A Framework for Optimal Decentralized Service-Choreography *

Saayan Mitra1
Ratnesh Kumar1
Samik Basu2

1 Department of Electrical and Computer Engineering
2 Department of Computer Science
Iowa State University, Ames, IA
Email:{saayan,rkumar,sbasu}@iastate.edu

Abstract

We address the problem of optimizing mediator-based service composition where the services and the desired composition (goal) functionality are represented as i/o-automata with loops. The objective of optimization is to minimize the costs of communications and computations necessary to realize the goal from the existing services. We develop an algorithm to compute the minimum cost of an automaton representing the choreographed behavior of services realizing the goal. This forms the central theme of our technique for developing automatically a strategy of decentralized mediation that will result in the optimized composition of services.

1 Introduction

Web Services composition, that composes existing services to realize a target service known as the goal service has followed two kinds of methodology. One depends on a centralized mediator (often referred to as orchestrator or choreographer) to realize a goal service [15, 4, 6, 8, 12]; while the other approach is decentralized choreography where mediators are placed physically close to the services and the client for optimality of performance (w.r.t. computation and communication costs) [7, 9].

The existing techniques for decentralized choreography require manual guidance. In our previous work [13], we provided an automata-theoretic composition algorithm which identifies appropriate decentralization necessary to minimize computation and communication costs of composition having loop-free goal specifications (workflow). In this paper, we extend that work to deal with loops in goal specifications. The introduction of loops, however, leads to a number of complications as cost is typically an additive feature, and as such simple additive cost computation will not suffice for compositions which may contain loops. Completely new insights are required to solve the problem which explores the trade-off between the “penalty” versus “payoff” of reaching a nonterminating final state (see Remarks 1 & 3).

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, where loops are considered in the goal service. 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 interleaving and transduced-closure of the given services), simulation relation, and worst-case path-cost minimization over graphs. The technique is provably sound and complete.

2 Illustrative Example

Figure 1(a) presents sequence diagrams of three services S1, S2 and S3. S1 takes as input a product name (p) and provides some information (inf) about the product, such as weight, size etc. Service S2 takes as input the information (inf) about (p) and its shipping address (a) and provides as output the price (prc) for shipping (p) to the address (a). The client can decide to cancel (c) the shipping after the quote (prc) is given out as output. S3 is a service similar to S2, the difference being it is located somewhere other than S2 and has different computation costs for its operations and communication costs of inputs and outputs from and to the client are different as well. Note that S2 and S3 are shown by the same sequence diagram. The developer wants to create a new service (goal service), S0, for a client which provides (p) and (a) and expects a quote (prc), additionally it can cancel the order as shown in Figure 1(a). The cost of communication (in terms of usage of the network) between the client and the services, and between the services is presented in Figure 1(b-i). Another table (Figure 1(b-ii)) illustrates the computation cost for each operation in each

*The research was supported in part by the National Science Foundation under the grants NSF-CCF-0702758, NSF-CNS-0709217, NSF-ECCS-0424048, NSF-ECCS-0601570, NSF-ECCS-0801763, NSF-CCF-0811541.
of the services.

Given a set of services, a goal, and communication and computation costs, our objective is to devise an optimal decentralization choreography scheme that will realize the goal from the existing services by incurring the minimum cost. To satisfy the above objective, multiple choreographers are deployed at servers which are at close proximity (physically or in terms of request/response delay) to the existing services. Not all sites may have choreographers in this scheme as they may not participate in the optimal choreography scheme. We will refer to choreographer at site \( i \) where \( S_1 \) resides as \( C_1 \); choreographer at site 0 (i.e. the client site) being \( C_0 \).

A possible choreography scheme is shown in Figure 1(c). Note that the service \( S_3 \) remains unused in the scheme. This is because both services \( S_2 \) and \( S_3 \) provide the same functionality. Thus, any one of them can be used to realize the goal. The realization of the goal involves the communication of \( \text{p} \) from \( C_0 \) at the client site to \( C_1 \) at site 1, \( \text{inf} \) from the choreographer \( C_1 \) to \( C_2 \) which is the choreographer at site 2, \( \text{a} \) from \( C_0 \) to \( C_2 \), and finally output \( \text{prc} \) from \( C_2 \) to \( C_0 \). Moreover the client can input a request \( \text{c} \) to cancel order, which similarly needs to be communicated to \( C_2 \). If \( S_3 \) is to be used, the communication needs to be between the choreographers \( C_0, C_1 \) and \( C_3 \) deployed at site 3. If \( S_2 \) is used, the overall cost of the cancelation operation from the perspective of the client would be communication cost from \( C_0 \) to \( C_2 \) added to the computation cost at \( S_2 \), which is \( 4 + 36 = 40 \). If instead \( S_3 \) is used, then by similar calculations the cost would turn out to be \( 3 + 57 = 60 \). In contrast, it can be shown that the cost for an optimal centralized choreographer is more, showing the advantage of decentralized choreography. The cost of the composite behaviors as presented in Figure 1(c) is equal to the sum of the cost of exchanging messages as shown between \( C_0 \), \( C_1 \) and \( C_3 \) and the respective computation costs at \( S_1 \) and \( S_2 \) to produce the desired outputs. Weighing the communication costs and the computation costs for each operation, \( S_2 \) might be a cheaper alternative than \( S_3 \).

