Optimum Decentralized Choreography for Web Services Composition

Saayan Mitra  
Dept. of Elec. & Comp. Eng.  
Iowa State University  
Ames, IA 50011, USA  
Email: saayan@iastate.edu

Ratnesh Kumar  
Dept. of Elec. & Comp. Eng.  
Iowa State University  
Ames, IA 50011, USA  
Email: rkumar@iastate.edu

Samik Basu  
Dept. of Computer Science  
Iowa State University  
Ames, IA 50011, USA  
Email: sbasu@cs.iastate.edu

Abstract

Typical solutions for Web services composition problem develop a single intermediary which mediates or choreographs computation and communication among the existing services to realize a target/goal service. However, such a centralized choreography mechanism can involve communication/computation overhead that can be reduced through its decentralized realization. With this as motivation, we study the problem of synthesizing a decentralized choreography strategy that will have an optimum overhead for service composition by developing a set of site-specific choreographers working concurrently to implement a desired goal service. Each communication/computation is quantified by a cost. We develop an algorithm that takes as input the existing services, the goal service, the costs and produces as an output a set of site-specific choreographers working concurrently to implement a desired goal service. Each communication/computation is quantified by a cost. We develop an algorithm that takes as input the existing services, the goal service, the costs and produces as an output a set of site-specific choreographers working concurrently to implement a desired goal service. The contribution lies in the formulation of the optimal decentralized choreographer synthesis problem as well as its solution and extends our earlier work [13, 12] in two ways: decentralization of solution and its optimality.

1. Introduction

In the recent past, service based software development where existing services are re-used to develop and deploy new services has gained significant interest both in academia and industry. As a result a number of techniques [17, 2, 4, 7, 19, 13, 12], formal and informal, have been proposed to (semi-)automatically develop composition of services that can realize the functionality of a desired new service, often referred to as the goal service. These techniques rely on developing a single choreographer which mediates the computation and communication among the existing services and the client. In such a centralized scheme of choreography, every computation and communication is coordinated by a single choreographer.

Such a centralized mechanism entails invoking services and relaying messages for all the required computation or communication transactions. However, in most practical scenarios, there is a cost associated with each computation and communication and as such, it is important to identify a choreography mechanism which incurs the minimum overall cost. The cost can be defined in terms of network bandwidth, congestion, or pure monetary value for using some network or service. Here, we address the problem of minimizing the overall cost of choreography by synthesizing an optimum decentralized choreography strategy. Decentralization amounts to generating multiple choreographers; one for each service in the composition (some sites may turn out to have no choreographer in the optimal solution).

Illustrative Example. We present a brief overview of the problem along with our solution mechanism using the following running example obtained from [8]. There are three existing services (Figure 1(a)) $S_1$, $S_2$ and $S_3$. $S_1$ takes as input name of a person ($nP$) and provides his/her address ($aP$). Service $S_2$ can perform all input/output operations of $S_1$ and can take as input name of a point of interest or business ($nP$) and output its address ($aC$). And finally, $S_3$ takes as input two addresses and computes the route ($r$) between them. The developer wants to create a new service, $G$, for a client which takes ($nP$) and ($nC$) as inputs and provides the route ($r$) between them. The developer wants to create a new service, $G$, for a client which takes ($nP$) and ($nC$) as inputs and provides the route between them. The developer wants to create a new service, $G$, for a client which takes ($nP$) and ($nC$) as inputs and provides the route between them. The developer wants to create a new service, $G$, for a client which takes ($nP$) and ($nC$) as inputs and provides the route between them. The developer wants to create a new service, $G$, for a client which takes ($nP$) and ($nC$) as inputs and provides the route between them. The developer wants to create a new service, $G$, for a client which takes ($nP$) and ($nC$) as inputs and provides the route between them.

Now consider that the cost of communication (in terms of network traffic, bandwidth etc.) between the client and services and between any services are as in the table of Figure 1(b). In the table we refer to the client site as $s_0$. It is desirable to develop a mechanism such that all the messages...
are not required to be relayed via a centralized choreographer such as $C_0$. Instead multiple choreographers are deployed at servers which are at close proximity (physically or in terms of request/response delay) to the existing services. The objective is to implement the goal service with minimum possible cost (following the cost table) where message exchanges do not necessarily require relaying through $C_0$.

