QoS-Aware Middleware for Web Services Composition

Liangzhao Zeng, Boualem Benatallah, *Member*, *IEEE*, Anne H.H. Ngu, Marlon Dumas, *Member*, *IEEE Computer Society*, Jayant Kalagnanam, and Henry Chang

Abstract—The paradigmatic shift from a Web of manual interactions to a Web of programmatic interactions driven by Web services is creating unprecedented opportunities for the formation of online Business-to-Business (B2B) collaborations. In particular, the creation of value-added services by composition of existing ones is gaining a significant momentum. Since many available Web services provide overlapping or identical functionality, albeit with different Quality of Service (QoS), a choice needs to be made to determine which services are to participate in a given composite service. This paper presents a middleware platform which addresses the issue of selecting Web services for the purpose of their composition in a way that maximizes user satisfaction expressed as utility functions over QoS attributes, while satisfying the constraints set by the user and by the structure of the composite service. Two selection approaches are described and compared: one based on local (task-level) selection of services and the other based on global allocation of tasks to services using integer programming.

Index Terms—Web services, quality of service, service composition, integer programming.

1 Introduction

Web services are autonomous software systems identified by URIs which can be advertised, located, and accessed through messages encoded according to XML-based standards (e.g., SOAP, WSDL, and UDDI [11]) and transmitted using Internet protocols [36]. Web services encapsulate application functionality and information resources and make them available through programmatic interfaces, as opposed to the interfaces provided by traditional Web applications which are intended for manual interactions. In addition, since they are intended to be discovered and used by other applications across the Web, Web services need to be described and understood both in terms of functional capabilities and Quality of Service (QoS) properties.

The emergence of Web services (e.g., for order procurement, finance, accounting, human resources, supply chain, and manufacturing) has created unprecedented opportunities for organizations to establish more agile and versatile collaborations with other organizations. Widely available and standardized Web services make it possible to realize Business-to-Business Interoperability (B2Bi) by inter-connecting Web services provided by multiple business partners according to some business process: a practice known as Web Services Composition [9], [5], [1], [24]. For example, an integrated financial management Web service can be created by composing more specialized Web services for payroll, tax preparation, and cash management.

Our work aims at advancing the current state of the art in technologies for Web service composition, by addressing the following key issues:

1. **QoS modeling**. In the presence of multiple Web services with overlapping or identical functionality, users will discriminate these alternatives based on their QoS. QoS is a broad concept that encompasses a number of nonfunctional properties such as price, availability, reliability, and reputation [28]. These properties apply both to standalone Web services and to Web services composed of other Web services (i.e., *composite Web services*). In order to reason about QoS properties in Web services, a model is needed which captures the descriptions of these from a user perspective. Such framework must take into account the fact that QoS involves multiple dimensions, and the fact that the QoS of composite services is determined by the QoS of its underlying *component services*. Also, it is worth noting that services are usually distributed across the Internet and that some of their QoS properties (e.g., availability and successful execution rate) are affected by the communication link and should be measured from the perspective of the requestor rather than the provider.
2. **QoS-aware composition of Web services**. When creating a composite service and, subsequently, when executing it following a user request, the

- *L. Zeng, J. Kalagnanam, and H. Chang are with the IBM T.J. Watson Research Center, 1101 Kitchawan Rd., RTE 134, PO Box 218, Yorktown Heights, NY 10598. E-mail: {lzeng, jayant, hychang}@us.ibm.com.*
- *B. Benatallah is with the School of Computer Science and Engineering, University of New South Wales, Sydney 2052, Australia. E-mail: boualem@cse.unsw.edu.au.*
- *A.H.H. Ngu is with the Department of Computer Science, Texas State University, 601 University Dr., San Marcos TX 78666. E-mail: hn12@txstate.edu.*
- *M. Dumas is with the Centre for IT Innovation, Queensland University of Technology, GPO Box 2434, Brisbane QLD 4001, Australia. E-mail: m.dumas@qut.edu.au.*

Manuscript received 24 Sept. 2003; revised 18 Feb. 2004; accepted 15 Mar. 2004.
Recommended for acceptance by J. Offutt.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-0150-0903.

number of component services involved in this composite service may be large, and the number of Web services from which these component services are selected is likely to be even larger. On the other hand, the QoS of the resulting composite service executions is a determinant factor to ensure customer satisfaction, and different users may have different requirements and preferences regarding QoS. For example, a user may require to minimize the execution duration while satisfying certain constraints in terms of price and reputation, while another user may give more importance to the price than to the execution duration. A QoS-aware approach to service composition is therefore needed, which maximizes the QoS of composite service executions by taking into account the constraints and preferences of the users.

3. **Composite service execution in a dynamic environment**. Web services operate autonomously within a highly variable environment (the Web). As a result, their QoS may evolve relatively frequently, either because of internal changes or because of changes in their environment (i.e., higher system loads). In particular, during an execution of a composite service, the component services involved may change their QoS properties, others may become unavailable, and still others may emerge. Consequently, approaches where Web services are statically composed are inappropriate. Instead, a dynamic composition approach is needed, in which runtime changes in the QoS of the component services are taken into account.

In this paper, we present AgFlow [39], [41], [40]: a middleware platform that enables the quality-driven composition of Web services. In AgFlow, the QoS of Web services is evaluated by means of an extensible multidimensional QoS model, and the selection of component services is performed in such a way as to optimize the composite service's QoS given a set of user requirements (i.e., constraints on QoS) and a set of candidate component services. Furthermore, AgFlow adapts to changes that occur during the execution of a composite service, by revising the execution plan in order to conform the user's constraints on QoS. The salient features of AgFlow are:

1. A *multidimensional QoS model* which captures nonfunctional properties that are inherent to Web services in general, e.g., availability and reputation. This model defines a number of QoS properties and methods for attaching values for these properties in the context of both stand-alone and composite Web services.
2. Two alternative *QoS driven service selection approaches* for composite service execution: one based on local optimization and the other on global planning. The local optimization approach performs optimal service selection for each individual task in a composite service without considering QoS constraints spanning multiple tasks and without necessarily leading to optimal overall QoS. The global planning approach on the other hand considers QoS constraints and preferences assigned to a composite service as a whole rather than to individual tasks, and uses integer programming to compute optimal plans for composite service executions.
3. An *adaptive execution engine* which reacts to changes occurring during the execution of a composite service (e.g., component services that become unavailable or change their predicted QoS), by replanning the execution in order to ensure that the QoS is optimal given the available information about the component services.

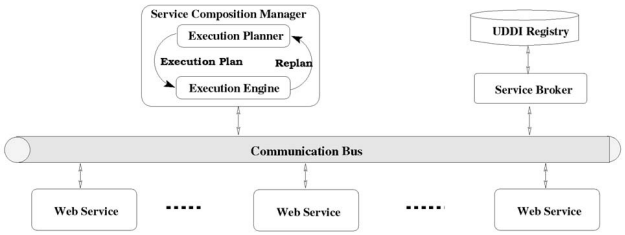


Fig. 1. AgFlow's architecture.

It should be noted that the proposed service composition and selection approaches can be applied to other distributed computing paradigms than Web services. However, the proposal has been designed to address certain aspects which appear more prominently in Web services (or more broadly in service-oriented architectures). First, the issue of optimizing the design and execution of composite services is more prominent for service-oriented architectures than for conventional middleware where composition technologies have also been developed but have not gained wide acceptance [1]. Second and foremost, in service-oriented architectures the need to capture QoS properties is more pronounced than in conventional middleware, since services are developed independently of their client applications, typically by different organizational units, and they operate in an open environment (the Web) where competition and differentiation are major factors. In any case, it is possible to leverage our approach to compose distributed components developed using conventional middleware, by exposing them as Web services and adding the QoS information required for the composition.

The remainder of the paper is organized as follows: Section 2 provides an overview of the AgFlow system and basic concepts of the underlying service composition model. Section 3 describes the proposed service quality model. In Section 3, two alternative service selection approaches are presented and compared. An implementation of the service composition and service selection model is then presented in Section 4, and experimental results are documented in Section 5. Finally, Section 6 discusses related work and Section 7 concludes the paper.

2 Preliminaries

In this section, the AgFlow system architecture is presented and some basic concepts and definitions are explained.

2.1 System Architecture

The architectural diagram of the AgFlow system is presented in Fig. 1. There are three distinct components in

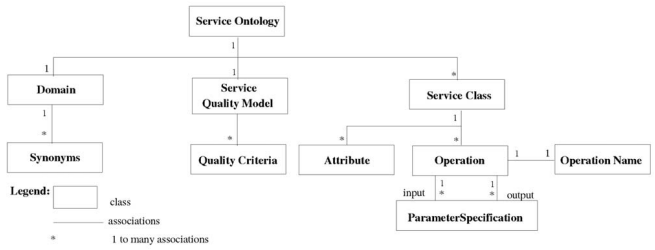


Fig. 2. UML class diagram for service ontologies.

the AgFlow system, namely, *Web services*, *service broker*, and *service composition manager*.

The service broker allows providers to register their service descriptions in an UDDI registry. A service description contains metadata that describe, among others, the capabilities and QoS of a Web service. The service composition manager is made up of an execution planner and an execution engine. When an instance of a composite service is initiated, the execution planner contacts the service broker to search for candidate component services, and, based on the candidate services retrieved, it generates an execution plan, i.e., an assignment of component services to the tasks in the schema of the composite service. Based on the execution plan, the adaptive execution engine then orchestrates the component services to execute the instance of the composite service. At runtime, the execution engine also monitors the component services. When the current status of the execution violates the execution plan, the execution engine triggers the execution planner to revise the current plan and resumes the execution using the new plan.

2.2 Service Ontologies and Service Description

A service ontology (see Fig. 2) consists of a common language agreed by a community (e.g., automobile industry). It defines a terminology that is used by all participants in that community. Within a community, service providers describe their services using the terms of the community's ontology, while service requesters use the terms of the ontology to formulate queries over the registry(ies) of the community.

