SERVICE SELECTION AND TRANSACTIONAL MANAGEMENT FOR WEB SERVICE COMPOSITION

Von der Fakultät Elektrotechnik und Informatik
der Gottfried Wilhelm Leibniz Universität Hannover

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

Dr. rer. nat.

genehmigte Dissertation von

M.Sc. Mohammad Al-Rifai

geboren am 8. März 1974, in Hodeidah, Jemen

2011

Referent: Prof. Dr. tech. Wolfgang Nejdl
Ko-Referent: Prof. Dr. Wolf-Tilo Balke
Tag der Promotion: 08. April 2011

ABSTRACT

In addition to functional aspects web services also expose non-functional properties, which describe how the offered functionalities are delivered. In service-oriented systems with real business settings, the non-functional properties play an important role in the service life cycle, including discovery, selection and operation of services. This PhD thesis focuses on two research challenges related to the non-functional aspects of web service composition.

The first part of the thesis addresses the problem of selecting web services for a composite application such that the end-to-end Quality of Service (QoS) values perceived by the user meet his/her requirements. This problem is often modeled as a Multiple-Choice Multi-Dimensional Knapsack problem, which is known to be NP-hard. Therefore, it is expected that any exact solution to this problem will have an exponential computation time. In order to solve this problem, new selection algorithms are presented in this thesis, which are able to satisfy all user's constraints much faster than exact solutions, while achieving close-to-optimal results. More specifically, a novel hybrid approach is presented, which combines global optimization with local selection in order to benefit form both worlds. In addition, skyline-based algorithms are presented to prune non-interesting services from the search space and thus reducing the computation time dramatically. Several experiments have been conducted using both real and synthetic datasets to evaluate the proposed selection methods. The results of these experiments indicate a significant improvement in performance compared to the state-of-the-art solutions.

Another important non-functional aspect that needs to be taken into account when dealing with service compositions is the transactional characteristics of the involved services. A transactional coordination of the composed services is needed in order to ensure a consistent outcome in case of service failures. Due to the inherent autonomy and heterogeneity of web services, current standards for web service transactions relax the ACID properties and rely on compensation models for failure recovery. However, due to relaxing the isolation property transactional dependencies start to emerge between concurrent long-running web service transactions. The second part of this thesis addresses this problem and proposes an extension to the standard framework for web service transactions to enable detecting and handling such transactional dependencies. Moreover, an optimistic protocol for concurrency control and decentralized solutions for handling global dependency cycles are presented. The proposed methods have the advantage that they can be deployed in a fully distributed fashion within the proposed architecture. Experimental evaluation of the concurrency control protocol using extensive simulation of long running web service transactions are also presented.

Keywords: *Web Services, Quality of Service, Transactions.*

ZUSAMMENFASSUNG

Neben funktionalen Aspekten besitzen Web Services nicht-funktionalen Eigenschaften, die bei der Entdeckung, Auswahl und Erbringung von Dienstleistungen in Service-orientierte Architekturen (SOA) mit realen Geschäftsbedingungen eine wichtige Rolle spielen. Diese Dissertation konzentriert sich auf zwei Forschungsherausforderungen im Zusammenhang mit der Problematik der Berücksichtigung von nicht-funktionalen Aspekten bei Web Service Kompositionen.

Der erste Teil der Dissertation befasst sich mit der Problematik der Auswahl von geeigneten Web Services für Kompositionen auf Basis von ihre Qualitätseigenschaften, so dass alle Anforderungen vom Benutzer erfüllt werden. Dieses Problem wird oft als Multiple-Choice Multi-dimensional Knapsack Problem modelliert, von dem bekannt ist, dass es NP-hard ist. Daher ist davon auszugehen, dass jede exakte Lösung für dieses Problem eine exponentielle Zeitkomplexität haben wird. Um dieses Problem zu bewältigen, werden in dieser Arbeit effiziente Lösungen für die Auswahl von Web Services vorgeschlagen, die im Vergleich zu vorhandenen Lösungen viel schneller sind und dabei nahezu optimale Ergebnisse erzielen können. Zunächst wird eine hybride Lösung vorgestellt, die globale Optimierung mit lokalen Selektion von Services kombiniert um von den Vorteilen beider Welten zu profitieren. Darüber hinaus werden Skyline-basierte Algorithmen vorgestellt, um uninteressante Services aus dem Suchraum auszuschließen und damit die Rechenzeit drastisch zu reduzieren. Die vorgestellte Lösungen wurden mit Hilfe von echten und synthetischen Datensätze evaluiert. Die Ergebnisse zeigen eine deutliche Verbesserung der Gesamtleistung im Vergleich zu den vorhandenen Lösungen.

Der zweite Teil dieser Arbeit befasst sich mit der Problematik der Berücksichtigung von den nicht-funktionalen transaktionalen Eigenschaften der Web Services bei der Ausfhrung im Rahemn einer Komposition. Die Lockerung der ACID-Anforderungen in der vorhandenen Standards für Web Service Transaktionen wegen der Autonomie und Heterogenität der Web Services fhrt oft dazu, dass transaktionale Abhängigkeiten zwischen nebenläufigen Geschäftsprozesse entstehen. Ohne vernnftige transaktionale Koordination der involvierten Web Services kann der konstistenzsicherung der gefassten Daten im Fehlerfall nicht gewärleistet werden. In dieser Arbeit wird eine Erweiterung des Transaktions-Frameworks für Web Services vorgeschlagen, die das Verfolgen von solche transaktionalen Abhängigkeiten ermöglicht. Darüber hinaus wird ein optimistisches Protokoll für Concurrency Control vorgestellt, das vollständig verteilt in die vorgeschlagene Architektur eingesetzt werden kann. Die Ergebnisse einer Evaluierung des vorgeschlagenen Protokolls mit umfangreichen Simulationen von langlaufenden Web Service Transaktionen werden vorgestellt.

Schlagwörte: *Web Dienste, Qualität von Services, Transaktionen.*

FOREWORD

The solutions presented in this thesis have been published in the proceedings of some conferences and workshops as well as at some scientific journals, as follows:

The contributions presented in Chapter 3 are included in the following publications list:

- *Combining Global Optimization with Local Selection for Efficient QoS-aware Service Composition*, Mohammad Alrifai and Thomas Risse, in: Proceedings of the 18th International World Wide Web Conference, WWW 2009, Madrid, Spain, April 20-24, 2009, pp. 881-890, 2009, ACM, 978-1-60558-487-4. [AR09]

- *Selecting Skyline Service for QoS-based Web Service Composition*, Mohammad Alrifai, Dimitrios Skoutas and Thomas Risse, in: Proceedings of the 19th International World Wide Web Conference, WWW 2010, Raleigh, NC, USA, April 26-30, 2010. [ASR10]

- *Efficient QoS-aware Web Service Composition*, Mohammad Alrifai and Thomas Risse, in Proceedings of the 3rd Workshop on Emerging Web Services Technology, at the 6th European Conference on Web Services (ECOWS), Dublin, November 2008. [AR10]

- *A Scalable Approach for QoS-based Web Service Selection*, Mohammad Alrifai, Thomas Risse, Peter Dolog and Wolfgang Nejdl, in: Proceedings of the 1st International Workshop on Quality-of-Service Concerns in Service Oriented Architectures (QoSCSOA'08) in conjunction with ICSOC 2008, Sydney, December 2008. [ARDN08]

- *Distributed and Scalable QoS Optimization for Dynamic Web Service Composition*, Mohammad Alrifai, presented at the PhD Symposium at the International Conference on Service Oriented Computing, Sydney, December 2008. [Alr08]

The contributions presented in Chapter 4 are included in the following publications list:

- *Distributed Management of Concurrent Web Service Transactions*, Mohammad Alrifai, Peter Dolog, Wolf-Tilo and Wolfgang Nejdl, in: IEEE

Transactions on Services Computing, vol. 2, no. 4, pp. 289-302, Oct.-Dec. 2009, doi:10.1109/TSC.2009.29. [ADBN09]

- *Transactions Concurrency Control in Web Service Environment*, Mohammad Alrifai, Peter Dolog and Wolfgang Nejdl, in: Proceedings of the fourth IEEE European Conference on Web Services (ECOWS), Zrich, Switzerland, Dec. 2006 Page(s):109 - 118. [ADN06]

- *Nonblocking Scheduling for Web Service Transactions*, Mohammad Alrifai, Wolf-Tilo, Peter Dolog and Wolfgang Nejdl, in: Proceedings of the 5th IEEE European Conference on Web Services (ECOWS),Halle, Germany, 26-28 Nov. 2007 Page(s):213 - 222. [ABDN07]

Contents

List of Figures

1 Introduction

Recently, there has been a growing trend for web-based businesses to outsource parts of their processes by delegating the operation and responsibility of these parts to a third-party service providers, so as to focus more on their core activities. In addition, there is often a need to combine different services to achieve a more complex task that cannot be fulfilled by an individual service. The Service-Oriented Architecture paradigm (SOA) and its realization through standardized Web Service technologies provide a promising solution for seamless integration of web applications. Industry standards, such as WSDL, UDDI, WS-BPEL, exist for describing, locating and composing web services. Semantic web based technologies like OWL-S, WSDL-S and WSMO have also been proposed to enrich the descriptions of the offered service functionalities and service requests with semantics and hence, improve the accuracy of the matchmaking algorithms.

