A Framework for Optimal Decentralized Service-Choreography *

Saayan Mitra[1] Ratnesh Kumar[1] Samik Basu[2]

[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.

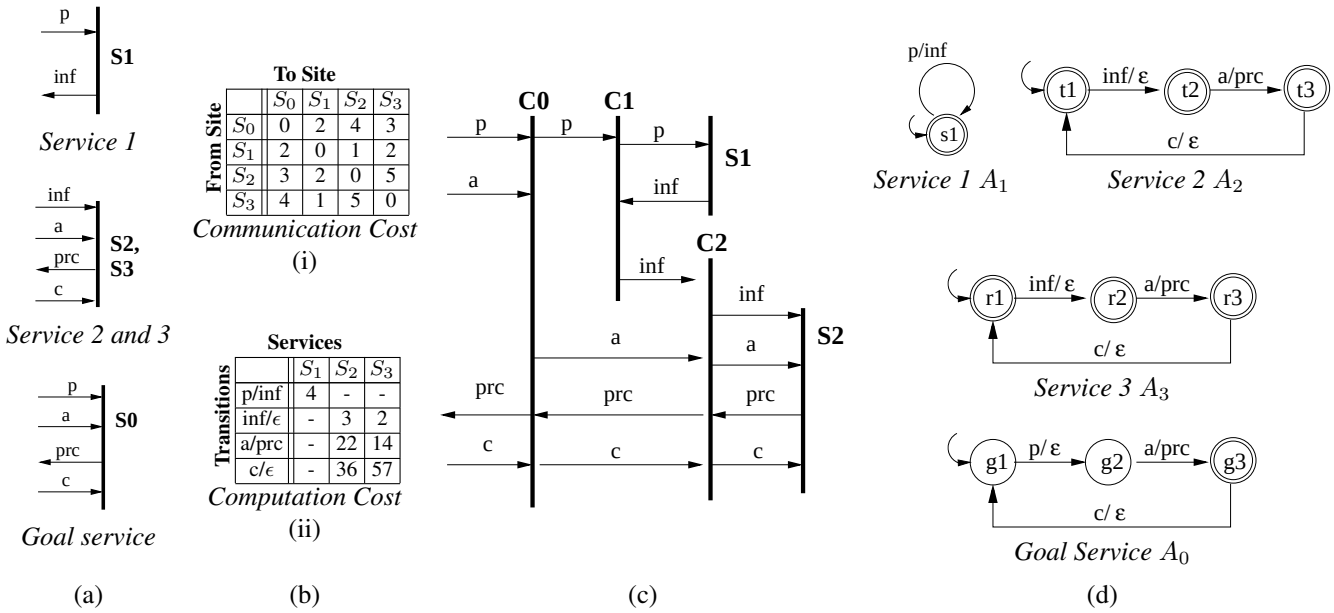

Figure 1: (a) Existing Services & goal service (b) cost metrics, (c) decentralized choreography, and (d) i/o-automata models

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 `Si` resides as `Ci`; choreographer at site 0 (i.e. the client site) being `C0`.

A possible choreography scheme is shown in Figure 1(c). Note that the service `S3` remains unused in the scheme. This is because both services `S2` and `S3` 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 `p` from `C0` at the client site to `C1` at site 1, `inf` from the choreographer `C1` to `C2` which is the choreographer at site 2, `a` from `C0` to `C2`, and finally output `prc` from `C2` to `C0`. Moreover the client can input a request `c` to cancel order, which similarly needs to be communicated to `C2`. If `S3` is to be used, the communication needs to be between the choreographers `C0`,`C1` and `C3` deployed at site 3. If `S2` is used, the overall cost of the cancelation operation from the perspective of the client would be communication cost from `C0` to `C2` added to the computation cost at `S2`, which is $4 + 36 = 40$. If instead `S3` 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 `C0`, `C1` and `C3` and the respective computation costs at `S1` and `S2` to produce the desired outputs. Weighing the communication costs and the computation costs for each operation, `S2` might be a cheaper alternative than `S3`.

