IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 18, 2018, accepted December 5, 2018, date of publication December 10, 2018,
date of current version January 7, 2019.

Digital Object Identifier 10.1109/ACCESS.2018.2885948

Composition of Partially-Observable Services

HIKMAT FARHAT
Computer Science Department, Notre Dame University-Louaize, Zouk Mosbeh, Lebanon
e-mail: hfarhat@ndu.edu.lb

ABSTRACT In this paper, we tackle the problem of controlling the behavior of independent, partially observable services so that they collectively achieve a desired behavior (specification). The solution consists of synthesizing an orchestrator to coordinate the actions of the services, modeled as labeled transition systems, while having partial knowledge about their states. We show that an orchestrator exists only if the set of services is controllable where controllability is defined in terms of a set of message controllable relations. We present two algorithms to solve the problem, prove their correctness, and study their complexity. One of them is a novel backtracking algorithm that builds the solution incrementally, which makes it suitable as a starting point for developing heuristics to solve this hard problem. The results of the backtracking algorithm on a test case are promising.

INDEX TERMS Automated planning, behavior composition, supervisory control, partial observation, web services.

I. INTRODUCTION

The composition of a set of independent services (or agents) to collectively perform a sequence of actions that none of them can perform individually is a recurring problem in service oriented architecture and multi-agent systems [3], [6]. Such a composition is typically performed by an *orchestrator* (controller) that communicates with the services to accomplish the required task. The composition problem is therefore the problem of synthesizing an orchestrator given a specification and a collection of services. The composition problem has been studied extensively [7], [10], [16], [18], [23] but most of the work to date has assumed that the orchestrator has perfect information about the state of the services. In some situations, particularly if the services are from different service providers, the orchestrator does not have complete information about the state of the agents and this situation is termed partial observation.

The service composition when the orchestrator has partial observation has a higher complexity than the full observation case which is already EXPTIME-hard [15]. This is because, typically, one transforms the partial observation case to the full observation case by considering subsets of states called *belief states* [12]. Such a construction is exponential in the worst case and it is performed *prior* to any synthesis even if the problem has no solution. Furthermore, even in the full observation case most of the algorithms to date use a backward search strategy which necessitates the visitation of all states. In backward search the starting point is the whole state space and states are removed iteratively until a fixed-point is reached. Such strategy is convenient in the case of supervisory control of discrete event systems [17] because it directly leads to the synthesis of the *most permissive* controller (''union'' of all possible controllers) even if there are other, less permissive, controllers. We have shown that the composition problem is equivalent to the supervisory control problem [9] but there is one *key difference*. In the case of service composition the *most permissive* orchestrator is not strictly required and any orchestrator that matches the requirement is also a solution. Therefore in the case of service composition building the solution incrementally, which does not require the algorithm to visit all states, is desirable even if it is not the most permissive. During the search, such an algorithm stops if it finds a solution even if there is a more permissive solution. Such an early stop of the algorithm is even more evident when no solution exists. Furthermore, an algorithm that builds the solution incrementally can be improved by developing local search heuristics, or a branch and bound [13] like technique since the algorithm is essentially a backtracking algorithm, to reduce the time complexity.

Our contribution is three fold. First, we present a model of the services under partial control and partial observation and we show that the existence of an orchestrator depends on the existence of a set of *message controlled* (m-controlled for

short) relations. Second, we present a fixed-point algorithm to find the set of m-controlled relations if it exists. Third, we develop a backtracking algorithm to find one such set (there could be multiple). We show the correctness of both algorithms and we study their complexity. Finally, we compare the two algorithms on a test case.

The paper is organized as follows. In section II we review related work. In section III we present our model for services and formulate the service composition problem. Also we prove that the existence of an orchestrator is related to the existence of a set of m-controllable relations. In sections IV and V we develop a fixed-point and a backtracking algorithms respectively. The correctness of both algorithms is proved and their complexity is studied. In section VI we show the results of the implementation of the two algorithms on a case study. We conclude in section VII.

II. RELATED WORK

In the vast majority of the methods that model and solve the service composition problem for the case when the orchestrator has partial information, the problem is converted to the case of the synthesis of an orchestrator with complete information over a groups of states called *belief states* [12]. The computation of the *belief state* space for the services precedes the synthesis of the orchestrator. Such computation groups all service states with the *same observation* together in a belief state. Once the belief state space is constructed, essentially a subset construction which is exponential in the number of states, the orchestrator is synthesized as an orchestrator with perfect information over belief states. Therefore, there is a preliminary step before the synthesis start even in the case when no orchestrator exists. By contrast, the backtracking algorithm proposed later in this paper, incrementally explores belief states only when needed while synthesizing the orchestrator.

Among the literature on the composition problem not many tackled the case of the orchestrator with partial information. The ones closest to our approach are [2], [3], and [11]. While [2] converts the problem to a satisfaction of a μ calculus formula without proposing an algorithm, [3] and [11] make use of the *belief state* concept [12]. At first glance, our model looks similar to the ones discussed in [3] and [11] but there are essential differences. First, in [11], all actions are considered as controllable (uncontrollability is accounted for by non-determinism). Partial observation is taken into account by using an observation function. In their model the orchestrator can determine the state(s) of the services by observing the sequence of actions. Therefore even if the observation function returns the same value for two states (i.e. they are indistinguishable) the orchestrator can distinguish between them if they have taken different paths. Hence, our model extends [11] both in terms of observability and controllability.

The model used in [3] includes uncontrollable and unobservable actions like our model. The key difference is that they consider unobservable actions as internal and are not required to satisfy the specification. By contrast, in this work unobservable actions must conform to the target specification.

In [14] develops an on-the-fly algorithm, similar to our backtracking algorithm, for the synthesis of a controller in supervisory control theory but their aim is to synthesize the *most permissive* controller with perfect information.

In this paper as in the above mentioned work the composition deals exclusively with the functional requirements of a request. In addition it is assumed that the set of available services is given and does not deal with the service discovery phase. In [21] a framework for service composition is developed which includes a discovery phase that takes non functional requirements into account. Among the various components of their framework, one component deals with service composition and our model can be integrated into their framework playing the part of the dynamic composition component.

Finally, our forward search algorithm can be used in dynamic and mobile environments where the services community changes dynamically as studied in [5].

III. SERVICE MODEL

The model follows closely the definitions given in [9] with the exception that the orchestrator has partial information. Before giving a formal definition of the various parts of the model it is convenient to give an informal description of its components. The model has the following components: a set of *independent services* whose behavior is coordinated by an *orchestrator* to satisfy a given *specification*. Each service is represented by a finite state labeled transition system (LTS). The goal is to synthesize an orchestrator to coordinate the action of the independent services so that their behavior conforms to the given specification.

A. EXAMPLE

In this section we give a motivational example to introduce the various concepts used in this paper. Figure 1 shows five translation services and a translation request by a user. Each service can read an input, action prefixed by "?", and produce an output, denoted by the prefix "!". For example, service A, can read an input in French ($?fr$) and produce an output in English ($!en$). Three scenarios for the setup shown in that figure will be used to clarify the concepts used in this paper.

