A Simple Formal Method to Synthesize an Orchestrator in Web Service Composition

Ahmed Khoumsi

Abstract—We study the following web service composition problem: given existing web services $S_1, \ldots, S_n$ and a desired web service $S_0$, the objective is to synthesize an orchestrator $\text{Orch}$ that coordinates $S_1, \ldots, S_n$ so that $S_0$ is provided. We develop a simple input-output automata-based method that solves this composition problem. We also study whether the desired service $S_0$ is totally or partially provided by the system $((S_i)_{i=1 \cdots n}, \text{Orch})$. The studied problem can be seen as a control problem, where $\text{Orch}$ is a controller that restricts the behaviors of $S_1, \ldots, S_n$ so that the controlled system conforms to $S_0$.

Keywords: Web service composition; Orchestrator synthesis; Input-output automata (IOA); Extended IOA.

I. INTRODUCTION

Web service (WS) composition consists in coordinating several existing WSs so that they cooperate to provide a new WS. However, several problems arise with WS composition. For example, the WSs to be combined may be incompatible, which requires to detect and resolve such eventual incompatibilities. To address the complexity of WS composition, several composition models have been developed, such as orchestration and choreography. Orchestration consists in designing a module called orchestrator that coordinates (or orchestrates) several WSs so that a desired service is provided. Choreography can be seen as a decentralized or distributed variant of orchestration. We will use the terms WS and service as synonyms.

We study and solve a composition problem whose inputs are: services $S_1, \ldots, S_n$ and a desired service $S_0$. Our objective is to synthesize automatically a module called orchestrator and denoted $\text{Orch}$ that coordinates $S_1, \ldots, S_n$ so that they provide $S_0$. We also analyze the provided service to determine whether it realizes $S_0$ totally or partially. The studied problem can be seen as a control problem, where $\text{Orch}$ is a controller that restricts the behaviors of $S_1, \ldots, S_n$ so that the controlled system conforms to $S_0$.

The rest of the paper is organized as follows: In Section II, we present an illustrative example of inputs and result of our synthesis method. We give two services $S_1$ and $S_2$ and a desired service $S_0$, and we present the orchestrator that is expected to result from our synthesis method. Section III presents Input-Output Automata (IOAs) used to model the inputs $(S_0, \ldots, S_n)$ and the result (Orch) of our synthesis method, and then our synthesis objective is formally defined. For the purpose of our synthesis method, we need to transform the IOAs modeling $(S_i)_{i=0 \cdots n}$ into Extended IOAs (or E-IOAs). The latter are presented in Section IV. In Section V, we develop our synthesis method and analyze its correctness and to what extent it permits to provide (partially or totally) the desired service. Section VI discusses related work and our contribution. We conclude in Section VII.

II. ILLUSTRATIVE EXAMPLE

We will use the example of Fig. 1 to clarify our explanations. This example is taken from from [1] to facilitate the comparison (in Sect. VI) of our work with [1]. For each sequence diagram, the corresponding sequence of interactions is obtained by top-down scanning of the diagram. The service $S_1$ of Fig. 1(a) receives an input SSN (social security number) and generates an output CCN (credit card number). The service $S_2$ of Fig. 1(b) receives an input CCN twice and generates respectively outputs Appr (credit approval) and Loan (loan approval). The desired service $S_0$ of Fig. 1(c) receives an input CCN and generates an output Loan. For each of these services $S_1, S_2, S_0$, an arrow from left to right represents an input coming from the user of the service, and an arrow from right to left represents an output of the service towards its user.

The design question is: how can we combine $S_1$ and $S_2$ to obtain $S_0$? A solution is represented in Fig. 1(d), where the two vertical bars at the right represent $S_1$ and $S_2$, and the vertical bar at the left represents Orch. Henceforth, we use the term environment as the user of the system consisting of the composed services and the orchestrator. The latter plays the role of an intermediary between the environment (at the left) and $S_1$ and $S_2$ (at the right). The service provided to the environment is represented by the sequence at the left of the left bar, which is similar to the sequence of $S_0$ (Fig. 1(c)), hence the desired service $S_0$ is provided. The sequence of arrows between the left bar and the right bars shows how Orch interacts with $S_1$ and $S_2$ to provide $S_0$. The whole sequence of interactions is interpreted as follows: Orch receives SSN from the environment and forwards it to $S_1$ which replies by sending CCN to Orch. The latter forwards CCN to $S_2$ which replies by sending Appr to Orch. Then, Orch sends for the second time CCN to $S_2$ which replies by sending Loan to Orch which forwards it to the environment.

