$ be a target protocol and let $k \in \mathbb{N}$. A (generic) protocol synthesis problem, noted $Compose(\mathcal{R}, S_T, k)$ is the problem of deciding whether there exist a composition of $P_T$ using $\mathcal{R}^k$.*

Note that, instances of this generic composition problem are characterized by the maximal number of instances of component protocols that are allowed to be used in a given composition. We distinguish in the following between two main cases, namely the bounded instance and the unbounded instance ones.

**Protocol synthesis problem : the bounded case.** Exiting work [6, 7, 5, 22] that investigated the protocol synthesis problem make the simplifying assumption that the number of instances of a service that can be involved in the composition of a target service is bounded and fixed *a priori*, i.e., they address the problem $Compose(\mathcal{R}, S_T, k)$ with $k$ finite and known *a priori*. Note that this particular case, called the *bounded instance protocol synthesis problem*, can be reduced w.l.o.g to the simplest case where $k = 1$. Indeed, if $k > 1$ the problem $Compose(\mathcal{R}, S_T, k)$ can be straightforwardly reduced to the problem $Compose(\mathcal{R}^k, S_T, 1)$.

The main idea described in [6, 7] consists in reducing the problem $Compose(\mathcal{R}, S_T, 1)$ into satisfiability of a suitable formula of Deterministic Propositional Dynamic Logic (DPDL) [15]. Interestingly, in [22] the protocol synthesis problem is reduced to the problem of testing a simulation relation between the target protocol and the product of the existing protocols. Using such a reduction, [22] shows the EXPTIME completness of this problem[4].

It is worth noting that the setting of bounded instances is very restrictive in the sense that some simple protocol synthesis problems, in which the solution may use an infinite number of instances of component protocols, cannot be solved. As an example, the problem depicted at Figure 3, and which consists in the synthesis of the target protocol $P_{T1}$ using the available protocols $P_1$ and $P_2$, cannot be solved by current state of the art approaches although a solution (i.e., the delegator $P_{C2}$) is not complex to construct. These strong limitations motivated our work on the unbounded instance case of the protocol synthesis problem.

**Protocol synthesis problem : the unbounded case.** In the remainder of this paper we study the protocol synthesis problem in the case where the number of protocol instances that can be used in a composition may be infinite (i.e., the problem $Compose(\mathcal{R}, S_T, +\infty)$). In the same spirit of [22], we formalize the protocol synthesis problem as that of testing the existence of a simulation relation. More precisely, given a repository $\mathcal{R} = \{P_1, \ldots, P_n\}$

[4]The EXPTIME upper bound is known from [6].

of service protocols, we consider the generation of new composite protocols that can be obtained by an asynchronous product of any subset of protocols in $\mathcal{R}$. The following proposition can easily be derived from existing results given in [6, 7, 5, 22].

**Proposition 1** *Let $Compose(\mathcal{R}, S_T, k)$ be a protocol synthesis problem with $k$ a finite positive integer fixed a priori. The problem $Compose(\mathcal{R}, S_T, k)$ has a solution iff $S_T \preceq \odot(\mathcal{R}^k)$ (or equivalently, iff $S_T \preceq (\odot(\mathcal{R}))^{\otimes_k}$).*

More precisely, we consider in this paper the decision problem underlying the general protocol synthesis problem, i.e., the problem $Compose(\mathcal{R}, S_T, +\infty)$.

**Problem 1** *Let $\mathcal{R}$ and $S_T$ defined as previously. Is the problem $Compose(\mathcal{R}, S_T, +\infty)$ decidable ?*

This problem has been left open in current related works. One way to answer this open question is to consider the related 'simulation relation' decision problem, i.e., is it decidable whether $S_T$ is simulated by $(\odot(\mathcal{R}))^{\otimes}$ ?

Up to our knowledge, simulation relation has never been studied in the context of product closure of FSMs. Since $\odot(\mathcal{R})$ is an FSM, we shall prove in next section that checking simulation between an FSM $A$ and a product closure of an FSM $M$ (i.e., $M^{\otimes}$) is decidable. This enables to derive the decidability of the protocol synthesis problem.