Scenario one: the user requests a translation of a French document to Japanese as shown in the request S in the top right of Figure 1. Services A,B,C and D participate in this scenario forming an asynchronous community (each service action is independent of the others) which is (partly)shown in the bottom left of the figure with the remaining possible states and sequence of actions that do not lead to a solution omitted for clarity. Since no single service can provide translation from French to Japanese the orchestrator has to coordinate the actions of the community of services to satisfy the request. There are two possible solutions: French-to-English-to-Japanese and French-to-German-to-English-to-Japanese.

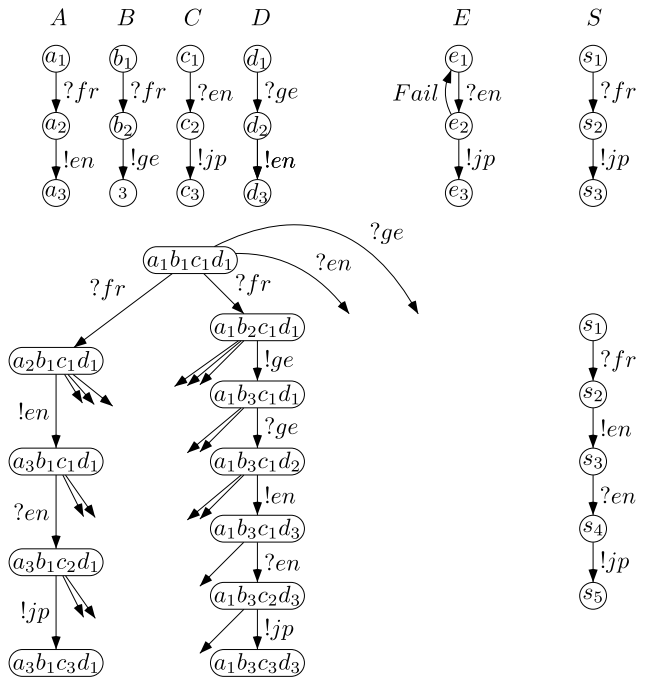

FIGURE 1. Example of services and specification used in the three scenarios in the text.

In this scenario all actions are controllable and observable by the orchestrator, however, only reading French and writing Japanese is observed by the user. This is a typical input-output model (no internal action) used for services in web service composition. In this scenario the user does not "care" about the intermediate actions, i.e. how the translation is done, as long as the input and output match the specification. There are two basic strategies for obtaining the solutions.

The first is a backward search, which will be given formally in Section IV, that starts with the whole state space of the community and the request (in this case it has 243 states). In each iteration of the backward search a set of states that do not contribute to a solution are removed until what remains are all possible solutions which in this scenario there are two.

The second strategy is a forward search (with backtracking), which will be given formally in Section V, that incrementally builds the solution. The forward search starts with the initial state $(a_1b_1c_1d_1)$ and tries the transitions one by one and if a transition does not lead to a solution it backtracks. For example the state $(a_1b_1c_1d_1)$ on the transitions ?*en* and ?*ge* does not match the request of ?*fr* therefore the algorithm backtracks and does not continue on that path. Depending on which transitions it tries first the algorithm could find a solution by visiting just 5 states $(a_1b_1c_1d_1) \xrightarrow{?fr} (a_2b_1c_1d_1) \xrightarrow{!en} \ldots \xrightarrow{!jp} (a_3b_1c_3d_1)$ and stops when it finds the solution. This is different from the case in supervisory control theory where one needs to find the *most permissible* solution. Obviously, this the best case scenario and in the worst case it might need to visit all states but in practice, on average, it will visit much less states than the backward search.

Scenario two: the user considers that something is lost in the translation so it allows a single intermediary language as shown in Figure 1 bottom right. In this case there is only one solution, the leftmost labeled path shown in Figure 1 on bottom left. In the example all actions, including the intermediate ones, are input-output but in different situations, like in multi-agent systems, it could be other actions such as ship item, load container,...etc.

Scenario three: service C is replaced by service E and the specification is the same as in scenario two. Service E is unreliable and when given a English document it can translate it to Japanese or fails to deliver as shown in the transition *Fail*. The action *Fail*, being internal to service E, is both uncontrollable and unobservable by the orchestrator. In this situation there is no solution since in the specification failure is not allowed, there is no *Fail* transition. In general, in addition to scenarios one and two, the specification might require an action which can only be performed by an internal action of a given service, and the action is both uncontrollable and unobservable by the orchestrator. To date a solution to this general case, to our knowledge, has not been presented. In Section III-B we present a necessary and sufficient condition for the existence of an orchestrator for this general case. Furthermore, when the orchestrator exists we show how it is constructed.

We restrict ourselves to the (reasonable) case where the uncontrollable actions are also unobservable. The orchestrator controls the actions of services by sending them messages from a set *Com*. First we give the formal definition of a service as a labeled transition system (LTS).

Definition 1 (Service): A Service $\mathcal{S}_i$ is a labeled transition system $\mathcal{S}_i = \langle S_i, \Sigma, Com_i, s_i^0, \theta_i \rangle$ where S_i is a finite set of states, Σ is a finite action alphabet, Com_i is a finite set of messages, s_i^0 is the initial state, $\theta_i \subseteq S_i \times (\Sigma_i \cup Com_i \times \Sigma_i) \times S_i$ is the transition relation.

Often we will write $s \xrightarrow{a} s'$ if $(s, a, s') \in \theta_i$. Also, if $m \in Com_i, a \in \Sigma$ we will often write $s \xrightarrow{m|a} s'$. Since any controllable action is prefixed by a message, e.g. $s \xrightarrow{m|a} s'$, the service moves from state s to state s' while performing action *a only if* it receives a message m from the orchestrator. An *uncontrollable* action is without a prefix, e.g. $s \xrightarrow{a} s'$. In this case the service could performs the transition (and action) regardless of the action of the orchestrator and without its knowledge (unobservable).

Definition 2 (Collection of Services (Community)): A set of n services $\mathcal{S}_i = \langle S_i, \Sigma, Com_i, s_i^0, \theta_i \rangle$, $i = 1 \ldots n$, are combined to form a collection (community) of services $\mathcal{P} = \langle P, \Sigma, Com, p^0, \theta \rangle$ where

- $P = S_1 \times \ldots \times S_n$.
- $p^0 = (s_1^0, \ldots s_n^0)$.
- $Com = \cup_i Com_i$
- $\theta \subseteq S \times (\Sigma \cup Com \times \Sigma) \times S$

θ is the transition relation of the collection defined as the asynchronous product of all relations θ_i:

$$(\langle s_1, \ldots, s_n \rangle, a, \langle s'_1, \ldots, s'_n \rangle) \in \theta$$

$$\text{iff } (s_k, a, s'_k) \in \theta_k \text{ for some } 1 \leq i \leq n$$
$$\text{and for all } k \neq i \text{ we have } s_k = s'_k$$

Alternatively, we can defined θ functionally as:

$$\theta(s_1, \ldots, s_n, a) = \bigcup_{i=1}^{n} \bigcup_{s'_i \in \theta_k(s_i,a)} (s_1, \ldots, s'_i, \ldots, s_n)$$

Controllable actions are prefixed by messages whereas uncontrollable ones are not. Given an action $a \in \Sigma$ and a community state p, $\theta(p, a)$ denotes the set of states that the services could be in after an *uncontrollable* a-transition and $\theta(p, m \mid a)$ after a controllable transition (if enabled by the orchestrator).