Concretely, a service ontology specifies a *domain* (e.g., `Automobile`, `Healthcare`, `Insurance`), a set of *synonyms*, used to facilitate flexible search for the domain (e.g., the domain `Automobile` may have synonyms like `Car`) and a set of *service classes* that are used to define the properties of services. A service class is further specified by its *attributes* and *operations*. For example, the attributes of a service class may include access information such as URL. Each operation is specified by its name and signature (i.e., inputs and outputs). A service ontology also specifies a *service quality model* that is used to describe non-functional properties of services, e.g., execution duration of an operation. The service quality model consists of a set of *quality dimensions* (or *criteria*). For each quality criterion, there are three basic elements: its definition, the service elements (e.g., services or operations) to which it is related, and how to compute or measure the value of the criteria. The service quality model is presented in Section 3.

There are two important elements in a service description:

1. **Service ontology and service class**. A Web service provider needs to specify which service ontology is used and which service classes are supported. For example, a travel service provider may specify that it uses the service ontology `Trip-planning` and support the service class `FlightTicketBooking`. The service ontology specifies the concepts and terminology used in the service description, and service class describes the capabilities (e.g., operations) of Web services and how to access them. Therefore, published services have the same attributes as the service class in the service ontology to which they belong, plus additional attributes to describe the provided capabilities.
2. **Service Level Agreements (SLA)**. An SLA defines the terms and conditions of service quality that a Web service delivers to service requesters. The major constituent of an SLA is the QoS information. There are a number of criteria (e.g., execution duration, availability) that contribute to a Web service's QoS in a SLA as discussed later in the paper. Some Web service providers publish QoS information in SLAs. Other Web service providers may not publish their QoS information in their service descriptions for confidential reasons. In this case, service providers need to provide interfaces that only authorized requesters can use to query the QoS information.

2.3 Composite Service Specifications

A composite service is specified as a collection of *generic service tasks* described in terms of service ontologies and combined according to a set of control-flow and data-flow dependencies. AgFlow uses statecharts [17] to represent these dependencies. This choice is motivated by several reasons. First, statecharts possess a formal semantics, which is essential for analyzing composite service specifications. Second, statecharts are a well-known and well-supported behavior modeling notation, following their integration into the Unified Modeling Language (UML). Finally, statecharts offer most of the control-flow constructs found in existing process modeling languages (branching, concurrent threads, structured loops) and they have been shown to be suitable for expressing typical control-flow dependencies [12]. Hence, it is possible to adapt the QoS-aware service selection mechanisms developed using statecharts to fit other alternative languages.

A statechart is made up of states and transitions. Transitions of a statechart are labeled with events, conditions, and operations. States can be *basic* or *compound*. Basic states (also called *tasks* in the sequel) are labeled with an operation name of a given service class (which is defined in a service ontology). Intuitively, when the basic state is entered, the operation that labels this state is invoked over one of the services belonging to the designated service class.

Compound states on the other hand provide means to structure the statechart into regions, and to express concurrent execution of regions. Compound states come in two flavors: OR-states and AND-states. An OR-state contains a single region whereas an AND-state contains several regions (separated by dashed lines) which are intended to be executed concurrently. Accordingly,

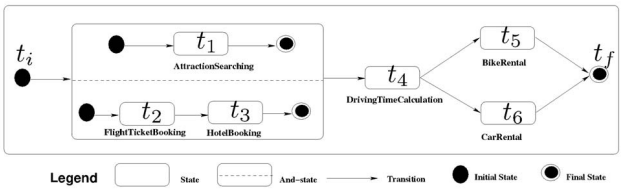


Fig. 3. Statechart of a "Travel Planner" composite service.

OR-states are used as a grouping 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.

A simplified statechart W specifying a "Travel Planner" composite Web service is depicted in Fig. 3. In this example, a search for attractions is done in parallel with a flight and an accommodation booking. After the searching and booking operations complete, 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, such as t_4, they denote a conditional branching and the transitions should be labeled with disjoint conditions.) In this example, it is assumed that the attractions do not change from one day to the next, so there is no need to know the flight arrival date and time to search for attractions.

Instance variables can be used in a composite service specification to capture the data manipulation perspective. Specifically, they can be used to express branching conditions and to provide (store) input (output) parameters to (from) the service operations invoked by the tasks. They can also be manipulated in the actions attached to the transitions of the statechart. Note that the data perspective is not relevant for the purposes of AgFlow since the methods for allocating services to tasks only need to consider the control-flow dependencies and the QoS of the component services. Data-flow is relevant for the execution of composite services by platforms such as Self-Serv [5].

2.4 Execution Paths and Plans

In this section, we define two concepts used in the remainder of the paper: *execution path* and *execution plan*. To simplify the discussion, 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. Details of the unfolding process are given in Section 4.2.3.

Definition 1: (Execution path). *An execution path of a statechart is a sequence of states* $[\mathrm{t}_1, \mathrm{t}_2, ..\mathrm{t}_n]$*, such that* t_1 *is the initial state,* t_n *is the final state, and for every state* t_i $(1 < i < n)$*:*

- t_i *is a direct successor of one of the states in* $[\mathrm{t}_1, \ldots, \mathrm{t}_{i-1}]$.
- t_i *is not a direct successor of any of the states in* $[\mathrm{t}_{i+1}, \ldots, \mathrm{t}_n]$.
- *There is no state* t_j *in* $[\mathrm{t}_1, \ldots, \mathrm{t}_{i-1}]$ *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 belongs to the set* $\{\mathrm{t}_1, \ldots, \mathrm{t}_{i-1}, \mathrm{t}_{i+1}, \ldots, \mathrm{t}_n\}$*. In other words, when an AND-state is entered, all the concurrent branches of this AND-state are executed.*

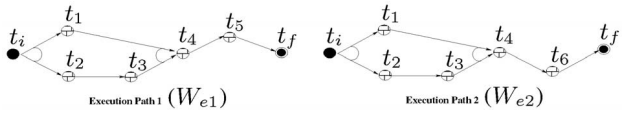


Fig. 4. DAG representation of the execution paths of the statechart of Fig. 3.

This definition relies on the concept of a *direct successor* of a state. Roughly stated, a basic state t_b in a statechart is a direct successor of another basic state t_a if there is a sequence of adjacent transitions[1] 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).

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

Definition 2: (DAG representation of an execution path). *Given an execution path* $[\mathrm{t}_1, \mathrm{t}_2, ..\mathrm{t}_n]$ *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* $\{\mathrm{t}_1, \mathrm{t}_2, ..\mathrm{t}_n\}$.
- *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 contains conditional branchings, it has multiple execution paths. Each execution path represents a sequence of tasks to complete a composite service execution. Fig. 4 gives an example of 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 the execution path W_{e1}, task t_5 is executed after task t_4, while in the execution path W_{e2}, task t_6 is executed after task t_4.

As stated earlier, each basic state of a statechart describing a composite service is labeled with an invocation to an operation provided by a given service class. Actual Web services belonging to the required service classes are selected during the execution of the composite service. Hence, it is possible to execute an execution path of a statechart in different ways by allocating different Web services to the basic states in the path. The concept of execution plan defined below captures the various ways of executing a given execution path.

1. Two transitions are adjacent if the target state of one is the source state of the other.

Definition 3: (Execution plan). *A set of pairs* $p = \{< t_1, s_1 > , < t_2, s_2 >, \ldots, < t_n, s_n >\}$ *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 pair* $< t_i, s_i >$ *in p, service* s_i *belongs to the service class associated with task* t_i. *In other words, service* s_i *provides the operation required by task* t_i.

It should be noted that, according to this definition, each task can be executed by a number of alternative services, but it is not possible for a service to execute a combination of tasks in a "single shot." To express that a combination of tasks can be executed by a single service, these tasks need to be assembled into a single one.

3 Web Service Quality Model

In the composition model presented in the previous section, Web services will typically be grouped together in a single community. To differentiate the members of a community during service selection, their nonfunctional properties need to be considered. For this purpose, we adopt a Web service quality model based on a set of quality criteria (i.e., nonfunctional properties) that are applicable to all Web services, for example, their pricing and reputation. 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 nonfunctional service characteristics such as those proposed by O'Sullivan et al. [28].

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 we 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**. Given an operation op of a service s, the execution price $q_{pr}(s, op)$ is the fee that a service requester has to pay for invoking the operation op. Web service providers either advertise the execution price of their operations, or provide means for potential requesters to inquire about it.
2. **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 inquire 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.

3. **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 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, end users are given a range to rank Web services. For example, in Amazon.com, the range is $[0, 5]$. Detail discussion on moderation of reputation rating is out of the scope of this paper.

4. **Successful execution rate**. The successful execution rate $q_{rat}(s)$ of a service s is the probability that a request is correctly responded (i.e., the operation is completed and a message indicating that the execution has been successfully completed is received by service requestor) within the maximum expected time frame indicated in the Web service description. The successful execution rate (or *success rate* for short) is a 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 success rate is computed from data of past invocations using the expression $q_{rat}(s) = N_c(s)/K$, where $N_c(s)$ is the number of times that the service s has been successfully completed within the maximum expected time frame, and K is the total number of invocations.
5. **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

TABLE 1
Aggregation Functions for Computing the QoS of Execution Plans

Criteria	Aggregation function
Price	$q_{pr}(p) = \sum_{i=1}^{N} q_{pr}(s_i, op(t_i))$
Duration	$q_{du}(p) = CPA(p, q_{du})$
Reputation	$q_{rep}(p) = \frac{1}{N}\sum_{i=1}^{N} q_{rep}(s_i)$
Success rate	$q_{rat}(p) = \Pi_{i=1}^{N}(q_{rat}(s_i)^{z_i})$
Availability	$q_{av}(p) = \Pi_{i=1}^{N}(q_{av}(s_i)^{z_i})$

Web services send notifications to the system about their running states (i.e., available, unavailable). Alternatively, if a given Web service does not support notification of its running state, the composite service manager can probe the service at certain intervals (determined by the system administrator) or obtain uptime information from a monitoring service. This aspect however is outside the scope of this paper (see for example reference [22]).

