Quality Driven Web Services Composition

Liangzhao Zeng
University of New South Wales
Sydney, Australia
zlzhao@cse.unsw.edu.au

Boualem Benatallah
University of New South Wales
Sydney, Australia
boualem@cse.unsw.edu.au

Marlon Dumas
Queensland University of Technology
Brisbane, Australia
m.dumas@qut.edu.au

Jayant Kalagnanam
IBM T.J. Watson Research Center
New York, USA
jayant@us.ibm.com

Quan Z. Sheng
University of New South Wales
Sydney, Australia
qsheng@cse.unsw.edu.au

ABSTRACT

The process-driven composition of Web services is emerging as a promising approach to integrate business applications within and across organizational boundaries. In this approach, individual Web services are federated into composite Web services whose business logic is expressed as a process model. The tasks of this process model are essentially invocations to functionalities offered by the underlying component services. Usually, several component services are able to execute a given task, although with different levels of pricing and quality. In this paper, we advocate that the selection of component services should be carried out during the execution of a composite service, rather than at design-time. In addition, this selection should consider multiple criteria (e.g., price, duration, reliability), and it should take into account global constraints and preferences set by the user (e.g., budget constraints). Accordingly, the paper proposes a global planning approach to optimally select component services during the execution of a composite service. Service selection is formulated as an optimization problem which can be solved using efficient linear programming methods. Experimental results show that this global planning approach outperforms approaches in which the component services are selected individually for each task in a composite service.

Categories and Subject Descriptors

H.3.5 [**Information Systems**]: Web-based services

General Terms

Management, Performance

Keywords

Web services, QoS, Service Composition

1. INTRODUCTION

Web services technologies are emerging as a powerful vehicle for organizations that need to integrate their applications within and across organizational boundaries. In particular, the process-based composition of Web services is gaining a considerable momentum as an approach for the effective integration of distributed, heterogeneous, and autonomous applications [1]. In this approach, applications are encapsulated as Web services and the logic of their interactions is expressed as a process model. This approach provides an attractive alternative to hand-coding the interactions between applications using general-purpose programming languages.

A Web service is a self-described application that uses standard Internet technologies to interact with other Web services. An example of a Web service is a SOAP-based interface to place bids in an auction house. Once deployed, services can be aggregated into *composite services*. An example of a composite service would be a "Travel Planner" system that aggregates multiple *component services* for flight booking, travel insurance, accommodation booking, car rental, and itinerary planning, which are executed sequentially or concurrently.

The process model underlying a composite service identifies the functionalities required by the services to be composed (i.e., the *tasks* of the composite service) and their interactions (e.g., control-flow, data-flow, and transactional dependencies). Component services that are able to provide the required functionalities are then associated to the individual tasks of the composite services and invoked during each execution of the composite service.

The number of services providing a given functionality may be large and constantly changing. Consequently, approaches where the development of composite services requires the identification at design-time of the exact services to be composed are inappropriate. The runtime selection of component services during the execution of a composite service has been put forward as an approach to address this issue [2, 6, 11]. The idea is that component services are selected by the composite service execution engine based on a set of criteria. However, previous approaches in this area have not identified a set of criteria (other than price and application-specific criteria) for selecting Web services. In addition, existing service selection approaches adopt a local selection strategy, meaning that they assign a component service to an individual tasks, one at a time. As a result, these approaches are not able to handle global user con-

WWW2003, May 20–24, 2003, Budapest, Hungary.
ACM 1-58113-680-3/03/0005.

straints and preferences. For example, the overall duration of the composite service execution should be minimized, or a given budget constraint should be satisfied.

In this paper, we present quality-driven approach to select component services during the execution of a composite service. The salient features of our approach are:

- A *Web services quality model*. We propose an extensible multi-dimensional Web services quality model. The dimensions of this model characterize non-functional properties that are inherent to Web services in general: *execution price*, *execution duration*, *reputation*, *reliability*, and *availability*.

- *Quality-driven service selection*. In order to overcome the limitations of local service selection outlined above, we propose a global planning approach. In this approach, quality constraints and preferences are assigned to composite services rather than to individual tasks within a composite service. Service selection is then formulated as an optimization problem and a linear programming method is used to compute optimal service execution plans for composite services. Experimental results show that the proposed service selection strategy significantly outperforms local selection strategies.

The rest of the paper is organized as follows. Section 2 presents the service composition model and defines some key concepts used throughout the paper. Section 3 defines the service quality criteria used for service selection and explains how the values of these quality criteria can be computed for a given service. Section 4 formulates the global service selection problem and describes a linear programming method to efficiently solve it. Section 5 presents a prototype implementation of the proposed approach, as well as a set of experiments comparing the global planning approach with the local selection approach. Finally, Section 6 discusses related work, and Section 7 draws some conclusions.

2. WEB SERVICE COMPOSITION MODEL

In this section, we will present some basic concepts in Web service composition first, then give some definitions on composite service execution planning.

2.1 Composite services and communities

A composite Web service is an umbrella structure aggregating multiple other elementary and composite Web services, which interact with each other according to a process model. Following our previous work [2], we choose to specify the process model of a composite service as a statechart [12]. The choice of statecharts for specifying composite Web services is motivated by two main reasons: (i) statecharts have a well-defined semantics; and (ii) they offer the basic flow constructs found in contemporary process modeling languages (i.e., sequence, conditional branching, structured loops, concurrent threads, and inter-thread synchronization). The first characteristic facilitates the application of formal manipulation techniques to statechart models, while the second characteristic ensures that the service composition mechanisms developed in the context of statecharts, can be adapted to other process modeling languages like, for example, those that are being designed by Web services standardization efforts (e.g., BPEL4WS, WSCI, BPML)[1].

A statechart is made up of states and transitions. In the proposed composition framework, the transitions of a statechart are labeled with events, conditions, and assignment operations over process variables. States can be *basic* or *compound*. Basic states are labelled with invocations to Web services operations. Compound states contain one or several statecharts within them. Specifically, compound states come in two flavors: OR and AND states. An OR-state contains a single statechart within it whereas an AND-state contains several statecharts (separated by dashed lines) which are intended to be executed concurrently. Accordingly, OR-states are used as a decomposition mechanism for modularity purposes, while AND-states are used to express concurrency: they encode a fork/join pair. The initial state of a statechart is denoted by a filled circle, while the final state is denoted by two concentric circles: one filled and the other unfilled.

A simplified statechart W specifying a "Travel Planner" composite Web Service is depicted in Figure 1. In this composite service, a search for attractions is performed in parallel with a flight and an accommodation booking. After these searching and booking operations are completed, the distance from the hotel to the accommodation is computed, and either a car or a bike rental service is invoked. Note that when two transitions stem from the same state (in this case the state t_4), they denote a conditional branching, and the transitions should therefore be labelled with disjoint conditions.