III. MODELING AND OBJECTIVE

A. Preliminaries

We consider services $S_1, \ldots, S_n$ to be composed and a desired service $S_0$ to be provided. Our solution is to synthesize an orchestrator $\text{Orch}$ as an intermediary between the environment and $(S_i)_{i=1 \cdots n}$, so that $((S_i)_{i=1 \cdots n}, \text{Orch})$
be the set of data that Orch may be only task is to interactions \(\alpha\) of and the environment. Hence, every interaction in the models \(\text{Appr}^{SSN}\) (resp. \(\text{Orch}\)) is exchanged between \(\text{SSN}\) and \(\text{Orch}\), \(\text{SSN}\) when it is exchanged between \(\text{Orch}\) and the environment. Since the same datum \(\sigma\) specifies the sequencing of the interactions between \(\text{Orch}\) and \(\text{S}\i\), and the model of \(S_0\) specifies the sequencing of the interactions between \(\text{Orch}\) and the environment. Hence, every interaction in the models of \(S_0\) and \((S_1)_{i=1\cdots n}\) represents a reception or an emission of a datum \(\sigma\) by \(\text{Orch}\). Since \(\text{Orch}\) is an intermediary, its only task is to forward data it has received. The forward may be multiple (i.e. data forwarded several times, possibly to different destinations) or null (data not forwarded). Let \(D\) be the set of data that \(\text{Orch}\) exchanges with \(S_1\), \(\cdots\), \(S_n\) and the environment. Since the same datum \(\sigma\in D\) can be exchanged in different places, we will note \(\sigma_i\) when it is exchanged between \(\text{Orch}\) and \(S_1\), \(i\in\{1,\cdots,n\}\), and \(\sigma_0\) when it is exchanged between \(\text{Orch}\) and the environment.

In our example, \(D=\{\text{SSN}, \text{CCN}, \text{Appr}, \text{Loan}\}\). \(\text{SSN}\) (resp. \(\text{SSN}_1\)) is \(\text{SSN}\) when it is exchanged between \(\text{Orch}\) and the environment (resp. \(\text{S}_1\)). \(\text{CCN}_1\) (resp. \(\text{CCN}_2\)) is \(\text{CCN}\) when it is exchanged between \(\text{Orch}\) and \(\text{S}_{\i}\) (resp. \(\text{S}_2\)). \(\text{Appr}_2\) when it is exchanged between \(\text{Orch}\) and \(\text{S}_2\). \(\text{Loan}_0\) (resp. \(\text{Loan}_2\)) is \(\text{Loan}\) when it is exchanged between \(\text{Orch}\) and the environment (resp. \(\text{S}_2\)).

\section{B. IOAs to model \((S_1)_{i=0\cdots n}\) and \(\text{Orch}\)}

Each of \((S_1)_{i=0\cdots n}\) (inputs of our synthesis problem) and \(\text{Orch}\) (the result) is modeled by an Input Output Automaton (or IOA). Usually, the transitions of an IOA are labeled in the form \(\sigma/\mu\), i.e. the reception of \(\sigma\) is followed by the sending of \(\mu\). But for the purpose of our study, we consider that every transition is labeled in the form \(\sigma/\mu\) or \(-/\mu\). Hence, a transition labeled \(\sigma/\mu\) is specified as two consecutive transitions labeled \(\sigma/-\mu\) and \(-/\mu\), respectively.

Let \(T_i=(Q_i, \Sigma_i, \alpha_i, q_i^0)\) be the IOA modeling \(S_{\i}\), for \(i = 0 \cdots n\), where \(Q_i\) is a set of states, \(q_i^0\) is the initial state, \(\Sigma_i\) is the alphabet, and \(\alpha_i \subseteq Q_i \times Q_i \times Q_i\) specifies the transitions. \(\Sigma_i\) contains interactions in the form \(\sigma_i/-\mu_j\) and \(-/\mu_i\). For \(i=1\cdots n\), \(\sigma_i/-\mu_i\) means \(S_i\) receives \(\sigma\) from \(\text{Orch}\), and \(-/\mu_i\) means \(S_i\) sends \(\mu\) to \(\text{Orch}\). \(\sigma_0/-\mu\) means \(\text{Orch}\) receives \(\sigma\) from the environment, and \(-/\mu\) means \(\text{Orch}\) sends \(\mu\) to the environment.