3 Choreographer Existence

I/O-automata naturally represent the behavior of Web services which are described as a set of sequences of input and output computations.

Definition 1 (I/O Automaton) An i/o-automaton \( A \) is defined by a tuple \(( S, S^0, S^F, I, O, \Delta ) \), where, \( S \) is the set of states, \( S^0 \subseteq S \) is the set of initial states, \( S^F \subseteq S \) is the set of final states, \( I \) is the set of inputs, \( O \) is the set of outputs, and \( \Delta \subseteq S \times ( I \cup \{ \epsilon \} ) \times ( O \cup \{ \epsilon \} ) \) 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, \( i \in I \cup \{ \epsilon \} \) is the input to the transition, \( o \in O \cup \{ \epsilon \} \) 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(d) presents the i/o-automata models for the three services and goal described in Figure 1(a). 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 and the final states are marked with double-circles. For a goal service, reaching any final state signifies completion of a task. In contrast, being at a non-final state signifies a pending task.
Furthermore, if a final state in the goal is non-deadlocking, another task initiates from that final state and its completion occurs when a subsequent final state is reached. When a final state is reached in the goal, any state can be reached in the services used to realize the goal. For this reason all states in the given services are treated final.

In order to realize the goal, each i/o operation of the goal needs to be realized by a sequence of i/o operations of the existing services, such that the input of the goal transition matches with the first input of the sequence and the output matches with the last output of the sequence. 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 $A_n = (S_n, S_n^0, \delta_n, \gamma_n, O_n, \Delta_n)$. Note each state is treated a final state (and so the third tuple-element is the same as the first tuple-element), since after reaching any state the service may no longer be required for realizing the goal. We designate site-$0$ as the site interfacing with the client or the goal service. Associated with each site-specific choreographer is a local history consisting of the inputs and outputs seen and stored at that site. The following definition of Interleaving Product With Distributed History ($\prod_n A_n$) captures all possible interleaved behaviors of the given service automata and the associated local histories.

**Definition 2 ($\prod_n A_n$ Automaton)** Given service automata $\{A_n = (S_n, S_n^0, \delta_n, \gamma_n, O_n, \Delta_n)\}_{1 \leq n \leq N}$, their interleaving product with distributed history is defined as the i/o-automaton $\prod_n A_n = (\prod_n S_n, \prod_n S_n^0, \prod_n \delta_n, \prod_n \gamma_n, O, \prod_n \Delta_n)$ where

$\prod_n S_n = \prod_{n=1}^N S_n, \quad \prod_n S_n^0 = \prod_{n=1}^N S_n^0, \quad \prod_n \delta_n = \prod_{n=1}^N \delta_n, \quad \prod_n \gamma_n = \prod_{n=1}^N \gamma_n, \quad \prod_n O_n, \quad \prod_n \Delta_n,$

$\prod_n \delta_n = \prod_{n=1}^N \delta_n, \quad \prod_n \gamma_n = \prod_{n=1}^N \gamma_n, \quad \prod_n O_n, \quad \prod_n \Delta_n,$

$\prod_n \Delta_n = \prod_{n=1}^N \Delta_n.$

**Example 1** Figure 2 depicts a part of the automaton $\prod_n A_n$ for $A_1, A_2$ and $A_3$ presented in Figure 1(d). 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.

### 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 the next transition of the service residing at that site. 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 nonnegative 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 over different routes between $n$ and $m$), and $c(n, m)$ denotes the cheapest option. By defining $c(n, m)$ this way we are able to abstract away the issue of optimum communication option from that of optimum choreography, although in the end, optimum choreography cost does depend on the optimum communication cost between a pair of sites. The table in Figure 1(b-i) 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. Further note that, each transition also incurs a computation cost (for our example it is summarized in Table 1(b-ii)). We will denote the computation cost of a transition $s \xrightarrow{i/o} s'$ as $w(s \xrightarrow{i/o} s')$. The uni-
verse of all choreographed behaviors of existing services that can be accomplished in the manner described above is computed via the transduced-closure of the automaton with distributed history \((\|_{\text{I}}^{\ell} A_n)^T\), and is defined as follows.