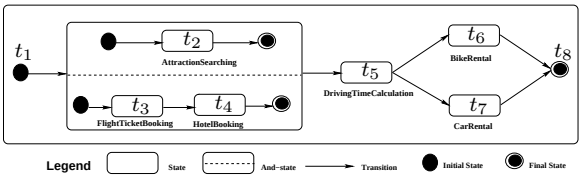


Figure 1: Statechart of a composite service "Travel Planner"

A basic state[2] of a statechart describing a composite service can be labelled with an invocation to either of the following:

- An elementary Web service, i.e., a service which does not transparently rely on other Web services.

- A composite Web service aggregating several other services.

- A *Web service community*, i.e., a collection of Web services with a common functionality although different non-functional properties (e.g., with different providers, different QoS parameters, reputation, etc.)

The concept of Web service community addresses the issue of composing a large and changing collection of Web services. Service communities provide descriptions of a desired functionality (e.g., flight booking) without referring to any actual service (e.g., Qantas flight booking Web service).

[1]See http://dev2dev.bea.com/techtrack/standards.jsp
[2]Also called task in remainder of this paper.

The set of members of a community can be fixed when the community is created, or it can be determined through a registration mechanism, thereby allowing service providers to join, quit, and reinstate the community at any time. When a community receives a request to execute an operation, this request is delegated to one of its current members. The choice of the delegatee is done at execution time based on the parameters of the request, the characteristics of the members, the history of past executions, and the status of ongoing executions. Sections 3 and 4 deal with the selection of delegatees during the execution of a composite service whose states are labelled with invocations to communities.

2.2 Execution paths and plans

In this section, we define two concepts used in the rest of the paper: *execution path* and *execution plan*.

Definition 1 (*Execution path*). An execution path of a statechart is a sequence of states [t_1, t_2, .. t_n], such that t_1 is the initial state, t_n is the final state, and for every state t_i ($1 < i < n$), the following holds:

- t_i is a direct successor of one of the states in [t_1,...,t_{i-1}]
- t_i is not a direct successor of any of the states in [t_{i+1},...,t_n]
- There is no state t_j in [t_1, ..., t_{i-1}] such that t_j and t_i belong to two alternative branches of the statechart.
- If t_i is the initial state of one of the concurrent regions of an AND-state *AST*, then for every other concurrent region C in *AST*, one of the initial states of C appears in [t_1, ..., t_{i-1}, t_{i+1}, ..., t_n]. In other words, when an AND-state is entered, all its concurrent branches are executed.

□

This definition relies on the concept of *direct successor* of a state. A basic state t_b is a direct successor of another basic state t_a if there is a sequence of adjacent transitions[3] going from t_a to t_b without traversing any other basic state. In other words, the first transition in the sequence stems from t_a, the last transition leads to t_b, and all intermediate transitions stem from and lead to either compound, initial, or final states, but are not incident to a basic state.

It is straightforward to see that an acyclic statechart has a finite number of execution paths. To simplify the presentation, we initially assume that all the statecharts that we deal with are acyclic. If a statechart contains cycles, a technique for "unfolding" it into an acyclic statechart needs to be applied beforehand. The method used to unfold the cycles of a statechart is to examine the logs of past executions in order to determine the average number of times that each cycle is taken. The states appearing between the beginning and the end of the cycle are then cloned as many times as the cycle is taken in average. Details about this unfolding process are omitted for space reasons.

Under the assumption that the underlying statechart is acyclic, it is possible to represent an execution path of this statechart as a Directed Acyclic Graph (DAG) as follows.

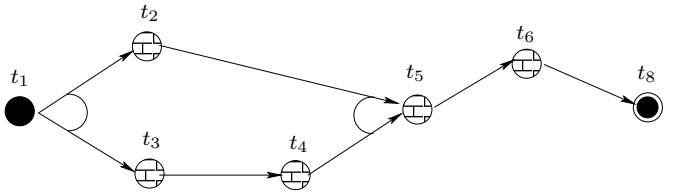
[3]Two transitions are adjacent if the target state of one is the source state of the other.

Definition 2 (*DAG representation of an execution path*). Given an execution path [t_1, t_2, .. t_n] of a statechart ST, the DAG representation of this execution path is a graph obtained as follows:

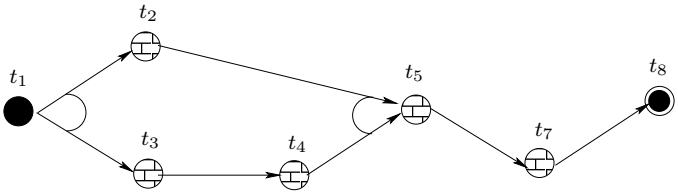
- The DAG has one node for each task t_1, t_2, .. t_n.
- The DAG contains an edge from task t_i to task t_j iff t_j is a direct successor of t_i in the statechart ST.

□

If a statechart diagram contains conditional branchings, it has multiple execution paths. Each execution path represents a sequence of tasks to complete a composite service execution. Figure 2 gives an example of a statechart's execution paths. In this example, since there is one conditional branching after task t_4, there are two paths, called W_{e1} and W_{e2} respectively. In execution path W_{e1}, task t_6 is executed after task t_5, while in execution path W_{e2}, task t_7 is executed after task t_5.



Execution Path 1 (W_{e1})



Execution Path 2 (W_{e2})

Figure 2: DAG representation of the execution paths of the statechart of Figure 1.

As stated before, the basic states of a statechart describing a composite service can be labelled with invocations to communities. If this is the case, actual Web services (i.e., members of communities) need to be selected during the execution of the composite service. Hence, it is possible to execute a path in very different ways by allocating different Web services to the states in the path. The concept of execution plan defined below captures the various ways of executing a given execution path.

Definition 3 (*Execution plan*). A set of pairs $p = \{< t_1, s_{i1} >, < t_2, s_{i2} >, \ldots, < t_N, s_{iN} >\}$ is an execution plan of an execution path W_e iff:

- $\{t_1, t_2, \ldots t_N\}$ is the set of tasks in W_e.
- For each 2-tuple $< t_j, s_{ij} >$ in p, service s_{ij} is assigned the execution of task t_j.

□

3. WEB SERVICE QUALITY MODEL

In a Web environment, multiple Web services may provide similar functionalities with different non-functional property values (e.g., different prices). In the composition model presented in the previous section, such Web services will typically be grouped together in a single community. To differentiate the members of a community during service selection, their non-functional properties need to be considered. For this purpose, we adopt a Web services quality model based on a set of quality criteria (i.e. non-functional properties) that are transversal to all Web services, for example, their pricing and reliability. Although the adopted quality model has a limited number of criteria (for the sake of illustration), it is extensible: new criteria can be added without fundamentally altering the service selection techniques built on top of the model. In particular, it is possible to extend the quality model to integrate non-functional service characteristics such as those proposed in [22], or to integrate service QoS metrics such as those proposed by [25].

In this section, we first present the quality criteria in the context of elementary services, before turning our attention to composite services. For each criterion, we provide a definition, indicate its granularity (i.e., whether it is defined for an entire service or for individual service operations), and provide rules to compute its value for a given service.