Definition 3 (Orchestrator): Let $Com = \cup_i Com_i$ be the set of messages accepted by the community of services. An orchestrator is a function $\Omega : A \times Com \longrightarrow \{0, 1\}$. Where A depends on whether the orchestrator has partial or complete information.

If the orchestrator has perfect information then it knows exactly in which state the services are in and the sequence of action they performed then $A = P \times \Sigma^*$. When the controller has partial information it can only "remember" its own messages then $A \subseteq Com^*$.

The orchestrator can affect only the controllable actions of the services. Let $p \in P, a \in \Sigma, m \in Com, n \in Com^*$ we define the interaction of the orchestrator with the services by the $\odot$ operator:

$$\theta(p, m \mid a) \odot \Omega(n, m) = \begin{cases} \emptyset & \text{If } \Omega(n, m) = 0 \\ \theta(p, m \mid a) & \text{If } \Omega(n, m) = 1 \end{cases}$$

The evolution of the orchestrated community (i.e. under the control of the orchestrator Ω), denoted by θ_Ω, can thus be divided into two parts: the uncontrollable and controllable evolution. Formally,

$$\theta_\Omega(p, a) = \theta(p, a) \bigcup_{m \in Com} \theta(p, m \mid a) \odot \Omega(n, m) \qquad (1)$$

Definition 4 (Behavior Specification): A behavior specification (or target) $\mathcal{Q}$ is the LTS $\mathcal{Q} = \langle Q, \Sigma, q^0, \theta_t \rangle$ where Σ is a finite action alphabet, Q is a finite set of states, $\theta_t \subseteq Q \times \Sigma \times Q$ is the transition relation, and q^0 is the initial state.

The multi-step evolution of the community on a sequence (trace) of actions, $\tau \in \Sigma^*$, $a \in \Sigma$, is defined inductively as:

$$\theta_\Omega(p, \tau a) = \bigcup_{p' \in \theta(p,\tau)} \theta_\Omega(p', a)$$

Similarly, we extend the transition function of the target inductively as:

$$\theta_t(q, \tau a) = \bigcup_{q' \in \theta_t(q,\tau)} \theta_t(q', a)$$

Definition 5 (Service Composition): Let $\mathcal{Q}$ be a behavior specification and $\mathcal{P}$ be a collection of independent services. Let Ω be an orchestrator with partial observation and denote by $\mathcal{P}_\Omega$ the orchestrated collection. We say that $\mathcal{P}_\Omega$ is a service composition of $\mathcal{Q}$ if and only if for all traces $\tau \in \Sigma^*$ and all $q \in \theta_t(q^0, \tau)$, $p \in \theta_\Omega(p^0, \tau)$ we have :

$$\forall a \in \Sigma, \quad \theta_t(q, a) \neq \emptyset \Leftrightarrow \theta_\Omega(p, a) \neq \emptyset$$

Informally, the above definition means that after both the target and the orchestrated community execute a sequence of actions τ if one of them can make an a transition then the other also can. Since this behavior is required to be true for any sequence, including the empty one, then the orchestrated community can mimic the behavior of the target and thus satisfy the specified behavior. We will show that the existence of an orchestrator for a service composition is characterized by the existence of a set of relations called *m-controllable relations*.

B. MESSAGE CONTROLLABLE RELATIONS

The set of m-controllable relations is a set of relations, $\mathcal{R}$, where each relation, $R \in \mathcal{R}$, is a set of pairs of states (p, q), one in the community and the other in the target and it has two properties. First, in the presence of uncontrollable actions, we don't want the services to reach a state where it can perform uncontrollable actions that the target cannot. This property is important enough to merit its own name, R is said to be a *controllable relation*.

Definition 6 (Controllable Relation): A relation $R \subseteq P \times Q$ is a controllable relation if and only if $\forall (p, q) \in R, a \in \Sigma$ we have $p \xrightarrow{a} p' \Rightarrow \exists q'.q \xrightarrow{a} q' \wedge (p', q') \in R$

The definition above guarantees that any uncontrollable service transition is matched by a target transition. Conversely, we want also that any target transition $q \xrightarrow{a} q'$ to be matched by a service transition, either uncontrollable $p \xrightarrow{a} p'$ or controllable $p \xrightarrow{m|a} p'$. In addition, since the message m can potentially enable other transitions, those transitions are also in R'. Formally,

Definition 7 (Message Controllable Relations): A set of controllable relations $\mathcal{R} \subseteq 2^{P \times Q}$ is said to be message controllable (m-controllable for short) if and only if $\forall R \in \mathcal{R}$ and $\forall (p, q) \in R, a \in \Sigma$ if $q \xrightarrow{a} q'$ then:

1) *Either $\exists p'.p \xrightarrow{a} p' \wedge (p', q') \in R$*
2) *Or $\exists p', m \in Com, R' \in \mathcal{R}$ with*

$$p \xrightarrow{m|a} p' \ \wedge \ (p', q') \in R'$$
$$\wedge$$
$$\forall (u, v) \ \in \ R, b \in \Sigma$$
$$(u \xrightarrow{m|b} u' \Rightarrow \exists v'.v \xrightarrow{b} v' \wedge (u', v') \in R')$$

As we will see below the concept of set of m-controllable relations is central for the concept of controllability (i.e. the existence of an orchestrator). Each controllable relation in $\mathcal{R}$ defines a state of the orchestrator and their transitions are the transitions of the orchestrator.

Therefore, to synthesize an orchestrator it is sufficient to design an algorithm to find the set of m-controllable relations which is done in Sections IV and V. It is worth noting that

in Petri Nets [1] a similar concept, *regions*, is used. Next we extend the concept of controllability [9] to the case when the orchestrator has partial observations.

Definition 8 (Controllability): Let $\mathcal{P} = \langle P, \Sigma, Com, p^0, \theta \rangle$ *be a community of services and* $\mathcal{Q} = \langle Q, \Sigma, q^0, \theta_t \rangle$ *be a specification (target).* $\mathcal{P}$ *is said to be controllable if and only if there exists a set of m-controllable relations* $\mathcal{R}$ *between* $\mathcal{P}$ *and* $\mathcal{Q}$ *and* $(p^0, q^0) \in R_0$ *for some* $R_0 \in \mathcal{R}$.

Next we show that an orchestrator with partial observation exists if and only the community is controllable. This reduces the orchestrator synthesis problem to the problem of finding the set of m-controllable relations.