Given the above considerations, the quality vector of an operation op of a service s is defined as

$$q(s, op) = (q_{pr}(s, op), q_{du}(s, op), q_{av}(s), q_{rat}(s), q_{rep}(s)).$$

Note that the method for computing the value of the quality criteria is not unique. Other computation methods can be designed to fit the needs of specific applications. The service selection approaches presented in Section 4 are independent of these computation methods.

3.2 Quality Criteria for Composite Services

The quality criteria defined above in the context of elementary Web services, are also used 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

$$p = \{< t_1, s_1 >, < t_2, s_2 >, \ldots, < t_n, s_n >\}.$$

A brief explanation of each criterion's aggregation function follows:

1. **Execution price**: The execution price $q_{pr}(p)$ of an execution plan p is a sum of the execution prices of the operations invoked over the services that participate in p. In the equation for the execution price given in Table 1, $op(t_i)$ denotes the operation invoked by task t_i.
2. **Execution duration**: The execution duration $q_{du}(p)$ of an execution plan p is computed using the Critical Path Algorithm (CPA) [32]. Specifically, the CPA is applied to the the execution path W_e 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 labeling its nodes. In the case at hand, a node corresponds to a task t in W_e, and its weight is the execution duration of the service operation invoked by t, that is: $q_{du}(sv_p(t), op(t))$, where $sv_p(t)$ is the service assigned to task t in plan p, and $op(t)$ denotes the operation invoked by task t. A task that belongs to the critical path is called a *critical task*, while a service assigned to a task that belongs to the critical path is called a *critical service*.

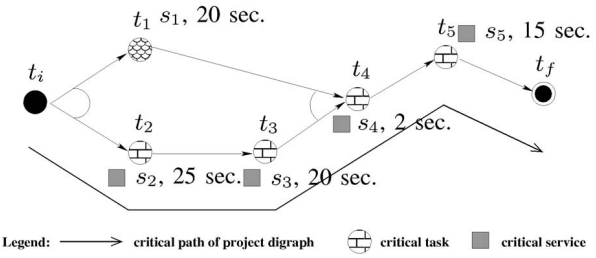


Fig. 5. Example of a critical path.

Fig. 5 provides an example of a critical path. This figure depicts an execution path as a project digraph, and an associated execution plan p, where

$$p = \{< t_1, s_1 >, < t_2, s_2 >, < t_3, s_3 >, < t_4, s_4 >, < t_5, s_5 >\}.$$

For each service, its execution duration is shown next to it. There are two *project paths* in this project digraph, where project path 1 is $< t_1, t_4, t_5 >$ and project path 2 is $< t_2, t_3, t_4, t_5 >$. The execution time of project path 1 (project path 2) is 37 seconds (62 seconds). The critical path is therefore path 2 and the execution duration of the plan is 62 seconds. Task t_2, t_3, t_4 and t_5 are critical tasks while services s_2, s_3, s_4 and s_5 are critical services.

3. **Reputation**: The reputation $q_{rep}(p)$ of an execution plan p is the average of the reputations of the services that participate in p.
4. **Successful execution rate**: The successful execution rate $q_{rat}(p)$ of an execution plan p is the product of the factors $q_{rat}(s_i)^{z_i}$, where 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 $q_{rat}(s_i)^{z_i} = 1$. Here, we assume that when the execution of noncritical services is not successful, the task can be re-executed without delaying the whole composite service execution. Hence, the success rates of noncritical services do not affect the overall plan's success rate.
5. **Availability**: The availability $q_{av}(p)$ of an execution plan p is given by the product of the factors $q_{av}(s_i)^{z_i}$, where $q_{av}(s_i)$ is the availability of service s_i and z_i indicates whether the service is a critical service or not. Again, we assume that, when the noncritical service is unavailable, a service can be reselected without delaying the whole composite service execution; hence, the availability of service s_i will not directly affect the overall plan's availability.

Given these functions, the quality vector of a composite service's execution plan is defined as

$$q(p) = (q_{pr}(p), q_{du}(p), q_{av}(p), q_{rat}(p), q_{rep}(p)).$$

4 QoS-Driven Service Selection for Web Service Composition

In this section, we present two service selection approaches, namely, *local optimization* and *global planning*.

4.1 Service Selection by Local Optimization

In this approach, the selection of the Web service that will execute a given task of a composite service specification is done at the last possible moment and without taking into account the other tasks involved in the composite service. When a task actually needs to be executed, the system collects information about the QoS of each of the Web services that can execute this task (namely the *candidate Web services* for this task). After collecting this QoS information, a *quality vector* is computed for each of the candidate Web services, and, based on these quality vectors, the system selects one of the candidate Web services by applying a Multiple Criteria Decision Making (MCDM) [7] technique. This selection process is based on the *weight* assigned by the user to each criterion, and a set of user-defined constraints expressed using a simple expression language. Examples of constraints that can be expressed include duration constraints and price constraints. However, constraints can only be expressed on individual tasks and not on combinations of tasks. In other words, it is not possible to express the fact that the sum of the durations for two or more tasks should not exceed a given threshold.

To illustrate the local optimization approach, we use the five quality dimensions discussed earlier, but other quality dimensions can be used instead without any fundamental changes. The dimensions are numbered from 1 to 5, with `1 = price`, `2 = duration`, `3 = availability`, `4 = success rate`, and `5 = reputation`. Given a task t_j in a composite service, there is a set of candidate Web services $S_j = \{s_{1j}, s_{2j}, \ldots, s_{nj}\}$ that can be used to execute this task. By merging the quality vectors of all these candidate Web services, a matrix $\mathbf{Q} = \big(Q_{i,j}; 1 \le i \le n, 1 \le j \le 5\big)$ is built, in which each row Q_j corresponds to a Web service s_{ij} while each column corresponds to a quality dimension.

A Simple Additive Weighting (SAW) [7] technique is used to select an optimal Web service. There are two phases in applying SAW:

1. **Scaling Phase**. Some of the criteria could be negative, i.e., the higher the value, the lower the quality. This includes criteria such as execution time and execution price. Other criteria are positive, i.e., the higher the value, the higher the quality. For negative criteria, values are scaled according to (1). For positive criteria, values are scaled according to (2).

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

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

In the above equations, Q_j^{max} is the maximal value of a quality criteria in matrix $\mathbf{Q}$, i.e., $Q_j^{max} = Max(Q_{i,j}), 1 \le i \le n$. While Q_j^{min} is the minimal value of a quality criteria in matrix $\mathbf{Q}$, i.e., $Q_j^{min} = Min(Q_{i,j}), 1 \le i \le n$. By applying these two equations on $\mathbf{Q}$, we obtain a matrix

$$V = \big(V_{i,j}; 1 \le i \le n, 1 \le j \le 5\big),$$

in which each row V_j corresponds to a Web service s_{ij} while each column corresponds to a quality dimension.

As an example, assume that there are eight Web services in S_5 for task t_5 and that their values for service reputation are given by the vector $Q_5 = (7.5, 8.4, 9, 8.3, 8.7, 9.1, 9.4, 9.2, 9.5)$. Since reputation is a positive criteria, (2) is used for scaling and, thus, $Q_5^{max} = 9.5$, $Q_5^{min} = 7.5$, and

$$V_5 = (0, 0.55, 0.45, 0.75, 0.4, 0.6, 0.8, 0.95, 1).$$

2. **Weighting Phase**. The following formula is used to compute the overall quality score for each Web service:

$$Score(s_i) = \sum_{j=1}^{5} (V_{i,j} * W_j), \quad (3)$$

where $W_j \in [0, 1]$ and $\sum_{j=1}^{5} W_j = 1$. W_j represents the weight of criterion j. End users express their preferences regarding QoS by providing values for the weights W_j.

For a given task, the system will choose the Web service which satisfies all the user constraints for that task and which has the maximal score. If there are several services with maximal score, one of them is selected randomly. If no service satisfies the user constraints for a given task, an execution exception will be raised and the system will propose the user to relax these constraints.

4.2 Service Selection by Global Planning

In the local optimization approach, service selection is done for each task individually. Although service selection is locally optimized, the global quality of the execution may be suboptimal. For example, if two tasks A and B are executed in parallel and need to synchronize upon completion, then it is not worth optimizing the duration of A, if it is known that B takes considerably more time to execute. Instead, it is preferable to optimize (for example) the price of A, while optimizing the duration of B. Furthermore, as explained above, when applying local optimization it is not possible to enforce intertask constraints over the composite service execution such as: "the total price of the composite service execution should be at most $500." In this section, we present a global planning approach for Web services selection which overcomes these limitations. We first present a *naive approach* for global planning, and then present a novel integer programming approach that avoids some obvious computational problems associated with the naive approach.

4.2.1 Optimal Execution Plan of an Execution Path

For each task t_j in an execution path, there is a set of candidate services $S_j = \{s_{1j}, s_{2j}, \ldots, s_{nj}\}$ that can execute task t_j. Assigning a candidate service s_{ij} to each task t_j in an execution path leads to a possible execution plan. In the global planning approach, all possible plans associated to a given execution path are generated (at least conceptually speaking) and the one which maximizes the user's preferences while satisfying the imposed constraints is then selected. The selection of an execution plan relies on the application of a MCDM technique on the quality matrix $\mathbf{Q} = \big(Q_{i,j}; 1 \leq i \leq n, 1 \leq j \leq 5\big)$ of the execution path. In this matrix, a row corresponds to the quality vector of a possible execution plan for the execution path.