A possible solution for the current example is presented in Figure 1(d) where in addition to $C_0$, three choreographers $C_1$, $C_2$, and $C_3$ are deployed near the existing service sites. Observe that, the messages ($a_P$ and $a_C$) are not required by the client and as such they are not sent to $C_0$. Also, observe that service $S_2$ is not used to obtain the $a_P$ from input $n_P$. This is because the cost of communication between the client and $S_2$ and between $S_2$ and $S_3$ is more than the corresponding communication cost for service $S_1$. In short, four choreographers are used to minimize the cost of exchanging input/output messages between the services and the client.

**Objective.** The objective of designing an *optimal decentralized choreography scheme* requires designing the behaviors of each site-specific choreographer such that an overall cost of choreography (as measured using communication and computation costs) is minimized. Note some site-specific choreographers may not be needed and their behaviors may turn out to be empty.

**Proposed solution.** The solution proposed in this paper has two steps. In the first step, we identify whether there exists a choreographed composition of existing services such that the given goal service, represented as i/o-automata (Section 2), can be realized. This is solved by computing the universal (all possible) choreographed behaviors (via inter-leaving and transduced-closure) that can be realized from the existing services and verifying (via simulation) whether the goal behavior can be subsumed by the universal behaviors (Section 3). The next step (Section 4) is to synthesize the choreographer at each service site such that an overall communication and computation cost of realizing the goal service is minimized. For this, the transitions in the universal composite behaviors of the existing services, that subsume the goal behavior, is annotated with their cost valuations. For example the cost of the composite behaviors as presented in Figure 1(c) is equal to the sum of the cost of exchanging messages between $C_0$ and $S_1$ twice, between $C_0$ and $S_2$ twice, and between $C_0$ and $S_3$ thrice. We identify the cost of all possible choreographed compositions and select the one which has the minimum cost by way of a backward search (as in dynamic programming).

The contributions of our work can be summarized as follows. This is a first approach which presents an automated solution to optimum decentralized choreography for Web services composition. Our approach is based on I/O-automata representation of the services and the goal, and identifies appropriate choreography scheme using the notions of universal service (obtained as a transduced-closure of given services), simulation relation and optimal worst-case path-cost computation for a graph. The technique is provably sound and complete. (Due to space limitation, a formal proof is not included.)

2. Services as I/O-automata

The behaviors of a Web service can be described as a set of sequences of input and output computations it can per-
form. I/O-automata can naturally represent such behaviors. The states in an I/O-automaton represent the configuration of a service and the state transitions represent the changes from one configuration to another. The transitions are labeled with input/output actions. For an alphabet set $\Sigma$ we use $\Sigma$ to denote $\Sigma \cup \{\epsilon\}$.

**Definition 1 (I/O Automaton)** An I/O-automaton $A$ is defined by a tuple $(S, S^0, I, O, \Delta)$, where, $S$ is the set of states, $S^0 \subseteq S$ is the set of initial states, $I$ is the set of inputs, $O$ is the set of outputs, and $\Delta \subseteq S \times I \times O \times S$ is the set of transitions. An element of $\Delta$, represented by $(s, i, o, s')$, is such that $s \in S$ is the origin state of the transition, $i \in I$ is the input to the transition, $o \in O$ is the output of the transition, and $s' \in S$ is the destination state of the transition. We use $s \xrightarrow{i/o} s'$ to denote $(s, i, o, s') \in \Delta$.

Figure 1(e) presents the I/O-automata models for the three services and goal described in Figure 1(a, b). The automaton $A_i$ ($i = 1, 2, 3$) corresponds to the $i$th service and the automaton $A_0$ corresponds to the goal. The start states of the automata have curved arrows pointed to them.

### 3. Choreographer Existence

The individual services can be used to implement a goal service by realizing each input/output computations of the existing services such that the input (resp., output) of the first (resp., last) computation in the sequence is the same as the input (resp., output) of the goal. In order to execute an input/output computation, input to a service is provided by the services or client which have seen and/or stored the said input. In other words, the input to a service comes from the “history” of information present in another service or the client. To formalize these concepts we define the notion of **interleaving product with distributed history**.