Theorem 1: Let $\mathcal{P} = \langle P, \Sigma, Com, p^0, \theta \rangle$ *be a community of services and* $\mathcal{Q} = \langle Q, \Sigma, q^0, \theta_t \rangle$ *be a specification. An orchestrator with partial information,* Ω*, exists such that* $\mathcal{P}_\Omega$ *is a service composition of* $\mathcal{Q}$ *if and only if* $\mathcal{P}$ *is controllable. Furthermore,* Ω *is the m-controllable relations graph and can be represented by a finite state labeled transition system.*

Proof: (*If*). Assume that $\mathcal{P}$ is controllable and let $\mathcal{R}$ be the set of m-controllable relations. We need to prove that for any $\tau \in \Sigma^*$ and for all $q \in \theta_t(q^0, \tau)$ and $p \in \theta_\Omega(p^0, \tau)$ we have:

$$\theta_t(q, a) \neq \emptyset \Leftrightarrow \theta_\Omega(p, a) \neq \emptyset$$

The proof is by induction on the length of τ.

Base case. Let ϵ be the empty trace then we can write $q^0 \in \theta_t(q^0, \epsilon)$ and $p^0 \in \theta_\Omega(p^0, \epsilon)$. By definition 8, $(p^0, q^0) \in R$ for some $R \in \mathcal{R}$. Let $q = \theta_t(q^0, a)$ thus $q^0 \xrightarrow{a} q$ and by definition 7 we have one of the two cases:

1) $\exists p.p^0 \xrightarrow{a} p \wedge (p, q) \in R$.
2) $\exists m \in Com, p \in P.p^0 \xrightarrow{m|a} p \wedge (p, q) \in R'$ for some $R' \in \mathcal{R}$. In addition, $\forall (u, v) \in R \left(u \xrightarrow{m|b} u' \Rightarrow v \xrightarrow{b} v' \wedge (u', v') \in R' \right)$. By setting $\Omega(\epsilon, m) = 1$ we get:

$$\theta_\Omega(p^0, a) \supseteq \theta(p^0, m \,|\, a) \odot \Omega(\epsilon, m) \neq \emptyset$$

From the above we get that $\theta_t(q^0, a) \neq \emptyset \Rightarrow \theta_\Omega(p^0, a) \neq \emptyset$.

Conversely suppose that $p \in \theta_\Omega(p^0, a)$ then there are two possibilities:

1) $p^0 \xrightarrow{a} p$ but R is a controllable relation therefore $\exists q$ with $q^0 \xrightarrow{a} q$ and $(p, q) \in R$.
2) $p^0 \xrightarrow{m|a} p$ and $\Omega(\epsilon, m) = 1$ for some m. The orchestrator is set to $\Omega(\epsilon, m) = 1$ only happen if for some $p', q', q^0 \xrightarrow{b} q'$ and $p^0 \xrightarrow{m|b} p'$. From the second part of definition 7, $c \in \Sigma$ with $p^0 \xrightarrow{m|c} u$ there exists v such that $q^0 \xrightarrow{c} v$ and $(u, v) \in R'$. In particular, $\exists q.q^0 \xrightarrow{a} q$ and $(p, q) \in R'$.

Combining both part we get $\theta_t(q^0, a) \neq \emptyset \Leftrightarrow \theta_\Omega(p^0, a) \neq \emptyset$. Moreover for all $p \in \theta_\Omega(p^0, a)$ and $q \in \theta_t(q^0, a)$ we have $(p, q) \in R$ for some $R \in \mathcal{R}$.

Induction Hypothesis: Suppose that $\forall \tau \in \Sigma^*, |\tau| = l - 1$ we have

$$\Big(p \in \theta_\Omega(p^0, \tau) \wedge q = \theta_t(q^0, \tau)\Big) \Rightarrow (\theta_t(q, a) \neq \emptyset \Leftrightarrow \theta_\Omega(p, a) \neq \emptyset) \wedge (p, q) \in R \quad \text{for some } R \in \mathcal{R}$$

Induction Step: Consider an arbitrary $b \in \Sigma$ then $|\tau b| = l$. Let $q^l \in \theta_t(q^0, \tau b)$, $p^l \in \theta_\Omega(p^0, \tau b)$ and $q^l \xrightarrow{a} q^{l+1}$. Then $q^l \in \theta_t(q, b)$ and $p^l \in \theta_\Omega(p, b)$ for some $p \in \theta_\Omega(p^0, \tau)$ and $q \in \theta_t(q^0, \tau)$. Now $|\tau| = l - 1$ and by hypothesis $(p^l, q^l) \in R$ therefore one the two cases is true:

1) $\exists p^{l+1}.p^l \xrightarrow{a} p^{l+1}$, which implies that $\theta_\Omega(p^l, a) \neq \emptyset$ and $(p^{l+1}, q^{l+1}) \in R$.
2) $\exists m, p^{l+1}.p^l \xrightarrow{m|a} p^{l+1}$ and $(p^{l+1}, q^{l+1}) \in R'$ for some $R' \in \mathcal{R}$. Set $\Omega(\tau b, m) = 1$, hence $\theta_\Omega(p^l, a) \supseteq \theta(p^l, m \,|\, a) \odot \Omega(\tau b, m) \neq \emptyset$.

Conversely, let $\theta_\Omega(p^l, a) \neq \emptyset$. Then there are two cases:

1) Either $\exists p^{l+1}.p^l \xrightarrow{a} p^{l+1}$. Since R is controllable then $\exists q^{l+1}.q^l \xrightarrow{a} q^{l+1}$, thus $\theta_t(q^l, a) \neq \emptyset$, and $(p^{l+1}, q^{l+1}) \in R$.
2) Or $\exists m, p^{l+1}.p^l \xrightarrow{m|a} p^{l+1}$ and $\Omega(\tau, m) = 1$. The orchestrator $\Omega(\tau, m)$ is set to 1 only if $\exists b \in \Sigma$ such that $q^l \xrightarrow{b} q'$ and $p^l \xrightarrow{m|b} p'$ with $(p', q') \in R'$ for some $R' \in \mathcal{R}$. But since $\mathcal{R}$ is m-controllable we know that this is done in such a way that for all $c \in \Sigma$ with $p^l \xrightarrow{m|c} u$, $\exists v.q^l \xrightarrow{c} u \wedge (u, v) \in R$. In particular $p^l \xrightarrow{\alpha|a} p^{l+1} \Rightarrow \exists q^{l+1}.q^l \xrightarrow{a} q^{l+1} \wedge (p^{l+1}, q^{l+1}) \in R'$, thus $\theta_t(q^l, a) \neq \emptyset$.

Collecting the above two results we get that $\forall \tau \in \Sigma^* \; \theta_t(q, a) \neq \emptyset \Leftrightarrow \theta_\Omega(p, a) \neq \emptyset$ for all $p \in \theta_\Omega(p^0, \tau)$ and $q^l \in \theta_t(q^0, \tau)$.