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 $\Delta \subseteq S \times (I \cup \{\epsilon\}) \times (O \cup \{\epsilon\}) \times S$ is the set of transitions. An element of Δ, 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 $A_n = (S_n, S_n^0, S_n, I_n, O_n, \Delta_n)$. 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 ($\|_n^{\vec{H}} A_n$) captures all possible interleaved behaviors of the given service automata and the associated local histories.

Definition 2 ($\|_n^{\vec{H}} A_n$ Automaton) Given service automata $\{A_n = (S_n, S_n^0, S_n, I_n, O_n, \Delta_n) | 1 \leq n \leq N\}$, their *interleaving product with distributed history* is defined as the i/o-automaton $\|_n^{\vec{H}} A_n = (\vec{S} \times \vec{H}, \vec{S^0} \times \vec{H^0}, \vec{S} \times \vec{H}, I_0, O_0, \Delta_{\vec{H}})$ where

$\vec{S} = \prod_{n=1}^{N} S_n,\ \ \vec{S^0} = \prod_{n=1}^{N} S_n^0,\ \ \vec{H} = \prod_{n=1}^{N} 2^{I_n \cup O_n},$
$\vec{H^0} = \prod_{n=1}^{N} \{\emptyset\}, I_0 = \bigcup_{n=1}^{N} I_n, O_0 = \bigcup_{n=1}^{N} O_n,$ and
$(\vec{s}, \vec{h}) \xrightarrow[n]{i/o} (\vec{s'}, \vec{h'}) \in \Delta_{\vec{H}}$ if and only if
$\vec{s}(n) \xrightarrow{i/o} \vec{s'}(n) \in \Delta_n \ \wedge\ \vec{h'}(n) = \vec{h}(n) \cup \{i, o\} \wedge$

$$\forall m \neq n : \vec{s'}(m) = \vec{s}(m) \wedge\ \vec{h'}(m) = \vec{h}(m).$$

In the definition of $\Delta_{\vec{H}}$, the first conjunct states that whenever a constituent service makes a move, the $\|_n^{\vec{H}} A_n$ 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 that other services do not change their states or their corresponding local histories. Thus, if a service gets the input i and produces output o, its local history is updated to include $\{i, o\}$.

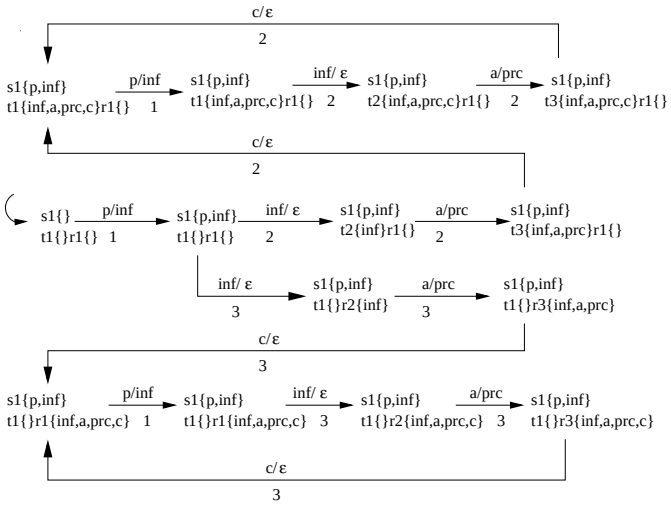

Figure 2: Interleaving Product Automaton $\|_n^{\vec{H}} A_n$

Example 1 Figure 2 depicts a part of the automaton $||_n^{\vec{H}} A_n$ 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 $c(n, m) \in \Re_+$, where $\Re_+$ 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 ($(\|_n^{\vec{H}} A_n)^T$), and is defined as follows.