#### 3.1. Product with Distributed History

We allow a choreographer to be associated with each service site and the client site. Suppose there are $N$ services located at sites $1, \ldots, N$. The service at site-$n$ ($n \in \{1, \ldots, N\}$) is modeled as an I/O-automaton $A_n = (S_n, S^0_n, I_n, O_n, \Delta_n)$. We designate site-0 as the site interfacing with the client. Associated with each site-specific choreographer is a site-specific history set consisting of the inputs and outputs seen and stored.

A certain input/output transition of the desired goal service, can be implemented by executing a sequence of input/output transitions at various sites. To facilitate the computation of all such sequences we define the notion of **interleaving product with distributed history**. A local history set of a site consists of all inputs and outputs that it has seen in past. Elements in this local history set can be used to supply the input of a transition to be executed in future; this will become clear in subsequent discussions.)

In the interleaving product, exactly one site participates in the execution of each transition, and accordingly each transition is tagged with a site-index $n$, that identifies the participant site. Execution of such a transition at site-$n$ augments its local history by the input-output pair of the transition. The following definition of **Interleaving Product With Distributed History** ($\|_{\|} \times_n A_n$) captures all possible interleaved behaviors of the service automata and the associated distributed histories. Note the distributed history is essential for the synthesis of a decentralized choreographer. Only a centralized history was considered in our earlier work [13, 12]. In the following, the notation $\vec{v}(n)$ is used to denote the $n$th element of a vector/tuple $\vec{v}$.

**Definition 2 (I/O $\|_{\|} \times_n A_n$ Automaton)** Given service automata

\[ A_n = (S_n, S^0_n, I_n, O_n, \Delta_n) \forall 1 \leq n \leq N, \]

their interleaving product with distributed history is defined as the I/O-automaton

\[ (\|_{\|} \times_n A_n) = (\|_{\|} \times_n S_n, \|_{\|} \times_n S^0_n, \|_{\|} \times_n I_n, \|_{\|} \times_n O_n, \|_{\|} \times_n \Delta_n) \]

where

\[ \|_{\|} \times_n S_n = \bigcap_{n=1}^{N} S_n, \|_{\|} \times_n S^0_n = \bigcap_{n=1}^{N} S^0_n, \]

\[ \|_{\|} \times_n I_n = \bigcup_{n=1}^{N} I_n, \|_{\|} \times_n O_n = \bigcup_{n=1}^{N} O_n, \]

and

\[ (\|_{\|} \times_n \vec{s}, \|_{\|} \times_n \vec{h}) = (\vec{s}, \vec{h}) \in \|_{\|} \times_n \Delta \text{ if and only if} \]

\[ \|_{\|} \times_n s(n) \xrightarrow{i/o} \|_{\|} \times_n s'(n) \in \Delta_n \land \|_{\|} \times_n h(n) = \|_{\|} \times_n h'(n) \cup \{i, o\} \land \]

\[ \forall m \neq n : \|_{\|} \times_n s(m) = \|_{\|} \times_n s(m) \land \|_{\|} \times_n h'(m) = \|_{\|} \times_n h(m). \]

In the definition of $\|_{\|} \times_n \Delta$, the first conjunct states that whenever a constituent service makes a move, the $\|_{\|} \times_n A_n$ automaton also makes a move with the same transition label. The second conjunct states that the local history of the participant service is updated with the input and output transition labels. The third and fourth conjuncts state the facts that other services do not change states or their corresponding local histories. Thus, if a service gets the input $i$ and produces output $o$, its local history is enriched by $\{i, o\}$. 

![Figure 2: Interleaving Product Automaton, $\|_{\|} \times_n A_n$](image-url)
Example 1  Figure 2 depicts a part of the automaton $||H|A_n$ for $A_1$, $A_2$ and $A_3$ presented in Figure 1(e). Observe that, the history of the start state is empty. Every state is shown with the local history associated with it and transitions are labeled with the participating service responsible for the transition. Thus, $sizzle \{ \{ r \} \}$ changes its configuration to $siz \{ \{ n \} \}$ as $A_2$ makes a move.