Let \(\mathcal{O}=(Q, \Sigma, \alpha, q^0)\) be the IOA modeling \(\text{Orch}\), where \(Q\) is a set of states, \(q^0\) is the initial state, \(\Sigma\) is the alphabet, and \(\alpha \subseteq Q \times \Sigma \times Q\) specifies the transitions. \(\Sigma\) contains interactions \(\sigma_i/-\mu_j\) for \(i, j \in \{0,\cdots,n\}\). \(\sigma_i/-\mu_i\) means Orch receives \(\sigma\) (from the environment if \(i = 0\), or from some \(S_i\) if \(i \neq 0\)). \(-/\mu_j\) means Orch sends \(\mu\) (to the environment if \(j = 0\), or to some \(S_j\) if \(j \neq 0\)).

The IOAs \((T_i)_{i=0\cdots n}\) have their alphabets \((\Sigma_i)_{i=0\cdots n}\) disjoints, because any \(\sigma_i\) and \(\sigma_j\) \((i \neq j)\) correspond to distinct interactions. Note that the inputs and outputs of \(T_0\) are also inputs and outputs of \(\mathcal{O}\), respectively, but the inputs and outputs of \((T_i)_{i=1\cdots n}\) are reversed into outputs and inputs of \(\mathcal{O}\), respectively. Due to this reversion and since \(\mathcal{O}\) is involved in all interactions of \((T_i)_{i=0\cdots n}\), we have that the alphabet \(\Sigma\) is obtained by reversing the inputs and outputs of the alphabets \((\Sigma_i)_{i=1\cdots n}\) and then making the union of all resulting alphabets and \(\Sigma_0\).

For our example, the IOAs \((T_i)_{i=0,1,2}\) modeling \((S_i)_{i=0,1,2}\) and the IOA \(\mathcal{O}\) modeling \(\text{Orch}\) are represented in Figure 2.

\section{C. Objective Specified Formally}

We consider two operators on automata: the synchronized product and the projection [2], [3]. The system \((S_1)_{i=1\cdots n}, \text{Orch}\) is modeled by the synchronized product \(T_1 \otimes \cdots \otimes T_n \otimes \mathcal{O}\). This product is computed after reversing the inputs and outputs of \((T_i)_{i=1\cdots n}\). The synchronization consists in executing every interaction of \(\mathcal{O}\) simultaneously with a similar interaction in some \(T_i\). To define the service provided by \((S_1)_{i=1\cdots n}, \text{Orch}\), we use the projection \(P\) in the alphabet \(\Sigma_0\), that keeps observable only the interactions \(\sigma_0/-\mu_0\). The service provided by \((S_1)_{i=1\cdots n}, \text{Orch}\) is modeled by \(P(T_1 \otimes \cdots \otimes T_n \otimes \mathcal{O})\).

\textbf{Notation 3.1:} Consider two IOAs \(A\) and \(B\) with the same alphabet and their two respective generated languages \(L_A\) and \(L_B\). \(A \equiv B\) means \(L_A = L_B\), \(A < B\) means \(L_A \subseteq L_B\), and \(A \preceq B\) means \(L_A \subseteq L_B\).

We characterize formally the total or partial provision of \(S_0\) as follows:

\textbf{Definition 3.1:} The desired service \(S_0\) is said provided by \((S_1)_{i=1\cdots n}, \text{Orch}\) if \(P(T_1 \otimes \cdots \otimes T_n \otimes \mathcal{O}) \preceq L_0\). More specifically, \(S_0\) is said totally (resp. partially) provided if \(P(T_1 \otimes \cdots \otimes T_n \otimes \mathcal{O}) \equiv L_0\) (resp. \(P(T_1 \otimes \cdots \otimes T_n \otimes \mathcal{O}) < L_0\)).

Intuitively, total (resp. partial) provision of \(S_0\) means that \((S_1)_{i=1\cdots n}, \text{Orch}\) can execute all and only (resp. only but not all) traces executable by \(S_0\).