(*Only if*) Assume that for all $a \in \Sigma, \tau \in \Sigma^*, p \in \theta_\Omega(p^0, \tau), q \in \theta_t(q^0, \tau)$ we have:

$$\theta_t(q, a) \neq \emptyset \Leftrightarrow \theta_\Omega(p, a) \neq \emptyset$$

We will show that a set of m-controllable relations $\mathcal{R}$ exists. In what follows $\tau = a_1 \ldots a_k \in \Sigma^*$ is a set of *uncontrollable* actions and for conciseness if $p \xrightarrow{a_1} p^1 \ldots \xrightarrow{a_k} p^k$ we write $p \xrightarrow{\tau} p^k$.

Given a controllable relation R we construct for every $m \in Com$ the relation:

$$R_m = \{(p, q) \mid p^0 \xrightarrow{m|a} p^1 \wedge p^1 \xrightarrow{\tau} p \wedge q^0 \xrightarrow{a\tau} q \text{ for some } a \in \Sigma, \tau \in \Sigma^*, p^1 \in \theta_\Omega(p^0, a)\}$$

where $(p^0, q^0) \in R$. First we show that R_m is a controllable relation. Let $(p, q) \in R_m$ then from the definition of R_m there exists $a \in \Sigma, \tau \in \Sigma^*, p^1 \in \theta_\Omega(p^0, a)$ such that $p^0 \xrightarrow{m|a} p^1 \xrightarrow{\tau} p \wedge q^0 \xrightarrow{a\tau} q$. Now assume that $p \xrightarrow{b} p'$ for some $b \in \Sigma$. It follows that $p' \in \theta_\Omega(p^0, a\tau b)$ and by assumption $\exists q'.q' \in \theta_t(q^0, a\tau b)$. Combining both facts we get $p^0 \xrightarrow{m|a} p^1 \xrightarrow{\tau} p \xrightarrow{b} p'$ and $q^0 \xrightarrow{a\tau} q^1 \xrightarrow{b} q'$. Therefore $(p', q') \in R_m$ and thus

R_m is a controllable relation. Next consider arbitrary $c \in \Sigma$ and $(p, q) \in R$ and suppose that $p \xrightarrow{m|c} p' \wedge \theta_\Omega(p, c) \neq \emptyset$. By assumption $\theta_t(q, a) \neq \emptyset$ then we can write $p \xrightarrow{m|c} p' \xrightarrow{\epsilon} p' \wedge q \xrightarrow{c\epsilon} q'$ therefore $(p', q') \in R_m$.

Using the above construction we build the set of relation $\mathcal{R}$ inductively. Given $R \in \mathcal{R}$ construct R_m using the above procedure. If $R_m \in \mathcal{R}$ stop otherwise add R_m to $\mathcal{R}$. The base case for the construction is:

$$R_\epsilon = \{(p, q) \mid p^0 \xrightarrow{\tau} p \wedge q^0 \xrightarrow{\tau} q\}.$$

That for every $p^0 \xrightarrow{\tau} p$ there exists q such $q^0 \xrightarrow{\tau} q$ follows inductively using the assumption that $\theta_t(q^0, a) \neq \emptyset \Leftrightarrow \theta_\Omega(p^0, a) \neq \emptyset$. Since the set of all relations, i.e. all subsets of $S \times S_t$, is finite our inductive procedure terminates in a finite number of steps. By construction $\mathcal{R}$ has the properties of a set of m-controllable relations and therefore $\mathcal{S}$ is controllable. □

Algorithm 1 CONT Returns True iff R is a Controllable Relation

```
CONT (R)
foreach (p, q) ∈ R do
    foreach p -a-> p' do
        if q -a-> q' then
            if (p', q') ∉ R then
                return false
        else
            return false
return true
```

IV. FIXED-POINT ALGORITHM

We develop a fixed-point algorithm in this section. The algorithm computes largest set of m-controllable relations, which corresponds to the *most permissive* orchestrator.

A. ALGORITHM

Since R is a controllable relation for all $R \in \mathcal{R}$, algorithm 1, based on definition 6, tests whether an input relation is controllable or not. The starting point of the algorithm is the set of all controllable relations, $\mathcal{R}_1$ defined as:

$$\mathcal{R}_1 = \{R \in \mathcal{R}_0 \mid CONT(R)\}$$

where $\mathcal{R}_0 = 2^{P \times Q}$. Let F be a function over the set of all relations $2^{P \times Q}$ defined as:

$$\begin{aligned} &F(\mathcal{R}_i) \\ &= \{R \in \mathcal{R}_i \mid \forall (p, q) \in R : \\ &\quad q \xrightarrow{a} q' \Rightarrow \left(\exists p'.p \xrightarrow{a} p' \wedge (p', q') \in R\right) && \text{(2a)} \\ &\quad \vee \\ &\quad \Big(\exists m \in Com, p' \in P, R' \in \mathcal{R}_i.p \xrightarrow{m|a} p' \wedge (p', q') \in R' && \text{(2b)} \\ &\quad \wedge \\ &\quad\quad \forall (u, v) \in R : u \xrightarrow{m|b} u' \Rightarrow v \xrightarrow{b} v' \wedge (u', v') \in R' \Big) && \text{(2c)} \end{aligned}$$

The different conditions in the above equation are a direct implementation of definition 7 in Section III-B. Equations 2a corresponds to part 1 of definition 7. Also equations 2b and 2c correspond to part 2 of the same definition. The set of m-controllable relations is the *largest* fixed-point of F. Therefore the algorithm computes the sequence:

$$\mathcal{R}_{i+1} = F(\mathcal{R}_i) \tag{3}$$

It is easily seen from equation (2) that for any set of relations $\mathcal{R}_i$, we have $F(\mathcal{R}_i) \subseteq \mathcal{R}_i$. This means that for each application of equation (2) relations are *removed* but never *added*. The first iteration is performed on $\mathcal{R}_1$, the set of all controllable relations, therefore any relation in $F(\mathcal{R}_i)$ is controllable.

Since $\mathcal{R}_1 \subseteq \mathcal{R}_0$ is finite then the above terminates in a finite number of iterations. Thus $F(\mathcal{R}_k) = \mathcal{R}_k$ for some k and $\mathcal{R}_k$ is, by construction of $\mathcal{R}_1$ and F, a set of m-controllable relations. Furthermore, by Tarski's fixed-point theorem [19] $\mathcal{R}_k$ is the largest such set.