3.1 Quality Criteria for Elementary Services

We consider five generic quality criteria for elementary services: (1) *execution price*, (2)*execution duration*, (3) *reputation*, (4) *reliability*, and (5) *availability*.

- **Execution price**. Given an operation *op* of a service s, the execution price $q_{price}(s, op)$ is the amount of money that a service requester has to pay for executing the operation *op*. Web service providers either directly advertise the execution price of their operations, or they provide means to enquire about it.

- **Execution duration.** Given an operation *op* of a service s, the execution duration $q_{du}(s, op)$ measures the expected delay in seconds between the moment when a request is sent and the moment when the results are received. The execution duration is computed using the expression $q_{du}(s, op) = T_{process}(s, op) + T_{trans}(s, op)$, meaning that the execution duration is the sum of the processing time $T_{process}(s, op)$ and the transmission time $T_{trans}(s, op)$. Services advertise their processing time or provide methods to enquire about it. The transmission time is estimated based on past executions of the service operations, i.e., $T_{trans}(s, op) = \frac{\sum_{i=1}^{n} T_i(s,op)}{n}$, where $T_i(s, op)$ is a past observation of the transmission time, and n is the number of execution times observed in the past.

- **Reliability.** The reliability $q_{rel}(s)$ of a service s is the probability that a request is correctly responded within a the maximum expected time frame (which is published in the Web service description). Reliability is a technical measure related to hardware and/or software configuration of Web services and the network connections between the service requesters and providers. The value of the reliability is computed from historical data about past invocations using the expression $q_{rel}(s) = N_c(s)/K$, where $N_c(s)$ is the number of times that the service s has been successfully delivered within the maximum expected time frame, and and K is the total number of invocations.

- **Availability.** The availability $q_{av}(s)$ of a service s is the probability that the service is accessible. The value of the availability of a service s is computed using the following expression $q_{av}(s) = T_a(s)/\theta$, where T_a is the total amount of time (in seconds) in which service s is available during the last θ seconds (θ is a constant set by an administrator of the service community). The value of θ may vary depending on a particular application. For example, in applications where services are more frequently accessed (e.g., stock exchange), a small value of θ gives a more accurate approximation for the availability of services. If the service is less frequently accessed (e.g., online bookstore), using a larger θ value is more appropriate. Here, we assume that Web services send notifications to the system about their running states (i.e., available, unavailable).

- **Reputation.** The reputation $q_{rep}(s)$ of a service s is a measure of its trustworthiness. It mainly depends on end user's experiences of using the service s. Different end users may have different opinions on the same service. The value of the reputation is defined as the average ranking given by to the service by end users, i.e., $q_{rep} = \frac{\sum_{i=1}^{n} R_i}{n}$, where R_i is the end user's ranking on a service's reputation, n is the number of times the service has been graded. Usually, the end users are given a range to rank Web services, for example, in Amazon.com, the range is $[0, 5]$.

Given the above quality criteria, the quality vector of a service s is defined as follows:

$$q(s) = (q_{price}(s), q_{du}(s), q_{av}(s), q_{re}(s), q_{rep}(s)) \quad (1)$$

Note that the method for computing the value of the quality criteria is not unique. The global planning model presented Section 4 is independent of these methods.

Table 1: Aggregation functions for computing the QoS of execution plans

Criteria	Aggregation function
Price	$Q_{price}(p) = \sum_{i=1}^{N} q_{price}(s_i, op_i)$
Duration	$Q_{du}(p) = CPA(q_{du}(s_1, op_1), ..., q_{du}(s_N, op_N))$
Reputation	$Q_{rep}(p) = \frac{1}{N} \sum_{i=1}^{N} q_{rep}(s_i)$
Reliability	$Q_{rel}(p) = \Pi_{i=1}^{N}(e^{q_{rel}(s_i)*z_i})$
Availability	$Q_{av}(p) = \Pi_{i=1}^{N}(e^{q_{av}(s_i)*z_i})$

3.2 Quality Criteria for Composite Services

The above quality criteria are also applied to evaluate the QoS of composite services. Table 1 provides aggregation functions for the computation of the QoS of a composite service CS when executed using plan $p = \{< t_1, s_{i1} >, < t_2, s_{i2} >, \ldots, < t_N, s_{iN} >\}$. A brief explanation of each criterion's aggregation function follows:

- **Execution price**: The execution price $Q_{price}(p)$ of an execution plan p is a sum of every service s_i's execution price $q_{price}(s_i, op_i)$.

- **Execution duration**: The execution duration $Q_{du}(p)$ of an execution plan p is computed using the Critical Path Algorithm (CPA) [23]. Specifically, the CPA is applied to the the execution path of execution plan p, seen as a project digraph. The critical path of a project digraph is a path from the initial state to the final state which has the longest total sum of weights labelling its nodes. In the case at hand, the weights labelling the nodes correspond to the maximum expected execution durations. A task that belongs to the critical path is a *critical task*, while a service that belongs to the critical path is a *critical service*.

  Figure 3 provides an example of critical path. In this example, the project digraph represents execution path W_{e1} and its execution plan p, where p=$\{ < t_2, s_{23} >, < t_3, s_{38} >, < t_4, s_{45} >, < t_5, s_{59} >, < t_6, s_{62} > \}$. Each task's execution duration is given in the project digraph. There are two *project paths* in this project digraph, where project path 1 is $< t_2, t_5, t_6 >$ and project path 2 is $< t_3, t_4, t_5, t_7 >$. The total execution time of project path 1 (project path 2) is 37 seconds (62 seconds). Since project path 2's total execution duration is longer than that of project path 1, the critical path for the project digraph is project path 2. Thus, the execution plan's total execution duration is 62 seconds. Task t_3, t_4, t_5 and t_7 are critical tasks. Services s_{38}, s_{45}, s_{59} and s_{62} are critical services.

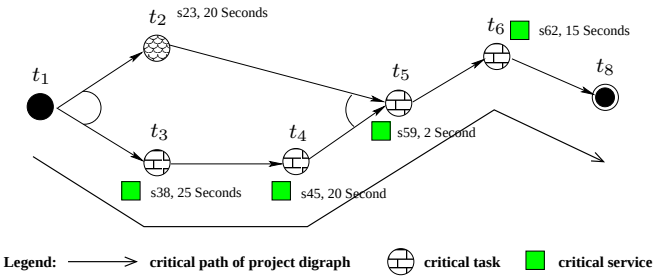


Figure 3: Critical Path

- **Reputation**: The reputation $Q_{rep}(p)$ of an execution plan p is the average of each service s_i's reputation $q_{rep}(s_i)$ in the execution plan p.