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

1. $\exists m.$ $\left[\begin{array}{l} (\vec{s}, \vec{h}) \xrightarrow[n_1]{i_1/o_1} (\vec{s}_2, \vec{h}_2) \in \Delta_{\vec{H}} \wedge \\ (\vec{s}_2, \vec{h}_2) \xrightarrow[n_2]{i_2/o_2} (\vec{s}_3, \vec{h}_3) \in \Delta_{\vec{H}} \wedge \ldots \\ (\vec{s}_m, \vec{h}_m) \xrightarrow[n_m]{i_m/o_m} (\vec{s'}, \vec{h'}) \in \Delta_{\vec{H}} \wedge \\ \forall 2 \leq k \leq m : i_k \in h_0 \cup \bigcup_{1 \leq n \leq N} \vec{h}_k(n), \\ i_1 = i, o_m = o \end{array}\right]$
2. $h_0' = h_0 \cup \{i, o\}$
3. $c := \left\{\begin{array}{l} c(0, n_1) \ + \ w(\vec{s}(n_1) \xrightarrow{i/o_1} \vec{s}_2(n_1)) + \\ \sum_{k=2}^{m} \left[\begin{array}{l} \texttt{MIN} \left[\begin{array}{l} \{c(n, n_k) \mid 1 \leq n \leq N : \\ i_k \in \vec{h}_k(n)\} \\ \cup \{c(0, n_k) \mid i_k \in h_0\} \end{array}\right] \\ + w(\vec{s}_k(n_k) \xrightarrow{i_k/o_k} \vec{s}_{k+1}(n_k)) \end{array}\right] \\ + c(n_m, 0) \end{array}\right.$
4. $\gamma = (\texttt{src}_2, \ldots, \texttt{src}_m)$ where for $2 \leq k \leq m$:

$$\texttt{src}_k := \texttt{arg} \left[\begin{array}{l} \texttt{MIN} \left[\begin{array}{l} \{c(n, n_k) \mid 1 \leq n \leq N : i_k \in \vec{h}_k(n)\} \\ \cup \{c(0, n_k) \mid i_k \in h_0\} \end{array}\right] \\ + w(\vec{s}_k(n_k) \xrightarrow{i_k/o_k} \vec{s}_{k+1}(n_k)) \end{array}\right]$$

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

Observe that, in the above definition, the states of U are represented by the states of $\|_n^{\vec{H}} A_n$ coupled with elements from $2^{I \cup O}$. The extra elements represent a history set of the client-site, i.e., the inputs and outputs seen by the client-site choreographer. According to the above definition, each transition in U automaton corresponds to the transduced-closure of a sequence of transitions present in the interleaving product automaton $\|_n^{\vec{H}} A_n$. Each transduced-closure transition is annotated with its cost c and the tuple γ 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) presents part of the transduced closure automaton U obtained from $\|_n^{\vec{H}} A_n$ (Figure 2) of A_1, A_2 and A_3 (Figure 1(d)). The history at the client site, h_0 is shown within [ ]. The dotted transitions correspond to the transitions obtained via the transduced-closure of a sequence of transitions. E.g., the transition `s1{}t1{}r1{}[]` $\xrightarrow[10,1]{\texttt{p}/\epsilon}$ `s1{p,inf}t1{}r2{inf}[p]` is obtained from the transduced-closure of `s1{}t1{}r1{}` $\xrightarrow[1]{\texttt{p/inf}}$ `s1{p,inf}t1{}r1{}` $\xrightarrow[3]{\texttt{inf}/\epsilon}$ `s1{p,inf}t1{}r2{inf}`. Cost of this transduced-closure transition is $c = 4 + 2$ (computation costs for the transitions involved)$+c(0,1)+c(1,3)$ (communication costs) $= 10$. Note that the ϵ 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, $A_0 = (S_0, S_0^0, S_0^F, I_0, O_0, \Delta_0)$. 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 $A_0 = (S_0, S_0^0, S_0^F, I_0, O_0, \Delta_0)$ and an universal service automata $U = (S_U, S_U^0, S_U, I_0, O_0, \Delta_U)$, 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: $s_1 \sqsubseteq s_2 \ \Rightarrow \ [\forall t_1 : s_1 \xrightarrow{i/o} t_1 \in \Delta_0 \Rightarrow (\exists t_2 : s_2 \xrightarrow[c,\gamma]{i/o} t_2 \in \Delta_U \wedge \ t_1 \sqsubseteq t_2)\ ]$. 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 from [13].