# 4 Composition decidability problem

In this section we are interested by the problem of testing the existence of a simulation relation between an FSM and a product closure of an FSM. To investigate this problem, we need first to define a state machine model that enables to describe a product closure. Shuffle automata, introduced in [19] to recognize the so-called shuffle languages, are an example of a candidate model that can be used to describe product closures of FSMs. However, as we deal only with a specific form of shuffle automata, i.e., automaton of the form $M^{\otimes}$ where $M =< \Sigma_M, S_M, F_M, q_M^0, \delta_M >$ is an FSM, we use in our work a simpler tool inspired from [26, 22]. Informally, the product closure $M^{\otimes}$ enables to run an infinite number of parallel instances of $M$. An instantaneous description of $M^{\otimes}$ may then be described by an FSM similar to $M$ with a (infinite) stack of tokens in each state. The size of the stack describes the number of parallel instances having reached that state. Let $w \in \Sigma_M^*$ the input of $M^{\otimes}$, a symbol $a \in w$ is recognized by the execution of such a state machine in two cases :

– (creation of a new instance of $M$) if there is an outgoing transition labeled $a$ from the initial state $q_M^0$ of $M$ to a state $q$. Upon such a transition, a token is added to $q$, or
– (moving an existing instance of $M$) if there exists two states $q$ and $q'$ such that $(q, a, q') \in \delta_M$ and $q$ has one or more tokens, then upon this transition, a token is moved from $q$ to $q'$.

Before providing a formal definition of the state machine $M^{\otimes}$, we first define below the notions of intermediate and hybrid states of an FSM $M$ which will be useful in the remainder of this paper. Let $M =< \Sigma_M, S_M, F_M, q_M^0, \delta_M >$ be a FSM. Then :

– The set of *hybrid states* of $M$, denoted $H_s(M)$, contains all the final states of $M$ that have at least one outgoing transition. Formally, $H_s(M) = \{q \in F_M \mid \exists q' \in S_M, a \in \Sigma_M, (q, a, q') \in \delta_M\}$.
– The set of *intermediate states* of $M$, denoted $I_s(M)$, contains the states of $S_M \backslash F_M$ that have at least one incoming and one outgoing transitions. Formally, $I_s(M) = \{q \in S_M \setminus F_M \mid \exists q', q'' \in S_M, a \in \Sigma_M, (q, a, q') \in \delta_M$ and $(q'', b, q) \in \delta_M\}$.

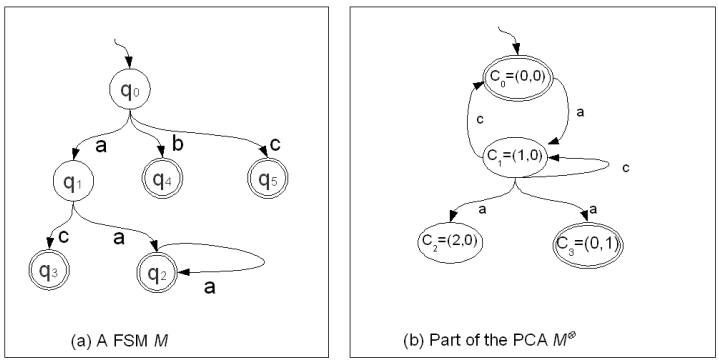


FIG. 4 – Example of an FSM $M$ and a part of its associated PCA $M^{\otimes}$.

For example, the intermediate states of the FSM $M$ depicted at Figure 4(a) are $I_s(M) = \{q_1\}$ while its hybrid states are $H_s(M) = \{q_2\}$

## Product Closure Automaton (PCA).

We provide now a formal definition of a state machine, called hereafter a *PCA* (Product Closure Automaton), that enables to describe the product closure of an FSM. A PCA is an infinite state machine which enables to describe : (i) all possible executions of a product closure of an FSM, and *(ii)* the branching choices at each state of the execution of such an automaton.