- **Reliability**: Reliability $Q_{rel}(p)$ of an execution plan p is a product of $e^{q_{rel}(s_i)*z_i}$. In the aggregation function, z_i is equal to 1 if service s_i is a critical service in the execution plan p, or 0 otherwise. If $z_i = 0$, i.e., service s_i is not a critical service, then $e^{q_{rel}(s_i)*z_i} = 1$, and hence, the reliability of service s_i will not affect the value of execution plan's reliability.

- **Availability**: The availability $Q_{av}(p)$ of an execution plan p is a product of $e^{q_{av}(s_i)*z_i}$, where $q_{av}(s_i)$ is service s_i's availability.

Using above aggregation functions, the quality vector of a composite service's execution plan is defined as:

$$Q(p) = (Q_{price}(p), Q_{du}(p), Q_{av}(p), Q_{re}(p), Q_{rep}(p)) \quad (2)$$

4. GLOBAL SERVICE SELECTION

As mentioned before, in existing approaches, the selection of component service to execute a task is determined independently to other tasks of composite services [2, 11, 6]. More precisely, in our previous work [2], service selection is done at each service community locally. The selection of a service is based on a selection policy involving parameters of the request, the characteristics of the members, the history of past executions, and the status of ongoing executions. Although service selection can be locally optimized, the global quality constraints may not be satisfied. For example, a global constraint such as composite services' execution price is less than 500 dollars can not be enforced. In this section, we present a global planning based approach for Web services selection. We first present an approach of selecting an optimal execution plan for a composite service, then present a novel linear programming based method for optimal execution plan selection.

4.1 Selecting an Optimal Execution Plan

The basic idea of global planning is the same as query optimization in database management systems. Several plans are identified and the optimal plan is selected. The foregoing discussion makes it clear that a statechart has multiple execution paths and each execution path has its own set of execution plans if the statechart contains conditional branchings. In this subsection, we assume that the statechart does not contain any conditional branchings and has only one execution path. We will discuss the case where a statechart has multiple execution paths in Section 4.2.

We also assume that for each task t_j, there is a set of candidate Web services S_j that are available to which task t_j can be assigned. Associated with each Web service s_{ij} is a quality vector (see equation 1). Based on the available Web services, by selecting a Web service for each task in an execution path, the global planner will generate a set of execution plans P :

$$P = \{p_1, p_2, ..., p_n\} \quad (3)$$

n is the number of execution plans. After a set of execution plans is generated, the system needs to select an optimal execution plan. When selecting the execution plan, instead of computing the quality vector of a particular Web service, each execution plan's global service quality vector needs to be computed.

The selection of execution plan uses Multiple Attribute Decision Making (MADM)[16] approach. Once the quality vector for each execution plan is derived, by accumulating all the execution plans' quality vectors, we obtain matrix $\mathbb{Q}$, where each row represents an execution plan's quality vector.

$$\mathbb{Q} = \begin{pmatrix} Q_{1,1} & Q_{1,2} & \dots & Q_{1,5} \\ Q_{2,1} & Q_{2,2} & \dots & Q_{2,5} \\ \vdots & \vdots & \vdots & \vdots \\ Q_{n,1} & Q_{n,2} & \dots & Q_{n,5} \end{pmatrix} \quad (4)$$

A Simple Additive Weighting (SAW) [4] technique is used to select an optimal execution plan. Basically, there are two phases in applying SAW:

4.1.1 Scaling Phase

Some of the criteria used could be negative, i.e., the higher the value is, the lower the quality is. This includes criteria such as execution time and execution price. Other criteria are positive criteria, i.e., the higher the value is, the higher the quality is. For negative criteria, values are scaled according to Equation 5. For positive criteria, values are scaled according to Equation 6.

$$V_{i,j} = \begin{cases} \frac{Q_j^{max} - Q_{i,j}}{Q_j^{max} - Q_j^{min}} & \text{if } Q_j^{max} - Q_j^{min} \neq 0 \\ 1 & \text{if } Q_j^{max} - Q_j^{min} = 0 \end{cases} \quad j = 1, 2 \qquad (5)$$

$$V_{i,j} = \begin{cases} \frac{Q_{i,j} - Q_j^{min}}{Q_j^{max} - Q_j^{min}} & \text{if } Q_j^{max} - Q_j^{min} \neq 0 \\ 1 & \text{if } Q_j^{max} - Q_j^{min} = 0 \end{cases} \quad j = 3, 4, 5 \quad (6)$$

In the above equations, Q_j^{max} is maximal value of a quality criterion in matrix $\mathbb{Q}$, i.e., $Q_j^{max} = Max(Q_{i,j}), 1 \leq i \leq n$. Q_j^{max} is minimal value of a quality criterion in matrix $\mathbb{Q}$, i.e., $Q_j^{min} = Min(Q_{i,j}), 1 \leq i \leq n$.

In fact, we can compute Q_j^{max} and Q_j^{min} without generating all possible execution plans. For example, in order to compute the maximum execution price (i.e., Q_{price}^{max}) of all the execution plans, we select the most expensive Web service for each task and sum up all these execution prices to compute Q_{price}^{max}. In order to compute the minimum execution duration (i.e., Q_{du}^{min}) of all the execution plans, we select the Web service that has shortest execution duration for each task and use CPA to compute Q_{du}^{min}. The computation cost of Q_j^{max} and Q_j^{min} is polynomial.

After the scaling phase, we obtain the following matrix $\mathbb{Q}'$:

$$\mathbb{Q}' = \begin{pmatrix} V_{1,1} & V_{1,2} & \dots & V_{1,5} \\ V_{2,1} & V_{2,2} & \dots & V_{2,5} \\ \vdots & \vdots & \vdots & \vdots \\ V_{n,1} & V_{n,2} & \dots & V_{n,5} \end{pmatrix}$$

4.1.2 Weighting Phase

The following formula is used to compute the overall quality score for each execution plan:

$$Score(p_i) = \sum_{j=1}^{5} (V_{i,j} * W_j) \qquad (7)$$

where $W_j \in [0, 1]$ and $\sum_{j=1}^{5} W_j = 1$. W_j represents the weight of each criterion. In (7), end users can give their preference on QoS (i.e., balance the impact of the different criteria) to select a desired execution plan by adjusting the value of W_j. The global planner will choose the execution path which has the maximal value of $Score(p_i)$ (i.e., $max(Score(p_i))$). If there are more than one execution plans which have the same maximal value of $Score(p_i)$, then an execution plan will be selected from them randomly.

4.2 Handling Multiple Execution Paths

In Section 4.1, we assume that the statechart only has one execution path. In this subsection, we discuss the case where statecharts have multiple execution paths. Assume that a statechart has n execution paths. For each execution path, an optimal execution plan can be selected. So, the global planner has n selected execution plans. Since each selected optimal execution plan only covers a subset of the entire statechart, then the global planner needs to aggregate these n execution plans into an overall execution plan to covers all the tasks in the statechart. This overall execution plan will be used to execute the statechart. For example, for `travel planner` statechart W (see Figure 1), there are two execution paths W_{e1} and W_{e2}. The optimal execution plans p_1 and p_2 of these two execution paths are selected. From the Figure 2, it can be seen that both execution paths W_{e1} and W_{e2} are subsets of W. Thus neither p_1 nor p_2 covers all tasks in W. Since the global planner conducts planning before the execution time, it does not know which execution path will eventually be used for the composite service. Therefore it needs to aggregate p_1 and p_2 into an overall execution plan which covers all the tasks in W.