**Definition 3 ((\|_{\text{I}}^{\ell} A_n)^T Automaton)** Given an interleaving product automaton with distributed history \(\|_{\text{I}}^{\ell} A_n = (S \times \overline{H}, S^0 \times H^0, S \times \overline{H}, I_0, O_0, \Delta_{\overline{H}})\) of \(\{A_n | 1 \leq n \leq N\}\), its transduced-closure is the automaton \((\|_{\text{I}}^{\ell} A_n)^T = (S \times (2^{I_0 \cup O_0} \times \overline{H}), S^0 \times (\emptyset \times H^0), S \times (2^{I_0 \cup O_0} \times \overline{H}), I_0, O_0, \Delta_{\overline{H}})^T)\) where \((\overline{s}, (h_0, \overline{h})) \xrightarrow{i/o, c, \gamma} (\overline{s}', (h_0', \overline{h}')) \in \Delta_{\overline{H}}^T \text{ if and only if}\)

1. \(\exists m. (\overline{s}, \overline{h}) \xrightarrow{i/o, n_1} (\overline{s}_2, \overline{h}_2) \in \Delta_{\overline{H}} \wedge (\overline{s}_2, \overline{h}_2) \xrightarrow{i/o, n_2} (\overline{s}_3, \overline{h}_3) \in \Delta_{\overline{H}} \wedge \ldots \)

\[ (\overline{s}_m, \overline{h}_m) \xrightarrow{i/o, n_m} (\overline{s}', \overline{h}') \in \Delta_{\overline{H}} \wedge \forall 2 \leq k \leq m: \exists i_k \in h_0 \cup \bigcup_{1 \leq n \leq N} \overline{h}_k(n), i_1 = i, o_m = o \]

2. \(h_0 = h_0 \cup \{i, o\} \)

3. \(c := \left\{ \begin{array}{ll}
\sum_{k=2}^{m} \min \left[ \{c(n, n_k) | 1 \leq n \leq N : i_k \in \overline{h}_k(n) \} \cup \{c(0, n_k) | i_k \in h_0 \} \right] \\
+w(\overline{s}_k(n_k) \xrightarrow{i/o, o} \overline{s}_k+1(n_k))
\end{array} \right. \)

4. \(\gamma = (s_{rc2}, \ldots, s_{rcm}) \text{ where for } 2 \leq k \leq m:\}

\[ \text{src}_k := \arg \left[ \min \left[ \{c(n, n_k) | 1 \leq n \leq N : i_k \in \overline{h}_k(n) \} \right] \right. \cup \{c(0, n_k) | i_k \in h_0 \} + w(\overline{s}_k(n_k) \xrightarrow{i/o, o} \overline{s}_k+1(n_k)) \]

We call \(U := (\|_{\text{I}}^{\ell} A_n)^T\) to be the universal service automaton for the service-automata \(\{A_n | 1 \leq n \leq N\}\).

**Example 2** Figure 3(a) presents part of the transduced-closure automaton \(U\) obtained from \(\|_{\text{I}}^{\ell} A_n\) (Figure 2) of \(A_1, A_2\) and \(A_3\) (Figure 1(d)). The history at the client site, \(h_0\) is shown within \(\emptyset\). The dotted transitions correspond to the transitions obtained via the transduced-closure of a sequence of transitions. E.g., the transition \(s1\{t1\}r1\{t2\}\) in \(\text{p/inf}_{10,1}\) \(s1\{p, inf\}t1\{r2\}i{inf}_{3}\) \(p/inf_{1}\) \(s1\{p, inf\}t1\{r2\}i{inf}_{3}\) is obtained from the transduced-closure of \(s1\{t1\}r1\{t2\}\) (\(c = 4 + 2\) (computation costs for the transitions involved)+\(c(0,1)+c(1,3)\) (communication costs) = 10). Note that the output does not need to be communicated and hence the communication cost associated with it is not added.

**3.3 Realizability of goal**

A goal service is specified as an i/o-automaton, \(A_0 = (S_0, S^0_0, S^F_0, I_0, O_0, \Delta_0)\). Note that \(I_0 = \cup_{n=1}^N I_n\) and \(O_0 = \cup_{n=1}^N O_n\) (see Definition 2), i.e., the inputs/outputs of the goal are the union of the inputs/outputs of the existing services. A 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 by checking whether \(A_0\) is simulated by \(U\).