In addition to the functional aspects, which describe what the service does, web services also expose non-functional properties, which describe how the offered functionalities are delivered. The non-functional aspects include quality of service (QoS) in terms of quantifiable attributes such as responsiveness, availability etc. as well as other non-quantifiable properties such as transactional support, privacy and security. In SOA based applications with real business settings non-functional properties of the invoked web services play a major role in determining the success or failure of the system. The performance of a composite application, for example, as perceived by the user (e.g. in terms of response time) strongly depends on the performance of the outsourced services. Therefore, modeling, provisioning and managing services based on non-functional properties become fundamental challenges in Service-Oriented Architectures.

1.1 Problems Addressed in this Thesis

This thesis focuses on two research challenges, which are related to the non-functional aspects of web service composition. In the following, we give a brief description of these research problems.

1.1.1 Problem 1: Service Selection for Compositions with End-to-End QoS Requirements

Quality of Service (QoS) properties play a major role in distinguishing similar services. Therefore, in applications with real business settings, service requests are usually associated with some requirements on the expected QoS level (e.g. response time, availability, throughput etc). QoS requirements are specified in terms of upper and/or lower bounds on the numerical values of the different QoS attributes. The goal of a QoS-based web service selection is to locate the *best* services that satisfy the given set of QoS constraints in addition to the functional requirements. The best service in this context is the service that satisfies all requirements and at the same time maximizes a given user's utility function.

Users of composite applications, however, are typically unaware of the involved services, and they specify their QoS requirements in terms of end-to-end QoS constraints (e.g. average end-to-end response time, minimum overall throughput, maximum total cost etc). Selecting the best service from a list of alternative services for each task such that all user's QoS requirements are satisfied, is a non-trivial task as the number of possible combinations can be very huge. This problem is a combinatorial problem, which is known to be NP-hard in the strong sense. Any exact solution to this problem is expected to have an exponential computational complexity with respect to the number of candidate services, which can be out of the run-time requirements.

The problem of QoS-based service selection for compositions with end-to-end constraints (or shortly QoS-based service composition) becomes especially important and challenging as the number of functionally-equivalent services offered on the web at different QoS levels increases. According to [AMM08], there has been a more than 130% growth in the number of published web services in the period from October 2006 to October 2007. The statistics published by the web services search engine

Seekda![1] also indicate an exponential increase in the number of web services over the last three years. Moreover, it is expected that with the proliferation of the Cloud Computing and Software as a Service (SaaS) concepts [CLPZ09], more and more web services will be offered on the web at different levels of quality. The pay-per-use business model promoted by the cloud computing paradigm will enable service providers to offer their (software) services to their customers in different configurations with respect to QoS properties. Therefore, it is expected that service requesters will be soon faced with a huge number of variation of the same services offered at different QoS levels and prices, and the demand for an automatic and scalable service selection method will increase.

The problem of designing efficient and scalable algorithms for QoS-based service compositions will be addressed in the first part of this thesis (in Chapter 3).

1.1.2 Problem 2: Transactional Management for Web Service Compositions

Business web applications very often involve activities that possess transactional characteristics, such as, placing an order, buying a product, charging a bank account etc. Web services that provide such activities are referred to as transactional web services. A key requirement of successful web service-based business applications, is to ensure reliable and consistent execution of their transactional web services despite the presence of concurrency and system/network failures. Similar to transaction management in database systems, it is important that handling concurrency and failures are factored out from the diversity of the applications, and are delegated to a generic run-time transactional management system. Such a clear separation of responsibilities enables seamless and rapid development of web services as developers do not need to take care of concurrency and failure issues. Following this separation of concerns principle, the transactional management of web services is typically considered as one of the non-functional aspects of web service compositions.

Several specifications have been proposed by industry and academia in the recent years for establishing a set of standards and protocols for managing and coordinating transactional web service compositions. The current de facto standards are the WS-Coordintaion [Comg], WS-AtomicTransaction [Come] and WS-BusinessActivity [Comf],

[1] http://webservices.seekda.com/

which were proposed by industrial companies such as IBM, Microsoft BEA Systems among others and approved by the Organization for Advancement of Structured Information Standards (OASIS). These standards focus mainly on handling failure recovery be means of a locking-based solution for short running transactions (e.g. WS-AtomicTransaction) and a compensation-based solution for long-running transactions (e.g. WS-BusinessActivity).

However, the concurrency problem is largely overlooked in the current specifications. In the open and dynamic web service environment, business transactions enter and exit the system independently. Under isolation relaxation transactional dependencies can emerge among independent business processes, which need to be taken into account when compensation is required in order to avoid inconsistency problems. Such transactional dependencies are currently overlooked in the web service transaction models and standards. Therefore, there is a need for extending the current web service transactions framework to handle such transactional dependencies and ensure consistency of the accessed data by concurrent web service transactions.

The concurrency control problem for composite web service transactions will be the focus of the second part of this thesis (in Chapter 4).

1.2 Contribution of the Thesis

In this section we briefly describe the contribution of this thesis in addressing the aforementioned research problems.

1.2.1 Efficient QoS-aware Service Selection for Service Compositions

The contribution of this thesis in addressing the performance and scalability issues of QoS-based service selection is twofold.

Firstly, we propose a novel hybrid and distributed solution that combines global optimization with local selection techniques. This method is more suitable for open web service environments, where centralized QoS-based selection is not desirable. For this purpose, we propose reformulating the QoS-aware service composition problem into two sub-problems that can be solved more efficiently in two subsequent phases: 1) Decomposition of end-to-end QoS constraints into local constraints on the com-

ponent service level, and 2) selection of best service candidates that satisfy all local constraints and optimize the overall value of a given utility function. The local search is performed in parallel for each service group to further improve the performance.

Secondly, we define QoS-based dominance relationship between services, following the skyline query model [BKS01], to determine the most relevant candidates, which we denote as skyline services. By pruning all non-skyline services from the search space we are able to reduce the computation time significantly. In order to handle situations, where the number of skyline services may still be too large, we present a clustering-based method for selecting representative services from the skyline. Moreover, we present a skyline-based method for extracting local quality levels of a group of candidate service to improve the success rate of the aforementioned hybrid approach.

In addition, we provide a mechanism for assisting the providers of non-skyline services in improving the competitiveness of their service, by measuring the required improvement in QoS attributes (and the associated cost) in order to bring their services into the skyline set.

1.2.2 Decentralized Concurrency Control for Web Service Compositions

The contribution of this thesis for maintaining consistency of concurrent web service-based business transactions is twofold.

Firstly, we propose extending the current standard web service transactions framework by introducing a new component, the WS-Scheduler, to implement the service-level concurrency control. The WS-Scheduler resides on the web service providers side and is responsible for managing the transactional dependencies and ensuring the consistency of concurrent invocations to local web services. The new architecture is grounded to a solid well-known multi-level transaction model, which allows the separation of the concurrency control responsibilities among the different layers in a multi-layer architecture.

Secondly, we present an optimistic concurrency control mechanism that can be deployed in a fully decentralized fashion in the extended web service transaction framework. More specifically, we propose a distributed serialization graph testing protocol, which applies a commit-differing policy for concurrency control in order to ensure global consistency of concurrent transactions.

In addition, distributed algorithms for handling global dependency cycles are also presented. Unlike existing solutions, our algorithms avoid any direct communication between independent business transactions, and thus prevent disclosing any possibly confidential business relationships among the participants of a business activity.

1.3 Structure of the Thesis

The rest of the thesis is organized as follows. In Chpater 2 we introduce general notions from the area of Service-Oriented Architecture and web services. We also review relevant industrial standards for web service composition. Related work and existing solutions to the research problems addressed in this thesis are also reviewed.