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 $A_0 \sqsubseteq U$ where U is the transduced-closure of the $(\|_n^{\vec{H}} A_n)$-automaton, and $(\|_n^{\vec{H}} A_n)$ 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).

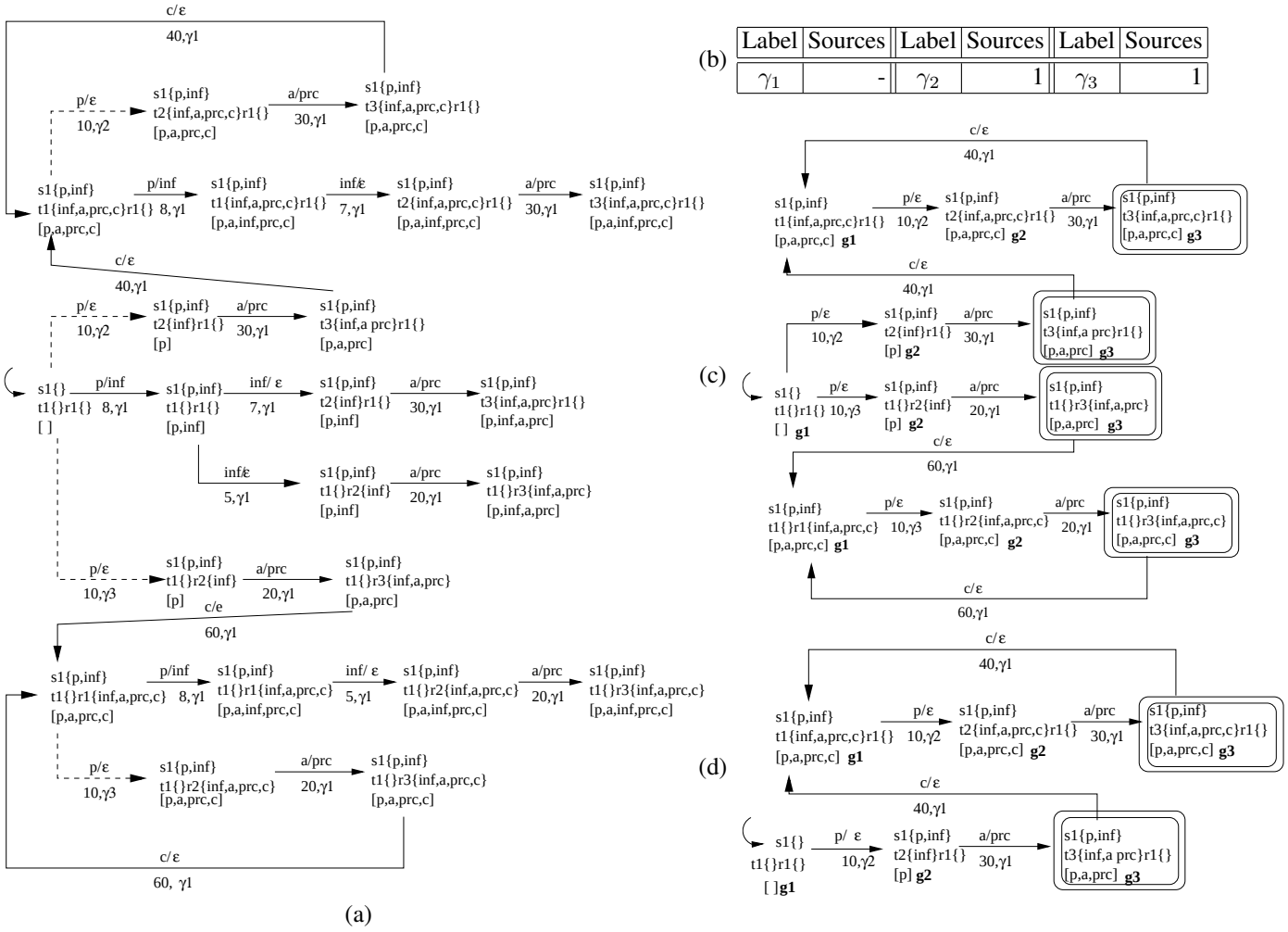

Figure 3: (a) Transduced Closure Automaton U, (b) Valuations of γ for each transition ("-" denotes input to the transition obtained from the client), (c) Simulating synchronous product $A_0 \times U$, (d) Mincost Choreography C.