As in the local selection approach, a SAW technique is used to select an optimal execution plan. The two phases of applying SAW are:

1. **Scaling Phase**. As in the previous section, we first scale the values of each quality criterion. For negative criteria, values are scaled according to (1). For positive criteria, values are scaled according to (2). Note that we can compute the value of Q_j^{max} and Q_j^{min} in these equations without generating all possible execution plans. For example, in order to compute the maximum execution price (i.e., Q_{pr}^{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_{pr}^{max}. In order to compute the minimum execution duration (i.e., Q_{du}^{min}) of all the execution plans, we select the service with the 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 thus polynomial. After the scaling phase, we obtain the matrix $V = \big(V_{i,j}; 1 \leq i \leq n, 1 \leq j \leq 5\big)$.
2. **Weighting Phase**. The following formula is used to compute the overall quality score for each execution plan:

$$Score(p_i) = \sum_{j=1}^{5} (V_{i,j} * W_j), \tag{4}$$

where $W_j \in [0,1]$ and $\sum_{j=1}^{5} W_j = 1$. W_j represents the weight of each criterion. End users can give their preferences 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 is more than one execution plan which has the same maximal value of $Score(p_i)$, then an execution plan will be selected from them randomly.

4.2.2 Handling Multiple Execution Paths

Assume that a statechart has multiple execution paths. For each of these paths, an optimal execution plan can be selected using the method described above. Since each of the selected plans only covers a subset of the statechart, the global planner needs to aggregate these "partial" execution plans into an overall execution plan. For example, for the `Travel Planner` statechart W (see Fig. 3), there are two execution paths W_{e1} and W_{e2}, each of which has its own optimal execution plan, say p_1 and p_2. Neither p_1 nor p_2 covers all tasks in W, so they need to be merged somehow.

Assume that statechart W has n tasks (i.e., $t_1, t_2, \ldots, t_k$) and m execution paths (i.e., $W_{e1}, W_{e2}, \ldots, W_{em}$). For each execution path, the global planner selects an optimal execution plan. Consequently, we obtain m optimal execution plans (i.e., $p_1, p_2, \ldots, p_m$) 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 the `Travel Planner` example, task t_5 (i.e., `BikeRental`) only belongs to execution path W_{e2}. In this case, W_{e2}'s execution plan p_2 is used to determine the service that will execute t_5. This fact is denoted by $<t_5, p_2>$.
2. Given a task t_i, if t_i belongs to more than one execution paths (e.g., $W_{ej}, W_{ej+1}, \ldots, W_{em}$), then there is a set of execution plans (i.e., $p_j, p_{j+1}, \ldots, p_m$) that can be used to execute W_{si}. Hence, the global planner needs to select one of these execution plans. The selection can be done by identifying the *hot path* for task t_i.

   For example, the hot path of a task t_i can be defined as the execution path that has been most frequently used to execute the task t_i in past instances of the composite service. For example, in the `Travel Planner` statechart, task t_4 (`Driving-TimeCalcultation`) belongs to both execution paths W_{e1} and W_{e2}. Assume that the composite service has been executed 25 times, 20 of which have followed execution path W_{e1}, while the other five have followed W_{e2}. Since the execution path W_{e1} is used more frequently to execute task t_4 (i.e., W_{e1} is the hot path for t_2), W_{e1}'s optimal execution plan p_1 is used to determine the service that will execute t_4. This is denoted by $<t_4, p_1>$. The system keeps the execution traces of the composite service. These traces allow the global planner to identify the hot path for each task. In the absence of (enough) traces, a human expert must indicate the hot path. It should be noted that hot path can also be defined based on a QoS criterion or a user defined utility function involving multiple QoS criteria. For example, the hot path can be defined as the path which has the highest execution price.

4.2.3 Unfolding Cyclic Statecharts

Hitherto, we have assumed that the statecharts are acyclic. If a statechart contains cycles, these need to be "unfolded" so that the resulting statechart has a finite number of execution paths. The method used to unfold a statechart is to examine the logs of past executions to determine the maximum number of times that each cycle is taken. The states appearing between the beginning and end of a cycle are then cloned as many times as the transition causing the

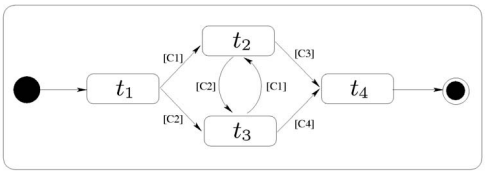


Fig. 6. "Unfoldable" statechart.

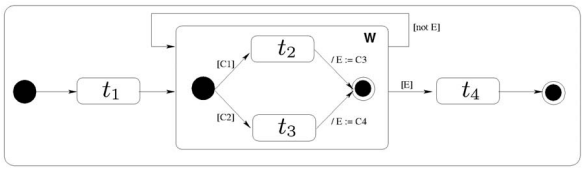


Fig. 7. Foldable statechart equivalent to that in Fig. 6.

cycle is taken.[2] This unfolding method works if the beginning and end of each cycle in the statechart can be identified. This is not the case for example in Fig. 6. In this example, it is not clear which is the first state and which is the last state in the cycle (is it t_2 or t_3?). However, an equivalent statechart which can be unfolded using the above method is shown in Fig. 7.

It can be proven that any arbitrary statechart can be transformed into an equivalent statechart in which the cycles do not cross the boundaries of conditional branches (as illustrated above). This proof is similar to that of the theorem stating that any program written using "goto" statements, can be transformed into an equivalent program which only uses structured loops (i.e., "while" statements). Oulsman [29], for example, describes a method for performing this kind of transformation with minimal number of auxiliary Boolean variables. Kiepuszewski et al. [19] points out to the fact that the algorithms for transforming "arbitrary cycles" into "structured" ones do not always apply in the presence of parallel branches. However, they do apply for statecharts since in statecharts it is not possible to have arbitrary transitions going from a parallel branch to another such as those put forward by reference [19].

Given the statechart with structured cycles of Fig. 7, the unfolding process proceeds by determining the maximum number of times that the transition causing the cycle is taken (i.e., the transition labeled with condition "not E"). If for example this transition has been taken a maximum of two times, then three copies of the compound state W will appear in the resulting acyclic statechart. From this acyclic statechart, we can then generate all possible execution paths.

4.2.4 Integer Programming Solution

The global planning approach by exhaustive searching outlined above requires the generation of all possible execution plans. Assuming that there are n tasks and m candidate Web services for each task, the total number of execution plans is m^n, making this approach impractical. Accordingly, we propose a method based on Integer Programming (IP) [18] for selecting an optimal execution plan without generating all possible execution plans.

There are three inputs in an IP problem: a set of *variables*, an *objective function*, and a set of *constraints*, where both the objective function and the constraints must be linear. IP attempts to maximize or minimize the value of the objective function by adjusting the values of the variables while enforcing the constraints. The output of an IP problem is the maximum (or minimum) value of the objective function and the values of variables at this maximum (minimum).

2. The idea of cloning the states in a loop for the purpose of transforming a cyclic statechart into an acyclic one has been proposed by Gillmann et al. [15].

The problem of selecting an optimal execution plan is mapped into an IP problem as follows. First, for every Web service s_{ij} that can be used to execute a task t_j, we include in the IP problem an integer variable y_{ij}, such that by convention y_{ij} is 1 if service s_{ij} is selected for executing task t_j, 0 otherwise. We also introduce a set of integer variables x_j, such that x_j denotes the expected start time of task t_j (if t_j is executed at all). This set of variables are used to express the constraints on the execution duration.

Next, since we rely on MCDM and SAW to determine the desirability of an execution plan, we use the following objective function, which is based on (1), (2), and (4):

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

where $W_l \in [0,1]$ and $\sum_{j=1}^{5} W_j = 1$. W_l is the weight assigned to the quality criteria. The remainder of this subsection describes the constraints of the IP problem.

Allocation constraint. For each task t_j, there is a set of Web services S_j that can be assigned (allocated) to it. However, for each task t_j, we should only select one Web service to execute this task. Given that y_{ij} denotes the selection of Web service s_{ij} to execute task t_j, the following constraint must be satisfied:

$$\sum_{i \in S_j} y_{ij} = 1,\ \forall j \in A, \tag{6}$$

where A is the set of tasks in the statechart. For example, assume that there are 100 potential Web services that can execute task j. Since only one of them will be selected to execute task j, we have that $\sum_{i=1}^{100} y_{ij} = 1$.

Constraints on Execution Duration, Price, and Reputation. Let x_j denote the expected start time of task t_j, p_{ij} denote the execution duration of task t_j when assigned to service s_{ij}, and p_j denote the expected duration of task t_j knowing which service has been assigned to it. Also, let $t_j \rightarrow t_k$ denote the fact that task t_k is a direct successor of task t_j. We have the following constraints:

$$\sum_{i \in S_j} p_{ij}\ y_{ij} = p_j, \forall j \in A, \tag{7}$$

$$x_k - (p_j + x_j) \geq 0, \forall t_j \rightarrow t_k, j, k \in A, \tag{8}$$

$$q_{du} - (x_j + p_j) \geq 0, \forall j \in A. \tag{9}$$

Constraint (7) indicates that the execution duration of a given task t_j must be the execution duration of one of the Web services in A since one and only one of these services will be selected to execute task t_j. Constraint (8) indicates

that if task t_k is a direct successor of task t_j, then the execution of t_k must start after task t_j has been completed. Constraint (9) indicates that the execution of a composite service plan is completed only when all the tasks in the plan are completed.

Now, let z_{ij} be an integer variable that has value 1 or 0: 1 indicates that Web service s_{ij} is a critical service and 0 indicates otherwise. The relationship between the duration of an execution plan and the duration of the critical services of the plan is captured by the following equation.

$$q_{du} = \sum_{j \in A} \sum_{i \in S_j} p_{ij} z_{ij}. \tag{10}$$

Similarly, assuming that variable c_{ij} represents the execution price of Web service s_{ij}, we impose the following constraint to capture the total execution price of a composite service:

$$q_{pr} = \sum_{j \in A} \sum_{i \in S_j} c_{ij} y_{ij}. \tag{11}$$

An alternative constraint (11) is the following:

$$\sum_{j \in A} \sum_{i \in S_j} c_{ij} y_{ij} \leq B, B > 0, \tag{12}$$

where B is the budget set by the user. This constraint indicates that the execution price of the composite service should not be greater than B.