In Chapter 3 we present our contribution in addressing the first problem, i.e. QoS-based service composition problem. We start by a short motivation and introduction to the problem in Section 3.1 and provide a formulation of the problem in Section 3.2. We then introduce our novel hybrid approach for tackling the problem in Section 3.3. Next, in Section 3.4 we show how using the skyline model can help in further improving scalability of both the standard QoS optimization approach as well as our hybrid approach. In Section 3.4.4 we tackle the problem from the service provider's side and present algorithms for assisting service providers in improving the competitiveness of their services. Finally, in Section 3.5 we provide a performance study of the proposed solutions based on extensive experimental evaluation.

In Chapter 4 we address the second problem, i.e. transactional management for web service-based processes. We start by introducing a motivating scenario in Section 4.1. We then describe the current transaction framework for web services and introduce our proposed extension based on the multi-level transaction model in Section 4.2. In Section 4.3 we present our optimistic protocol for distributed concurrency control as well as algorithms for detecting global dependency cycles. Finally, in Section 4.4 we describe our experimental evaluation of the presented solutions and detailed analyses of the results.

In Chapter 5 we give a short summary of the contribution of the thesis and an outlook to future research directions and possible improvements to the presented solutions.

2 Background and State-of-the-art

This chapter will introduce background information of the research problems addressed in this thesis. We will also clarify some general terms used in this thesis such as Quality of Service and Web Service Transactions and explain their relation to web service composition. A review of the state-of-the-art in both industrial standards as well as related research work will also be presented.

2.1 SOA and Web Services

Services are first-class citizens in Service-Oriented Architectures (SOA). According to SOA principles, functionalities are exposed as services that can be described, discovered and consumed by different users. There are three main roles and associated set of operations in the service life cycle as illustrated in Figure 2.1: a *service provider*, a *service client* and a *service broker* [Tea, TP02].

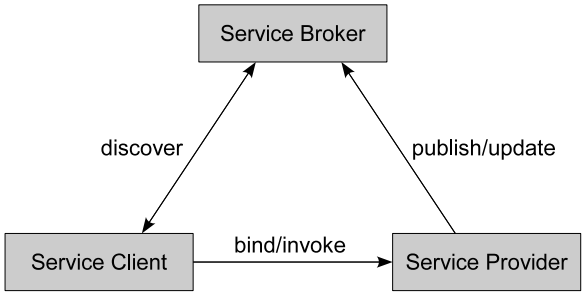


Figure 2.1 Main Roles and Operations in SOA

- ***Service provider***: From the business perspective, this is the owner of the service. Service providers encapsulate parts of their business activities and expose them as programmatically accessible components using standard interfaces. Providers then publish the functional descriptions of their services in order to make them available for potential users.

- ***Service client***: From the business perspective, this is the party that has a need that can be fulfilled by using one of the published services. Service clients send their inquiries to the service brokers, who in turn perform a matchmaking between the requests and the offers and return a list of matching services to the client. The clients then contact the service providers directly to use the offered functionalities.

- ***Service broker***: This is the party that provides storage, indexing and search functionalities for service providers and requesters. Service brokers maintain service descriptions in searchable repositories and apply matchmaking algorithms for finding relevant service offers for each service request.

Web services have recently been widely accepted by both industry and academia as a means of realizing the SOA concepts. The World Wide Web Consortium (W3C) defines a web service as follows [BHM+]:

> *'A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.'*

Using standard web service technologies for implementing the basic interactions and communications between the different parties allows developers to design SOA-based applications in a language and platform independent way. Figure 2.2 shows the conceptual architecture stack of web services [BHM+]. The first layer in this stack is the communication layer. Web services use standard transport protocols like HTTP for communication over the web. The Messages, Description and Composition layers use XML-based standard formats. In the messages layer, service clients and

service providers use the Simple Object Access Protocol (SOAP) [BEK+], which is an XML-based protocol for exchanging messages. Service providers use the Web Service Description Language (WSDL) [CCMW] to describe the web interface of their services, and publish them in public or enterprise service registries such like Universal Description, Discovery and Integration (UDDI) [Coma] repositories. Service clients define processes and compositions that involve several services using XML-based process modeling languages such as the Business Process Execution Language for Web Service (WS-BPEL) [Comb].

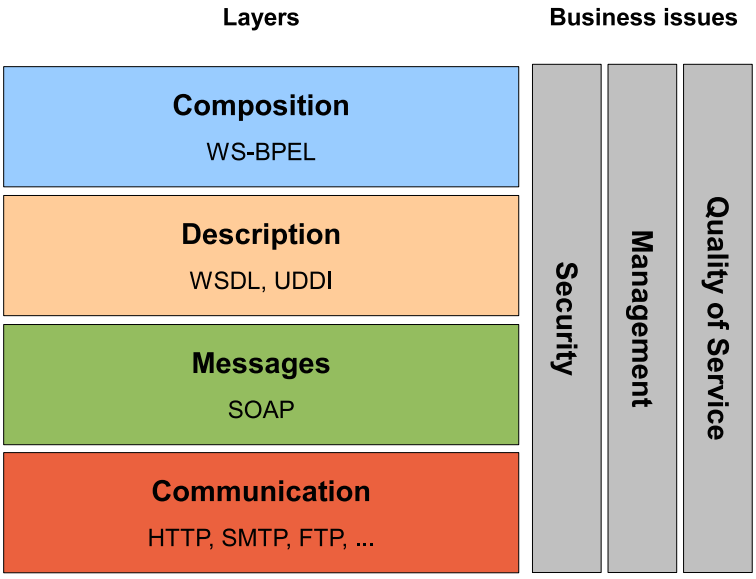


Figure 2.2 Web Services Architecture Stack

In addition to the standards and specifications that support the basic activities for describing, publishing and interacting with web services, there has been also a wide range of specifications for supporting non-functional aspects such as security, management and coordination and Quality of Service (QoS). These aspects are of great importance for building reliable web service-based applications. Examples of industrial specifications that deal with the non-functional properties of web services are the Web Services Coordination (WS-Coordination) [Comg], Web Service Atomic Transactions (WS-AtomicTransaction) [Come] and Web Service Business Activity (WS-BusinessActivity) [Comf] for transactional management of web services. There is also Web Services Security (WS-Security) [Comd] and Web Service Reliable Messaging (WS-ReliableMessaging) [Comc] for supporting secure and reliable service execution respectively. In addition, Web Services Policy (WS-Policy) [VOH+] extends WSDL

to allow the encoding and attachment of QoS information to service descriptions and Web Services Agreement (WS-Agreement) [ACD+] provides an XML-based language for establishing a contractual agreement on the expected services level between service clients and service providers .

2.2 Web Service Composition

One of the important features of web services is their reusability and composability, i.e. the ability to combine a set of available services in order to solve more complex problems and create new value-added services. The services that are created by aggregating other services are referred to as *composite services*, while the process of aggregating and combining services is referred to as *service composition*. Web service composition accelerates the development of complex applications and allows providers of business applications to focus on their core activities and delegate the execution of other activities to third-party services. For example, the providers of on-line shop applications usually delegate the delivery of the sold products to transport companies and focus on their core activities such like the management of the orders, products, billing and accounting issues. In this section we give an overview of the web service composition process and discuss some related issues and research problems.

Figure 2.3 gives a conceptual overview of the web service composition. At design-time the design and structure of the composition is defined either manually or automatically. In the latter case, AI planning and Semantic Web techniques are often used to describe the pre-conditions and goals of the composition, decompose the overall task into sub-tasks and locate appropriate web services for implementing these sub-tasks. The automatic composition methods are useful for composing web services on the fly in order to fulfill arbitrary complex tasks that are not known a priori. A survey of such methods can be found in [RS04].