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 $A_0 \sqsubseteq U$ holds if and only if $A_0 \sqsubseteq A_0 \times U$ 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 $A_0 = (S_0, S_0^0, S_0^F, I_0, O_0, \Delta_0)$ and an universal service automaton $U = (S_U, S_U^0, S_U, I_0, O_0, \Delta_U)$, their simulating synchronous product is the automaton $A_0 \times U = (S_0 \times S_U, S_0^0 \times S_U^0, S_0^F \times S_U, I_0, O_0, \Delta_\times)$, where

$$(s_0, s_u) \xrightarrow[c,\gamma]{i/o} (s_0', s_u') \in \Delta_\times \Leftrightarrow \left\{ \begin{array}{l} s_0 \xrightarrow{i/o} s_0' \wedge \; s_u \xrightarrow[c,\gamma]{i/o} s_u' \\ \wedge\, s_0 \sqsubseteq s_u \wedge s_0' \sqsubseteq s_u'. \end{array} \right\}$$

The size of $A_0 \times U$ is usually 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 $A_0 \sqsubseteq A_0 \times U$ holds (as opposed to checking whether $A_0 \sqsubseteq U$ 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 `inf/`ϵ, `a/prc` as well as `c/`ϵ and other path 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 $\{A_n \mid 1 \leq n \leq N\}$ is guaranteed by the satisfaction of $A_0 \sqsubseteq A_0 \times U$. 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: $(k = 0)$

$$\texttt{cost}^k(s_0, s_u) = \begin{cases} 0 \quad \text{if } (s_0, s_u) \text{ is a deadlocking state} \\ \underset{s_0 \xrightarrow{i/o} s_0' \in \Delta_0}{\texttt{MAX}} \left[\underset{(s_0,s_u) \xrightarrow[\overline{c},\overline{\gamma}]{i/o} (\overline{s_0},\overline{s_u}) \in \Delta_\times}{\texttt{MIN}} \left\{ \begin{array}{l} \{\overline{c} + \texttt{cost}^k(\overline{s_0}, \overline{s_u}) \mid (\overline{s_0}, \overline{s_u}) \notin S_0^F \times S_U\} \\ \cup \{\overline{c} \mid (\overline{s_0}, \overline{s_u}) \in S_0^F \times S_U\} \end{array} \right\} \right] \\ \quad \text{otherwise} \end{cases}$$

$$\texttt{cost}^k = \texttt{MAX}\{\texttt{cost}^k(s_0, s_u) \mid (s_0, s_u) \in S_0 \times S_U\}$$

Iteration: $(k \geq 1)$

$$\texttt{cost}^k(s_0, s_u) = \begin{cases} 0 \quad \text{if } (s_0, s_u) \text{ is a deadlocking state} \\ \underset{s_0 \xrightarrow{i/o} s_0' \in \Delta_0}{\texttt{MAX}} \left[\underset{(s_0,s_u) \xrightarrow[\overline{c},\overline{\gamma}]{i/o} (\overline{s}_0,\overline{s}_u) \in \Delta_\times}{\texttt{MIN}} \left\{ \begin{array}{l} \{\overline{c} + \texttt{cost}^k(\overline{s_0}, \overline{s_u}) \mid (\overline{s_0}, \overline{s_u}) \notin S_0^F \times S_U\} \\ \cup \{\overline{c} \mid (\overline{s_0}, \overline{s_u}) \in S_0^F \times S_U, \\ \quad \forall m < k : \texttt{cost}^m(s_0, s_u) \neq \texttt{cost}^m\} \\ \cup \{\infty \mid (\overline{s_0}, \overline{s_u}) \in S_0^F \times S_U, \\ \quad \exists m < k : \texttt{cost}^m(s_0, s_u) = \texttt{cost}^m\} \end{array} \right\} \right] \\ \quad \text{otherwise} \end{cases}$$

$$\texttt{cost}^k = \texttt{MAX}\{\texttt{cost}^k(s_0, s_u) \mid (s_0, s_u) \in S_0 \times S_U, \forall m < k : \texttt{cost}^m(s_0, s_u) \neq \texttt{cost}^m\}$$