Finally, assuming that variable r_{ij} represents the reputation of Web service s_{ij}, we impose the following constraint to capture the overall reputation of an execution plan:

$$q_{rep} = \sum_{j \in A} \sum_{i \in S_j} r_{ij} y_{ij}. \tag{13}$$

Note that other criteria with a simple linear aggregation function could be integrated in the same way as the reputation.

Constraints on Success Rate and Availability. Among the criteria used to select Web services, the availability and the success rate are associated with nonlinear aggregation functions (see Table 1). In order to capture them in the IP problem, we need to transform nonlinear to linear function. Assume that variable a_{ij} represents the success rate of Web service s_{ij}. Since z_{ij} indicates whether Web service s_{ij} is a critical service or not, the success rate of the execution plan is:

$$q_{rat} = \Pi_{j \in A} \left(\Pi_{i \in S_j} a_{ij}^{z_{ij}} \right).$$

By applying the logarithm function ln, we obtain:

$$ln(q_{rat}) = \sum_{j \in A} ln \left(\Pi_{i \in S_j} a_{ij}^{z_{ij}} \right).$$

Since for each task t_j, $\sum_{i \in Sj} z_{ij} = 1$ and $z_{ij} = 0$ or 1, we have that:

$$ln(q_{rat}) = \sum_{j \in A} \left(\sum_{i \in S_j} \big(ln(a_{ij}) z_{ij} \big) \right).$$

Let $q'_{rat} = ln(q_{rat})$, we introduce the following constraint into the IP problem in order to capture the success rate criterion:

$$q'_{rat} = \sum_{j \in A} \sum_{i \in S_j} \big(ln(a_{ij}) z_{ij} \big). \tag{14}$$

It should be noted that by transforming nonlinear functions to linear functions, we actually change the objective function if the weight of the other criteria is not equal to zero. However, the constraints on the success rate still hold. Similarly, assume that b_{ij} represents the availability of the Web service s_{ij}. We introduce the following constraint:

$$q'_{av} = \sum_{j \in A} \sum_{i \in S_j} \big(ln(b_{ij}) z_{ij} \big), \tag{15}$$

where $q'_{av} = ln(q_{av})$.

Constraints on the Uncertainty of Execution Duration. Hitherto, we have assumed that the execution duration p_{ij} of a Web service is deterministic. In reality, the execution duration p_{ij} of a Web service s_{ij} is uncertain, in the sense that some deviations between the actual duration and p_{ij} are likely to occur. In order to capture this uncertainty, we assume that p_{ij} is a random variable that follows a normal distribution with mean μ_{ij} and standard deviation σ_{ij}, which can be computed from the history of past executions.

Since $q_{du} = \sum_{i \in A} \sum_{i \in S_j} p_{ij} z_{ij}$ is a linear combination of random variables with normal distribution, q_{du} itself is a random variable with normal distribution [35] and its deviation σ_{du} is $\sigma_{du}^2 = \sum_{j \in A} \sum_{i \in S_j} \sigma_{ij}^2 z_{ij}$. With this equation at hand, it is possible to extend the objective function in order to incorporate the deviation from the expected execution duration as an optimization criteria. The extended objective function is:

$$Max \left(\sum_{l=0}^{2} \left(\frac{Q_l^{max} - Q_{i,l}}{Q_l^{max} - Q_l^{min}} * W_l \right) + \sum_{l=3}^{5} \left(\frac{Q_{i,l} - Q_l^{min}}{Q_l^{max} - Q_l^{min}} * W_l \right) \right), \tag{16}$$

where $Q_0 = \sigma_{du}^2$ and $W_0 \in [0, 1]$ are the values assigned to the criterion "deviation from expected execution duration." Given the above variables, objective function, and constraints, an IP solver is able to compute the values of y_{ij} corresponding to an optimal execution plan. As a side effect, the IP solver will also provide a tentative schedule for the tasks in the statechart by assigning values to the variables x_j. This assignment of variables can be used by the execution engine, but they are not strictly necessary since the execution engine can schedule the tasks using its own scheduling mechanisms.

Note that the proposed method for translating the problem of selecting an optimal execution plan into an IP problem is generic and, although it has been illustrated with the five criteria introduced in Section 3 (plus the "duration deviation" criterion), other criteria can be accommodated.

4.2.5 Replanning the Execution of Composite Services

When using the global planning approach, an execution plan is built at the beginning of the execution of the composite service. Once the execution has started, several contingencies may occur, e.g., a component service becomes unavailable or the QoS of one of the component services

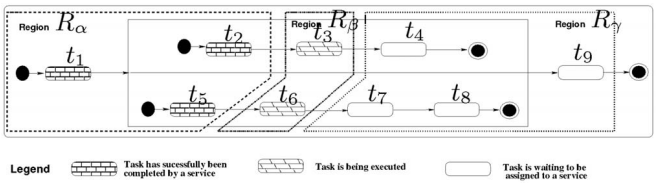


Fig. 8. Partition of a composite service into regions for replanning.

changes significantly. In these situations, a replanning procedure may be triggered in order to ensure that the QoS of the composite service execution remains optimal. Note that it is conceivable that the computation time spent on replanning is more than the duration of the remaining tasks. Even in this situation, however, replanning may be beneficial as it may improve the values of the other QoS criteria, e.g., lower execution price. In this section, we discuss how this replanning procedure is conducted.

Consider a composite service consisting of a set of tasks $T = \{t_1, t_2, \ldots, t_n\}$. Based on the execution status, T can be partitioned into four regions:

1. region R_α containing tasks that have been completed;
2. region R_β containing tasks that are being executed;
3. region R_γ containing tasks that have not yet started; and
4. region R_δ containing tasks which belong to a conditional branch that is no longer accessible from any of the tasks in R_β (i.e., tasks that have not been executed but will not be executed).

An example of a partitioning of a composite service is given in Fig. 8. Note that this composite service has no conditional branches, so $R_\delta = \emptyset$.

When using the global planning approach, the composite service execution needs to be replanned in the following cases:

1. One or several exceptions have occurred during the execution of component services in region R_β. Execution exceptions include 1) component services failing to execute tasks and 2) a service not being able to attain its expected QoS.
2. Changes have been reported in the expected QoS of candidate services for tasks in region R_γ. Changes include: 1) services selected during the global planning becoming unavailable or changing their QoS properties, 2) new candidate services offering better QoS than existing services appearing, and 3) existing services raising their advertised QoS.

As discussed in Section 4.2.4, there are three inputs in an IP problem: a set of *variables*, an *objective function*, and a set of *constraints*. When an execution plan is revised at runtime, the *variables* and the *objective function* are not modified, but instead, a number of additional constraints are introduced in order to capture the current execution status (i.e., the fact that some of the tasks have already completed). Specifically, the actual QoS delivered by the tasks in region R_α, and the fact that the tasks in region R_α and R_β cannot be reassigned to new candidate services, need to be encoded in the constraints of the IP problem. For example, assuming that task t_1 has been successfully completed by service s_{41}, that the actual execution duration was 20 seconds, and that the actual execution cost was $10, the following constraints will be added to the IP problem by the global planner:

$$y_{41} = 1, \tag{17}$$

$$a_{41} = 1, b_{41} = 1, \tag{18}$$

$$p_{41} = 20, c_{41} = 10. \tag{19}$$

Constraint (17) indicates that service s_{41} executed task t_1. Constraint (18) indicates that service s_{41} was available and reliable when invoked. Finally, constraint (19) encodes the actual execution duration and cost of t_1. The introduction of these constraints force the IP solver to select an execution plan for region R_γ which takes into account what has already been accomplished during the composite service execution.

4.3 Comparison of Service Selection Approaches

In the following, we compare the local and the global service selection approaches based on the following metrics:

1. **QoS of Composite Service**. This is measured along two perspectives:
   - **Quality Criteria**. As we discussed in earlier section, QoS of composite service is measured by a set of quality criteria. And since there are often trade offs among different quality criterion (e.g., execution time and cost), it is important that the system is able to find a combination of these dimensions that fits the user's preferences.
   - **Ability to satisfy user's requirements**. Although good quality composite service execution requires optimal QoS of services, the satisfaction of end users' constraints is an equally important aspect. There are two kinds of constraints: constraints on a single task and constraints on multiple tasks. The system's ability to accept both kinds of constraints is key to satisfying user requirements.
2. **System Cost**. When the system receives a request to execute a composite service, it utilizes resources to locate candidate Web services for the required tasks and to select among them. Specifically, the system consumes network resources (*bandwidth cost*) to contact the service broker(s) to identify candidate Web services. It then consumes computational resources (*computational cost*) to process the search results and select one among the candidate Web services. Also, when the composite service is being executed, the system interacts with the Web services in order to orchestrate them which also consumes bandwidth and computational resources.

The computational cost of local optimization is polynomial. The bandwidth cost is very limited: for each task, there are two messages that flow between the composite service execution engine and the service broker (`query` and `result`) and three that flow between the execution engine

and the selected Web services (`enable`, `start`, and `completed`).

On the negative side, the local optimization approach has two shortcomings:

1. It cannot consider global trade offs between quality dimensions, especially in the case of composite services involving concurrent threads. For example, in the `Travel Planner` statechart, task t_2 (`AttractionSearching`) and task t_3 (`FlightTicketBooking`) are executed concurrently. If the execution duration of task t_3 is always longer than that of task t_2, the system should select for task t_2 the candidate service that offers the lowest price, regardless of the duration. In the local selection approach however, the system does not take advantage of this fact.
2. When selecting Web services, the local optimization approach can consider constraints on individual tasks, but it cannot consider global constraints, i.e., constraints that cover multiple (or all) tasks in the composite service. Also, although it is always able to select a Web service with minimal execution price or minimal execution duration for each task, it fails when both the execution price and execution duration need to be considered at a global level. For example, it cannot enforce a constraint stating that the composite service's execution price cannot exceed $500 and the execution duration cannot exceed three days.