We introduce below the notion of a configuration which provides an *instantaneous description (ID)* of a PCA. Unlike finite state automata, where the ID of a given automaton is given by its current state, an ID of a PCA involves the set of its states as well as the number of tokens in each state (number of instances having reached that state when recognizing a word). Let $M =< \Sigma_M, S_M, F_M, q_M^0, \delta_M >$ be an FSM and let $|I_s(M)| = l$ and $|H_s(M)| = n$. We assume states of $I_s(M)$ (respectively, $H_s(M)$) ordered according to the lexicographical order and relabeled accordingly with integers from 1 to $l$ (respectively, from $l+1$ to $l+n$). The configurations of $M^{\otimes}$ are formally defined below.

**Definition 4** ***(Configuration)*** *A configuration $C$ of the product closure $M^{\otimes}$ is a tuple of size $l+n$ of (may be infinite) integers. The $i^{th}$ element of $C$, written $C[i]$, denotes the number of tokens (i.e., instance of $M$) that are at state $i$. We say that $C[i]$ is the witness of the state $i$ in a configuration $C$. Note that, if $i \leq l$ (respectively, $i > l$) then $C[i]$ is a witness of an intermediate state (respectively, an hybrid state).*

*A configuration $C$ is an initial configuration of $M^{\otimes}$ if $C[i] = 0$, $\forall i \in [1, l+n]$ and $C$ is a final configuration if $C[i] = 0$, $\forall i \in [1, l]$.*

Note that, a configuration keeps only the information about intermediate and hybrid states. Indeed, it is useless to store information about the number of tokens (i.e., instances of $M$) that are in final, not hybrid, states. In the same spirit, as the number of instance of $M$ that can be created is infinite (i.e., the set of tokens in the initial state is infinite) we do not describe the initial state in a configuration unless it is also an intermediate state.

Continuing with the example of Figure 4, the FSM $M$ contains only one intermediate state (state $q_1$) and one hybrid state (state $q_2$). Hence, a configuration associated with $M^{\otimes}$ is a pair of integers where the first (respectively, the second) integer is the witness of the state $q_1$ (respectively, $q_2$). For instance, a configuration $C = (2, 3)$ indicates an instantaneous description of $M^{\otimes}$ in which there are two instances of $M$ at state $q_1$ and three instances at state $q_2$.

We define below two partial orders on configurations.

**Definition 5** ***(Configuration inclusion and cover)*** *Let $C_1$ and $C_2$ be two configurations of size $m$ of $M^{\otimes}$. We define the two following partial orders on configurations.*

- *Inclusion. $C_1 \subseteq C_2$, iff $C_1[i] \leq C_2[i] \forall i \in [1, m]$,*
- *Cover. $C_1 \lhd C_2$, iff ($C_1 \subseteq C_2$ and $C_1[i] = C_2[i]$, $\forall i \in [1, |I_s(M)|]$). In this case, we say that $C_1$ is covered by $C_2$.*

Using the notion of configuration, we formally define below PCAs.

**Definition 6** *Let $M =< \Sigma_M, S_M, F_M, q_M^0, \delta_M >$ be a FSM with $|I_s(M)| = l$ and $|H_s(M)| = n$. The associated PCA of $M$ is an infinite state machine $M^{\otimes} =< \Sigma_M, \mathcal{C}, F_C, C_0, \phi >$, where :*

- *$\mathcal{C}$ is an (infinite) set of states consisting of all the configurations of $M^{\otimes}$,*
- *$F_C$ is the set of final configurations of $M^{\otimes}$, i.e., $\{C \in \mathcal{C} \mid C[i] = 0, \forall i \in [1, l]\}$,*
- *$C_0$ is the initial state of $M^{\otimes}$ and corresponds to the initial configuration, i.e., $C_0[i] = 0$, $\forall i \in [1, l + n]$,*
- *$\phi \subseteq \mathcal{C} \times \Sigma_M \times \mathcal{C}$ is an infinite set of transitions. The set $\phi$ is built as follows. Let $C_1$ and $C_2$ be two configurations in $\mathcal{C}$. We have $(C_1, C_2, a) \in \phi$ if $(q, a, q') \in \delta_M$ and one of the following conditions holds :*
  - *$q = q_M^0$ and $q' \in (F_M \backslash H_s(M))$ with $C_1[i] = C_2[i]$, $\forall i \in [1, l + n]$, or*
  - *$q = q_M^0$ and $q' \in (I_s(M) \cup H_s(M))$ with $C_2[q'] = C_1[q'] + 1$, $C_1[i] = C_2[i]$, $\forall i \in [1, l + n]$ and $i \neq q'$, or*
  - *$\{q, q'\} \subseteq (I_s(M) \cup H_s(M))$ with $C_1[q] > 0$, $C_2[q] = C_1[q] - 1$, $C_2[q'] = C_1[q'] + 1$, $C_1[i] = C_2[i]$, $\forall i \in [1, l + n]$ and $i \notin \{q, q'\}$, or*
  - *$q \in (I_s(M) \cup H_s(M))$ and $q' \in (F_M \backslash H_s(M))$ with $C_2[q] = C_1[q] - 1$, $C_1[i] = C_2[i]$, $\forall i \in [1, l + n]$ and $i \neq q$.*