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

Figure 4: Algorithm for computing minimum cost choreography

cost realization of A_0, which can be obtained by first finding an optimal cost subautomaton of $A_0 \times U$ that simulates A_0, and next projecting that subautomaton over individual sites to obtain the site-specific choreographers. The objective of optimization is to choreograph the existing services in such a way that regardless of the history of evolution, any pending task is completed in a minimal cost (the worst cost between any state and its nearest reachable final states is minimized). With this in mind, we define the cost of an automaton of $A_0 \times U$ as follows.

Definition 6 (Automaton Cost) Given an i/o-automaton with each of its transitions labeled by some cost, we define the cost of a path to be the sum of the costs of all the transitions in that path. We define the cost of a state to be the maximum cost among all paths originating at that state and terminating at a final state. The cost of an automaton is defined to be the maximum cost among all its states.

We refer to the cost of an i/o-automaton A as $\texttt{cost}(A)$. 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 $A_0 \sqsubseteq A_0 \times U$, 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 $\texttt{cost}(C) \leq \texttt{cost}(C')$.

4.1 Computing optimum cost

The algorithm for computing the optimum cost of the subautomaton in $A_o \times U$ that simulates A_o 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 $\texttt{cost}^0$.

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 kth 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 $\texttt{cost}^m$, 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 kth iteration will require $O(|S_0|.|S_U|)$ 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 $O(|S_0^F|.|S_U|)$ number of iterations. Thus the overall computational complexity of Algorithm 1 is $O(|S_0|.|S_0^F|.|S_U|^2)$.

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 optimal cost will be ∞, 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 synchronous product $A_0 \times U$ is shown in Figure 3(c). Observe that $A_0 \times U$ has one initial state, and two final states. Upon starting from the initial state, when a final state is reached a certain task of the goal is completed. If the final state reached is non-deadlocking, a new task begins from this state and its completion occurs when a subsequent final state is reached. Thus $A_0 \times U$ can execute three distinct tasks, starting from either the initial state or one of the final states. In order to simplify the illustration of the algorithm, the graph of $A_0 \times U$ is accordingly split into three subgraphs (that are trees), one for each task, as shown Figure 5($k = 0$).

For our example, $\texttt{cost}^0 = 90$. Therefore, for $k = 1$ iteration, the worst costing state is avoided by assigning it cost of ∞. Furthermore, since this state is to be avoided, it can no longer act as a restarting point of a task, and hence there are only two trees to consider in Iteration $k = 1$ (Figure 5). Proceeding further, $\texttt{cost}^1 = 80$. In the next iteration for $k = 2$, $\texttt{cost}^2 = \infty$ as the state with worst cost ($= 80$) is avoided and we will be left with one subgraph whose leaf-states have the cost ∞. This results in termination of our algorithm $\texttt{cost}^2 > \texttt{cost}^1$ and we output $\texttt{cost}^1$ as the optimum cost. In other words, we identify the optimum decentralized choreography which uses service 2 to realize the goal instead of service 1 (see Figure 3(d)).

Remark 3 It should be noted that if the algorithm of loop-free specifications [13] is used, then the iterations $k = 1$ and $k = 2$ won't be performed, and the optimum answer reported would be $cost^0 = 90$ which obviously is incorrect since a cheaper alternative with $cost^1 = 80$ exists.