Assume that statechart W has k tasks (i.e., t_1, t_2, ..., t_k) and n execution paths (i.e., W_{e1}, W_{e2},..., W_{en}). Thus, for each execution path, the global planner selects an optimal execution plan. Consequently, we obtain n optimal execution plans (i.e., $p_1, p_2, ..., p_n$) for these execution paths. The global planner adopts the following approach to aggregate multiple execution plans into an overall execution plan.

1. Given a task t_i, if t_i only belongs to one execution path (e.g., W_{ej}), then the global planner selects W_{ej}'s execution plan p_j to execute the task t_i. We denote this as $t_i(p_j)$. For example, in trip planning statechart, task t_7 (i.e.,`CarRental`) only belongs to execution path W_{e2}. In this case, W_{e2}'s execution plan p_2 is used to execute t_7, i.e., $t_7(p_2)$.

2. Given a task t_i, if t_i belongs to more than one execution paths (e.g., W_{ej}, W_{ej+1}, ..., W_{em}), then there is a set of execution plans (i.e., p_j, p_{j+1}, ..., p_m) that can be used to execute W_{si}. In this case, the global planner needs to select one of the execution plans from p_j, p_{j+1}, ... , p_m. The selection can be done by identifying the *hot path* for task t_i. Here, the hot path of a task t_i is defined as the execution path that has been most frequently used to execute task t_i in the past. For example, in `travel planner` statechart, task t_3 (`FlightTicketBooking`) belongs to both execution path W_{e1} and W_{e2}. Assume that the statechart W has been used to execute the composite service for 25 times. Also assume that, in 20 times the execution of the composite service follows the execution path W_{e1}; while in 5 times, the execution of the composite service follows the execution path W_{e2}. This indicates that execution path W_{e1} has been more frequently used to execute task t_3 (i.e., W_{e1} is the hot path for t_3). Thus, W_{e1}'s execution plan p_1 is used to execute t_3, i.e., $t_3(p_1)$.

The system keeps composite service execution traces in an execution history [10]. This allows the global planner to identify hot path for each task.

4.3 Linear Programming Solution

The approach of selecting an optimal execution plan given in the previous section requires the generation of all possible

execution plans. Assume that there are N tasks in a statechart and there are M potential Web services for each task. The total number of execution plans is M^N. The computation cost of selecting an optimal execution plan is $O(M^N)$. Such an approach is impractical for large scale composite services, where both the number of tasks in the composite services and the number of candidate Web services in communities are large. For example, assume that a composite service has one execution path and 10 tasks, and for each task, there are 10 candidate Web services. Then the total number of execution plans is 10^{10}. It is very costly to generate all these 10^{10} plans and select an optimal one. In this subsection, we present a method based on linear programming (LP) [15], which can be used to select an optimal execution plan without generating all the possible execution plans.

There are three inputs in LP: *variables*, an *objective function* and *constraints* on the variables, where both the objective function and constraints must be linear. LP attempts to maximize or minimize the value of the objective function by adjusting the values of variables based on the constraints. The output of LP is the maximum (or minimum) value of the objective function as well as the values of variables at this maximum or minimum point.

In order to use LP to select an optimal execution plan, we model the selection of an optimal execution plan as an LP problem. The variables of the LP problem are y_{ij} representing the participation of service s_{ij} in the selected execution plan. The value of each variable y_{ij} is 1 if service s_{ij} is in the selected plan, 0 otherwise. The objective function of the LP problem, which is based on equations 5,6, and 7, is:

$$Max\left(\sum_{l=1}^{2}\left(\frac{Q_l^{max}-Q_{i,l}}{Q_l^{max}-Q_l^{min}}*W_l\right)+\sum_{l=3}^{5}\left(\frac{Q_{i,l}-Q_l^{min}}{Q_l^{max}-Q_l^{min}}*W_l\right)\right) \quad (8)$$

where $W_l \in [0,1]$ and $\sum_{j=1}^{5} W_j = 1$. W_l is the weight assigned to quality criteria l.

In the following subsections, we discuss the constraints on the variables of the LP problem.

4.3.1 Constraints on Execution Duration and Execution Price

In this subsection, we consider constraints on the execution duration and the execution price of an execution plan. Assume that A is the set of all tasks (i.e., basic states) of the statechart. For each task t_j, there is a set of Web services S_j that can be assigned to this task, but on the end, for each task t_j, only one Web service should be selected. Given that y_{ij} ($y_{ij} = 0$ or 1) denotes the participation of Web service s_{ij} in the selected plan, this latter fact is captured by the following constraints:

$$\sum_{i \in S_j} y_{ij} = 1,\ \forall j \in A \quad (9)$$

For example, there are 100 potential Web services that can execute task j, since only one of them will be selected to execute the task j, then we have $\sum_{i=1}^{100} y_{ij} = 1$.

Assume that variable x_j represents the earliest start time of task t_j, variable p_j represents the execution duration for task j, and variable p_{ij} represents the execution duration for task t_j by service s_{ij}. We use the notation $t_j \rightarrow t_k$ to denote that task t_k is task t_j's direct successor task. We have the following constraints:

$$\sum_{i \in S_j} p_{ij}\ y_{ij} = p_j, \forall j \in A \quad (10)$$

$$x_k - (p_j + x_j) \geq 0, \forall t_j \rightarrow t_k, j, k \in A \quad (11)$$

$$Q_{du} - (x_j + p_j) \geq 0, \forall j \in A \quad (12)$$

Constraint 10 indicates that the execution duration of a given task t_j is equal to the execution duration of one of the Web services in A. Constraint 11 captures the fact that if task t_k is a direct successor of task t_j, the execution of task t_k must start after task t_j has been completed. Constraint 12 indicates that the execution of a composite service is completed only when all its tasks are completed.