Figure 4(b) describes a part of $M^{\otimes}$, the PCA of the FSM $M$ depicted at Figure 4(a). As mentioned before, configurations of $M^{\otimes}$ are pairs $(i, j)$ where $i$ (respectively, $j$) is the witness of the state $q_1$ (respectively, $q_2$). The infinite state machine $M^{\otimes}$ is initially in the configuration $C_0 = (0, 0)$ then it can, for example, execute the activity $a$, upon which it moves to the configuration $C_1 = (1, 0)$. At this stage, $M^{\otimes}$ has two possibilities to execute the activity $c$ : (i) by moving the current instance of $M$ that is at state $q_1$ into the final state $q_3$, or (ii) by creating a new instance of $M$ and moving it from state $q_0$ into state the final $q_5$. Note that, as the final states $q_3$ and $q_5$ are not described in configurations, case (i) make the $M^{\otimes}$ moving back to the configuration $C_0$ while case (ii) makes it looping on configuration $C_1$.

## Simulation existence decision problem.

We are now able to state formally the simulation decision problem we are interested in.

**Problem 2** *Let $A$ and $M$ be two FSMs. Is it decidable whether $A \preceq M^{\otimes}$ (or, equivalently, is decidable whether $q_A^0 \preceq C_0$ ?).*

This section answers positively to this problem by providing a sound and complete algorithm that checks the existence of a simulation relation between a FSM and a product closure of a FSM.

Note that, the main difficulty to devise our algorithm comes from the fact that we have to check the existence of a simulation relation between an FSM and a PCA, this later one being an infinite state machine. The corner stone of our proof is to show that to check the existence of such a simulation relation we need only to explore a finite part of the corresponding PCA. We propose an algorithm made of three main parts : Check-Sim, Check-Candidate and Check-Cover. When checking the simulation between a given state $q$ and a configuration $C$, the Check-Sim procedure will recursively generate new simulation tests by making calls to the Check-Candidate procedure for each transition $(q, a, q')$ in $A$. This later procedure enables to check if the state $q'$ is simulated by at least one configuration $C'$ such that $(C, a, C')$ is in $M^{\otimes}$. Informally speaking, the execution of the algorithm can be seen as a tree where the nodes are labeled with pairs $(q, C)$ and correspond to the calls of the Check-Sim algorithm. As an example, Figure 5(b) shows an execution of a Check-Sim between the initial state $q_1$ of the FSM of Figure 5(a) and the initial configuration $C_0 = (0, 0)$ of the product closure of the FSM of Figure 4(a).