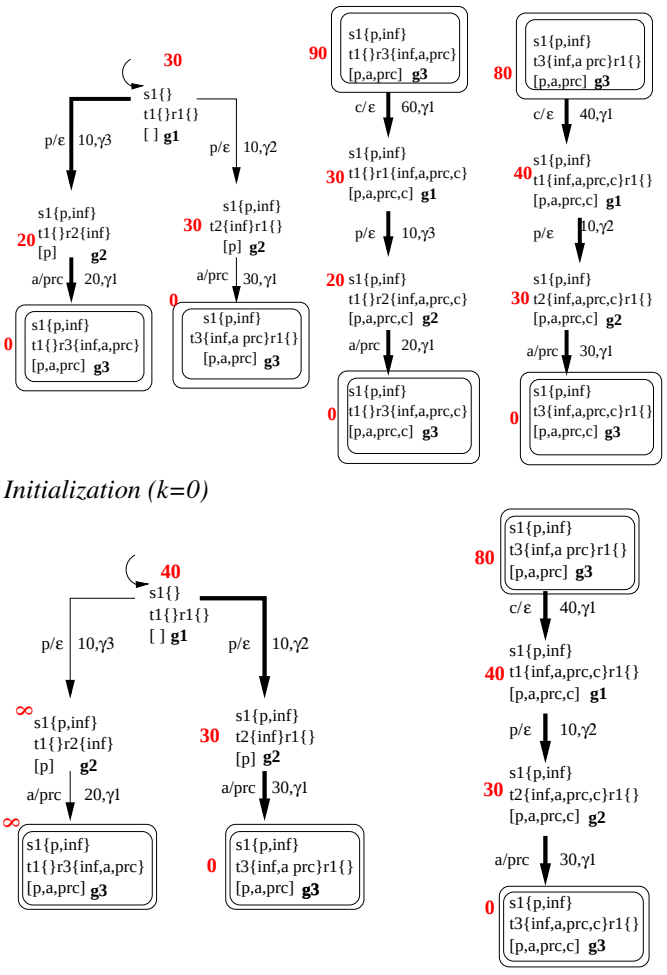

Figure 5: Iterations of Algorithm 1

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_n function is shown within $\langle\rangle$. In Figure 6(a), we obtain the choreographer C_0 at the client-site. C_0 communicates $\texttt{p}$ to the choreographer C_1 at site-1, and $\texttt{a}$ and $\texttt{c}$ 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.

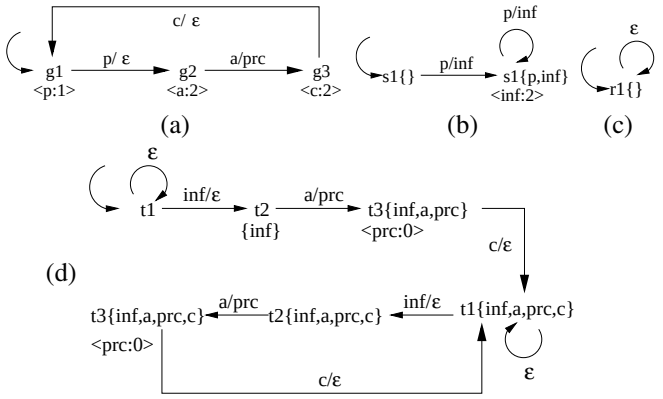

Figure 6: Site specific Choreography Automata: (a) C_0, (b) C_1, (c) C_3 and (d) C_2

5 Related Work

Automated composition of Web Services has been well researched both in the industry and academia for a few years now (See representative work [16, 5, 6, 14]). Decentralization of computation aims to move away from the simple mediator based approach to a more sophisticated scheme. [1] gives an overview of recent and ongoing approaches, such as BondFlow, Symphony, OSIRIS. Decentralization of composition has been researched in authors in [9] based on program dependence and graph partitioning. The industry standard workflow models are moving from simple centralized scheme towards a more loosely coupled approach by distributing the workflow, more suited to lightweight mobile technologies [2, 10, 17]. [19] discusses late binding of services for communication optimization. To this effect, [3] proposes a hybrid model specifying centralized control-flow and distributed data-flow. [18] introduces a peer-to-peer approach of service communication, from a centralized specification. In contrast, our approach guarantees the minimization of the overall cost, the valuation of which can be generic and context based. No manual intervention for obtaining the optimal solution is necessary as our approach is automated. In [13], we introduced the optimal decentralization scheme, goals being restricted to be loop-free. Here, the algorithm we introduced can be applied to a generalized class of services and a goal with loops.