The global planning approach overcomes these shortcomings, but at the price of higher computational and bandwidth cost. Indeed, the global planner first needs to select an optimal execution plan using an expensive algorithm (exponential in some cases). It then needs to monitor all the candidate Web services (whether they are included in the plan or not) thereby consuming considerable bandwidth resources. Finally, when it detects exceptions or changes, it may need to revise the execution plan, again using an expensive algorithm. Another issue with global planning is that users are required to provide relatively complex input (i.e., global constraints and trade offs).

An experimental comparison between local optimization and global planning is given in Section 6.

5 Implementation

In this section, we describe the implementation of two major components of the AgFlow architecture (shown in Fig. 1): the service broker and the service composition manager.

5.1 Implementing the Service Broker

There are two metadata repositories in the AgFlow system, namely, the *service ontology repository* and the *Web service repository*. We adopt the UDDI registry to implement both metadata repositories. In the UDDI registry, every Web service is assigned to a tModel. A tModel provides a semantic classification of a service's functionality and a canonical description of its interface.

We define an XML schema for service ontologies. Each service ontology is represented as an XML document conforming to this XML schema. A separate tModel of type `serviceOntologySpec` is created for each service ontology. The information that makes up a `serviceOntologySpec` tModel is simple. There is a tModel key, a name (i.e., service ontology's name), an optional description, and a URL that points to the location of the service ontology description document. In the Web service repository, we adopt WSDL to specify services. It should be noted that the Web service's tModel contains the key of a service ontology's tModel in `categoryBag`.

Using the UDDI API, the service broker provides two kinds of interfaces for both repositories, namely the *publish interface* and the *search interface*. The publish interface is to used to create and edit service ontologies. The search interface is used to search and browse service ontologies.

5.2 Implementation of the Service Composition Manager

The service composition manager consists of two modules, namely, the *Execution Planner*, and *Execution Engine*.

- **Execution Planner** is the module that selects a Web service for each task in a composite service using either local optimization or global planning. It provides a specific method called `select()` that can be used to perform local optimization or IP-based global planning. The IP-based global planning approach is implemented as an integer programming solver based on IBM's Optimization Solutions and Library (OSL) [27].
- **Execution Engine** is the heart of the service composition manager. It manages composite service executions from beginning to the end. It determines which tasks need to be executed based on the control and data dependencies. It also maintains the state of the composite service, including the execution state and the events/messages triggered by Web services. The prototype uses the execution engine of the Self-Serv system [5].

6 Experimentation

In order to evaluate the proposed service selection approaches, we developed a travel planning application based on the one presented in Zeng et al. [42] and conducted experiments using the prototype system. Services were developed using IBM's Web Services Toolkit [37] and deployed on a cluster of PCs. All PCs had the same configuration: Pentium III 933MHz with 512M RAM, Windows 2000, Java 2 Enterprise Edition V1.3.0, and Oracle XML Developer Kit. They were connected to a LAN through 100Mbits/sec Ethernet cards.

QoS data is retrieved by the service execution engine in different ways depending on the QoS dimension. The execution duration and the execution cost are retrieved via two operations: getExecutionDuration() and getExecutionPrice(), respectively. These operations are defined in the underlying service ontology. The reliability and reputation on the other hand are calculated by the service composition

manager using the formulas presented in Section 3.1. For this purpose, the service composition manager logs appropriate QoS information during task executions. Finally, the availability is calculated by the service broker based on the information that it records about the up and down time of each service.

Experiments were conducted in two types of environments: *static* and *dynamic*. In a static environment, there is no change in the QoS of any component service during a given composite service execution. In addition, all component services are able to execute the tasks successfully and in conformance with their expected QoS. In a dynamic environment, on the other hand, the QoS of component services may undergo changes during the execution of a composite service. Existing component services may become unavailable, new component services with better QoS may become available, component services may not be able to complete the execution of tasks, or they may complete them but without meeting their expected QoS.

The experiments involved composite services with varying numbers of basic states. The composite services were created by randomly repeating existing states in the composite service shown in Fig. 3. The number of states varied from 10 to 80 with steps of 10 (e.g., 10, 20, ... 80). Also, we varied the number of candidate component services per task from 10 to 40 with steps of 10.

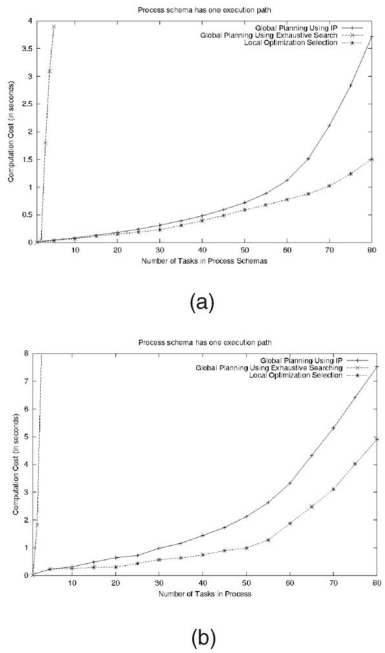


Fig. 9. Experimental results (computation cost) in a static environment, varying the number of tasks in the statechart and the number of candidate services per task.

6.1 Measuring Computation Cost

The first series of experiments aimed at comparing the three selection approaches previously described (local optimization, global planning by exhaustive search, and global planning by integer programming) with respect to the computational overhead involved by their planning phase. The idea is to provide a basis for determining when should global planning be preferred over local optimization. Accordingly, we measured the computation cost (in seconds) of selecting component services to create execution plans under different selection approaches. For each test case, we executed the composite service 10 times and computed the average computation cost.

6.1.1 Static Environments

In a static environment and when using a global planning approach, once a process execution plan is created the execution planner does not need to replan the process execution. So, the global planner is invoked only once during a composite service execution. In the case of local optimization, on the other hand, if we assume that the number of tasks that are executed is N, then the service selector in the execution planner is invoked N times to select a component service for each task.

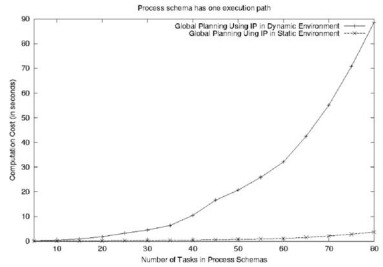
Fig. 9 plots computation cost (in seconds) of selecting services for composite services with only one execution path. In this experiment, we varied the number of tasks and the number of candidate services per task. In all the approaches, the computation cost increases when the number of tasks increases and the number of candidate services increases. The computation cost of global planning by exhaustive searching is very high even in very small scale in aspect of the number of tasks and candidate service. Although the computation cost of global planning by IP is higher than that of local optimization, it still acceptable if the number of tasks and candidate services is not very large. For example, when there are 40 tasks and 40 candidate Web services for each task, the computation cost of the global planning by IP (1.6 seconds) is almost 1.5 times higher than the local optimization approach (0.7 seconds).

We also conducted experiments involving composite services with more than one execution path. Using IP-based global planning, we observed that the computation cost increases proportionally to the number of the execution paths given a constant number of tasks in each execution path. This is normal since for each execution path, an optimal execution plan needs to be generated. It should be noted that the number of execution paths does not affect the computation cost of the local optimization approach since the selection is done incrementally and, therefore, only for the execution path taken.

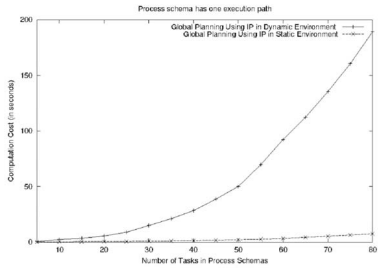
6.1.2 Dynamic Environments

Dynamic environments were simulated by randomly changing the QoS of the component services during a composite service execution according to three QoS properties: execution price, duration, and availability. Changes are done so that, if the global planning approach is used, the execution of a composite service needs to be replanned whenever a task is completed. Hence, the global planner needs to be invoked as many times as there are tasks in the composite service.

Fig. 10 presents computation cost (in seconds) of service selection for composite services with only one execution path, where we vary the number of tasks and the number of candidate services per task. In both selection approaches, the computation cost increases with the number of tasks and the number of candidate services. The computation cost of global planning using IP is much higher than in a static one. For 80 tasks and 40 candidate services per task, the



(a)



(b)

Fig. 10. Experimental results (computation cost) in a dynamic environment and static environment, varying the number of tasks and the number of candidate services per task.

computation cost of the global planning using IP in a dynamic environment (190 seconds) is almost 25 times higher than in a static one (7.8 seconds). However, this may be considered to be an extreme case, and for reasonably sized cases, global planning delivers acceptable computation cost. Furthermore, the computation cost in a dynamic environment is spread across the lifespan of the composite service execution.

6.2 Measuring QoS of Composite Services

The second series of experiments aimed at evaluating the QoS of composite service executions in both static and dynamic environments. Table 2 presents experiment results on composites' execution duration (i.e., $Q_{du}(CS)$, in seconds) and execution price (i.e., $Q_{price}(CS)$, in dollars) in static environment, where we vary the number of tasks. The experiment results show that the global planning approach yields significantly better QoS than the local optimization approach. For example, the execution duration is consistently shorter when using global planning approach than that of local optimization approach.