A crucial question is then to ensure that the algorithm terminates. Observe that for each state $q'$, the number of candidates $C'$ generated by the Check-Candidate procedure is linear in the size of $M$ since for any configuration $C$ of a PCA $M^{\otimes}$, the number of outgoing transitions is finite and bounded by the total number of transitions in $M$. Therefore, to ensure termination of the algorithm it remains to show that there are no infinite branches in the execution tree of the algorithm. In the simple case where $A$ is a loop-free FSM, it is easy to see that the corresponding execution tree of the algorithm is finite since the length of the branches are bounded by the size of the maximal path in $A$. For the general case, a state $q$ belonging to a loop in $A$ may appear (infinitely) many times in a branch of the execution tree of the algorithm. Such a case is illustrated on the Figure 5(b) where the branch depicted in bold involves many times the state $q_1$ which belong to the loop $(a.b)^*$ of the FSM $A$. An important technical contribution of this work is to provide necessary and sufficient conditions that enable to cut early such infinite branches. This is achieved by the second terminating condition of the Check-Sim (i.e., the call to the Check-Cover procedure) which is based on the following property : if a state $q$ appears infinitely many times in a given branch then there is necessarily a sub-path in this branch from a node $(q, C)$ to a node $(q, C')$ such that $C'$ is a cover of $C$. Interestingly, this condition characterizes the cases where a loop in $A$ is simulated by $M^{\otimes}$.

Continuing with the example of Figure 5(b), the bold branch which is potentially infinite is cut at node $(q_1, (0,1))$ since the configuration $(0,1)$ is a cover of the configuration $(0,0)$ which appear previously in a node $(q_1, (0,0))$ in the same branch. Note that, to verify such a condition, the Check-Cover procedure maintains for each state $q$ in a given branch a list, noted $L(q)$, of all the configurations $C'$ corresponding to the nodes $(q, C')$ of this branch. In our example, we have at node $(q_1, (0,1))$ of the bold branch the sequence $L(q_1) = [(0,0), (1,0)]$.

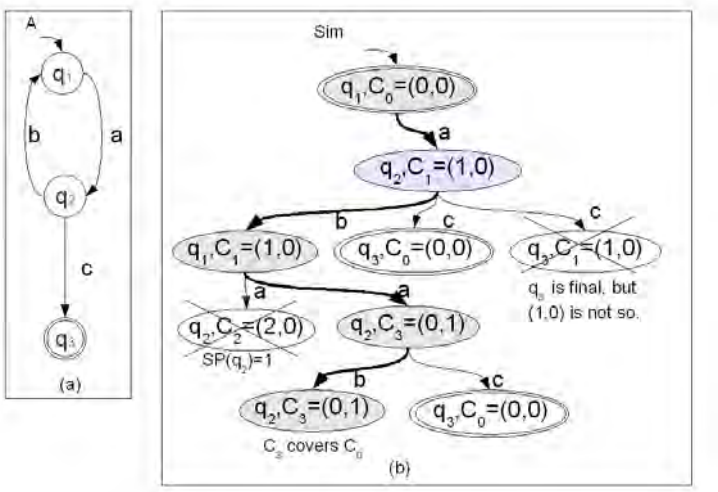


FIG. 5 – Simulation

**Algorithm 1**: Check-Sim

**Input**: Two FSM $A$ and $M$, a state $q$ of $A$, a configuration $C$ of $M^{\otimes}$
**Output**: boolean
**begin**
  **if** $q \in F_A \setminus H_s(A)$ **then**
    return($\sum_{i=1}^{|I_s(M)|} C[i] = 0$);
  **if** *Check-Cover(q,C)* **then**
    Return(true);
  **for** *each transition* $(q, a, q')$ *in* $\delta_A$ **do**
    **if** *not(Check-Candidate*$(q', C, a)$*)* **then**
      return(false);
  return(true);
**end**

**Algorithm 2**: Check-Candidate

**Input**: a state $q'$ of $A$, a configuration $C$ of $M^{\otimes}$, $a \in \Sigma_M$
**Output**: boolean
**begin**
  Candidates=$\emptyset$;
  **for** *each transition* $(C, a, C')$ *in* $\phi$ **do**
    **if** $\sum_{i=1}^{|I_s(M)|} C'[i] \leq SP(q')$ **then**
      Candidates= Candidates $\cup\{C'\}$;
  flag=0;
  **while** *Candidates*$\neq \emptyset$ *and (not flag)* **do**
    $C'$ =first element in Candidates;
    flag= Check-Sim(q',C');
  return(flag);
**end**

**Algorithm 3**: Check-Cover

**Input**: a state $q$ of $A$, a configuration $C$ of $M^{\otimes}$
**Output**: boolean
**begin**
  **for** $C' \in L(q)$ **do**
    **if** $C' \triangleleft C$ **then**
      return(true);
  return(false);
**end**

## Correction of the algorithm.

In the remainder of this section, we shall prove that the procedure Check-Sim halts and is sound and complete.