On the other hand, there are many (business) applications, which have a pre-defined set of tasks and activities structured in a pre-defined set of (business) processes. Each process is designed for a specific goal (e.g. purchasing a product from an online shop). In such applications, automatic composition of services is not needed. A workflow-like language (e.g. WS-BPEL [Comb] and WSML [Gro]) is usually used to model the abstract representation of the composition including a functional description of the required services and the expected inputs and outputs etc. Next,

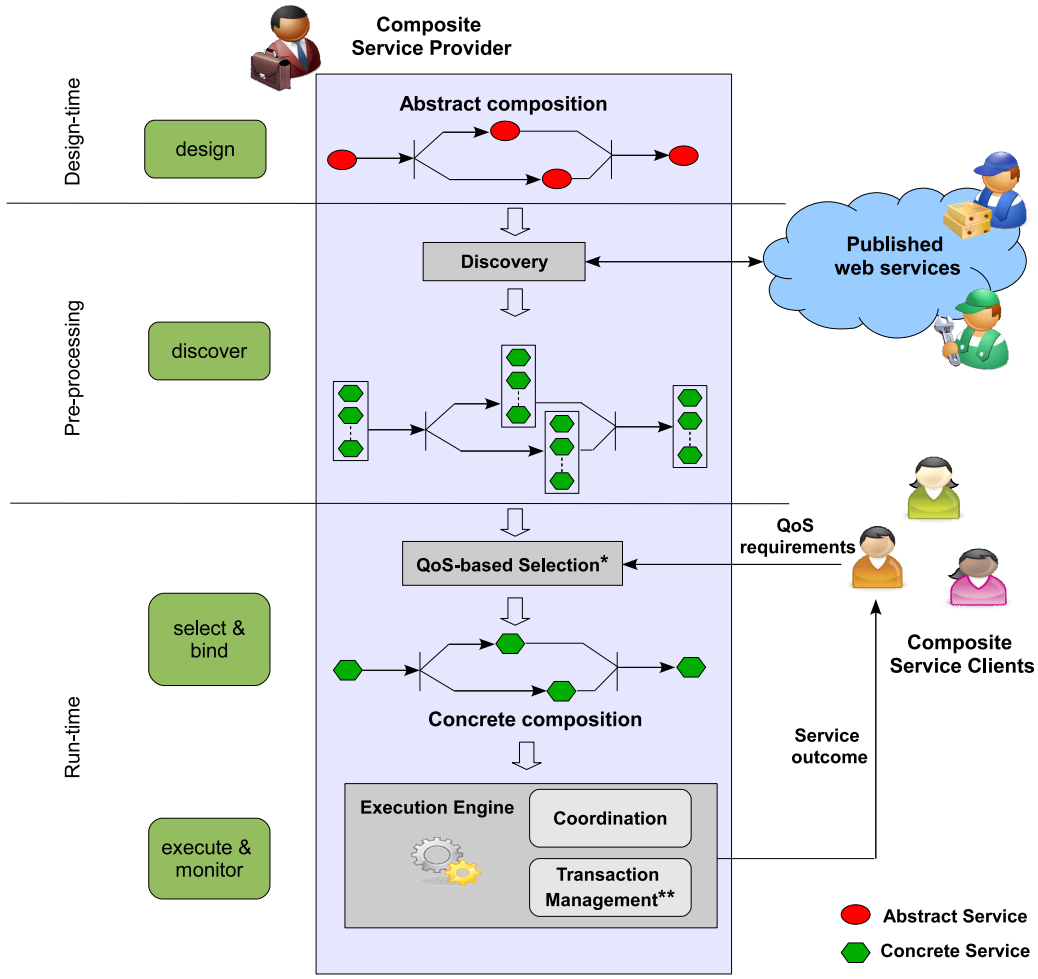


Figure 2.3 Conceptual Overview of Web Service Composition
* Focus of 1st part of the thesis
** Focus of 2nd part of the thesis

service discovery is performed by exploiting the existing infrastructure (e.g. UDDI) to locate available web services for each task in the workflow using syntactic (and probably semantic) functional matching between the tasks and service descriptions. As a result, a list of functionally-equivalent web services (referred to as *candidate services*) is obtained for each task.

Unlike *static composition*, where abstract services are bound to concrete web services already at design-time, in *dynamic composition* the decisions about service selection are made at run-time based on users' requirements. This increases the flexibility and adaptability of the designed system. At run-time, users of the dynamic composite service specify their QoS requirements in terms of end-to-end constraints on

the different QoS parameters (e.g. total average response time, supported security protocols, total cost etc.). The goal of QoS-aware service selection in this stage is to select one component service from each list of candidate services such that the aggregated QoS values satisfy the user's end-to-end QoS requirements. The efficiency of the QoS-aware service selection methods will be **the focus of the first part of this thesis**.

After binding the selected services, the actual execution of the composite service can start. Process execution engines are usually used for monitoring and controlling the execution of the service composition (eg. BPEL Engine for WS-BPEL models and WSMX [1] for WSML models). This activity is referred to as the *orchestration* [GA04] of the involved services. As the interactions between web services in a composition may span distributed applications and/or organizations, and result in long-lived distributed transactions, careful coordination and transactional management of the involved services is usually needed. The distributed coordination of concurrent transactional web service compositions will be **the focus of the second part of this thesis**.

2.3 State-of-the-art

This section will give a review of the state-of-the-art in the area of web service composition. A special focus will be given to the key related work addressing the problems addressed by this thesis. We will discuss the advantages and disadvantages of the related work and explain why do we need to readdress the same problems.

2.3.1 Quality of Service

Quality of Service (QoS) is one of the most important subsets of the non-functional properties of web services. The term QoS is traditionally used to refer to network related issues such as bandwidth, reliability, message routing etc. In the context of SOA the term is used to refer to the quality of the service delivery in terms of some indicators such as response time, availability of the service, throughput (i.e. number of processed requests per sec) etc. These properties can have a great impact on user's experience and thus play an important role in differentiating between similar

[1] http://www.w3.org/Submission/WSMX/

web services. Consequently, managing and optimizing QoS levels of the offered web services becomes also important for service providers to stay competitive.

In the recent years, several, solutions, specifications and frameworks have been proposed for modeling, monitoring and negotiating the different QoS aspects of web services. In the following we give a brief review of the state-of-the-art in this field.

In general, QoS metrics can be classified into three categories, based on the approaches to obtaining them [ZLC07]:

- Provider-advertised metrics: this type of metrics is usually provided by service providers, which is subjective to service providers. An example is the execution prices advertised by service providers.

- Consumer-rated metrics: this type of metrics can be computed based on service consumer's evaluations and feedback, which is therefore subjective to service consumers. For example, the service reputation according to service consumers' evaluations.

- Observable metrics: this type of metrics can be observed, i.e., computed, based on monitored service operational events, which is objective to both service providers and consumers.

The monitoring can be conducted at the service consumer and/or service provider side or by a third-party. In [MRLD09] a framework that combines the advantages of client- and server-side QoS monitoring was presented. A service QoS monitoring system, which provides a user-friendly programming model that allows users to define the QoS metrics and associated evaluation rules is presented in [ZLC07].

In [LNZ04] the authors propose an extensible QoS computation model that supports open and fair management of QoS data. Figure 2.4 gives an conceptual overview of a QoS-aware service infrastructure as suggested in [LNZ04]. The QoS information of web services are stored in a QoS repository, which is maintained by a service broker and associated to the published service descriptions. The QoS information are either published by the service providers as part of the service description or collected from service consumers' feedback.

As WSDL does not provide a means for describing the non-functional properties of web services, there have been some proposals for extending the WSDL model to support QoS descriptions. An example is the work presented in [D'A06], where

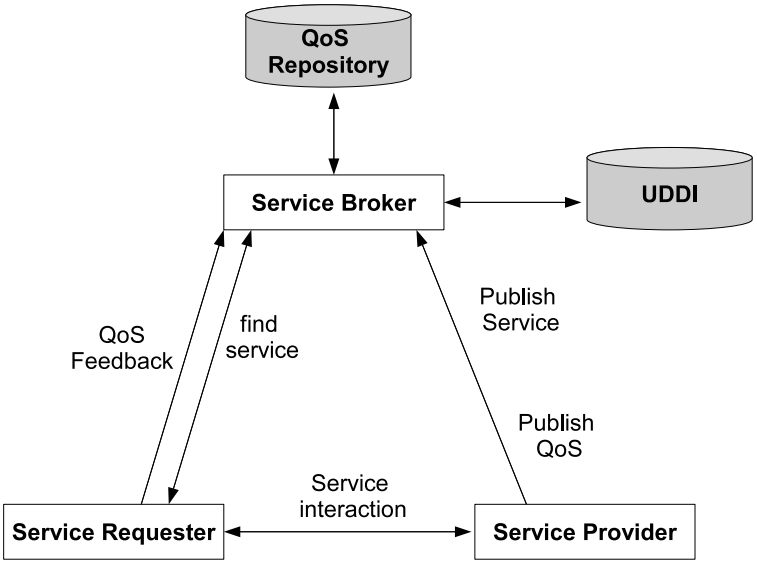


Figure 2.4 QoS-Aware Service Infrastructure

the WSDL extension is carried out as a meta-model transformation, according to principles and standards recommended by the Model Driven Architecture (MDA). In [ZCL04, BS04] ontology-based representations for describing QoS properties and requests were proposed to support semantic and dynamic QoS-based discovery of web services. Another solution is the WS-Policy framework [VOH+], which can be used for expressing the QoS attributes of web services. A policy is defined as a collection of alternatives which is, itself, defined as a collection of assertions. An assertion is used to represent a requirement, capability or a behavior of a Web service. Example of such a solution is the work presented in [CBB08].