Table 3 shows that global planning approach gives better QoS of composite services than that of local optimization approach in most of the cases in dynamic environments (for clarity and space reasons, we only show the execution duration and price in the table). At the same time, the average QoS of composite services in the global planning approach is better than that of the local optimization approach. However, in some cases (see left side of Table 3, composite service 4), the global planning approach may create worse execution result compared to the local optimization approach. The reason is that the global planning approach makes the decision based on the set of currently available Web services. Since the availability of Web services is dynamic, some services that are selected by the optimal execution plan may become unavailable when the task needs to be executed. Although the system can replan the unexecuted part of composite services, the executed part may become suboptimal, making the entire composite service execution suboptimal. Note also that, even in dynamic environments, the global planner is still able to handle constraints spanning multiple tasks when

TABLE 2
Experimental Results on QoS of Composite Services in Static Environments

Composite Service	$Q_{du}(CS)$		$Q_{price}(CS)$	
	GP	LO	GP	LO
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:	6627.16	9305.9	1191.08	1753.42

1. Composite service has 20 tasks and one execution path.

Composite Service	$Q_{du}(CS)$		$Q_{price}(CS)$	
	GP	LO	GP	LO
1	10212.4	14110.2	2216	3626
2	11221.7	15330.5	2327	3502
3	10945.2	14280.4	2424	3445
4	12535.5	15292.2	2221	3524
5	12340.5	14302.5	2123	3324
6	11350.5	14470.5	2285	3528
7	12445.2	13980.4	2184	3589
8	12440.5	14470.5	2485	3487
9	11970.4	15920.4	2160	3616
10	11250.3	15628.3	2352	3459
11	11490.3	15520.3	2373	3412
12	12890.3	14920.5	2163	3519
Average:	11757.73	14852.22	2276.08	3502.58

2. Composite service has 80 tasks and one execution path.

TABLE 3
Experimental Results on QoS of Composite Services in Dynamic Environments

Composite Service	$Q_{du}(CS)$		$Q_{price}(CS)$	
	GP	LO	GP	LO
1	6433.2	9322.4	1341	1834
2	6434.9	8123.9	1123	1873
3	6343.6	8654.5	1512	1836
4	7443.8	7302.2	1123	1873
5	6334.3	8463.3	1134	1838
6	7112.7	9474.8	1324	1834
7	6235.4	8674.4	1241	1835
8	6542.7	8354.5	1132	1873
9	6234.7	8363.4	1312	1835
10	6435.2	8733.3	1153	1736
11	5823.6	8340.3	1212	1873
12	7245.1	9473.5	1142	1546
Average:	6609.93	8673.37	1229.08	1815.5

1. Composite service has 20 tasks and one execution path.

Composite Service	$Q_{du}(CS)$		$Q_{price}(CS)$	
	GP	LO	GP	LO
1	11225.5	18346.3	2341	3834
2	11342.5	18324.6	2651	3834
3	12341.4	18345.4	2234	3345
4	13221.4	17344.3	2612	3873
5	16512.6	18734.7	2123	3345
6	12315.2	17232.3	2234	3873
7	13123.6	17234.3	2612	3873
8	13123.7	14235.7	2512	3458
9	12341.9	17623.8	2124	3456
10	13123.2	17232.2	2342	3443
11	13213.9	14343.7	2651	3983
12	12322.8	17623.7	2213	3436
Average:	12850.65	17218.42	2387.42	3646.08

2. Composite service has 80 tasks and one execution path.

performing execution replanning (see Section 4.2.5). For example, in the left side of Table 3, the composite service executions are performed under a execution duration constraint ($Q_{du} < 7,500$ sec).

6.3 Discussion

From the experimental results, we conclude that the IP-based global planning approach leads to significantly better QoS of composite service executions with little extra system cost in static environments. For example, in a static environment, a composite service execution with 40 tasks spends: 1) 1.6 seconds for selecting Web services using global planning; 2) 0.7 seconds using local optimization. In a dynamic environment, however, global planning imposes a perceivable overhead. For example, a composite service with 80 tasks spends 190 seconds for selecting Web services using global planning, while it spends 4.9 seconds using local optimization.

These results reinforce the conclusions of the analytical considerations of Section 4.3. If there is no requirement for specifying global constraints, then local optimization is preferable, especially in dynamic environments. On the other hand, global planning is superior when it comes to selecting services that satisfy certain global constraints and which optimize global tradeoffs. Given the fact that, a global selection approach considers both local and global selection constraints, the results clearly demonstrate the benefit of using a IP-based method for global service execution planning, even if we take into consideration the modest planning overhead.

7 Related Work

In this section, we first review some related work on QoS management in middleware before discussing the relationship between our work and research and standardization efforts in the area of (Web) service composition.

QoS management has been widely discussed in the area of middleware systems [3], [25], [15]. The focus of these works is essentially on the following issues: QoS specification to allow description of application behavior and QoS parameters, QoS translation and compilation to translate specified application behavior into candidate application configurations for different resource conditions, QoS setup to appropriately select and instantiate a particular configuration, and, finally, QoS adaptation to runtime resource fluctuations. Most efforts in QoS-aware middleware however are centered on the network transport and system level. Very limited work has been done at the application and business process levels.

Other work in the area of QoS-aware middleware include those of Mecella et al. [23] and Naumann et al. [26] which consider data quality management in cooperative information systems. They investigate techniques to select the best available data from various service providers based on dimensions such as accuracy, completeness, and consistency. However, they do not consider QoS-based selection for service composition.

Service composition is a very active area of research and standardization [4], [10]. Notations for service description and composition have also been proposed in other standardization efforts such as ebXML [30] and DAML-S [2]. DAML-S supports the description of Semantic Web-enabled services based on a generic ontology in which both functional and QoS aspects of services are expressed as rule-based preconditions and postconditions on service operations. This rule-driven approach is also adopted in SWORD [33] where forward chaining is proposed as a mechanism for deriving composite service schemas given a set of required preconditions and postconditions. Unlike our proposal, however, neither DAML-S nor SWORD consider specific QoS criteria nor do they address the issue of dynamic service selection using these criteria. BPEL4WS [31] provides a process-based language for service composition. Our approach builds upon the building blocks of these standards to provide a QoS-aware and dynamic service composition model. In fact, our proposal could be applied to composite services specified in BPEL4WS, as statecharts support similar control-flow primitives as BPEL4WS.

Despite the relatively large body of work in the area of service composition, few efforts have specifically addressed the topic of QoS-aware service composition. A notable exception is the work by Gu and Nahrstedt [16] and Xu and Nahrstedt [38]. In this work, the authors propose global planning algorithms for dynamic QoS-aware service composition which, like our approach, compute an initial plan at the start of a composite service execution and then revise the plan as necessary during the execution. Unlike our approach however, the underlying service composition model does not support parallelism nor branching. Instead, a composite service (called a *service path* by the authors) is defined as a chain of service operations. Still, many of the ideas proposed in [16], [38] could be integrated into our approach. In particular, the probing techniques of Xu and Narhstedt [38] could be used to efficiently collect QoS information during composite service execution.

CMI [14] and eFlow [9] have investigated the possibility of performing dynamic service selection based on user requirements. 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 by a concrete activity type. A selection policy determines 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 specific service. Both CMI and eFlow focus on optimizing service selection at a task level. In addition, no QoS model is explicitly supported. In contrast, our approach focuses on optimizing service selection at the composite service level and handles various types of QoS criteria.

The SAHARA [34] project proposed a reference model for service composition which recognizes two different models: the cooperative composition model and the brokered composition model. Our approach can be seen as adding dynamic QoS-aware service selection to SAHARA's brokered model. Indeed, the service composition

manager in our architecture acts as a broker between the composite service clients and the services participating in the composition.

Some related work on QoS has been done in the area of workflow. Most efforts in this area focus on specifying and enforcing temporal constraints [13], [6], [15]. Some projects such as METEOR [8] and CrossFlow [21] consider other QoS criteria than time. METEOR [8] considers four quality dimensions: time, cost, reliability, and fidelity. However, it does not consider the dynamic composition of services. Instead, it focuses on analyzing, predicting, and monitoring workflow QoS. CrossFlow proposes the use of continuous-time Markov chains to estimate execution time and cost of workflow instances. However, the contributions of CrossFlow on this topic are complementary to ours, insofar as they do not deal with dynamic service selection. Klingemann [20] proposes modeling primitives for capturing points of flexibility in workflows where dynamic selection can occur, but does not propose specific selection methods as we do.

8 Conclusion

AgFlow is a QoS-aware middleware supporting quality driven Web service compositions. The main features of the AgFlow system are: 1) a service quality model to evaluate overall quality of (composite) Web services and 2) two service selection approaches for composite service execution.

AgFlow has been implemented as a platform that provide tools for: 1) defining service ontologies, 2) specifying composite services using statecharts, 3) assigning services to the tasks of a composite service. The AgFlow platform has been used to validate the feasibility and benefits of the proposed approaches. The experimental results show that the computation cost of IP-based global planning is acceptable when the number of tasks and candidate services is not very large. The results also show that the global planning approach leads to better QoS and, specifically, to lower execution prices and execution durations. We argue that large scale service composition should be done in a hierarchical fashion [43], in such a way that a global planning approach can be used in each layer of the composition without major performance penalty.

The proposed global planning approach does not take into account the computation cost of the planning/replanning algorithm as part of QoS of the composite service. This is a limitation of the current solution as the computation cost affects the composite service's execution time. A possible extension to the approach is to develop techniques for estimating the computation cost of global planning in order to evaluate its benefit and to determine if it outweighs its overhead in a given situation. Another direction for further work is to combine global planning and local optimization approaches in order to leverage their relative advantages.

Acknowledgments

The authors would like to thank Phuong Nguyen for the fruitful discussions regarding the QoS criteria.

References

[1] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services.* Springer Verlag, 2003.

[2] A. Ankolekar, M. Burstein, J.R. Hobbs, O. Lassila, D. McDermott, D. Martin, S.A. McIlraith, S. Narayanan, M. Paolucci, T. Payne, and K. Sycara, "DAML-S: Web Service Description for the Semantic Web," *Proc. First Int'l Semantic Web Conf. (ISWC 02),* 2002.

[3] C. Aurrecoechea, A.T. Campbell, and L. Hauw, "A Survey of QoS Architectures," *Multimedia Systems,* vol. 6, no. 3, pp. 138-151, 1998.