**Theorem 1** *The algorithm Check-Sim halts.*

**Proof 1** *Let us suppose that the procedure Check-Sim does not halt, i.e. there exists an infinite branch in its execution tree. This means that a given state $q \in S_M$ may appear infinitely many times in this branch, i.e. $|L(q)|$ is infinite. Thus $L(q)$ corresponds to a cover-free sequence $(C_i)_{i\in\mathbb{N}}$ of configurations,i.e. $\forall j, k \in \mathbb{N}\ j < k \Rightarrow C_j \not\triangleleft C_k$.*

*Since the sum of tokens in intermediate state are bounded by $SP(q)$ then $(C_i)_{i\in\mathbb{N}}$ may be split into a finite number of sub-sequences $(C_{ik})_{i_k\in\mathbb{N}}$, such*

*that for all* $C, C' \in (C_{ik})_{k\in\mathbb{N}}$, $C[j] = C'[j]\ \forall j \in [1, l]$ *. In other words, each* $(C_{ik})_{i_k\in\mathbb{N}}$ *represents a sequence of configurations where the witnesses for intermediate states are the same. Hence,* $(C_i)_{i\in\mathbb{N}}$ *is an infinite cover-free sequence of configurations iff* $(C_{ik})_{i_k\in\mathbb{N}}$ *is an infinite sequence of configurations without inclusion, i.e.* $\forall C_1, C_2 \in (C_{ik})_{i_k\in\mathbb{N}}\ \ C_1 \lhd C_2 \Leftrightarrow C_1 \subseteq C_2$.

*Hereafter, we will prove that such a sequence* $(C_{ik})_{i_k\in\mathbb{N}}$ *of configurations without inclusion cannot exist. This proof is based on the following lemma established by [9] and reported in [12].*

***Lemma 1*** *Let* $n$ *be any integer such that* $n > 1$*. Given any infinite sequence* $(C_i)_{i\geq 1}$ *of* $n$*-tuples of natural numbers, there exists positive integers* $i, j$ *such that* $i < j$ *and* $C_i \preceq_n C_j$*, where* $\preceq_n$ *is the partial order on* $n$*-tuples of natural numbers induced by the natural ordering* $\leq$ *on* $\mathbb{N}$*.*

*Lemma1 states that there does not exist an infinite sequence of configurations without inclusion. Thus* $(C_{ik})_{i_k\in\mathbb{N}}$ *is not infinite without inclusion. Thus we conclude that* $(C_i)_{i\in\mathbb{N}}$ *can not be an infinite without cover and therefore the procedure* Check-Sim *halts.*

**Theorem 2** *The Algorithm* Check-Sim *is correct.*

**Proof 2**
- *Soundness. Suppose that Algorithm* Check-Sim *returns true. We show that there exists simulation between* $q_0$ *and* $C_0$*, and thus, there exists simulation between* $A$ *and* $M^{\otimes}$*. Let us consider a call to the Algorithm* Check-Sim *with* $q$ *a state of* $A$ *and* $C$ *a configuration of* $M^{\otimes}$*. We can distinguish three acceptance cases :*
  - $q \in F_A \setminus H_s(A)$ *and* $(\sum_{i=1}^{|I_s(B)|} C[i] = 0$*; i.e.* $C$ *is final). Then* $q \preceq C$*.*
  - *For each transition* $(q, a, q') \in \delta_A$ *:* $q'$ *is simulated by a given* $C'$*. Then* $q \preceq C$*.*
  - $Cover(q, C) = 1$*. This case represents the difference between our algorithm and classic simulation algorithms. It corresponds to an execution of a loop in* $A$ *which go through* $q$*. That is to say, there exists a sub-path in the execution tree from* $(q, C)$ *to* $(q, C')$ *such that* $C \lhd C'$*. This cover condition allows us to avoid the test of simulation between* $q$ *and* $C'$*, because* $C'$ *possesses the same number of tokens as* $C$ *on intermediate states of* $M$ *and more tokens than* $C$ *for hybrid states of* $M$*. Since* $C'$ *and* $C$ *need to simulate the same state* $q$*, we can restrict* $C'$ *to be equal to* $C$ *by deleting the extra tokens in hybrid states of* $C'$*.*
- *Completness. Now suppose that Algorithm* Check-Sim *returns false. First we show that Algorithm* Check-Sim *looks for all the possibilities to simulate the state* $q$ *by a configuration* $C$*. In order to simulate* $q$ *by* $C$*,*