Standards and frameworks for modeling and expressing Service Level Agreements for web services also exist. Examples are the Web Service Service Level Agreement (WSLA) language and framework and the Web Services Agreement Specification (WSAgreement). The WSLA framework was mainly proposed by IBM [LKD+], whereas the WS-Agreement was developed by the Global Grid Forum [ACD+]. The WS-Agreement specification provides a more expressive language for specifying expected quality levels. It defines an extensible XML language and protocol for establishing Service Level Agreements (SLA) between service consumers and service providers. It allows specifying the nature of the agreement and agreement templates to facilitate discovery of compatible agreement parties. An agreement specifies one or more service level objectives (SLO), which state the requirements and capabilities

of each party on service qualities [ACD+, OVSH06]. For example, an agreement may provide assurances on the bounds of service response time and/or service availability. WS-Agreement is more expressive than previous policy standards because it allows defining scopes for which the guarantees on service level holds, preconditions for the guarantees on SLO to be valid and business values such as penalties and rewards, which incur if the SLO is not satisfied [ACD+, OVSH06].

A key difference between the WS-Agreement and WSLA is that the former does not provide a means to specify the metrics associated with parameters used in the agreement. Instead, metrics are defined in any structure required by a domain-specific extension.

Related Work

In addition to the above mentioned web service specifications and standards, Quality of Service management has been widely discussed in the area of middleware systems [ACH98, CS01, CN01, GWW02]. Most of these works focus on QoS specification, representation, monitoring and management. However, the focus of the work presented in this thesis is not on these issues, rather on the design of web service selection algorithms for composite applications with end-to-end QoS constraints. Recently, the QoS-based web service selection and composition in service-oriented applications has gained the attention of many researchers [ZBD+03, ZBN+04, LNZ04, AP05, AP07, YZL07a, KP09, ZZL09].

Two general approaches exist for the QoS-aware service composition: local selection and global optimization.

Local Selection: The local selection approach is especially useful for distributed environments where central QoS management is not desirable and groups of candidate web services are managed by distributed service brokers [BSND02, LYSS07]. The idea is to select one service from each group of service candidates independently on the other groups. Using a given utility function, the values of the different QoS criteria are mapped to a single utility value and the service with maximum utility value is selected. This approach is very efficient in terms of computation time as the time complexity of the local optimization approach is $O(l)$, where l is the number of service candidates in each group. Even if the approach is useful in decentralized environments, local selection strategy is not

suitable for QoS-based service composition, with end-to-end constraints (e.g. maximum total price), since such global constraints cannot be verified locally.

Global Optimization: The global optimization approach was put forward as a solution to the QoS-aware service composition problem [ZBD+03, ZBN+04, AP05, AP07, KP09]. This approach aims at solving the problem on the composite service level. The work of Zeng et al. [ZBD+03, ZBN+04] focuses on dynamic and quality-driven selection of services. The authors use global planning to find the best service components for the composition. They use Integer Linear Programming techniques (ILP) [NW88] to find the optimal selection of component services. Similar to this approach Ardagna et al. [AP05, AP07] extend the Integer Linear Programming model to include local constraints. In [KP09] Kritikos and Plexousakis claim that (mixed) Integer Linear Programming should be used as a matchmaking technique instead of Constrained Programming (CP) [RBW06] and provide experimental results proving it. In [ZZL09] Zhai et al. propose a solution for repairing failed service compositions by replacing the failed services only and reconfiguring the composition in a way that still meets the user's end-to-end QoS requirements. The reconfiguration of the composition and the suggestion of new services is based on ILP. Generally, ILP methods are very effective when the size of the problem is small. However, these methods suffer from poor scalability due to the exponential time complexity of the applied search algorithms [Mar03]. Already in larger enterprises and even more in open service infrastructures with a few thousands of services the response time for a service composition request could already be out of the real-time requirements.

Heuristic Solutions:

As discussed earlier, the problem of QoS-aware service selection can be modeled as a Multiple-choice Multi-dimensional Knapsack Problem (MMKP). In MMKP problem, a set of groups of items, where each item has a profit value and consumes some resources exist. The goal of this problem is to select exactly one item from each group such that the total profit value is maximized under some constraints on the total resource consumptions. The groups and items in this problem correspond to the service classes and the candidate services in the web service scenario respectively. The profit value of an item corresponds to the utility value of a web service and the constraints on the resource consumption correspond to the QoS constraints.

There exist a number of heuristics in the literature for solving the Knapsack Problem in general and the MMKP variant of this problem in particular. In [Kha98] a heuristic named HEU for solving the MMKP was presented. HEU uses a measurement called *aggregate resource consumption* to decide upon which item from each group should be upgraded in each round of selection. In [KLMA02] a modified version of HEU named M-HEU was presented, where a pre-processing step to find a feasible solution and a post-processing step to improve the total value of the solution with one upgrade (i.e. item selection that increases the total profit value) followed by one or more downgrades (i.e item selection that decreases the total profit value) were added. In [ARK+06] the authors propose another heuristic for solving the MMKP named C-HEU and evaluate its performance and optimality against several heuristics including the M-HEU algorithm. The results of their evaluation show that C-HEU outperforms M-HEU in terms of computation time. However, the experiments have also shown that M-HEU produces the nearest to the optimal solution among all the heuristics, while the optimality of C-HEU decreases as the number of items in each group increases. Furthermore, the results have shown that the C-HEU algorithm performs better in systems, where the objective value to be maximized (i.e. the utility value in the web service scenario) is not proportional to the resource requirements (i.e. the QoS values of web services). Since the utility value if a given web service is proportional to the QoS level of the service, the C-HEU algorithm is not applicable to the QoS-aware service selection problem.

A modified version of the M-HEU algorithm named WS-HEU, designed for the QoS-aware service selection problem was proposed in [YZL07a]. The authors propose two models for the QoS-based service composition problem: 1) a combinatorial model and 2) a graph model. A heuristic algorithm is introduced for each model: the WS-HEU algorithm for the combinatorial model and the MCSP-K for the graph model. The time complexity of WS-HEU is polynomial, whereas the complexity of MCSP-K is exponential. Despite the significant improvement of these algorithms compared to exact solutions, both algorithms do not scale with respect to an increasing number of web services and remain out of the real-time requirements. In our experimental evaluation, which we present in Section 3.5.2 we compare our hybrid approach (Section 3.3) against the WS-HEU algorithm. The results indicate the the hybrid approach outperforms the WS-HEU.

Moreover, the WS-HEU algorithm is not suitable for the distributed setting of

web services. This due to the fact that WS-HEU (following the original M-HEU algorithm) starts with a pre-processing step for finding an initial service combination that satisfies all constraints but not necessarily is the best solution, and improves this solution in several rounds of upgrades and downgrades of one of the selected component services. Applying this algorithm in a distributed setting where the QoS data of the different service classes is managed by distributed service brokers would raise very high communication cost among these brokers to find the best composition. The hybrid approach, which we propose in this thesis solves the composition problem more efficiently and fits well to the distributed environment of web services.

2.3.2 Web Service Transactions

A *business transaction* can be defined as a consistent change in the state of the business that is driven by a well-defined business function [Pap03]. Ordering some product from a company is an example of a business transaction. The state of the back-end database of the selling company is affected by the execution of this transaction. More complex business transactions may involve a sequence of several activities such as processing orders, managing payments and shipping products. A *web service transaction* is a business transaction that is implemented as a web service composition and executed by invoking one or more web services.

In composite web services, transactional dependencies may occur among the involved services. Such dependencies can affect the overall outcome of the composite service. Consider for example a travel planning application that is composed of two basic services: a web service for booking flight tickets and a web service for booking hotel rooms. Although each of these services can be provided by a different business partner, there exists a strong dependency relation between them within the context of the invoking application. If, for example, the payment for booking the flight ticket fails, any successful reservation of the hotel rooms need to be canceled. Given that web services are inherently autonomous, heterogeneous and loosely-coupled, the coordination and transactional management of services that share a common context becomes essential.

For this purpose, the Organization for the Advancement of Structured Information Standards (OASIS) [2] has approved a set of specifications as the de facto standards

[2] http://www.oasis-open.org

for managing and coordinating web service transactions. In the following we give an overview on these standards and the relations between them (see Figure 2.5).