In Section III-A, we have noted that \(\text{Orch}\) can only forward (no time, one time or more times) data it has received. Formally, \(\text{Orch}\) must respect the following rule:

\textbf{Rule 3.1:} Let \(L_\mathcal{O}\) be the language accepted by \(\mathcal{O}\). In every trace of \(L_\mathcal{O}\), every \(-/\sigma_i\) is necessarily after \(-/\sigma_j\).
In Rule 3.1, $i$ may be different from $j$. We have now all the ingredients to define formally our objective as follows:

**Objective 1:** From $(T_i)_{i=0}^{n}$, we have to synthesize $O$ that respects Rule 3.1 and the following two points: 1) $S_0$ is provided by $(S_i)_{i=1}^{n}$, Orch); 2) if $S_0$ is partially provided by $(S_i)_{i=1}^{n}$, Orch), there exists no orchestrator $X$ respecting Rule 3.1 such that $(S_i)_{i=1}^{n}$ provides $S_0$ totally. In a word, the synthesized orchestrator is the one that best provides the desired service.

**IV. EXTENDED IOAS TO MODEL $(S_i)_{i=0}^{n}$**

To synthesize in a simple way an IOA $O$ modeling the orchestrator that respects Rule 3.1, we need to transform the IOAs $(T_i)_{i=0}^{n}$ into extended IOAs (or E-IOAs) which are presented in this section. To each datum $\sigma \in D$, we associate a variable $x_\sigma \in \{0,1\}$ that has the following meaning:

- $x_\sigma$ is set to 1 (resp. 0) to indicate that Orch has (resp. has not) received $\sigma$.

Then, to each transition of $T_i$, we associate an enabling condition or an assignment that have the following forms and semantics:

- When a transition is executed, its assignment is applied, if any. An assignment is in the form “$x_\sigma = 1$” and means setting $x_\sigma$ to 1, i.e. it indicates that Orch has received $\sigma$. Hence, “$x_\sigma = 1$” is used in every transition representing the reception by Orch of $\sigma$.

- An enabling condition is in the form “if $x_\sigma = 1$” and is used to respect Rule 3.1, i.e. to permit that Orch sends $\sigma$ only if $x_\sigma$ evaluates to 1, i.e. only if Orch has received such $\sigma$. Hence, “if $x_\sigma = 1$” is used in every transition representing the sending by Orch of $\sigma$.

Therefore, to respect Rule 3.1, we have to respect the following rules:

**Rule 4.1:** To every transition representing the reception by Orch of $\sigma$, we associate the assignment “$x_\sigma = 1$” and no enabling condition.

**Rule 4.2:** To every transition representing the sending by Orch of $\sigma$, we associate the enabling condition “if $x_\sigma = 1$” and no assignment.

**A. Modeling the Services $(S_i)_{i=1}^{n}$ by E-IOAs**

In this subsection, when we write $S_i$, we consider $i \in \{1,\cdots,n\}$. The E-IOA modeling each $S_i$ is defined as $S_i = \langle L_i, \Sigma_i, D_i, \nu_i, \delta_i, \ell^i_0 \rangle$, where $L_i$ is a set of locations, $\ell^i_0$ is the initial location, $\Sigma_i$ is the set of variables $x_\sigma$ associated to every data $\sigma \in D_i$, $\delta_i \subseteq L_i \times \Sigma_i \times \nu_i \times A_i \times L_i$ specifies the transitions, where $E_i$ is the set of enabling conditions “if $x_\sigma = 1$” and $A_i$ is the set of assignments “$x_\sigma = 1$”, for all variables $x_\sigma \in \nu_i$.

We use the term location instead of state, because a state is defined by a location and a valuation of each variable $x_\sigma$. Let us consider separately the input and output transitions:

An input transition of $S_i$ is defined by $[\ell, \sigma_i, -; \text{if } x_\sigma = 1, -, \ell']$, which means that if $\sigma_i$ occurs while the current location is $\ell$ and $x_\sigma$ evaluates to 1, then the location $\ell'$ is reached. “-” in the fourth position means the absence of assignment. Such a transition represents the reception by $S_i$ of $\sigma$ coming from Orch, i.e. the sending by Orch of $\sigma$. Hence, Rule 4.2 becomes:

**Rule 4.3:** In $(S_i)_{i=1}^{n}$, an input transition executing $\sigma_i/-$ has the enabling condition “if $x_\sigma = 1$” and no assignment.