B. COMPLEXITY

In this section we compute an upper bound for the complexity of the algorithm. First, we compute the complexity of a single application of equation (3). For each $R \in \mathcal{R}_i$ the function F in equation (2) decides if $R \in \mathcal{R}_{i+1}$. Denote by $|p \xrightarrow{a}|$ and $|q \xrightarrow{a}|$ the number of a-transitions that states $p \in P$ and $q \in Q$, respectively, can make. Then the complexity of processing one relation $R \in \mathcal{R}_i$ is at most:

$$\sum_{(p,q)\in R} \sum_{a\in\Sigma} \sum_{m\in Com} |q \xrightarrow{a}| \cdot \left(|p \xrightarrow{a}| + |p \xrightarrow{m|a}| \cdot \sum_{(u,v)\in R} \sum_{b\in\Sigma} |u \xrightarrow{m|b}| \right)$$

This is because for every $q \xrightarrow{a}$ transition, F needs to find either a matching $p \xrightarrow{a}$ or $p \xrightarrow{m|a}$. In addition, for a controllable action with message m, F has to check *all* transitions in R enabled by m, hence the last sum in the above cost. If we keep only the largest contribution in the above (double sum) and that $|q \xrightarrow{a}| = 1$ because the target is deterministic then for each relation $R \in \mathcal{R}_i$ the cost is at most:

$$\begin{aligned} &\sum_{(p,q)\in R} \sum_{a\in\Sigma} \sum_{m\in Com} |p \xrightarrow{m|a}| \cdot \sum_{(u,v)\in R} \sum_{b\in\Sigma} |u \xrightarrow{m|b}| \\ &\leq \left(\sum_{(p,q)\in R} \sum_{a\in\Sigma} \sum_{m\in Com} |p \xrightarrow{m|a}| \right) \cdot \left(\sum_{(u,v)\in R} \sum_{n\in Com} \sum_{b\in\Sigma} |u \xrightarrow{n|b}| \right) \\ &= |E|^2 \cdot |R|^2 \end{aligned}$$

where $|E|$ is the number of controllable transitions in the community P. The above is the cost of one $R \in \mathcal{R}_i$. Knowing that the algorithm terminates at iteration k and by summing over all relations in $\mathcal{R}_i$ we get the following upper bound:

$$O\left(|E|^2 \sum_{i=1}^{k} \sum_{R \in \mathcal{R}_i} |R|^2\right)$$

Since F is monotone decreasing then $\mathcal{R}_1 \supset \ldots, \supset \mathcal{R}_j$. The worst case for the algorithm occurs when $\mathcal{R}_j = \emptyset$, i.e. the problem has no solution. Furthermore, in the worst case $\mathcal{R}_i$ and $\mathcal{R}_{i+1}$ differ by only one relation, i.e. $|\mathcal{R}_i| = |\mathcal{R}_{i+1}| + 1$. Adding these two properties we get that in the worst case it takes $|\mathcal{R}_1|$ iterations for the algorithm to determine that there is no solution. In addition $\mathcal{R}_1 \supset \mathcal{R}_i$ for all i then the complexity becomes:

$$O\left(|E|^2 \sum_{i=1}^{|\mathcal{R}_1|} \sum_{R \in \mathcal{R}_1} |R|^2\right) = O\left(|E|^2 \cdot |\mathcal{R}_1| \sum_{R \in \mathcal{R}_1} |R|^2\right)$$

The expression $\sum_{R \in \mathcal{R}_1} \mid R \mid^2$ is the sum of the square of number of elements in all relations in $\mathcal{R}_1$. The above complexity depends on the size of $\mathcal{R}_1$ which is at most equal to the size of $\mathcal{R}_0 = 2^{P \times Q}$. Let R_k be a subset of $P \times Q$ (i.e. an element of $\mathcal{R}_0$) with size k. Clearly the range of k is from zero for the empty set up to $N = |P \times Q|$ for $P \times Q$ itself. The number of subsets of $P \times Q$ of size k is given by the binomial coefficient $\binom{N}{k}$. Therefore the sum can be written as

$$\begin{aligned}\sum_{R \in \mathcal{R}_1} |R|^2 &\leq \sum_{R \in \mathcal{R}_0} |R|^2 \\ &= \sum_{k=0}^{N} k^2 \binom{N}{k} \\ &= (N + N^2) 2^{N-2}\end{aligned}$$

Finally the complexity of the fixed-point algorithm has the following upper bound

$$O\left(|E|^2 \cdot N^2 \cdot 2^{2N}\right)$$

Note that the above does not include the cost of computing the set $\mathcal{R}_1$.

V. BACKTRACKING ALGORITHM

In this section we develop a backtracking algorithm, to incrementally construct the set of m-controlled relations. We believe this approach will lead to a smaller portion of the search space being visited thus saving time even if the complexity in the worst case is exponential in the number of states.

A. ALGORITHM

Our aim is construct the set of m-controllable relations, $\mathcal{R}$. As we have seen, any relation $R \in \mathcal{R}$ has to be a controllable relation. In the course of computation we need to determine if a given pair of states (p, q) belongs to a controllable relation or not. This closure operation is performed using the *CLOSURE* function given in Algorithm 3. The return value of the function is $\emptyset$ if (p, q) is not in a controllable relation, and returns the controllable relation otherwise.

The function $OBS(R)$ is the backtracking function and all other functions are helpers. It determines if the input relation R is to be added to the solution $\mathcal{R}$. Note that R when used as input for *OBS* it is guaranteed to be a controllable relation. Basically, the function $OBS(R)$ checks that R belongs to an m-controllable set according to definition 7. Otherwise, R is added to a set Z that contains all the relations that *should not* be in $\mathcal{R}$.

The main algorithm, algorithm 2 adds relations to $\mathcal{R}$ in preorder and checks them in postorder fashion. Because of that it is possible a relation R' is removed from $\mathcal{R}$ even if some other relation R depends it. Therefore, we maintain a variable *DONE* set to false if a given relation is removed from $\mathcal{R}$ and the algorithm is runs again with $\mathcal{R}$ reset to empty. When *DONE* is true the algorithm stops. It should be noted that while $\mathcal{R}$ is reset to empty the set Z, which contains

Algorithm 2 Forward Search Algorithm

```
cl ← CLOSURE((∅, p^0, q^0))
if cl = ∅ then
    return ∅

DONE ← false
while ¬DONE do
    DONE ← true
    R ← ∅
    res = OBS(cl)
    if res = false then
        break
return R
```

Algorithm 3 Function CLOSURE

```
CLOSURE (R,(p, q))
if (p, q) ∈ bad then
    return ∅
if (p, q) ∈ R then
    return R
R' ← R ∪ {(p, q)}
foreach p -a-> p' do
    if q -a-> q' then
        R' ← CLOSURE (R', (p', q'))
        if R' = ∅ then
            bad ← bad ∪ {(p', q')}
            return ∅
    else
        bad ← bad ∪ {(p, q)}
        return ∅
return R'
```

Algorithm 4 Function OBS

```
OBS (R)
if R ∈ Z then
    return false
if R ∈ R then
    return true
R ← R ∪ {R}
foreach (p, q) ∈ R do
    foreach q -a-> q' do
        res = FindMatch (R,p,q -a-> q')
        if res = false then
            GOTO END
return true
END: R ← R − R
Z ← Z ∪ R
DONE ← false
return false
```

Algorithm 5 Function FindMatch

```
FindMatch (R, p, q -a-> q')
if p -a-> p' then
    return true
foreach m ∈ Com do
    if p -m|a-> p' then
        /* the result is a controllable
           relation                          */
        R' ←TEST (R, m)
        if R' = ∅ then
            /* the result is not a
               controllable relation         */
            Continue
        else
            res =OBS (R')
            if res = true then
                return true
return false
```

Algorithm 6 Function TEST

```
TEST (R, m)
R' ← ∅
foreach (p, q) ∈ R do
    if p -m|b-> q' then
        if q -b-> q' then
            cl = CLOSURE(∅, p', q')
            if cl = ∅ then
                return ∅
            else
                R' ← R' ∪ cl
        else
            return ∅
return R'
```

the relations that *are not* in $\mathcal{R}$ is maintained after each run because if $R \notin \mathcal{R}$ at some point it cannot be in $\mathcal{R}$ in a subsequent run. Now we explain the various functions used by the algorithm.