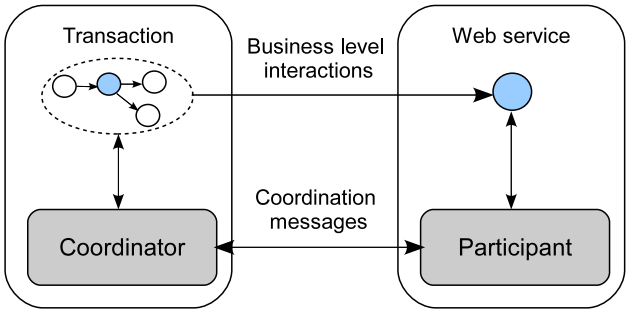


Figure 2.5 Web service transactions framework

- **WS-Coordination** [Comg] defines a framework that provides a coordination context for loosely coupled partners in a distributed application. The framework defines two key concepts:

  1. The coordinator is responsible for creating the context and coordinating the different partners according to the applied protocol. The coordinator role can be taken by the initiator of a distributed application or by a (trusted) third party.
  2. The participant is an entity that resides on the web service provider side and represents an instance of the web service that has been invoked within the distributed application. This entity is responsible for communicating with the coordinator according to the applied protocol on behalf of the web service.

  The WS-Coordination framework enables participants to reach consistent agreement on the outcome of distributed activities. The coordination protocols that can be defined in this framework can accommodate a wide variety of activities, including protocols for simple short-lived operations and protocols for complex long-lived business activities. There are currently two transactional coordination types: *WS-AtomicTransaction* [Come] and *WS-BusinessActivity* [Comf].

- **WS-AtomicTransaction** [Come] is intended for short-duration interactions among trusted partners, which is referred to as *Atomic Transactions*. Atomic transactions have an all-or-nothing property. The actions taken by a transaction

participant prior to commit are only tentative; typically they are neither persistent nor made visible outside the transaction. When an application finishes working on a transaction, it requests the coordinator to direct all participants to either all commit or all cancel using the well known Two Phase Commit (2PC) protocol. The coordinator determines if there were any processing failures by asking the participants to vote. If the participants all vote that they were able to execute successfully, the coordinator commits all actions taken. If a participant votes that it needs to abort or a participant does not respond at all, the coordinator aborts all actions taken. Commit directs the participants to make the tentative actions final so they may, for example, be made persistent and be made visible outside the transaction. Abort directs the participants to make the tentative actions appear as if they never happened.

- **WS-BusinessActivity** [Comf] on the other hand is intended for long-duration activities that apply business logic to handle exceptions that occur during the execution of activities of a business process. Actions in such long-running business activities are applied immediately and their effects are made immediately visible. Compensating actions may be invoked in the event of an error. However, the assumption that all service operations can always be compensated is not realistic. When the number of transactions having access to intermediate results increases, the compensation of some operations becomes either too expensive or even impossible. This raises the need to a concurrency control mechanism for web service transactions.

Related Work

Ensuring a fault-tolerant execution of Web service transactions has been the focus of recent research work (e.g. [BPG05, MM06, SDN08, CKJ+08]). The adopted transaction models in these works rely on the notion of compensations [Elm92, AAA+96, Gra88] which are triggered whenever a subset of a transaction fails. Compensations are introduced either at the client level as part of the business process execution [KHC+05] or on both, client and participant sides [SDN08]. However, maintaining consistency of the concurrent transactions is neither addressed by these papers nor by the existing industrial specifications. An advanced database transaction model that deals with both atomicity and consistency of distributed applications is the Con-

Tract model [Elm92]. This model provides a user defined mechanism to control the correctness of the distributed transaction. The transaction initiator specifies so-called invariant predicates which have to remain unchanged from his application specific view to insure correctness. However, this model is not practical for loosely-coupled and autonomous Web services, where service providers cannot accept such predicates on their local resources by the clients. Moreover, as the service implementation is usually not visible to the service consumers, it is difficult to specify the right invariant predicates.

Similar to our work in addressing the concurrency control problem of Web services is the work in [CJK+05] and [HST05]. Both solutions share the idea of handing over the concurrency control to the transaction coordinators, who in turn maintain and update local partial views of the global serialization graph by direct communication among them. The main disadvantage of these approaches is that they rely on information exchange among independent transactions to decide upon committing or aborting transactions. We argue that the assumption that independent transactions would like to exchange information about their own business relations and activities is unrealistic. The exchanged dependency information can be interpreted as mission-critical information such as confidential contracts between organizations. In contrast to this approach, the solution presented in this thesis separates the roles of the transaction coordinators (commitment protocol) and transaction schedulers (concurrency control protocol) and does not require any direct communication or information exchange between independent transactions. Deciding upon committing or canceling transactions as well as detecting possible global dependency cycles are accomplished in our approach without disclosing any business related information.

3 Efficient QoS-aware Service Selection

In this chapter we will address the problem of QoS-aware web service composition. More specifically, we will focus on improving the efficiency of the web service selection algorithms. We start in Section 3.1 by introducing the problem and presenting a motivating example. Then, in Section 3.2 we provide a formulation of the problem and definition of what is considered as an optimal selection for a given composition request. After describing the problem, we introduce our contributions in solving this problem in Sections 3.3, 3.4, 3.4.4. In Section 3.5 we present the results of our experimental evaluation of the proposed solutions. The presented solutions in this chapter have been mainly published in [AR09] and [ASR10].

3.1 Introduction

In Section 2.2 we presented the conceptual overview of web service composition and discussed all the steps starting from the abstract modeling of the composition at design-time until the execution and delivery of the services outcome. For the ease of reference we show the conceptual overview here in Figure 3.1 with the focus on the service selection part.

The run-time QoS-aware service selection aims at providing an adapted instantiation of the composed service to the users based on their QoS preferences. Given an abstract representation of the service composition, which can be stated as an abstract process using any process modeling language, and a list of candidate web services for each task in the process, the goal of the QoS-aware service selection is to select one service from each list such that the aggregated QoS values satisfy the

user's end-to-end QoS requirements.

However, as the number of similar services providing the same functionality increases, the total number of possible combinations to be considered by the selection algorithm increases exponentially. Performing exhaustive search to find the best combination of services is, therefore, impractical.

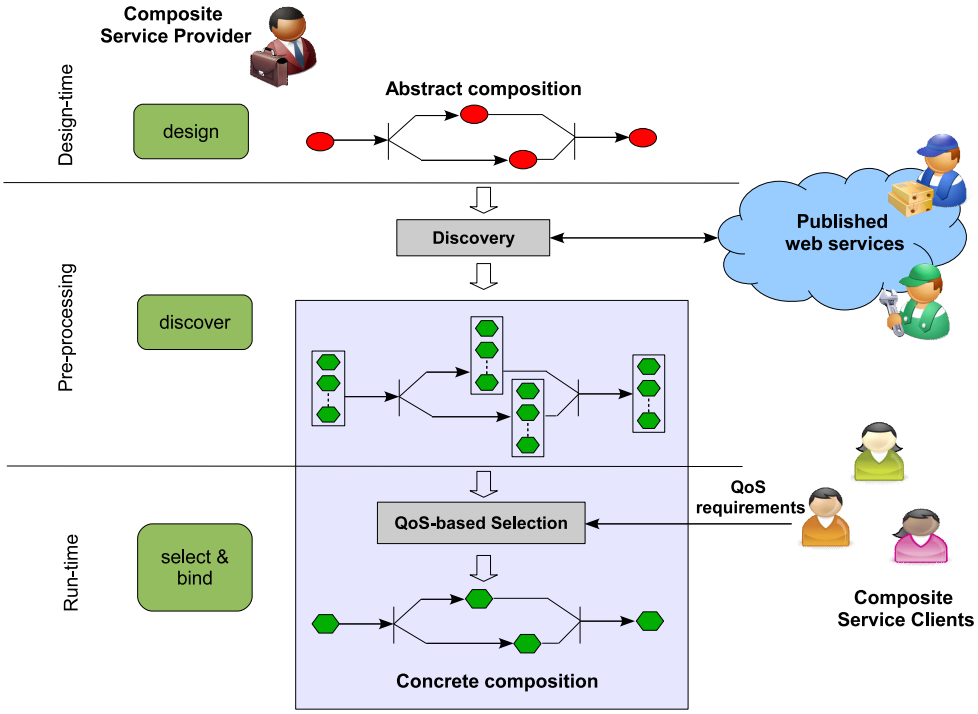


Figure 3.1 QoS-based Selection - Conceptual Overview

The QoS-aware service selection for web service compositions problem is a combinatorial problem [YZL07b], which can be modeled as a Multiple-choice Multi-dimensional Knapsack Problem (MMKP) (a variant of the classical 01 Knapsack Problem) [MT90]. Given a set of n groups of items. Each item has a particular value and it requires some resources. The objective of the MMKP is to pick exactly one item from each group for maximum total value of the collected items, subject to a set of resource constraints of the knapsack. Similarly, the goal of QoS-aware service selection is to pick exactly one service from each list of candidate services for each task in the composition model. The aggregated QoS values of the selected services must meet the user's QoS constraints. The MMKP is known to be NP-hard problem [MT90], therefore, any exact solution to the QoS-aware service selection problem is expected to have a n exponential computational complexity with respect to the

size of the problem, i.e. the number of composed tasks and the number of candidate services per task.