3.2. Transduced Closure Automata

A site can get data from its own local history or from the local history of another site to execute future transitions. There is a cost associated with any such communication of data, and we use $c(n, m) \in \mathbb{R}_+$, where $\mathbb{R}_+$ is the set of non-negative reals, to denote the (cheapest) cost of communicating a data from site-$n$ to site-$m$. The cost can be any numeric valuation quantifying various aspects of communication; e.g., network traffic, distance between servers, number of hops for each communication. Note that, communication between a pair of sites $n$ and $m$ will in general involve multiple options (such as different routes either directly between $n$ and $m$ or via intermediate nodes), and $c(n, m)$ denotes the cheapest option. The Table in Figure 1(b) presents the communication cost for our example. Utilizing the local histories, a sequence of input/output computations can be performed by the various site-services without the intervention of the client-site choreographer. The inputs for these computations are produced from the history of the nearest site repository, whereas the outputs are sent to the client-site only when needed. The universe of all target or goal services that can be accomplished in this manner is computed via the transduced-closure of an automaton with distributed history ($||H|A_n$), defined as follows. Note this definition is significantly different from the notion of transduced-closure introduced in our earlier work [13, 12], for it is designed to support decentralization and optimality.

Definition 3 ($||H|A_n$) Automaton) Given an interleaving product automaton with distributed history $||H|A_n = (S \times \hat{H}, S^0 \times \hat{H}^0, I_0, O_0, \delta_H)$ of $\{A_n\}_{1 \leq n \leq N}$, its transduced-closure is the automaton $(||H|A_n)^T = (S \times (2^{0 \cup O_0} \times \hat{H}), S^0 \times (0 \times \hat{H}^0), I_0, O_0, \delta^T_H)$, where $\langle s, (h_0, \vec{h}) \rangle \xrightarrow{i_n \in \gamma}_{\delta^T_H} \langle \hat{s}, (h'_0, \vec{h'}) \rangle \in \Delta^T_H$ if and only if

1. $m \in m.$

\[ h'_0 = h_0 \cup \{ i, o \} \]

\[ c := \sum_{k=2}^{m} \left[ \{ c(n, k) | 1 \leq n \leq N : i_k \in \tilde{h}_k(n) \} \right] \cup \{ c(0, n) | 1 \leq n \leq N : i_k \in \tilde{h}_k(n) \}

\[ + c(0, n) + c(n, m) \]

4. $\gamma = (src_2, \ldots, src_m)$ where for $2 \leq k \leq m$

\[ src_2 := \arg \min \left( \{ c(n, k) | 1 \leq n \leq N : i_k \in \tilde{h}_k(n) \} \right) \]

We call $U := (||H|A_n)^T$ to be the universal service automaton corresponding to the service-automata $\{A_n | 1 \leq n \leq N\}$.

Observe that, in the above definition, the states of $U$ are represented by the states of $||H|A_n$ coupled with elements from $2^{O_0}$. The extra elements represent a history set of the client-site, i.e., the inputs and outputs seen by the client-site choreographer. The above definition states that every transition in $||H|A_n$ is also a transition in $U$, as the cost associated with any such communication of data from site-$n$ to site-$m$.

The transduced-closure of a sequence of service-transitions is possible when certain conditions are satisfied. The first condition of the states that the source state, the input, the destination state, and the output of the transduced-closure transition matches respectively with the source state and the input of the first transition in the sequence, and destination state and the output of the last transition in the sequence. Furthermore, the input of each transition in the sequence should be present in some local history associated with its source state.

The second condition states that the history at the client site is updated by the input and output of the transduced-closure transition (since such a transition implements a goal transition whose input and output are relayed by the client-site choreographer). The third condition computes the overall cost of a transduced-closure transition as the sum of the costs of all the individual transitions in the sequence. The last condition identifies the site from which the input of an intermediate transition is obtained—it is the site possessing the input in its local history and nearest (in terms of communication cost) to the site executing the transition. Each transduced-closure transition is annotated with its cost $c$ and the tuple $\gamma$ of the nearest sources from where the inputs (for the sequence of transitions implementing the transduced-closure transition) are obtained.