An output transition of $S_i$ is defined by $[\ell, \sigma_i, -; \text{if } x_\sigma = 1, \ell']$, which means that if $x_\sigma = 1$ occurs while the current location is $\ell$, then the location $\ell'$ is reached and $x_\sigma$ is set to 1. “-” in the third position means the absence of enabling condition. Such a transition represents the sending by $S_i$ of $\sigma$ to Orch, i.e. the reception by Orch of $\sigma$. Hence, Rule 4.1 becomes:

**Rule 4.4:** In $(S_i)_{i=1}^{n}$, an output transition executing $/\sigma_i$ has the assignment “$x_\sigma = 1$” and no enabling condition.

**B. Modeling the Desired Service $S_0$ by an E-IOA**

The E-IOA modeling $S_0$ is defined as $S_0 = \langle L_0, \Sigma_0, D_0, \nu_0, \delta_0, \ell^0_0 \rangle$, where $L_0$ is a set of locations, $\ell^0_0$ is the initial location, $\Sigma_0$ is the set of inputs and outputs of $S_0$, $D_0$ is the set of data exchanged between Orch and the environment, and $\nu_0$ is the set of variables $x_\sigma$ associated to every datum $\sigma \in D_0$. $\delta_0 \subseteq L_0 \times \Sigma_0 \times E_0 \times A_0 \times L_0$ specifies the transitions, where $E_0$ is the set of enabling conditions “if $x_\sigma = 1$” and $A_0$ is the set of assignments “$x_\sigma = 1$”, for all variables $x_\sigma \in \nu_0$. Let us consider separately the input and output transitions:

An input transition of $S_0$ is defined by $[\ell, \sigma_0, -, -; \text{if } x_\sigma = 1, \ell']$, which means that if $\sigma_0/x_\sigma$ occurs while the current location is $\ell$, then the location $\ell'$ is reached and $x_\sigma$ is set to 1. Such a transition represents the reception by Orch of $\sigma$ from the environment. Hence, Rule 4.1 becomes:

**Rule 4.5:** In $S_0$, any input transition executing $\sigma_0/-$ has the assignment “$x_\sigma = 1$” and no enabling condition.

An output transition of $S_0$ is defined by $[\ell, \sigma_0, -; \text{if } x_\sigma = 1, -, \ell']$, which means that if $x_\sigma = 1$ occurs while the current location is $\ell$ and $x_\sigma$ evaluates to 1, then the location $\ell'$ is reached. Such a transition represents the sending by Orch of $\sigma$ to the environment. Hence, Rule 4.2 becomes:

**Rule 4.6:** In $S_0$, any output transition executing $/\sigma_0$ has the enabling condition “$x_\sigma = 1$” and no assignment.

**C. Example of E-IOAs $(S_i)_{i=0}^{n}$**

Figure 3 shows the E-IOAs $S_1, S_2, S_0$ obtained from the IOAs $T_1, T_2, T_0$ of Fig. 2. We have $\Sigma_1 = \{\text{SSN}_1/-, \text{/CCN}_1\}$, $D_1 = \{\text{SSN}, \text{CCN}\}$, $\nu_1 = \{x_{\text{SSN}}, x_{\text{CCN}}\}$, $\Sigma_2 = \{\text{CCN}_2/-, \text{/Appr}_2, \text{/Loan}_2\}$, $D_2 = \{\text{CCN}, \text{Appr}, \text{Loan}\}$, $\nu_2 = \{x_{\text{CCN}}, x_{\text{Appr}}, x_{\text{Loan}}\}$, $\Sigma_0 = \{\text{SSN}_0/-, \text{/Loan}_0\}$, $D_0 = \{\text{SSN}, \text{Loan}\}$, and $\nu_0 = \{x_{\text{SSN}}, x_{\text{Loan}}\}$. Locations are represented by circles, where the initial location is numbered 0. A transition from location $\ell$ to location $\ell'$ is represented by an arrow from $\ell$ to $\ell'$, where the interaction (\sigma_i/- or /\sigma_i) is above the arrow, and the enabling condition or assignment is below the arrow.
Since the three E-IOAs depend with each other due to the use of common variables \(x_\sigma\) (e.g. \(x_{\text{SSN}}\) is used by \(S_0\) and \(S_1\)), let us explain them together. “\(x_{\text{SSN}}:=1\)” associated to \(\text{SSN}_0/-\) in \(S_0\) and “\(x_{\text{SSN}}:=1\)” associated to \(\text{SSN}_1/-\) in \(S_1\), model the fact that \(\text{SSN}_1\) (Orch sends SSN to \(S_1\)) occurs after \(\text{SSN}_0\) (Orch receives SSN from the environment). “\(x_{\text{CCN}}:=1\)” associated to /CCN1 in \(S_1\) and “\(x_{\text{CCN}}:=1\)” associated to CCN2/- in \(S_2\), model the fact that CCN2 (Orch sends CCN to the environment) occurs after CCN1 (Orch receives CCN from \(S_1\)). “\(x_{\text{Loan}}:=1\)” associated to /Loan_2 in \(S_2\) and “\(x_{\text{Loan}}:=1\)” associated to /Loan_0 in \(S_0\), model the fact that Loan_0 (Orch sends Loan to the environment) occurs after Loan_2 (Orch receives Loan from \(S_2\)).