[4] *Distributed and Parallel Database,* special issue on Web Services, B. Benatallah and F. Casati, eds., Kluwer Academic, 2002.

[5] B. Benatallah, M. Dumas, Q.Z. Sheng, and A.H. Ngu, "Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services," *Proc. Int'l Conf. Data Eng. (ICDE),* pp. 297-308, Feb. 2002.

[6] C. Bettini, X. Wang, and S. Jajodia, "Temporal Reasoning in Workflow Systems," *Distributed and Parallel Databases,* vol. 11, no. 3, pp. 269-306, 2002.

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

[8] J. Cardoso, "Quality of Service and Semantic Composition of Workflows," PhD thesis, Univ. of Georgia, 2002.

[9] F. Casati and M.-C. Shan, "Dynamic and Adaptive Composition of E-Services," *Information Systems,* vol. 26, no. 3, pp. 143-162, May 2001.

[10] *Very Large Databases J.,* special issue on E-Services, F. Casati et al., eds., Springer-Verlag, 2001.

[11] F. Curbera et al., "Unraveling the Web Services: An Introduction to SOAP, WSDL, and UDDI," *IEEE Internet Computing,* vol. 6, no. 2, Mar./Apr. 2002.

[12] M. Dumas and A.t. Hofstede, "UML Activity Diagrams as a Workflow Specification Language," *Proc. Int'l Conf. Unified Modeling Language (UML),* pp. 86-90, Oct. 2001.

[13] J. Eder, E. Panagos, and M. Rabinovich, "Time Constraints in Workflow Systems," *Lecture Notes in Computer Science,* vol. 1626, 1999.

[14] 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, vol. 24, no. 6, pp. 429-456, 1999.

[15] M. Gillmann, G. Weikum, and W. Wonner, "Workflow Management with Service Quality Guarantees," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 228-239, June 2002.

[16] X. Gu and K. Nahrstedt, "A Scalable QoS-Aware Service Aggregation Model for Peer-to-Peer Computing Grids," *Proc. 11th IEEE Int'l Symp. High Performance Distributed Computing (HPDC),* pp. 73-82, July 2002.

[17] D. Harel and A. Naamad, "The STATEMATE Semantics of Statecharts," *ACM Trans. Software Eng. and Methodology,* vol. 5, no. 4, pp. 293-333, 1996.

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

[19] B. Kiepuszewski, A.t. Hofstede, and C. Bussler, "On Structured Workflow Modelling," *Proc. Int'l Conf. Advanced Information Systems Eng. (CAiSE),* June 2000.

[20] J. Klingemann, "Controlled Flexibility in Workflow Management," *Proc. Int'l Conf. Advanced Information Systems Eng. (CAiSE),* pp. 126-141, June 2000.

[21] J. Klingemann, J. Wäsch, and K. Aberer, "Deriving Service Models in Cross-Organizational Workflows," *Proc. Ninth Int'l Workshop Research Issues in Data Eng.: Virtual Enterprise (RIDE-VE '99),* Mar. 1999.

[22] Y. Liu, A.H.H. Ngu, and L. Zeng, "QoS Computation and Policing in Dynamic Web Service Selection," *Proc. 13th Int'l Conf. World Wide Web (WWW),* May 2004.

[23] M. Mecella, M. Scannapieco, A. Virgillito, R. Baldoni, T. Catarci, and C. Batini, "Managing Data Quality in Cooperative Information Systems," *Proc. 10th Int'l Conf. Cooperative Information Systems (CoopIS),* 2002.

[24] B. Medjahed, A. Bouguettaya, and A.K. Elmagarmid, "Composing Web Services on the Semantic Web," *The VLDB J.,* vol. 12, no. 4, pp. 333-351, 2003.

[25] K. Nahrstedt, D. Xu, D. Wichadakul, and B. Li, "QoS-Aware Middleware for Ubiquitous and Heterogeneous Environments," *IEEE Comm. Magazine,* vol. 39, no. 11, pp. 2-10, 2001.

[26] F. Naumann, U. Leser, and J.C. Freytag, "Quality-Driven Integration of Heterogenous Information Systems," *Proc. Int'l Conf. Very Large Databases (VLDB),* pp. 447-458, 1999.
[27] IBM Optimization Solutions and Library, http://www-3.ibm.com/software/data/bi/osl/index.html, 2003.
[28] J. O'Sullivan, D. Edmond, and A.t. Hofstede, "What's in a Service?" *Distributed and Parallel Databases,* vol. 12, nos. 2-3, pp. 117-133, Sept. 2002.
[29] G. Oulsnam, "Unravelling Structured Programs," *The Computer J.,* vol. 25, no. 3, pp. 379-387, 1982.
[30] S. Patil and E. Newcomer, "ebXML and Web Services," *IEEE Internet Computing,* vol. 7, no. 3, pp. 74-82, May/June 2003.
[31] C. Peltz, "Web Services Orestrestration and Choreography," *Computer,* vol. 36, no. 10, pp. 46-52, Oct. 2003.
[32] M. Pinedof, *Scheduling: Theory, Algorithms, and Systems,* second ed. Prentice Hall, 2001.
[33] S. Ponnekanti and A. Fox, "SWORD: A Developer Toolkit for Building Composite Web Services," *Proc. Alternate Tracks of the 11th World Wide Web Conf.,* May 2002.
[34] B. Raman, S. Agarwal, Y. Chen, M. Caesar, W. Cui, P. Johansson, K. Lai, T. Lavian, S. Machiraju, Z. Morley-Mao, G. Porter, T. Roscoe, M. Seshadri, J.S. Shih, K. Sklower, L. Subramanian, T. Suzuki, S. Zhuang, A.D. Joseph, R.H. Katz, and I. Stoica, "The SAHARA Model for Service Composition Across Multiple Providers," *Proc. First Int'l Conf. Pervasive Computing,* pp. 1-14, May 2002.
[35] D.D. Wackerly, W. Mendenhall, and R.L. Scheaffer, *Mathematical Statistics with Application.* Duxbury Press, 1996.
[36] Web Services Architecture Requirements Working Group, http://www.w3.org/TR/wsa-reqs, 2004.
[37] IBM Web Services Toolkit, http://alphaworks.ibm.com/tech/webservicestoolkit, 2003.
[38] D. Xu and K. Nahrstedt, "Finding Service Paths in a Media Service Proxy Network," *Proc. SPIE/ACM Multimedia Computing and Networking Conf. (MMCN),* Jan. 2002.
[39] L. Zeng, "Dynamic Web Services Composition," PhD thesis, Univ. of New South Wales, 2003.
[40] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q.Z. Sheng, "Quality Driven Web Services Composition," *Proc. 12th Int'l Conf. World Wide Web (WWW),* May 2003.
[41] L. Zeng, B. Benatallah, and A.H.H. Ngu, "On Demand Business-to-Business Integration," *Proc. Ninth Int'l Conf. Cooperative Information Systems,* 2001.
[42] L. Zeng, B. Benatallah, A.H.H. Ngu, and P. Nguyen, "AgFlow: Agent-Based Cross-Enterprise Workflow Management System (Demonstration Paper)," *Proc. 27th Int'l Conf. Very Large Data Bases,* 2001.
[43] L. Zeng, J.-J. Jeng, S. Kumaran, and J. Kalagnanam, "Reliable Execution Planning and Exception Handling for Business Process," *Proc. VLDB Workshop Technologies for E-Services (VLDB-TES),* 2003.

Liangzhao Zeng received the PhD degree in computer science from University of New South Wales, Sydney, Australia in 2003. He is a postdoctorial researcher in the Business Informatics department in IBM T.J. Watson Research Center. His research interests are in the areas of web services, business process management, and data stream management.

Boualem Benatallah received the PhD degree in computer science from Grenoble University (IMAG, France). He is senior lecturer at the University of New South Wales, Sydney, Australia. His research interests lie in the areas of Web services, Workflows semantics Web, and mobile data management. He was a visiting scholar at Purdue University, West Lafeyette, Indiana, and a visiting professor at INRIA-Loria, France. He is member of the editorial board of the *International Journal of Business Process Integration and Management*. He has been a program committee member of several conferences. He was guest editor for the special issue on E-Services of the *Journal on Parallel and Distributed Databases* and for the special issue on M-Services, Web Services for the Wireless World of *IEEE Transactions on Systems, Man, and Cybernetics*. He is a member of ACM and IEEE.

Anne H.H. Ngu currently is an associate professor with the Department of Computer Science at Texas State University—San Marcos. She has over 15 years of experience in research and development in IT with expertise in integrating data and applications, automating business processes on the web, databases, and object-oriented technologies. From 1992-2000, she worked as a senior lecturer in the School of Computer Science and Engineering, University of New South Wales (UNSW). She had held research scientist positions with Telecordia Technologies and MCC. She has been awarded summer faculty scholarship by Lawrence Livermore National Laboratory in 2003 and 2004.

Marlon Dumas received the PhD degree in computer science from the University of Grenoble, France, in 2000. Since then he has been taken successive positions as postdoctoral fellow and lecturer at the Queensland University of Technology, Brisbane, Australia. His research interests are in the areas of Web services and business process technologies. He is member of the IEEE Computer Society. More information can be found at his web page: http://www.fit.qut.edu.au/dumas He is a member of the IEEE Computer Society.

Jayant Kalagnanam received the PhD degree in engineering and public policy from Carnegie Mellon University in 1991. He is a research staff member in the Mathematical Sciences Department at IBM T.J. Watson Research Center. His research interests include business analytics and optimization. He is a member of the Institute for Operations Research and Management Science (INFORMS). His home page is http://www.research.ibm.com/people/j/Jayant.

Henry Chang received the PhD degree in computer sciences from UW-Madison in 1987. He is a senior technical staff member in the Business Informatics department in IBM T.J. Watson Research Center. His recent research interests include business process monitoring and management and business collaboration infrastructure across design chain and supply chains. He is a member of the ACM.