Example 2 Figure 3(a) depicts a part of the transduced closure automaton $U$ obtained from $||H|A_n$ (Figure 2) of $A_1$, $A_2$ and $A_3$ (Figure 1(e)). The history at the client site, $h_0$ is shown within $[\ldots]$. The dotted transition corresponds to the transitions obtained via the transduced-closure of a sequence of transitions. E.g., the transition
Figure 3: (a) Transduced Closure Automaton, $U$, (b) Valuations of $\gamma_l, s$, (c) Simulating synchronous product $A_0 \times U$, (d) Mincost Choreography Automaton, $C$, (e) $C_0$, (f) $C_1$, (g) $C_2$ and (h) $C_3$.

$A = \langle S, S_0, \delta, \gamma_l, s, \Delta, \mu, G \rangle$ is a goal automaton.

3.3. Realizability of goal

A given goal service $A_0$ is realizable from the existing services under a centralized/decentralized choreographer if and only if all input/output behaviors of $A_0$ are also present in the universal service automaton $U$. Note that the inputs in $A_0$ come from the client and the outputs from $A_0$ go to the client. Similarly the transition labels in $U$ have inputs coming from the client and the outputs going to the client. The realizability of a goal using the existing services is verified using checking whether $A_0$ is simulated by $U$.

Definition 4 (Simulation [11]) Given a goal automaton $A_0 = \langle S_0, S_0^0, I_0, O_0, \Delta_0 \rangle$ and an universal service automaton $U = \langle S_U, S_U^0, I_0, O_0, \Delta_U \rangle$, a state $s_1 \in S_0$ is simulated by a state $s_2 \in S_U$ if and only if they are related by the largest simulation relation denoted by $s_1 \sqsubseteq s_2$ and defined as: $s_1 \sqsubseteq s_2 \Rightarrow [\forall t : s_1 \xrightarrow{i/o} t_1 \in \Delta_0 \Rightarrow (\exists t_2 : s_2 \xrightarrow{i/o}_{c, \gamma} t_2 \in \Delta_U \wedge t_1 \sqsubseteq t_2)]$. $A_0$ is said to be simulated by $U$, denoted by $A_0 \sqsubseteq U$, if all states in $S_0^0$ are simulated by some state in $S_U^0$.

Then we have the following result.
Theorem 1 Given a goal $A_0$ and a set of services $\{A_n \mid 1 \leq n \leq N\}$, the goal is realizable from the choreography of $\{A_n \mid 1 \leq n \leq N\}$ if and only if $A_0 \subseteq U$ where $U$ is the transduced-closure of the $\prod \hat{A}_n$-automaton, and $\prod \hat{A}_n$ is the interleaving product with distributed history of the automata $\{A_n \mid 1 \leq n \leq N\}$.

Example 3 It can be seen that the goal $A_0$ given in Figure 1(e) is simulated by the $U$-automaton in the Figure 3(a). Thus $A_0$ can be realized by choreographing the services $A_1, A_2, A_3$ of Figure 1(e).

It can be verified that $A_0 \subseteq U$ holds if and only if $A_0 \subseteq A_0 \times U$ holds, where $A_0 \times U$ denotes “simulating synchronous product” of $A_0$ and $U$ as defined below:

**Definition 5 (Simulating Synchronous Product)** Given a goal $A_0 = (S_0, S_0^U, I_0, O_0, \Delta_0)$ and an universal service automaton $U = (S_U, S_U^0, I_0, O_0, \Delta_U), their simulating synchronous product is the automaton $A_0 \times U = (S_0 \times S_U, S_0^U \times S_U^0, I_0, O_0, \Delta \times)$, where

$$(s_0, s_u) \xrightarrow{i/o, c, \gamma} (s'_0, s'_u) \in \Delta \times \iff \begin{cases} s_0 \xrightarrow{i/o} s'_0 \land s_u \xrightarrow{i/o} s'_u, \\ \land s_0 \subseteq s_u \wedge s'_0 \subseteq s'_u. \end{cases}$$

Usually the size of $A_0 \times U$ is smaller compared to the size of $U$ (since size of $A_0$ is smaller compared size of $U$). Hence it is preferable to check whether $A_0 \subseteq A_0 \times U$ holds (as opposed to checking whether $A_0 \subseteq U$ holds).