**V. ORCHESTRATOR SYNTHESIS**

**A. Preliminaries**

In Section III-C, we have defined our objective as the synthesis of an IOA \(\mathcal{O}\) from IOAs \((\mathcal{T}_i)_{i=0\ldots n}\), such that the two points of Objective 1 are respected. We have developed a synthesis procedure whose principle consists in transforming \((\mathcal{T}_i)_{i=0\ldots n}\) into \((\mathcal{S}_i)_{i=0\ldots n}\) and then computing a synchronized product \(S_0 \otimes S_1 \otimes \cdots \otimes S_n\). Note that we compute a synchronized product of E-IOAs and not of IOAs. This product is computed by the developed synthesis procedure which is presented further. The result of the product is indeed the IOA \(\mathcal{O}\) modeling Orch. The E-IOAs \((\mathcal{S}_i)_{i=0\ldots n}\) have disjoint alphabets, since any \(\sigma_i\) and \(\sigma_j\) (for \(i \neq j\)) correspond to distinct interactions. It is the variables \(x_\sigma\) and the corresponding assignment and enabling condition that permit to respect the order constraints required by Rule 3.1 while computing the product \(S_0 \otimes S_1 \otimes \cdots \otimes S_n\). Since \(\mathcal{O}\) is involved in all interactions of \((\mathcal{S}_i)_{i=0\ldots n}\), the set of data exchanged in \(\mathcal{O}\) is the union of all data exchanged in \((\mathcal{S}_i)_{i=0\ldots n}\). Before explaining how the product is realized, we need the following definitions.

**Definition 5.1:** Consider a transition \(T\) of \(S_i\) and let \(\ell\) be its origin location and \(E\) be its enabling condition. \(T\) is said **eligible** when \(\ell\) is the current location. \(T\) is said **enabled** when it is eligible and \(E\) evaluates to true. By transition of index \(i\), we mean a transition of \(S_i\).

**Definition 5.2:** Let \(p\) be the number of data \(x_\sigma\) exchanged in \(\mathcal{O}\). \(p\) is also the number of variables \(x_\sigma\) used in \(\mathcal{O}\), which are ordered from 1 to \(p\). The **index of** \(x_\sigma\) denotes its order.

In the computation of the IOA \(\mathcal{O}\), a state of \(\mathcal{O}\) is defined as \((\ell_0, \cdots, \ell_n; v_1, \cdots, v_p)\). Each \(\ell_i\) is a location of \(S_i\), and each \(v_k \in \{0, 1\}\) is a value of the variable \(x_\sigma\) of index \(k\).

For our example, we have \(p = 4\) because we have four data \(\text{SSN}, \text{CCN}, \text{Appr}, \text{Loan}\) whose respective indexes can for example be 1, 2, 3, 4. A state of \(\mathcal{O}\) is defined as \((\ell_0, \ell_1, \ell_2; v_1, v_2, v_3, v_4)\) where each \(\ell_i\) is a location of \(S_i\), for \(i = 0, 1, 2,\) and \(v_1, \cdots, v_4\) are respective valuations (0 or 1) of \(x_{\text{SSN}}, x_{\text{CCN}}, x_{\text{Appr}}, x_{\text{Loan}}\).

**Notation 5.1:** Let \(q = (\ell_0, \cdots, \ell_n; v_1, \cdots, v_p)\) be a state of \(\mathcal{O}\). Consider a location \(\ell\) of \(S_i\) (\(0 \leq i \leq n\)) and a value \(v \in \{0, 1\}\). \(q[i; \ell, j; v]\) denotes the state obtained from \(q\) by replacing \(\ell_i\) and \(v_j\) by \(\ell\) and \(v\), respectively. In particular, \(q[i; \ell]\) denotes the state obtained from \(q\) by replacing \(\ell_i\) by \(\ell\).