Assume that z_{ij} is an integer variable that has value 1 or 0: 1 indicates that Web service s_{ij} is a critical service and 0 indicates otherwise. We have the following constraint on execution plan's execution duration Q_{du}:

$$Q_{du} = \sum_{j \in A} \sum_{i \in S_j} p_{ij} z_{ij} \quad (13)$$

For execution price, assume that variable c_{ij} represents the execution price of Web service s_{ij}, then we have the following constraint on total execution price of composition service:

$$Q_{price} = \sum_{j \in A} \sum_{j \in S_j} c_{ij}\ y_{ij} \quad (14)$$

An alternative of constraint 14 is as follows:

$$\sum_{j \in A} \sum_{j \in S_j} c_{ij} y_{ij} \leq B, B > 0 \quad (15)$$

where B is the budget constraint given by the user. This constraint indicates that the entire composite service's execution price can not be greater than B. By introducing a budget constraint the above problem needs to be explicitly solved as an integer programming problem. This problem is a special case of the knapsack problem and hence it is NP-hard [19]. Notice that constraints on other criteria can be easily incorporated into LP if the aggregation function is a linear function. For example, assume that variable r_{ij} represents the reputation of Web service s_{ij}, we can have the following constraint on the execution plan's reputation:

$$Q_{rep} = \sum_{j \in A} \sum_{j \in S_j} r_{ij} y_{ij} \quad (16)$$

4.3.2 Constraints on Reliability and Availability

In this subsection, we consider constraints on criteria where the aggregation function is not a linear function. Among the criteria that are used to select Web services, both the availability's and the reliability's aggregation functions are non-linear (see Table 1). We can linearize them using a logarithm function as shown below. Assume that variable a_{ij} represents the reliability of Web service s_{ij}. Since z_{ij} indicates whether Web service s_{ij} is a critical service or not, the reliability of execution plan is:

$$Q_{rel} = \Pi_{j \in A}\left(\sum_{j \in S_j} e^{a_{ij} z_{ij}}\right)$$

By applying the logarithm function ln, we obtain:

$$ln(Q_{rel}) = \sum_{j \in A} ln \left(\sum_{j \in S_j} e^{a_{ij} z_{ij}} \right)$$

Since $\sum_{j \in A} z_{ij} = 1$ and $z_{ij} = 0$ or 1, we obtain:

$$ln(Q_{rel}) = \sum_{j \in A} \left(\sum_{j \in S_j} a_{ij} z_{ij} \right)$$

Let $Q'_{rel} = ln(Q_{rel})$, we have the following constraint on the execution plan's reliability:

$$Q'_{rel} = \sum_{j \in A} \sum_{j \in S_j} a_{ij} z_{ij} \quad (17)$$

Similarly, assuming that b_{ij} represents the availability of the Web service s_{ij}, the following constraint is introduced:

$$Q'_{av} = \sum_{j \in A} \sum_{i \in S_j} b_{ij} z_{ij} \quad (18)$$

where $Q'_{av} = ln(Q_{av})$.

Criteria that can be introduced into the LP problem are not limited to what we defined in Section 3. Other criteria can be added once the aggregation functions are given.

4.3.3 Constraints on Uncertainty of Execution Duration

In the previous sections, we assume that the execution durations p_{ij} of Web services are deterministic. In reality, the execution duration of a Web service s_{ij} may be uncertain. For example, Service s may advertise that the execution duration is 5 seconds, but the actual execution duration may be 4.5, 4.6, or 5.2 seconds. To address this issue, we model each p_{ij} using a normal distribution. Variable p_{ij} has therefore the following probability function:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} exp\left[-\frac{1}{2} \left(\frac{x-\mu}{\sigma} \right)^2 \right], -\infty < x < \infty$$

where the mean μ and the std. deviation σ are given by:

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i \quad (19)$$

$$\sigma^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \mu)^2 \quad (20)$$

By applying formulas 19 and 20 to the execution logs of a Web service s_{ij}, we can obtain μ_{ij} and σ_{ij} for this Web service. Since $\sum_{i \in A} \sum_{i \in S_j} p_{ij} z_{ij} = Q_{du}$, the total execution duration Q_{du} must follow a normal distribution[4] whose deviation σ_{du} is:

$$\sigma^2_{du} = \sum_{i \in A} \sum_{i \in S_j} \sigma^2_{ij} z_{ij} \quad (21)$$

So in order to consider the deviation of the total execution duration in the LP problem, we should adopt the following objective function:

$$Max \left(\sum_{l=0}^{2} \left(\frac{Q_l^{max} - Q_{i,l}}{Q_l^{max} - Q_l^{min}} * W_l \right) + \sum_{l=3}^{5} \left(\frac{Q_{i,l} - Q_l^{min}}{Q_l^{max} - Q_l^{min}} * W_l \right) \right) \quad (22)$$

where $Q_0 = \sigma^2_{du}$ and $W_0 \in [0, 1]$ is the weight assigned to the deviation of the total execution duration.

Given the above inputs for the LP problem, the output of the LP solver will be a set of values for variables y_{ij}, which indicate the selection or exclusion of Web services s_{ij}. The selected Web services compose an optimal execution plan.

5. VALIDATION

In this section, we present the implementation of the QoS-driven selection of services and some experimental results to evaluate the proposed approach.

5.1 Implementation

The proposed QoS-driven selection technique is implemented in the SELF-SERV prototype. Detailed description of SELF-SERV can be found in [24]. In this section, we briefly overview the prototype architecture and discuss its extension to support the service selection approach. The prototype architecture (Figure 4) features an *interface*, a *service manager* and a *pool of services*. Services communicate via Simple Object Access Protocol (SOAP) messages.

The *service manager* consists of four modules, namely the *service discovery engine*, the *service editor*, the *composite service orchestrater* and the *global planner*. The *service discovery engine* facilitates the advertisement and location of services. It is implemented using the Universal Description, Discovery and Integration (UDDI), the Web Service Description Language (WSDL), and SOAP. In the implementation, we make extensive use of the IBM Web Services Toolkit 2.4 (WSTK2.4) [14], which is a showcase package for Web services emerging technologies.

The *service editor* provides facilities for defining new services and editing existing ones. A service is edited through a visual interface, and translated into an XML document for subsequent analysis and processing by the *service orchestrater*. The orchestrater is responsible for scheduling, initiating, and monitoring the invocations to the tasks of a composite service during its execution, and for routing events and data items between these components.

Finally, *global planner* is the module that plans the execution of a composite service using the global planning based approach. The global planner is implemented as a linear programming solver based on IBM's Optimization Solutions and Library (OSL) [13]. It should be noted that QoS information is retrieved via pre-defined operations of services (e.g., `getExecutionTime()`).

5.2 Experimentation

We conducted experiments using the implemented prototype system to evaluate our approach. In the experiments, a cluster of PCs were used to run the prototype system. All PCs have the same configuration of Pentium III 933MHz with 512M RAM. Each PC runs Windows 2000, Java 2 Edition V1.3.0, and Oracle XML Developer Kit (Oracle XDK, for XML parsing). They are connected to a LAN through 100Mbits/sec Ethernet cards. This section presents two experimental results. The first experiment compares the QoS

[4] A detailed proof can be found in [26].