As the selection and binding of web services has to be performed at run-time, the efficiency of the applied selection mechanism becomes crucial to the performance of the composition engine, and hence, affects the user's experience of the system.

Example 1. Consider the example shown in Figure 3.2 of a web application for finding used car offers. The users submit their requests to the system, specifying some criteria for selecting the cars (e.g. brand, type, model). The system then returns a list of the best offers along with a credit and an insurance offer for each car on the list. The composed application can be exposed to users as a web service, API or widget, programmatically accessible or directly integrated into their web applications using a Mashup tool for example.

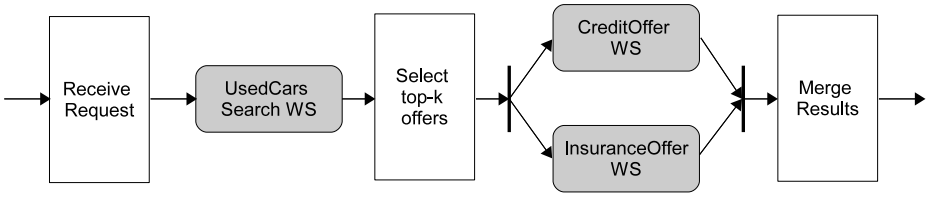


Figure 3.2 Example of Service Composition

In this example, some tasks, illustrated as gray boxes in Figure 3.2, are outsourced and integrated via web service calls. For these outsourced tasks, multiple services may be available providing the required functionality but with different QoS values such as response time, maximum number of requests per minute and price.

Users of the composed service (i.e. the used cars application in our example) are typically unaware of the structure of the application and, therefore, specify their QoS requirements in terms of end-to-end constraints in a the format of a Service Level Agreement (SLA). In order to fulfill the user's QoS requirements, the provider of the composed service creates an adapted instance of the composed application by selecting appropriate service providers for each of the outsourced services, (i.e. the UsedCars Search WS, CreditOffer WS and InsuranceOffer WS). The selection is performed by matching the QoS information of the candidate services with the uesr's QoS constraints and, therefore, has to be carried out at run-time. Furthermore, a re-selection of the services may be required at run-time to adapt to any changes in the QoS of the selected services (e.g. some services become slower because of the load or some of the selected services become unavailable). A quick response to such

adaptation requests is important in order to continue delivering the services at the promised quality level in the SLA.

3.2 Problem Formulation

Assume a set $\mathbb{S}$ of *service classes*, which classify the universe of available web services according to their functionality. Each service class $S_j = \{s_{j_1}, ..., s_{j_n}\}, S_j \in \mathbb{S}$, consists of all web services that deliver the same functionality (e.g. used car search) but potentially differ in terms of non-functional properties. Some service providers might provide the same service in different quality levels, e.g. at different response times and different prices. For the sake of simplicity, we model each variation of the service as a different service. According to the SOA principles, descriptions of functional and non-functional properties of web services are stored and managed by service registries (e.g. UDDI registries), which are maintained by *service brokers* [LNZ04, LYSS07]. We assume that service brokers maintain and update information about existing service classes and candidate services of each class in their registries, making them accessible to service requesters.

3.2.1 Abstract vs. Concrete Composite Services

As shown in Figure 3.1 we distinguish in the context of web service composition between the following two concepts:

- An abstract composite service $P = \{S_1, \ldots, S_n\}$, which is an abstract representation of a service composition. P describes the structure of the composition and the service classes $S_1, \ldots, S_n$, which are required to achieve the different tasks in the composition.

- A concrete composite service $CS = s_1, \ldots, s_n$, which is an instantiation of an abstract composite service P. CS is obtained by binding each abstract service class S_j in P (e.g. used car search service) to a concrete web service s_j (e.g. AutoScout24 WS or Mobile.de WS), such that $s_j \in S_j$.

3.2.2 QoS Parameters

We consider a set quantitative non-functional properties of web services Q, which describe the quality criteria of a web service [ZBD+03, LNZ04]. These can include generic QoS attributes like response time, availability, price, reputation etc, as well as domain-specific QoS attributes, for example bandwidth for multimedia web services, as long as these attributes can be quantified and represented by real numbers. QoS attributes may be positive or negative. The values of positive attributes need to be maximized (e.g. throughput and availability), whereas the values of negative attributes need to be minimized (e.g. price and response time). For simplicity, in this thesis we assume that all attributes are negative (positive attributes can be easily transformed into negative by multiplying their values by -1). We use the vector $Q_s = \{q_1(s), \ldots, q_r(s)\}$ to represent the QoS values of service s. The function $q_i(s)$ determines the value of the i-th attribute of the service s. These values can be either collected from service providers directly (e.g. price), from previous execution monitoring records (e.g. average response time), or from user feedbacks (e.g. reputation) [LNZ04].

3.2.3 QoS Computation of Composite Services

The QoS vector for a composite service $CS = \{s_1, ..., s_n\}$ is defined as $Q_{CS} = \{q'_1(CS), \ldots, q'_r(CS)\}$, where $q'_i(CS)$ is the estimated end-to-end value of the i-th QoS attribute and can be aggregated from the QoS values of the component services $\{s_1, ..., s_n\}$. The computation of the end-to-end QoS values depends on the structure of the composite service.

In our study we consider the following four elementary composition constructs, which can be used for building more complex compositions:

1. *Sequential*: a sequence of services $\{s_1, ..., s_n\}$ are executed in a strict sequential order one after another.

2. *Loop*: a block of one or more services is executed repeatedly up to a maximum number of k executions. The aggregated QoS values of a loop construct are computed based on the worst case scenario, where the number of iterations equals k.

3. *Parallel* (and split/and join): multiple services $\{s_1, ..., s_n\}$ are executed concurrently and merged synchronization.

4. *Conditional* (exclusive split/exclusive join): a set of services $\{s_1, ..., s_n\}$ are associated with a logical condition, which is evaluated at run-time and based on its outcome one service is executed. The estimated QoS values of a conditional construct are the worst values of the services $\{s_1, ..., s_n\}$. For example, the estimated execution price of the conditional construct is computed as the price of the most expensive service among the services $\{s_1, ..., s_n\}$.

Depending on the QoS attribute and the composition pattern, there can be three different types of aggregation relations: 1) summation, 2) product or 3) minimum/maximum relations. Table 3.1 shows examples of these aggregation functions.

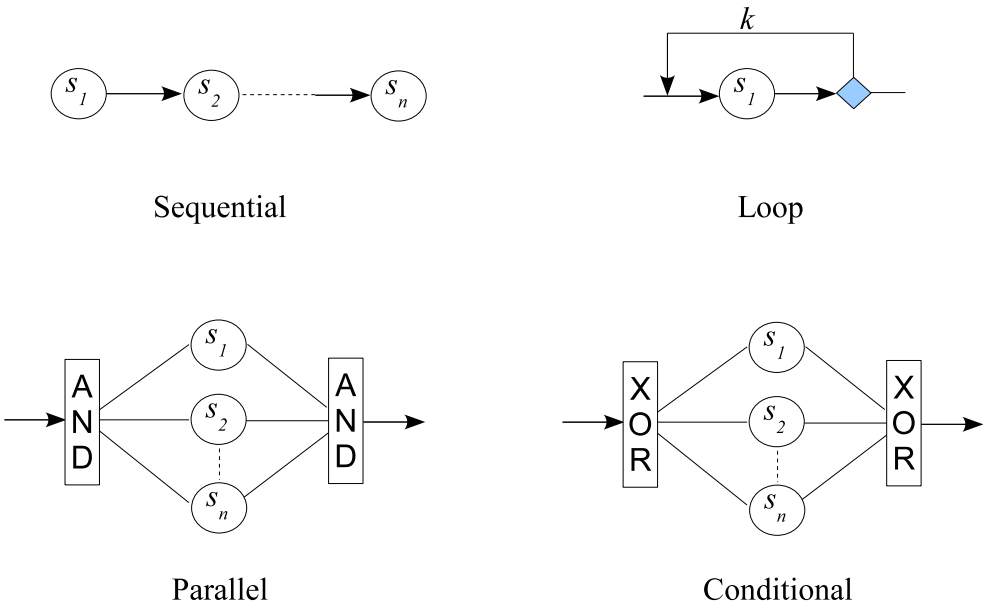


Figure 3.3 Composition Patterns

3.2.4 Global QoS Constraints

Global QoS constraints represent user's end-to-end QoS requirements. These can be expressed in terms of upper (and/or lower) bounds for the aggregated values of the different QoS criteria. As mentioned earlier, we only consider negative QoS criteria.