**Definition 4 (Simulation [11])** Given a goal automaton \(A_0 = (S_0, S^0_0, S^F_0, I_0, O_0, \Delta_0)\) and an universal service automaton \(U = (S_U, S^0_U, S^F_U, I_U, O_U, \Delta_U)\), 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: \(s_1 \sqsubseteq s_2 \Rightarrow [\forall t_1 : s_1 \xrightarrow{i/o, o_{\gamma}} t_1 \in \Delta_0 \Rightarrow (\exists t_2 : s_2 \xrightarrow{i/o, o_{\gamma}} t_2 \in \Delta_U \wedge t_1 \sqsubseteq t_2)]\). \(A_0\) is said to be simulated by \(U\), denoted by \(A_0 \sqsubseteq U\), if all states in \(S^0_U\) are simulated by some state in \(S^0_0\).

Then we have the following result from [13].

**Theorem 1** Given a goal \(A_0\) and a set of services \(\{A_n | 1 \leq n \leq N\}\), the goal is realizable from the choreography of \(\{A_n | 1 \leq n \leq N\}\) if and only if \(A_0 \sqsubseteq U\) where \(U\) is the transduced-closure of the \((\|_{\text{I}}^{\ell} A_n)^T\)-automaton, and \((\|_{\text{I}}^{\ell} A_n)^T\) is the interleaving product with distributed history of the automata \(\{A_n | 1 \leq n \leq N\}\).

**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(d).

It can be verified that $A_0 \subseteq U$ holds if and only if $A_0 \not\subseteq A_0 \times U$ 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 $A_0 = (S_0, S_0^p, S_0^c, I_0, O_0, \Delta_0)$ and an universal service automaton $U = (S_U, S_U^p, S_U^c, I_U, O_U, \Delta_U)$, their simulating synchronous product is the automaton $A_0 \times U = (S_0 \times S_U^p, S_0^c \times S_U^c, S_0 \times S_U^c, I_0, O_0, \Delta_U)$, where

$$\begin{align*}
(s_0, s_u) \xrightarrow{i/o} (s_0', s_u') & \in \Delta_x \iff \\
& s_0 \xrightarrow{i/o} s_0' \land s_u \xrightarrow{i/o} s_u' \land s_0 \not\subseteq s_0' \land s_u \not\subseteq s_u'.
\end{align*}$$

The size of $A_0 \times U$ is usually smaller compared to the size of $U$ (since size of $A_0$ is smaller compared to $U$). Hence it is preferable to check whether $A_0 \subseteq A_0 \times U$ holds (as opposed to checking whether $A_0 \not\subseteq U$ holds).

**Example 4** Figure 3(c) shows the simulating synchronous product of the goal automaton $A_0$ in Figure 1(d) and the universal service automaton $U$ in Figure 3(a). Here, $A_0 \times U$ has two paths from the start state both of which can yield choreographers; one path uses automaton $A_2$ for computing the transitions $i/o, a/prc$ as well as $c/e$ and other uses service $A_3$ for the same. In the following sections we introduce the algorithm for choosing the optimal choreography.

**4 Optimum Decentralization**

Realizability of a goal $A_0$ by choreographing a set of services $\{A_n | 1 \leq n \leq N\}$ is guaranteed by the satisfaction of $A_0 \subseteq A_0 \times U$. 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
Algorithm 1 (MinCost Choreography)
Initialization: \( k = 0 \)

\[
\text{cost}^k(s_0, s_u) =
\begin{cases}
0 & \text{if } (s_0, s_u) \text{ is a deadlocking state} \\
\max & \\
\min & \left\{ (s_0, s_u) \in \Delta_x \right\} \\
\text{otherwise}
\end{cases}
\]

\[
\text{cost}^k(s_0, s_u) = \begin{cases}
0 & \text{if } (s_0, s_u) \text{ is a deadlocking state} \\
\max & \\
\min & \left\{ (s_0, s_u) \in \Delta_x \right\} \\
\text{otherwise}
\end{cases}
\]

\[
\text{cost}^k = \max\{\text{cost}^k(s_0, s_u) | (s_0, s_u) \in S_0 \times SU\}
\]

\[
\text{cost}^m = \max\{\text{cost}^m(s_0, s_u) | (s_0, s_u) \in S_0 \times SU, \forall m < k : \text{cost}^m \neq \text{cost}^m\}
\]

Termination: If \( \text{cost}^k < \text{cost}^{k-1} \), set \( k := k + 1 \) and repeat Iteration step; otherwise stop and output \( \text{cost}^{k-1} \).