*the Algorithm* Check-Sim *checks for each transition* $(q, a, q')$ *in* $\delta_A$ *if* $q'$ *can be simulated by a configuration* $C'$ *such that* $(C, a, C') \in \phi$. *The Algorithm* Check-Candidate *computes all configurations that may be candidate to simulate* $q'$. *Candidates that do not satisfy the condition* $\sum_{i=1}^{|I_s(B)|} C'[i] \leq SP(q')$ *are rejected. Indeed, these configurations cannot simulate* $q'$, *since there exists a path from* $q'$ *to a final state in* $F_A$ *such that the tokens on the intermediate states of* $C'$ *cannot be all consumed. From the list of candidates, the Algorithm Check-Candidate try to find a candidate configuration that simulates* $q'$. *The algorithm returns false if no such a configuration exist.*

*Now suppose that the Algorithm* Check-Sim *returns false. We distinguish two cases :*

- $q \in F_A \setminus H_s(A)$ *and* $\sum_{i=1}^{|I_s(B)|} C[i] \neq 0$. *This means that* $q$ *is a final state and* $C$ *is not a final state. Thus* $q$ *cannot be simulated by* $C$.
- *There is a transition* $(q, a, q')$ *in* $\delta_A$ *such that the state* $q'$ *cannot be simulated. Since all candidate configurations* $C'$ *such that* $(C, a, C') \in \phi$ *are checked, we conclude that* $q$ *cannot be simulated by* $C$.

*We conclude that Algorithm* Check-Sim *is correct.*

It is worth noting that the proposed proof is constructive in the sense that if the answer is true, the algorithm may be easily modified to exhibit a simulation relation between its inputs. This is an interesting point in the context of the protocol synthesis problem since such a simulation relation can be effectively used to build a delegator.

# 5 Conclusion

We have studied the web service protocol synthesis problem in the general case where the number of protocol instances that can be used in a composition is unbounded. We made a reduction of this problem to that of of checking simulation between a FSM and a product closure of a FSM. To cope with this later problem, we first proposed PCAs as a suitable tool for describing the behavior of a product closure of an FSM and built upon this formal framework to prove the decidability of checking the simulation relation between a FSM and a PCA.

As a perspective of this work, we point out several interesting issues :

- the algorithmic issues related to the optimization of the proposed algorithm as well as the development of suitable implementation strategies,
- complexity, by identifying particular cases that either reduce the complexity of the problem or can be solved using classical simulation algorithms,

– extension of our technique to more expressive models that enable for example modeling message exchanges and impacts on the real world such as the *Colombo* model [5],
– application of our technique to tackle close problems in the formal language theory, as for example, identifying restricted classes of shuffle automata for which fundamental problems such as languages inclusion and the universe problems are decidable.

# Références


[1] Gustavo Alonso, Fabio Casati, Harumi A. Kuno, and Vijay Machiraju. *Web Services - Concepts, Architectures and Applications*. Springer, 2004.

[2] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, and others. « Business Process Execution Language for Web Services, Version 1.1 ». *Specification, BEA Systems, IBM Corp., Microsoft Corp., SAP AG, Siebel Systems*, 2003.

[3] Boualem Benatallah, Fabio Casati, and Farouk Toumani. « Web Service Conversation Modeling : A Cornerstone for E-Business Automation ». *IEEE Internet Computing*, 08(1) :46–54, 2004.

[4] Boualem Benatallah, Fabio Casati, and Farouk Toumani. « Representing, analysing and managing web service protocols ». *Data Knowledge. Engineering*, 58(3) :327–357, 2006.