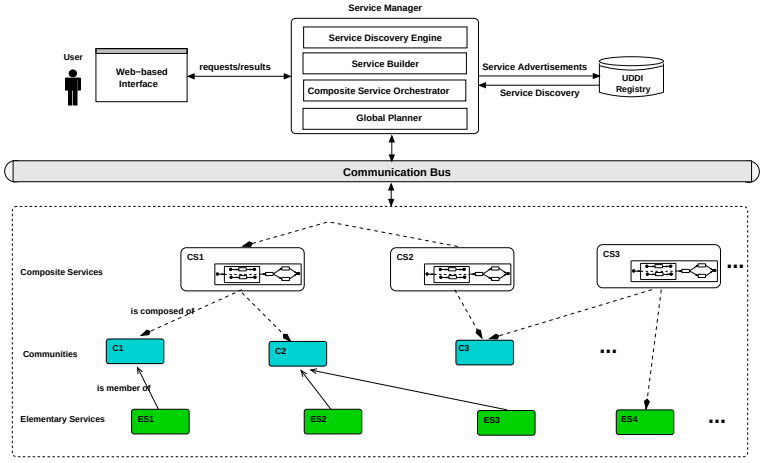


Figure 4: Architecture of the prototype.

metrics of the execution of composite services using the global planning and local selection approaches. The second experiment shows the system costs (i.e., computation cost and bandwidth cost) of these two approaches.

5.2.1 QoS Metrics

The purpose of this experiment is to compare the QoS values of the global planning approach with that of the local selection. The comparison was done by measuring price and execution time of the composite services using both approaches. In the experiment, we created several composite services with different number of basic states. The services were created by randomly adding states to the composite service shown in Figure 1. The number of states ranges over the values 10, 20, 30, 40, 50, 60, 70, and 80. For each composite service, we executed the service 12 times and recorded the price and execution time. Since we obtained similar experimental results for all created composite services, only the result of one composite service (with 20 states) is shown (see Table 2) for clarity reasons.

From the table, we can see that for every instance of the composite service, the global planning approach gives better QoS than local selection approach. For example, for the instance 7 of Table 2, the time required to execute the composite service is: (i) 6451 seconds in the global planning approach, (ii) 9480 seconds in the local selection approach. Similarly, the execution price spends for executing this composite service is: (i) 1231 dollars in the global planning approach, (ii) 1789 dollars in the local selection approach. Overall, the average execution time (resp., execution price) is: (i) 6627.2 seconds (resp., 1191 dollars) in the global planning approach, (ii) 9305.9 seconds (resp., 1753 dollars) in the local selection approach.

5.2.2 System Costs

The aim of this experiment is to investigate the system costs of executing composite services using the LP-based global planning and local selection approaches. The experiment was done by measuring: (i) computation cost (i.e., time used for selecting a Web service for each task of the composite service); (ii) bandwidth cost (i.e., network bandwidth consumed between global planner and Web services).

Instance No	$Q_{time}(W)$ (second)		$Q_{price}(W)$ (dollar)	
	Global	Local	Global	Local
1	6523.2	8322.4	1023	1642
2	6634.4	9123.9	1117	1728
3	6843.2	9234.5	1123	1825
4	6432.5	9292.2	1132	1824
5	6347.3	8943.3	1121	1723
6	6512.3	9902.8	1185	1888
7	6451.2	9480.4	1231	1789
8	6440.5	9470.5	1275	1787
9	6970.4	9920.4	1324	1625
10	6890.3	9628.3	1235	1759
11	6590.3	9520.3	1267	1852
12	6890.3	8920.5	1250	1599
Average:	6627.2	9305.9	1191	1753

Table 2: The QoS of a composite service with 20 states

In the experiment, we created several composite services with different number of tasks. The services were created by randomly adding states to the composite service shown in Figure 1. The number of tasks ranges over the values 10, 20, 30, 40, 50, 60, 70, and 80. In addition, we created a set of Web services which were assigned to tasks of composite services as the candidate Web services. For each task, the number of candidate Web services we used varies as follows: 5, 10, 20, and 40 services. We executed composite services with different number of states and candidate Web services. The computation and bandwidth costs for selecting Web services were recorded. The results for composite services with 40 candidate Web services of each state are shown in figures 5 and 6 respectively. Similar results were obtained for other cases.

From Figure 5, we can see that for both global and local selection approaches, the computation cost increases when the number of tasks and the number of candidate Web services increases. As expected, the computation cost of global planning approach is a little bit higher than that of local selection approach. For example, a composite service with 40 tasks spends: (i) 1.6 seconds for selecting Web services in global planning approach, (ii) 0.7 seconds in local selection approach.

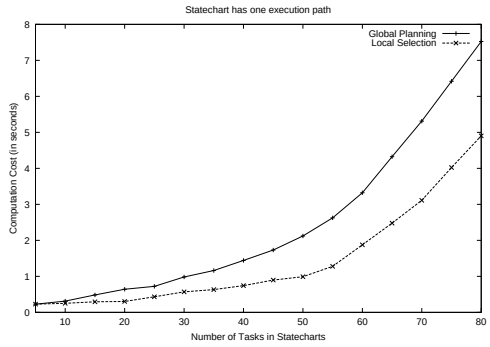


Figure 5: The computation cost of selecting services for composite services (each state has 40 candidate Web services)

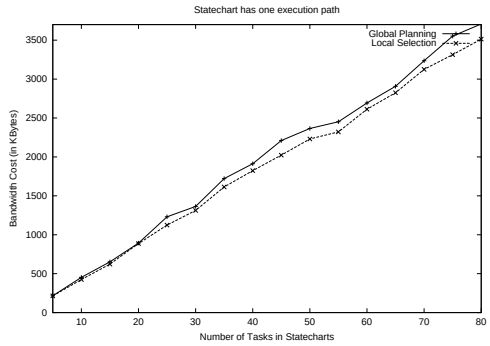


Figure 6: The bandwidth cost of selecting service for composite services (each state has 40 candidate Web services)

Similar observations are found regarding bandwidth cost. More specifically, for both approaches, the linear increase of the number of tasks and the number of candidate Web services leads almost a linear increase of bandwidth cost (see Figure 6). The bandwidth cost in global planning is slightly higher than that of local selection approach. For example, a composite service with 40 tasks consumes about 2080 KBytes of network bandwidth for selecting Web services in global planning approach, while it consumes 1910 KBytes in local selection approach.

6. RELATED WORK

In this section, we briefly discuss the relationships between our work and existing Web service standards, Web service composition approaches, and QoS-driven workflow management.

Several standardisation proposals aiming at providing infrastructure to support Web services composition have recently emerged including SOAP, WSDL, UDDI, and BPEL4WS. SOAP defines an XML messaging protocol for communication among services. WSDL is an XML-based language for describing web service interfaces. UDDI provides the directory and a SOAP-based API to publish and discover services. Finally, BPEL4WS provides a language for process-based service composition. Other proposed notations for service description and composition include ebXML and DAML-S. The above proposals however are complementary to ours. Indeed, our proposal aims at leveraging the above standards (e.g., SOAP, UDDI) to provide a quality-driven and dynamic service composition model.