**B. Synthesis Procedure**

The synthesis procedure is outlined below. Its principle is to first construct the initial state of the IOA \(\mathcal{O}\), and then it constructs iteratively all the states and transitions of \(\mathcal{O}\).

**Synthesis of the orchestrator**

**Input:** IOAs \((\mathcal{T}_i)_{i=0\ldots n}\).

**Result:** IOA \(\mathcal{O}\).

**BEGIN**

Transform the \((T_i)_{i=0\ldots n}\) into E-IOAs \((S_i)_{i=0\ldots n}\)

Create the initial state \(q^0 = (p^0_0, \cdots, p^0_n; 0, \cdots, 0)\)

Initialize \(Q\) and \(R\) as \(\{q^0\}\)

Initialize \(\alpha\) as empty

while \(R \neq \emptyset\) do

| Select any state \(q = (\ell_0, \cdots, \ell_n; v_1, \cdots, v_p) \in R\) |
| for every \(S_i\) \((i = 0 \cdots n)\): |
| for every transition \(T = (\ell_i, e, C, A, \ell) \in \delta_i\): |
| if \(C\) is true: |
| if \(A\) is empty: Create the state \(r = q[i; \ell]\) |
| else: Let \(x_\sigma = v\) be the assignment \(A\) |
| Let \(j\) be the index of \(x_\sigma\) |
| Create the state \(r = q[i; \ell, j; v]\) |
| end-if |
| if \(r \notin Q\): Insert \(r\) in \(Q\) and \(R\) |
| Insert \((q, e, r)\) in \(\alpha\) |
| end-if |
| end-for |
| end-for |
| Remove \(q\) from \(R\) |
| end-while |

Reverse every \(\sigma_{i}/-\) into /\(\sigma_{i}\), and every /\(\mu_{i}\) into \(\mu_{i}/-\), for \(i = 1 \cdots n\)

Combine every consecutive \(\sigma_{i}/-\) an /\(\mu_{i}\) into \(\sigma_{i}/\mu_{j}\).

**END**

Here are explanations of the above procedure.

**Transformations:** As shown in Section IV, we must transform the IOAs \((T_i)_{i=0\ldots n}\) into E-IOAs \((S_i)_{i=0\ldots n}\).

**Initializations:** The initial state \(q^0\) of \(\mathcal{O}\) is created by combining the initial locations \((\ell^0_i)_{i=0\ldots n}\) of \((S_i)_{i=0\ldots n}\) and the initial values 0 of all \(x_\sigma\). All \(x_\sigma\) evaluate initially to 0 because Orch has not yet interacted with \((S_i)_{i=1\ldots n}\) and the environment. \(Q\) is the set that will contain all states of \(\mathcal{O}\), and \(R\) is the set that contains the current states that have been created but not treated, i.e. whose outgoing transitions have not been computed. These two sets are initialized with \(q^0\), \(\alpha\).
is the set that will contain all transitions of $O$. It is initially empty because no transition has not yet been computed.

**While-loop:** In each iteration, a state in $R$ is treated by constructing its outgoing transitions and the corresponding destination states. The while-loop iterates until $R$ is empty, which occurs when all states and transitions of $O$ have been constructed. The first instruction of the while-loop is to select a state $q$ of $R$ to be treated, and its last instruction is to remove $q$ from $R$ after it has been treated.

**Outer and inner for-loops:** the combination of the two nested for-loops inside the while-loop consists in considering every eligible transition of every $S_i$, for $i = 0 \cdots n$. More precisely, the outer-loop considers every $S_i$, and the inner-loop considers every eligible transition of $S_i$.

**Outer if ($C = true$):** the considered eligible transition is treated if it is enabled.

**Inner if-else ($A$ empty or not):** The “if” (resp. “else”) part treats the case where the considered enabled transition has no (resp. has an) assignment. The treatment consists in constructing the destination state $r$ reached in $O$ by the execution of the considered enabled transition.

**Inner if ($r \notin Q$):** It inserts the constructed state $r$ in $Q$ and $R$ if $r$ has not been previously constructed. After this “if”, the transition leading from $q$ to $r$ is inserted in $\alpha$.