The input to the function *OBS* is a relation R and returns true if $R \in \mathcal{R}$. This is done with the help of function *FindMatch* explained below. The *FindMatch* function checks for the first and second part of the disjunction in equation (2). We can write the function F in equation (2) as:

$$F(\mathcal{R}_i) = \{R \in \mathcal{R}_i \mid \forall (p,q) \in R : FindMatch(R, p, q \xrightarrow{a} q')\} \tag{4}$$

Finally, the return value of the function *TEST* is either $\emptyset$ or

$$\bigcup_{\substack{(u,v)\in R \\ u \xrightarrow{\alpha|b} v'}} CLOSURE(u, v)$$

B. CORRECTNESS

We will show that the algorithm is correct by "unrolling" the functions that are used. From Algorithm 4 it is clear that a relation R is included in $\mathcal{R}$ if and only if the return value of $OBS(R)$ is true. Now, the return value of $OBS(R)$ is true if R has the following properties:

$$\forall (p,q) \in R, \forall a \in \Sigma : \quad q \xrightarrow{a} q' \Rightarrow FindMatch(R, p, q \xrightarrow{a} q')$$

We unroll the code of *FindMatch*:

$$\forall (p,q) \in R, \forall a \in \Sigma : \quad q \xrightarrow{a} q' \Rightarrow \left(\exists p'.p \xrightarrow{a} p'\right) \vee \left(\exists m \in Com, p' \in P.p \xrightarrow{m|a} p' \wedge (TEST(R,m) \neq \emptyset) \wedge OBS(TEST(R,m))\right)$$

Since the input to *OBS* and *FindMatch* is a controllable relations then the term $\exists p'.p \xrightarrow{a} p'$, can be written as:

$$\exists p'.p \xrightarrow{a} p' \wedge (p', q') \in R$$

By unrolling the function *TEST* and the fact that the return value of $OBS(R)$ is true if and only if $R \in \mathcal{R}$ we get:

$$\forall (p,q) \in R, \forall a \in \Sigma : \quad q \xrightarrow{a} q' \Rightarrow \left(\exists p'.p \xrightarrow{a} p' \wedge (p', q') \in R\right) \vee \left\{\exists m \in Com, p' \in P. \left[p \xrightarrow{m|a} p' \wedge \forall (u,v) \in R \left(u \xrightarrow{m|b} u' \Rightarrow \exists v'.v \xrightarrow{b} v' \wedge closure(u', v')\right)\right.\right.$$

$$\wedge \bigcup_{\substack{(u,v)\in R \\ u \xrightarrow{m|b} u' \\ v \xrightarrow{b} v'}} closure(u', v') \in \mathcal{R} \Big]\Big\} \tag{5}$$

Let:

$$R' = \bigcup_{\substack{(u,v)\in R \\ u \xrightarrow{m|b} u' \\ v \xrightarrow{b} v'}} closure(u', v')$$

Then the unrolled code for $OBS(R)$ can be written as:

$$\begin{aligned}
&\forall (p,q) \in R, \forall a \in \Sigma : \\
&\quad q \xrightarrow{a} q' \Rightarrow \left(\exists p'.p \xrightarrow{a} p' \wedge (p', q') \in R\right) \\
&\quad \vee \\
&\quad \Big\{\exists m \in Com, p' \in P, R' \in \mathcal{R}. \Big[p \xrightarrow{m|a} p' \wedge (p', q') \in R' \\
&\qquad \forall (u,v) \in R \left(u \xrightarrow{m|b} u' \Rightarrow \exists v'.v \xrightarrow{b} v' \wedge (u', v') \in R'\right)
\end{aligned} \tag{6}$$

By comparing equation (6) with equation (2) it is clear that $\mathcal{R}$ is a fixed-point of F and therefore the algorithm is correct. Note that $\mathcal{R}$ is not necessarily the *largest* fixed-point.

C. COMPLEXITY

Our starting point is determining the cost of single call to OBS.

$$\sum_{(p,q)\in R} \sum_{a\in\Sigma} |q \xrightarrow{a}| \cdot |FindMatch(R, p, q \xrightarrow{a} q')|$$

The complexity of *FindMatch* is dominated by the controllable transitions thus:

$$\sum_{(p,q)\in R} \sum_{a\in\Sigma} \sum_{m\in Com} |p \xrightarrow{m|a}| \cdot |TEST(R, m)|$$

Including the cost of the *TEST* function we get

$$\begin{aligned}
&\sum_{(p,q)\in R} \sum_{a\in\Sigma} \sum_{m\in Com} |p \xrightarrow{m|a}| \\
&\quad \cdot \sum_{(u,v)\in R} \sum_{b\in\Sigma} |u \xrightarrow{m|b} u'| \cdot |v \xrightarrow{b} v'| \cdot |closure(\emptyset, u', v')|
\end{aligned}$$

Since the closure function visits a pair at most once then in the worst case it will visit all the pairs in $P \times Q$ thus

$$\begin{aligned}
|closure(\emptyset, u', v')| &\leq \sum_{(p,q)\in P\times Q} \sum_{a\in\Sigma} |p \xrightarrow{a}| \cdot |q \xrightarrow{a}| \\
&\leq |E_u|
\end{aligned}$$

where $|E_u|$ is the number of uncontrollable transitions of the community (the target is deterministic so $|q \xrightarrow{b} q'| \leq 1$). E does not depend on R and can be removed outside the summation. Therefore the cost of a single call of $OBS(R)$ is:

$$\leq |E_u| \sum_{(p,q)\in R} \sum_{a\in\Sigma} \sum_{m\in Com} |p \xrightarrow{m|a}| \cdot \sum_{(u,v)\in R} \sum_{b\in\Sigma} |u \xrightarrow{m|b} u'|$$

TABLE 1. Results of the four experiments.