Web service composition is a very active area of research and development [1, 7, 8]. Previous efforts in this area such as CMI [11] and eFlow [6] have investigated dynamic service selection based on user requirements. In particular, CMI's service definition model features the concept of a *placeholder activity* to cater for dynamic composition of services. A placeholder is an abstract activity replaced at runtime with a concrete activity type. A selection policy is specified to indicate the activity that should be executed in lieu of the placeholder. In eFlow, the definition of a service node contains a *search recipe* represented in a query language. When a service node is invoked, a search recipe is executed in order to select a reference to a specific service. Both CMI and eFlow focus on optimizing service selection at single task level (i.e. local selection). In addition, no QoS model is explicitly supported. In contrast, our approach focuses on optimizing service selection at a composite service level, based on a generic QoS model, and using established linear programming techniques.

Related work on QoS has been conducted in the area of workflow. In particular, there are a number of research proposals addressing the specification and verification of temporal constraints in workflows [9, 3]. Other proposals such as METEOR [5] and CrossFlow [18, 17] have considered QoS models with other parameters than time. Specifically, [5] considers four quality dimensions, namely time, cost, reliability and fidelity. However, it does not consider the dynamic composition of services. Instead, it focuses on analyzing, predicting, and monitoring QoS of workflow processes. Similarly, [18] proposes the use of continuous-time Markov chain to estimate execution time and cost of a workflow. Closer to our proposal is the one reported in [17], which explores the issue of dynamically selecting several alternative tasks within a workflow, based on quality parameters, in a similar way as eFlow does using search recipes. As stated before, this local selection strategy contrasts with the global planning approaches that we advocate.

Other related research proposals include [20, 21], which focus on data quality management in cooperative information systems. They investigate techniques to select best available data from different service providers based on a set of data quality dimensions such as accuracy, completeness, and consistency.

7. CONCLUSION

Dynamic selection of component services is an important issue in Web services composition. In this paper, we have presented a general and extensible model to evaluate QoS of both elementary and composite services. Based on the QoS model, a global service selection approach that uses linear programming techniques to compute optimal execution plans for composite services has been described.

We have conducted experiments to compare the proposed technique with the local selection approach. The results show that the proposed approach effectively selects high quality execution plans (i.e., plans which have higher overall

QoS). Our ongoing research includes the support for exception handling during composite service executions. For example, after an execution plan has been built and while it is being executed, an exception may occur (e.g., unavailability of a component service). We will explore the possibility of performing dynamic plan revision during composite service execution, as a means to respond to runtime exceptions.

Acknowledgments

The work of Boualem Benatallah is partially supported by the Australian Research Council's Discovery GRANT DP02-1120.

8. REFERENCES

[1] B. Benatallah and F. Casati, editors. *Distributed and Parallel Database, Special issue on Web Services*. Springer-Verlag, 2002.

[2] B. Benatallah, M. Dumas, Q. Z. Sheng, and A. Ngu. Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services. In *Proc. of ICDE'02, IEEE Computer Society*, pages 297–308, San Jose, 2002.

[3] C. Bettini, X. Wang, and S. Jajodia. Temporal Reasoning in Workflow Systems. *Distributed and Parallel Databases*, 11(3):269–306, 2002.

[4] H. C.-L and K. Yoon. *Multiple Criteria Decision Making*. Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, 1981.

[5] J. Cardoso. Quality of Service and Semantic Composition of Workflows. *Ph.D. Thesis, University of Georgia*, 2002.

[6] F. Casati, S. Ilnicki, L.-J. Jin, V. Krishnamoorthy, and M.-C. Shan. eFlow: a Platform for Developing and Managing Composite e-Services. Technical Report HPL-2000-36, HP Laboratoris, Palo Alto, 2000.

[7] F. Casati, M.-C. Shan, and D. Georgakopoulos, editors. *VLDB Journal, Special Issue on E-Services*. Springer-Verlag, 2001.

[8] A. Dogac, editor. *ACM SIGMOD Record 31(1), Special Section on Data Management Issues in E-Commerce*. ACM, March 2002.

[9] J. Eder, E. Panagos, and M. Rabinovich. Time Constraints in Workflow Systems. In *Proc. of the 11th International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 286–300, Heidelberg, Germany, June 1999.

[10] M.-C. Fauvet, M. Dumas, and B. Benatallah. Collecting and Querying Distributed Traces of Composite Service Executions. In *Proc. of the 10th International Conference on Cooperative Information Systems (CoopIS)*, Irvine, CA, USA, 2002.

[11] D. Georgakopoulos, H. Schuster, A. Cichocki, and D. Baker. Managing Process and Service Fusion In Virtual Enterprises. *Information System, Special Issue on Information System Support for Electronic Commerce*, 24(6):429–456, 1999.

[12] D. Harel and A. Naamad. The STATEMATE Semantics of Statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, 1996.

[13] IBM Optimization Solutions and Library, 2002. http://www-3.ibm.com/software/data/bi/osl/index.html.

[14] IBM WSTK Toolkit. http://alphaworks.ibm.com/tech/webservicestoolkit.

[15] H. Karloff. *Linear Programming*. Birkhauser, 1991.

[16] M. Kksalan and S. Zionts, editors. *Multiple Criteria Decision Making in the New Millennium* . Springer-Verlag, 2001.

[17] J. Klingemann. Controlled Flexibility in Workflow Management. In *Proc. of the 12th International Conference on Advanced Information Systems (CAiSE)*, pages 126–141, Stockholm, Sweden, June 2000. Springer.

[18] J. Klingemann, J. Wasch, and K. Aberer. Deriving Service Models in Cross-Organizational Workflows. In *Ninth International Workshop on Research Issues in Data Engineering: Virtual Enterprise, RIDE-VE'99*, Sydney, Australia, March 1999.

[19] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley and Sons, 2001.

[20] M. Mecella, M. Scannapieco, A. Virgillito, R. Baldoni, T. Catarci, and C. Batini. Managing Data Quality in Cooperative Information Systems. In *Proc. of the 10th International Conference on Cooperative Information Systems (CoopIS)*, Irvine, CA, USA, 2002.

[21] F. Naumann, U. Leser, and J. C. Freytag. Quality-driven Integration of Heterogenous Information Systems. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 447–458, Edinburgh, UK, 1999.

[22] J. O'Sullivan, D. Edmond, and A. ter Hofstede. What's in a Service. *Distributed and Parallel Databases*, 12(2–3):117–133, September 2002.

[23] M. Pinedof. *Scheduling: Theory, Algorithms, and Systems (2nd Edition)*. Prentice Hall, 2001.

[24] Q. Z. Sheng, B. Benatallah, M. Dumas, and E. Mak. SELF-SERV: A Platform for Rapid Composition of Web Services in a Peer-to-Peer Environment. In *Proc. of the 28th VLDB Conference*, Hong Kong, China, August 2002.

[25] A. van Moorsel. Metrics for the Internet Age: Quality of Experience and Quality of Business. Technical Report HPL-2001-179, HP Labs, August 2001. Also published in 5th Performability Workshop, September 2001, Erlangen, Germany.

[26] D. D. Wackerly, W. Mendenhall, and R. L. Scheaffer. *Mathematical Statistics with Application*. Duxbury Press, 1996.