Example 4 Figure 3(c) shows the simulating synchronous product of the goal automaton $A_0$ in Figure 1(e) and universal service automaton $U$ in Figure 3(a). It can be seen from inspecting $A_0 \times U$ and $A_0$ that $A_0 \subseteq A_0 \times U$ holds. Here, $A_0 \times U$ has two paths from the start state both of which can yield choreographies. From the associated histories in the paths it can be seen that one path uses service $A_1$ for computing $a\#a/a\#a$ and other path uses service $A_2$ for the same. In the following sections we introduce the algorithm for choosing the optimal solution.

**4. Optimum Decentralization**

Realizability of a goal $A_0$ by choreographing a set of services $\{A_n \mid 1 \leq n \leq N\}$ is guaranteed by the satisfaction of $A_0 \subseteq \prod \hat{A}_n$. It is possible that $A_0$ can be simulated by $A_0 \times U$ in multiple ways since $A_0 \times U$ can possess multiple subautomata each of which can simulate $A_0$. Thus there can be multiple realizations of $A_0$, each with its own cost (as defined below). Our goal then is to find an optimum cost realization of $A_0$, which we approach by finding an optimal cost subautomaton of $A_0 \times U$ that simulates $A_0$. In what follows next, we assume for simplicity of presentation that $A_0$ is loop-free. This then implies that $A_0 \times U$ is also loop-free. We proceed by defining the cost of a subautomaton of $A_0 \times U$.

**Definition 6 (Cost of an Automaton)** Given a loop-free I/O-automaton with its each transition labeled by a cost, we define the cost of a path to be the sum of the costs of all the transitions in the path. We define the cost of a state to be the maximum cost among all paths originating at that state and terminating at a deadlocking state (state with no outgoing transitions). The cost of an automaton is defined to be the maximum cost among all its initial states. We represent the cost of a loop-free I/O-automaton $A$ as $\text{cost}(A)$.

Using the notion of cost from Definition 6, we define the minimum cost choreography automaton for realizing $A_0$ from $\{A_n \mid 1 \leq n \leq N\}$.

**Definition 7 (MinCost Choreography Automaton)** Given a goal automaton $A_0$ and an universal service automaton $U$ such that $A_0 \subseteq A_0 \times U$, the minimum cost for choreography is obtained as the cost of a subautomaton $C$ of $A_0 \times U$ such that $A_0 \subseteq C$ and for all subautomata $C'$ of $A_0 \times U$ with $A_0 \subseteq C'$, it holds that $\text{cost}(C') \leq \text{cost}(C)$.

**4.1. Computing optimum cost**

In the following we present an algorithm for computing a subautomaton of $A_0 \times U$ from which an optimum choreographer can be extracted. Given a state $(s_0, s_u)$ of $A_0 \times U$, a certain goal transition $s_0 \xrightarrow{i/o} s'_0$ may be simulated by multiple transitions of the type $(s_0, s_u) \xrightarrow{i/o} (s_0, s_u)$ in $A_0 \times U$. The algorithm identifies the minimum cost option by searching over all alternatives. The computation starts from the deadlocking states by assigning their cost to be zero, and recursively proceeds backwards by assigning costs to the predecessor states.

**Algorithm 1**

$$\text{cost}(s_0, s_u) = \begin{cases} 0 & \text{if } (s_0, s_u) \text{ is a deadlocking state} \\ \max_{s_0 \xrightarrow{i/o} s'_0 \in \Delta_0} \min_{(s_0, s_u) \xrightarrow{i/o} (s_0, s_u) \in \Delta \times} \{c + \text{cost}(s_0, s_u)\} & \text{otherwise} \end{cases}$$

In the above, the first case states that the cost of a deadlocking state $(s_0, s_u)$ is 0 as there is no path from such a state. The second case corresponds to non-deadlocking states. The cost of such a state $(s_0, s_u)$ can be understood in two steps. In the first step (minimization), we identify the cheapest way of simulating a goal transition $s_0 \xrightarrow{i/o} s'_0$ originating at $s_0$. This corresponds to a transition of $A_0 \times U$ labeled by $i/o$ having the least sum of the cost of the transition and the cost of its destination state. In the second step (maximization), we identify the worst cost of simulating a transition of the type $s_0 \xrightarrow{i/o} s'_0$ originating at $s_0$. 