**Figure 4:** Algorithm for computing minimum cost choreography

Definition 7 (MinCost Choreography Automaton)

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

4.1 Computing optimum cost

The algorithm for computing the optimum cost of the subautomaton in \( A_0 \times U \) that simulates \( A_0 \) is presented in Algorithm 1 in Figure 4. In the initialization phase \( (k = 0) \), cost of all deadlocking states is assigned 0. For the non-deadlocking state \( (s_0, s_u) \), its cost is computed in two steps. In the first step (minimization-step), the cheapest way to simulate the goal transition \( s_0 \xrightarrow{i/o} s_0' \) is identified. The cost for such simulation is computed as the sum of the corresponding transition from \( (s_0, s_u) \) and the cost of the destination state. If the destination state is a final state, its cost is assumed to be 0 as it denotes completion of a task. In the second step (maximization-step), the worst cost of simulating a transition originating from \( s_0 \) is computed. The maximum cost of any state as computed in the initialization phase is denoted \( \text{cost}^0 \).

Remark 1 As shown in our earlier work [13], in the absence of loops in the goal specification, the optimum cost computation is already obtained at this point, i.e., no further iteration is required. In the presence of loops however, it is possible that the goal specification doesn’t terminate at a final state, and thus the cost of such a final state is non-zero. So there is a possibility of lowering the overall cost if the “payoff” of reaching a final state (namely lowering of costs of other states by completing tasks through reaching the final state) is overshadowed by the “penalty” of reaching it (namely incurring the cost of the final state). Thus further iterations are required to explore this trade-off, which makes the algorithm for specification with loops non-trivially different from the loop-free specifications, where a straightforward backward search starting from terminating states suffices.

The \( k \)th iteration \( (k \geq 1) \) explores the possibility of reducing the overall cost by way of avoiding reaching the worst-costing final states (cost of which is equal to \( \text{cost}^m \), for some \( m < k \)) of the earlier iterations. Doing this results in raising the cost of all the other states (since they no longer have access to the worst-costing final states of the earlier iterations), but the overall cost may still reduce since the costs of the worst-costing states of the earlier iterations no longer affect the overall cost. The worst-costing final states of the earlier iterations are avoided by simply setting their cost contribution to be infinity. This is the only difference between the 0th iteration, and the subsequent iterations. A
new iteration is executed only if the current iteration results in a reduction in the overall cost compared to the previous iteration.

It can be concluded that the $k$th iteration will require $O([S_0].|S_\Pi|)$ number of computations. Further since in each iteration at least one final state is avoided (by forcing its cost contribution to be infinity), there can be at most $O([S_0].|S_\Pi|)$ number of iterations. Thus the overall computational complexity of Algorithm 1 is $O([S_0].|S_\Pi|^2)$.

**Remark 2** While Algorithm 1 works for general services and goals, a finite-cost optimum solution will exist if all cycles in the goal service either possess a final state or cost zero to simulate. (Otherwise the optimum cost will be $\infty$, and in which case any decentralized choreographer is an optimum one.)

**Theorem 2** Given $A_0 \times U$, where $A_0$ and $U$ are goal and universal service automata respectively, Algorithm 1 in Figure 4 terminates with the cost equaling that of a mincost choreography subautomaton $C$ of $A_0 \times U$.

The proof is omitted due to space constraint. For details refer to http://www.public.iastate.edu/~saayan/th2-proof.pdf

**Example 5** Figure 5 presents the computation of Algorithm 1 as applied to our running example. The simulating algorithm 1 as applied to our running example. The simulating algorithm

**Initialization (k=0)**

3. **Iteration 1 (k=1)**

4.2 Synthesizing optimum choreographers

Starting from a subautomaton $C$ of $A_0 \times U$ representing an optimum choreography scheme, our objective is to synthesize choreographers at each site such that the transduced closure of their product replicates $C$. In addition to normal i/o-behavior at each transition, a site-specific choreographer also needs to record information regarding the i/o's that must be sent from one choreographer to another for minimal cost communication. Site-specific choreographers are obtained using the algorithm presented in [13].

Figure 6(a, b, c, d) shows the various site-specific choreographers as obtained from optimum choreography in Figure 3(d). The labeling of states as implied by the $E_m$ function is shown within $\langle \rangle$. In Figure 6(a), we obtain the choreographer $C_0$ at the client-site. $C_0$ communicates $p$ to the choreographer $C_1$ at site-1, and $a$ and $c$ to the site-2 choreographer $C_2$ from the initial and the next successor states, respectively. Note that, choreographer at site 3 does not do anything as service 3 does not participate in the minimum cost choreography in our example.
from the loop-free case to the case with loops is non-trivial, where in the former case a straightforward backward search starting from the terminating states suffices.

References