Experiment	Fixed-point	Backtracking
Case 1	32 s	23 s
Case 2	871 s	614 s
Case 3	36 s	18 s
Case 4	1038 s	734 s

The value of the multiple sum was computed in section IV-B. Therefore the complexity of the algorithm is

$$O(|E_u| \cdot |E|^2 \cdot N^2 \cdot 2^N)$$

VI. EXPERIMENTAL

We implemented the two algorithm to solve the system first presented in [20]. The system consists of four services: the user, a producer, a shipper and a delivery. The user would like to purchase some items from the producers and have them delivered at her home. When the user requests an item an offer, including the cost and delay, is made which can be accepted or refused by the user. Therefore each item requested by the user can have four properties: location, delay, cost and size. We tested two of the case mentioned in [20] (cases 5 and 6) which we refer two as case 1 and case 2 in the table to compare the difference in performance between the fixed-point and the forward search algorithm. The test were run on a MAC book with Intel Core i5 2.4MHZ with 4GB or RAM. Furthermore, in each case we made a small variation in order to make the target unachievable (i.e. no solution) which we label as case 3 and case 4. The results are shown in the table below. As we can see the backtracking algorithm performs better in all cases than the fixed-point algorithm.

VII. CONCLUSION

In this paper we have developed methods to find a solution to the composition problem under partial information. First, we have shown that an orchestrator exists if and only if a set of m-controllable relations exists and the orchestrator is the same as the graph of the m-controllable relations. Then we developed two algorithms to find the m-controllable relations. One is a fixed-point algorithm that obtains the *most permissive* orchestrator and the second is a backtracking algorithm that obtains an orchestrator which is not necessarily the most permissive. We proved the correctness and computed the complexity of both algorithms. The backtracking algorithm performed better when used on a test case.

The backtracking algorithm developed in this paper can be improved by using heuristics based on abstraction as done for the complete information case in [8]. Also, the algorithm can be improved further by using binary decision diagrams(BDD) [4] as done in supervisory control problems such as [22]. Finally, it is interesting to try our approach for the distributed orchestration case where the specification is divided into sub-specifications and an orchestrator is synthesized for each then a super-orchestrator will coordinate the actions with the synthesized orchestrators.

REFERENCES

[1] E. Badouel and P. Darondeau, "Theory of regions," in *Lectures on Petri Nets I: Basic Models: Advances in Petri Nets*. Berlin, Germany: Springer, 1998, pp. 529–586.
[2] P. Balbiani, F. Cheikh, and G. Feuillade, "Composition of interactive Web services based on controller synthesis," in *Proc. IEEE Congr. Services I*, Jul. 2008, pp. 521–528.
[3] P. Bertoli, M. Pistore, and P. Traverso, "Automated composition of Web services via planning in asynchronous domains," *Artif. Intell.*, vol. 174, nos. 3–4, pp. 316–361, Mar. 2010.
[4] R. E. Bryant, "Symbolic Boolean manipulation with ordered binary-decision diagrams," *ACM Comput. Surv.*, vol. 24, no. 3, pp. 293–318, 1992.
[5] N. Chen, N. Cardozo, and S. Clarke, "Goal-driven service composition in mobile and pervasive computing," *IEEE Trans. Services Comput.*, vol. 11, no. 1, pp. 49–62, Jan./Feb. 2018.
[6] G. De Giacomo, F. Patrizi, and S. Sardina, "Agent programming via planning programs," in *Proc. 9th Int. Conf. Auton. Agents Multiagent Syst. (AAMAS)*, Richland, SC, USA, vol. 1, 2010, pp. 491–498,
[7] G. De Giacomo and S. Sardina, "Automatic synthesis of new behaviors from a library of available behaviors," in *Proc. 20th Int. Joint Conf. Artif. Intell. (IJCAI)*. San Francisco, CA, USA: Morgan Kaufmann, 2007, pp. 1866–1871.
[8] H. Farhat, "Composition services behavior via orchestrator synthesis," Ph.D. dissertation, Inst. Recherche en Inf. de Toulouse, Paul Sabatier Univ., Toulouse, France, 2014.
[9] H. Farhat, "Web service composition via supervisory control theory," *IEEE Access*, vol. 6, pp. 59779–59789, 2018.
[10] Y. Feng, A. Veeramani, R. Kanagasabai, and S. Rho, "Automatic service composition via model checking," in *Proc. IEEE Asia–Pacific Services Comput. Conf. (APSCC)*, Dec. 2011, pp. 477–482.
[11] G. De Giacomo, R. De Masellis, and F. Patrizi, "Composition of partially observable services exporting their behaviour," in *Proc. ICAPS*, 2009, pp. 90–97.
[12] R. P. Goldman and M. S. Boddy, "Expressive planning and explicit knowledge," in *Proc. AIPS*, 1996, pp. 110–117.
[13] E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Hoboken, NJ, USA: Wiley, 1985.
[14] S. Miremadi and B. Lennartson, "Symbolic on-the-fly synthesis in supervisory control theory," *IEEE Trans. Control Syst. Technol.*, vol. 24, no. 5, pp. 1705–1716, Sep. 2016.
[15] A. Muscholl and I. Walukiewicz, "A lower bound on Web services composition," *Logical Methods Comput. Sci.*, vol. 4, no. 2, pp. 1–14, 2008.
[16] P. Papapanagiotou and J. Fleuriot, "Formal verification of Web services composition using linear logic and the pi-calculus," in *Proc. 9th IEEE Eur. Conf. Web Services (ECOWS)*, Sep. 2011, pp. 31–38.
[17] P. J. G. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proc. IEEE*, vol. 77, no. 1, pp. 81–98, Jan. 1989.
[18] J. Rao and X. Su, "A survey of automated Web service composition methods," in *Proc. 1st Int. Conf. Semantic Web Services Web Process Composition (SWSWPC)*. Berlin, Germany: Springer-Verlag, 2005, pp. 43–54.
[19] A. Tarski, "A lattice-theoretical fixpoint theorem and its applications," *Pacific J. Math.*, vol. 5, no. 2, pp. 285–309, 1955.
[20] P. Traverso and M. Pistore, "Automated composition of semantic Web services into executable processes," in *The Semantic Web—ISWC* (Lecture Notes in Computer Science), vol. 3298, S. McIlraith, D. Plexousakis, and F. Harmelen, Eds. Berlin, Germany: Springer, 2004, pp. 380–394.
[21] A. K. Tripathy and P. K. Tripathy, "Fuzzy QoS requirement-aware dynamic service discovery and adaptation," *Appl. Soft Comput.*, vol. 68, pp. 136–146, Jul. 2018.
[22] A. Vahidi, M. Fabian, and B. Lennartson, "Efficient supervisory synthesis of large systems," *Control Eng. Pract.*, vol. 14, no. 10, pp. 1157–1167, 2006.
[23] E. Zahoor, O. Perrin, and C. Godart, "Web services composition verification using satisfiability solving," in *Proc. IEEE 19th Int. Conf. Web Services (ICWS)*, Jun. 2012, pp. 242–249.

HIKMAT FARHAT received the B.Sc. degree in physics from the American University of Beirut in 1993 and the Ph.D. degree in artificial intelligence from the University of Paul Sabatier in 2014. He is currently an Associate Professor of computer science at Notre Dame University-Louaize, Lebanon. His research interests include artificial intelligence broadly construed and in particular formal methods and machine learning.