Example 5 For our running example, the automaton $C$, shown in Figure 3(d), represents the optimum choreographer (from the two candidate choreographers: Figure 3(c)). One choreography obtains the $\pi_n/\omega$ from $A_1$ (service 1) while the other does the same operation using $A_2$ (service 2). In both case, $\omega$ is sent to $A_3$ (service 3) for outputting $v$. Note that the sum of communication cost between client-site and service 1 and between service 1 and service 3 is less than the sum of cost of communication between client-site and service 2 and between service 2 and service 3 (see Figure 1(b)). As a result, the transition simulating $\pi_n/\epsilon$ has two different costs associated with it in Figure 3(c): in one it is the sum of communication cost between client to service 1, service 1 to service 3 and service 3 to client (total: 40); in the other it is the sum of the communication cost between client to service 2, service 2 to service 3 and service 3 to client (total: 45). The first case produces a minimum cost choreography strategy.

4.2. Synthesizing optimum choreographers

Starting from a subautomaton $C$ of $A_0 \times U$ representing an optimum choreography scheme, a set of site-specific choreographers achieving the optimum cost can be obtained using the following algorithm.

Algorithm 2 (Site-specific Choreographer) Given a minimum cost choreography subautomaton $C$ of $A_0 \times U$, the choreographer at site-$n$ ($0 \leq n \leq N$) is a communicating I/O-automaton $C_n = (S_n \times 2^{I_n \cup O_n}, S^0_n \times \{\emptyset\}, I_n, O_n, E_n, \Delta^C_n)$, where $S_n, S^0_n, I_n, O_n$ are as in the site-$n$ service or goal automaton $A_n$, while $E_n : S_n \times 2^{I_n \cup O_n} \rightarrow 2^{(I_n \cup O_n) \times \{0, \ldots, N\}}$ labels each state of $C_n$ with a set of data (that $C_n$ should send in this state) along with their destinations. $E_n$ and $\Delta^C_n$ are obtained as follows. Suppose there exists a transition $(s_0, s_1, (h_0, h_1^*)) \xrightarrow{i/o} (n, h_0^*, h_1^*)$ in $C$, which implies:

\[
\begin{align*}
(s_0, \tilde{h}) \xrightarrow{i/o}_{n_1} (s_2, \tilde{h}_2) & \in \Delta^C_n \land \\
(s_2, \tilde{h}_2) \xrightarrow{i/o}_{n_2} (s_3, \tilde{h}_3) & \in \Delta^C_n \land \\
(s_m, \tilde{h}_m) \xrightarrow{i/o}_{n_m} (s_{m+1}, \tilde{h}_{m+1}) & \in \Delta^C_n \land \\
\forall 2 \leq k \leq m : i_k & \in h_0 \cup \bigcup_{1 \leq n \leq N} \tilde{h}_k(n)
\end{align*}
\]

- $h_0' = h_0 \cup \{i, o\}$
- $c = \sum_{k=2}^{m} \min \{c(n, n_k) | 1 \leq n \leq N : i_k \in \tilde{h}_k(n)\} + c(0, n_1) + c(m, 0)$

\[\gamma = (s_{c2}, \ldots, s_{cm})\text{ where for }2 \leq k \leq m:\]

\[s_{ck} := \arg \left[ \min \left( \left\{ c(n, n_k) | 1 \leq n \leq N : i_k \in \tilde{h}_k(n) \right\} \cup \{c(0, n_k) | i_k \in h_0\} \right) \right] \]

Then

1. $\forall 1 < k \leq m : [(n_k = n) \Rightarrow \left( s_k(n), \tilde{h}_k(n) \right) \xrightarrow{i/o} \left( s_{k+1}(n), \tilde{h}_{k+1}(n) \right) \in \Delta^C_n]$.
2. $\forall 1 < k \leq m : [(s_{ck} = n) \Rightarrow \left( i_k, n_k \right) \in E_n(s_k(n), \tilde{h}_k(n))]$.
3. $(n_m = n) \Rightarrow (0, 0) \in E_n(s_m+1(n), \tilde{h}_{m+1}(n))$.
4. $(i, n_1) \in E_0(s_0, h_0)$.
5. $(s_0, h_0) \xrightarrow{i/o} (s_0', h_0') \in \Delta^C_0$.