[5] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Richard Hull, and Massimo Mecella. « Automatic Composition of Transition-based Semantic Web Services with Messaging ». In *VLDB*, pages 613–624, 2005.

[6] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Massimo Mecella. « Automatic Composition of E-services That Export Their Behavior. ». In *ICSOC*, pages 43–58, Dec. 2003.

[7] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Massimo Mecella. « Automatic Service Composition Based on Behavioral Descriptions ». *Int. J. Cooperative Inf. Syst.*, 14(4) :333–376, 2005.

[8] T. Bultan, X. Fu, R. Hull, and J. Su. « Conversation specification : a new approach to design and analysis of e-service composition ». In *WWW'03*. ACM, 2003.

[9] Leonard Eugene Dickson. « Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. ». *Amer. Journal Math*, 35 :413-422, 1913.

[10] M. Domaratzki and K. Salomaa. « RESTRICTED SETS OF TRAJECTORIES AND DECIDABILITY OF SHUFFLE DECOMPOSITIONS ». *International Journal of Foundations of Computer Science*, 16(5) :897–912, 2005.

[11] Schahram Dustdar and Wolfgang Schreiner. « A survey on web services composition ». *International Journal of Web and Grid Services*, 1(1) :1–30, 2005.

[12] Jean H. Gallier. « What's so special about Kruskal's theorem and the ordinal $\Gamma_0$ ?A survey of some results in proof theory ». *Annals of Pure and Applied Logic*, 53 :199–260, 1991.

[13] Jay Gischer. « Shuffle languages, Petri nets, and context-sensitive grammars ». *Communications of the ACM*, 24(9) :597–605, 1981.

[14] R. Hamadi and B. Benatallah. « A Petri net-based model for web service composition ». *Proceedings of the Fourteenth Australasian database conference on Database technologies 2003-Volume 17*, pages 191–200, 2003.

[15] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic logic Foundations of computing.* the MIT Press, 2000.

[16] Richard Hull, Michael Benedikt, Vassilis Christophides, and Jianwen Su. « E-services : a look behind the curtain ». In *PODS, San Diego, USA*. ACM, Jun' 03.

[17] M. Ito. « Shuffle decomposition of regular languages ». *Journal of Universal Computer Science*, 8(2) :257–259, 2002.

[18] Kazuo Iwama. « The universe problem for unrestricted flow languages ». *Acta Informatica*, 19(1) :85–96, 1982.

[19] Joanna Jedrzejowicz and Andrzej Szepietowski. « Shuffle languages are in P. ». *Theor. Comput. Sci.*, 250(1-2) :31–53, 2001.

[20] Joanna Jkedrzejowicz. « Undecidability results for shuffle languages ». *Journal of Automata, Languages and Combinatorics*, 1(2) :147–159, 1996.

[21] S. McIlraith and T.C. Son. « Adapting Golog for Composition of Semantic Web Services ». *Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning (KR2002)*, pages 482–493, 2002.

[22] Anca Muscholl and Igor Walukiewicz. « A lower bound on web services composition ». In *Proceedings FOSSACS*, volume 4423 of *LNCS*, pages 274–287. Springer, 2007.

[23] S. Narayanan and S.A. McIlraith. « Simulation, verification and automated composition of web services ». *Proceedings of the eleventh international conference on World Wide Web*, pages 77–88, 2002.

[24] M. Pistore, P. Traverso, and P. Bertoli. « Automated Composition of Web Services by Planning in Asynchronous Domains ». *Proc. ICAPS'05*, 2005.

[25] P. Traverso and M. Pistore. « Automated Composition of Semantic Web Services into Executable Processes ». *International Semantic Web Conference*, 3298 :380–394, 2004.

[26] Manfred K. Warmuth and David Haussler. « On the complexity of iterated shuffle. ». *Journal of Computer and System Sciences*, 28(3) :345–358, 1984.

[27] D.M. Yellin and R.E. Storm. « Protocol Specifications and Component Adaptors ». *ACM Trans. Program. Lang. Syst.*, 19(2) :292–333, March 1997.