6 Conclusion

We studied the problem of optimum decentralized choreography with the objective of minimizing the computation and communication overhead required to realize a goal service from the existing services. A universal service automaton is obtained to capture all possible decentralized-choreographed behaviors as well as the associated costs (of computation and communication). To the best of our knowledge, this is the first automated technique for solving the optimum decentralized choreography problem, in which the goal service is allowed to have *loops*. The extension 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

[1] J. Balasooriya. *Distributed Web Service Coordination for Collaborative Applications and Biological Workflows*. PhD thesis, Georgia State University, 2006.

[2] L. Baresi, A. Maurino, and S. Modafferi. Towards distributed bpel orchestration. *Electronic Communications of the EASST*, 3, 2006.

[3] A. Barker, J. Weissman, and J. van Hemert. Eliminating the middleman: Peer-to-peer dataflow. In *17th International Symposium on High-Performance Distributed Computing*, pages 191–200, MA, USA, 2008.

[4] B. Benatallah, F. Casati, D. Grigori, H. Nezhad, and F. Toumani. Developing adapters for web services integration. In *International Conference on Advanced Information Systems Engineering*, Berlin, Germany, 2005.

[5] B. Benatallah, Q. Sheng, and M. Dumas. The self-serv environment for web services composition. *IEEE Intelligent Systems*, 7(1):40–48, 2003.

[6] D. Berardi, D. Calvanese, G. Giacomo, R. Hull, and M. Mecella. Automatic composition of transition-based semantic web services with messaging. In *Conference on Very large data bases*, Trondheim, Norway, 2005.

[7] W. Binder, I. Constantinescu, and B. Faltings. Decentralized orchestration of composite web services. In *IEEE International Conference on Web Services (ICWS'06)*, Chicago, IL, September 2006.

[8] T. Bultan, J. Su, and X. Fu. Analyzing conversations of web services. *IEEE Internet Computing*, 10(1):18–25, 2006.

[9] G. Chafle, S. Chandra, V. Mann, and M. Nanda. Decentralized orchestration of composite web services. In *International World Wide Web Conference on Alternate track papers and posters*, New York, NY, 2004.

[10] R. Jiménez-Peris, M. Patino-Martínez, and E. Martel-Jordán. Decentralized web service orchestration: A reflective approach. In *23rd Annual ACM Symposium on Applied Computing*, pages 494–498, March 2008.

[11] R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., 1982.

[12] S. Mitra, R. Kumar, and S. Basu. Automated choreographer synthesis for web services composition using i/o automata. In *IEEE International Conference on Web Services (ICWS)*, Salt Lake City, 2007.

[13] S. Mitra, R. Kumar, and S. Basu. Decentralized choreography for web service composition. In *IEEE Services Computing Conference (SCC)*, Honolulu, 2008.

[14] J. Pathak, S. Basu, R. Lutz, and V. Honavar. Parallel web service composition in moscoe: A choreography-based approach. In *ECOWS '06: Proceedings of the European Conference on Web Services*, pages 3–12. IEEE Computer Society, 2006.

[15] J. Peer. Web service composition as ai planning - a survey, technical report. Technical report, University of St. Gallen, 2005.

[16] M. Trainotti, M. Pistore, G. Calabrese, G. Zacco, G. Lucchese, F. Barbon, P. Bertoli, and P. Traverso. Astro: Supporting composition and execution of web services. In *International Conference on Automated and Planning Sheduling (ICAPS'05)*, 2005.

[17] Q. Xiaoqiang and W. Jun. A decentralized services choreography approach for business collaboration. In *IEEE Services Computing Conference (SCC)*, Honolulu, 2008.

[18] U. Yildiz and C. Godart. Centralized versus decentralized conversation-based orchestrations. In *The 9th IEEE International Conference on E-Commerce Technology and The 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services((CEC-EEE)*, 2007.

[19] M. Zaremba, T. Vitvara, and M. Moran. Towards optimized data fetching for service discovery. In *Fifth European Conference on Web Services*, pages 191–200, 2007.