Observe that the service-site choreographer $C_n$ is a sub-automaton of the “history-augmented” service automaton $A_n$, but with the added feature that it can perform certain transmissions at its states (as determined by the labeling function $E_n$). If the $k$th service-transition in a transduced-closure transition is such that $n_k = n$ (item 1), it implies that the $k$th service-transition is executed at site-$n$, and in which case site-$n$ performs this computation (and doesn’t initiate any communication). On the other hand if the $k$th service-transition is such that $s_{ck} = n$ (item 2), then site-$n$ is the source for the input $i_k$, which it sends to the site-$n_k$ (where the $k$th service-transition is executed). When $n_m = n$ (item 3), the last service-transition in a transduced-closure transition is executed at site-$n$ which sends the output $o$ to site-0. Similarly, site-0 sends the input $i$ to the site-$n_1$ (item 4) that executes the 1st service-transition in the sequence.

Example 6 Figure 3(e, f, g, h) shows the various site-specific choreographers as obtained by applying Algorithm 2. The labeling of states as implied by the $E_n$ function is shown within $<>$. In Figure 3(e), we obtain the choreographer $C_0$ at the client-site. $C_0$ communicates $\pi$ to the choreographer $C_1$ at site-1, and $\omega$ to the site-2 choreographer $C_2$ from the initial and the next successor states, respectively. In Figure 3(b), it can be seen that after $C_1$ (at service 1 site) obtains $\pi$ from $C_0$, it computes the output $\omega$ and sends that to $C_3$. Other choreographers can be explained in similar fashion. They realize the goal in with minimum cost (as per cost table in Figure 1(b)) as the choreographers are obtained from the minimum cost automaton.
5. Related Work

A number of recent papers [18, 3, 4, 16, 10] have addressed the automatic Web service composition problem. In the context of decentralized computation, the merits of distributing the data storage and execution structure are discussed in [9]. Several approaches, e.g., BondFlow, Symphony, OSIRIS, have been developed to address the problem of decentralization in the industry (see [1] for details). Traditionally, partition-based approaches are employed to handle decentralization. [14] used workflow partitioning of state chart models while [15] relied on program dependence graph partitioning. Following the same path, the decentralizing the execution of web services has been dealt in [8]. [5] proposes a complete decentralization scheme with service invocation triggers to route traffic efficiently. More recently, in [20] a formal model is proposed which handles synchronization and concurrency constraints in decentralizing service compositions. In [6], the authors discuss the security related issues of decentralization.

In contrast to all the existing work which require manual guidance for realizing desired decentralization, our methodology is automatic. Our technique also generates optimum decentralization with respect to a cost valuation which can quantify any desired feature, i.e., network bandwidth, traffic, monetary value of service etc.

6. Conclusion

In this paper we have addressed the problem of decentralizing the choreography of web services for achieving cost benefits. We have systematically formulated this problem and presented algorithms for finding an optimal decentralized scheme of choreography. The state space exploration of the Universal Automaton has been reduced by taking the simulating synchronous product with the goal. The notions of interleaving product with distributed history, transduced-closure, and simulating synchronous product are introduced for formulating and solving the problem of optimum decentralized choreography. For the solution, we developed an algorithm for computing an optimum cost subautomaton of the universal service automaton, and also presented another algorithm to “extract” the decentralized choreographers from such an optimum cost subautomaton. The paper presents a first automated technique for solving the problem of optimum decentralized choreography for web services composition. Implementing the technique with various use cases is among future avenues of research.

Acknowledgment. The research was supported in part by the National Science Foundation under the grants NSF-CCF-0702758, NSF-CNS-0709217, NSF-ECS-0218207, NSF-ECS-0244732, NSF-EPNES-0323379, NSF-0424048, and NSF-ECS-0601570.

References