Finally, inputs $\sigma_i/\omega$ and outputs $\sigma_i/\nu$, for $1 \leq i \leq n$ are reversed into outputs and inputs (see Sect. III-B), and then every pair of consecutive transitions labeled $\sigma_i/\omega$ and $\sigma_j/\nu$ are combined into a single transition labeled by $\sigma_i/\nu_j$.

For the $(T_i)_{i=0,1,2}$ of Fig. 2(a,b,c), we have seen in Section IV-C that after the transformation into E-IOAs, we obtain the $(S_i)_{i=0,1,2}$ of Fig. 3. The IOA $O$ that results from the synthesis procedure before the final combination is shown in Fig. 4(a), where each state $(\ell_0, \ell_1, \ell_2; v_1, v_2, v_3, v_4)$ is specified in two lines “$\ell_0, \ell_1, \ell_2$” and “$v_1, v_2, v_3, v_4$”. After the final combination, we obtain the IOA $O$ of Fig. 4(b). The latter IOA is similar to the expected IOA of Fig. 2(d).

![Fig. 4. Synthesized $O$: (a) before combination; (b) after combination.](image)

**VI. RELATED WORK, DISCUSSION AND CONTRIBUTION**

Automated synthesis of WS composition has been intensively studied in the last decade. Due to the great number of
work in the area, several interesting surveys and overviews have been published such as [4], [5], [6], [7], [8], [9]. These surveys categorize approaches for composing WSs, identify requirements that should be respected in WS composition, and analyze and compare design approaches with regard to their capabilities to respect the identified requirements.

Due to our experience in automata-based design methods in various subjects, we have been interested by approaches of WS composition using finite state automata. For example, the authors of [10] study how to solve the problem of synthesizing automatically WS composition using automata and the control theory of discrete event systems [11]. They study conditions that make this problem decidable.

[1], [12], [13], [14] have particularly drawn our interest. [1] develops an Input-output-automata (IOA)-based method to synthesize a choreographer that coordinates several services to realize a desired service. [12] improves [1] by reducing state space explosion. [13], [14] adapt the work in [1], [12] for decentralized choreography. The synthesis methods developed in [1], [12], [13], [14] are based on known notions (closure, universal automaton, simulation) and other new notions based on Transduce (transduced-closure, transduced-realizability, transducing-choreographer) and History. All these notions require developing new operators which are uselessly complex. This observation motivated us to develop the simpler approach presented in this paper. Since [12], [13], [14] are continuations of the original approach of [1], let us discuss our contributions compared to [1]:

- The above mentioned notions (e.g. transducing, history) are avoided by using Extended IOAs (E-IOAs) which consist in associating enabling conditions and assignments to transitions. E-IOAs may be complex in general, but this is not our case because we use trivial enabling condition and assignments. The most important (but simple) operator we use to synthesize the orchestrator, is a synchronized product between the extended E-IOAs modeling the services to be composed and the desired service. E-IOAs have not to be constructed or used by the designer of WS. Instead, the synthesis method generates automatically E-IOAs from the IOAs modeling the services.
- The orchestrator we have developed corresponds to the transducing choreographer of [1] which is a complex sophistication of a first simple choreographer. Interestingly, our approach generates naturally the equivalent of the complex version, and we need to adapt it to obtain the simple choreographer. The adaptation can be simply done as follows. We can specify that the orchestrator forwards at most once a received σ, by adding the assignment “$x_{σ} := 0$” to every sending of σ by the orchestrator. We can also specify that the orchestrator forwards σ immediately after its reception, by using adequate enabling conditions and assignments that block every interaction of the orchestrator until σ has been forwarded. We conjecture that we will be capable to design more sophisticated and valued orchestrators, by using various kinds of enabling conditions and assignment.
- In addition to synthesizing the orchestrator, we show how to verify whether the provided service realizes totally or partially the desired service. Moreover, we show how to determine which portion of the desired service is actually provided.
- The formal proofs of our propositions are in a longer version of this paper [15].

VII. CONCLUSION

We have presented an automata based method to synthesize an orchestrator that coordinates several services so that a desired service is provided. The method has the advantage of being simpler than other synthesis methods thanks to the use of specific extended input-output automata.

For future work, we will study how the same approach can be simply adapted for using more sophisticated orchestrators that do more than just forwarding received data. We will investigate how the adaptation can be realized by identifying various adequate enabling conditions and assignments. We also plan to implement our method in order to apply it to real-life examples.

REFERENCES