QoS Attribute	Aggregation Function			
	Sequential	Loop	Parallel	Conditional
Response Time	$\sum_{j=1}^{n} q(s_j)$	$\sum_{i=1}^{k} q(s)$	$\max_{j=1}^{n} q(s_j)$	$\max_{j=1}^{n} q(s_j)$
Price	$\sum_{j=1}^{n} q(s_j)$	$\sum_{i=1}^{k} q(s)$	$\sum_{j=1}^{n} q(s_j)$	$\max_{j=1}^{n} q(s_j)$
Availability	$\prod_{j=1}^{n} q(s_j)$	$\prod_{i=1}^{k} q(s)$	$\prod_{j=1}^{n} q(s_j)$	$\min_{j=1}^{n} q(s_j)$
Throughput	$\min_{j=1}^{n} q(s_j)$	$q(s)$	$\min_{j=1}^{n} q(s_j)$	$\min_{j=1}^{n} q(s_j)$

Table 3.1 Examples of QoS aggregation functions

Therefore in our model we only have upper bound constraints.

Definition 1. (Feasible Selection) For a given abstract composition $P = \{S_1, \ldots, S_n\}$ and a given vector of global QoS constraints $C' = \{c'_1, \ldots, c'_m\}, 1 \leq m \leq r$, we consider a selection of concrete services CS to be a *feasible selection*, iff it contains exactly one service for each service class appearing in P and its aggregated QoS values satisfy the global QoS constraints, i.e. $q'_k(CS) \leq c'_k, \forall k \in [1, m]$.

3.2.5 Utility Function

Since each web service is typically characterized by several QoS attributes, a utility function is used to evaluate the overall, multi-dimensional quality of a given service. In particular, it maps the quality vector Q_s of the service into a single real value, to enable sorting and ranking of the alternative services. In this paper, we use a Multiple Attribute Decision Making approach for the utility function, and in particular the *Simple Additive Weighting (SAW)* technique from [YH95]. The utility computation involves scaling the QoS attributes' values to allow a uniform measurement of the multi-dimensional service qualities independent of their units and ranges. The scaling process is then followed by a weighting process for representing user priorities and preferences. In the scaling process, each QoS attribute value is transformed into a value between 0 and 1, by comparing it with the minimum and maximum possible value according to the available QoS information about alternative services. For a

composite service CS, the aggregated QoS values are compared with minimum and maximum possible aggregated values, which can be easily estimated by aggregating, respectively, the minimum or maximum possible value of each service class in CS. For example, the maximum execution price of a given composite service can be computed by summing up the execution price of the most expensive service in each service class in CS. Formally, the minimum and maximum aggregated values of the k-th QoS attribute for a given composite service $CS = \{s_1, \ldots, s_n\}$ of an abstract process $P = \{S_1, \ldots, S_n\}$ are computed as follows:

$$Qmin'(k) = F_{j=1}^{n}(Qmin(j,k)) \qquad (3.1)$$
$$Qmax'(k) = F_{j=1}^{n}(Qmax(j,k))$$

with

$$Qmin(j,k) = \min_{\forall s \in S_j} q_k(s) \qquad (3.2)$$
$$Qmax(j,k) = \max_{\forall s \in S_j} q_k(s)$$

where $Qmin(j,k)$ is the minimum value (e.g. minimum price) and $Qmax(j,k)$ is the maximum value (e.g. maximum price) that can be expected for the k-th QoS attribute of the service class S_j, according to the available information about the service candidates in this class. The function F denotes an aggregation function that depends on QoS criteria e.g. summation, multiplication (s.a. Table 3.1). Now the utility of a component web service $s \in S_j$ is computed as

$$U(s) = \sum_{k=1}^{r} \frac{Qmax(j,k) - q_k(s)}{Qmax(j,k) - Qmin(j,k)} \cdot w_k \qquad (3.3)$$

and the overall utility of a composite service is computed as

$$U'(CS) = \sum_{k=1}^{r} \frac{Qmax'(k) - q'_k(CS)}{Qmax'(k) - Qmin'(k)} \cdot w_k \qquad (3.4)$$

with $w_k \in R_0^+$ and $\sum_{k=1}^{r} w_k = 1$ being the weight of q'_k to represent user's priorities.

3.2.6 Problem Statement

QoS-based service composition is a constraint optimization problem which aims at finding the composition that maximizes the overall utility value, while satisfying all the global QoS constraints. Formally:

Definition 2. (Optimal Selection) For a given abstract process P and a vector of global QoS constraints $C' = \{c'_1, \ldots, c'_m\}$, $1 \leq m \leq r$, we consider as *optimal selection* the feasible selection (see Definition 1) that maximizes the overall utility value U'.

A straightforward method for finding the optimal composition is enumerating and comparing all possible combinations of candidate services. For a composition request with n service classes and l candidate services per class, there are l^n possible combinations to be examined. Hence, performing an exhaustive search can be very expensive in terms of computation time and, therefore, inappropriate for run-time service selection in applications with many services and dynamic needs.

The global optimization approach has been recently widely accepted by researches as a solution for the QoS-aware service composition problem (e.g. [ZBD+03, ZBN+04, AP05, AP07]). The problem is modeled as a (Mixed) Integer Linear Program [NW88] (ILP) and solved using existing well-optimized ILP solvers such as CPLEX[1], MATLAB[2], LP_SOLVE[3], etc.

Each candidate service is represented in the mathematical ILP model by a binary decision variable. The end-to-end QoS constraints are specified as a set of equalities and/or inequalities over these variables.

A service candidate s_{ij} is selected in the optimal composition if its corresponding variable x_{ij} is set to 1 in the solution of the model and discarded otherwise. By re-writing (3.4) to include the decision variables, the problem of solving the model can be formulated as a maximization problem of the overall utility value given by

$$\sum_{k=1}^{r} \frac{Qmax'(k) - \sum_{j=1}^{n} \sum_{i=1}^{l} q_k(s_{ji}) \cdot x_{ji}}{Qmax'(k) - Qmin'(k)} \cdot w_k \tag{3.5}$$

subject to the global QoS constraints

$$\sum_{j=1}^{n} \sum_{i=1}^{l} q_k(s_{ji}) \cdot x_{ji} \leq c'_k, 1 \leq k \leq m \tag{3.6}$$

[1] http://www-01.ibm.com/software/integration/optimization/cplex/
[2] http://www.mathworks.com/products/matlab/
[3] http://lpsolve.sourceforge.net/5.5/

while satisfying the allocation constraints on the decision variables as

$$\sum_{i=1}^{l} x_{ji} = 1, 1 \leq j \leq n. \tag{3.7}$$

Because the number of variables in this model depends on the number of service candidates (number of variables $= n \cdot l$), this ILP model may not be solved satisfactorily, except for small instances. Another disadvantage of this approach is that it requires that the QoS data of available web services be imported from the service broker into the ILP model of the service composer, which raises high communication.

Since the QoS values (e.g. response times, throughput or availability) are only approximate, in this thesis we argue that finding a "reasonable" set of services that avoid obvious violations of constraints at acceptable costs is more important than finding "'the optimal"' set of services with a very high cost. Therefore, we advocate developing approximate selection algorithms that are able to achieve close-to-optimal solutions very efficiently. An approximate selection algorithm in this sense is a selection algorithm that is able to find a combination of services that satisfy *all* user QoS requirements while maximizing the overall utility value but *does not guarantee* finding the selection with maximum utility value. The closer the utility value of the found selection to the maximum utility value the closer the selection to the optimal selection.

The research problem we are addressing in this part of the thesis can be stated as follows: "*develop an approximate QoS-aware service selection algorithm that, for a given abstract composition and a set of user's QoS end-to-end constraints, produces a close-to-optimal selection of services much faster than producing the optimal selection by an exact algorithm.*"

In the rest of this chapter we describe our contributions in solving this problem.

3.3 The Hybrid Approach

To cope with the limitations of the global optimization approach, we propose a hybrid approach for solving the QoS-aware service composition problem. The hybrid approach combines global optimization with local selection techniques in order to benefit from the advantages of both worlds.

Figure 3.4 gives an overview of our approach. The hybrid approach divides the QoS-aware service composition problem into two sub-problems that can be solved more efficiently in two subsequent phases. In the first phase, the service composer decomposes the end-to-end SLA into local SLA's on the component service level. In other words, global QoS constraints in the end-to-end SLA are decomposed into local constraints on the component services' level and sent to the corresponding service brokers. In addition, user's preferences, which are expressed in terms of weights of the QoS attributes are also sent to the service brokers. In the second phase, each service broker performs local selection to find the best component service that satisfy the local SLA. The two phases of our approach are described in the next subsections in more details.

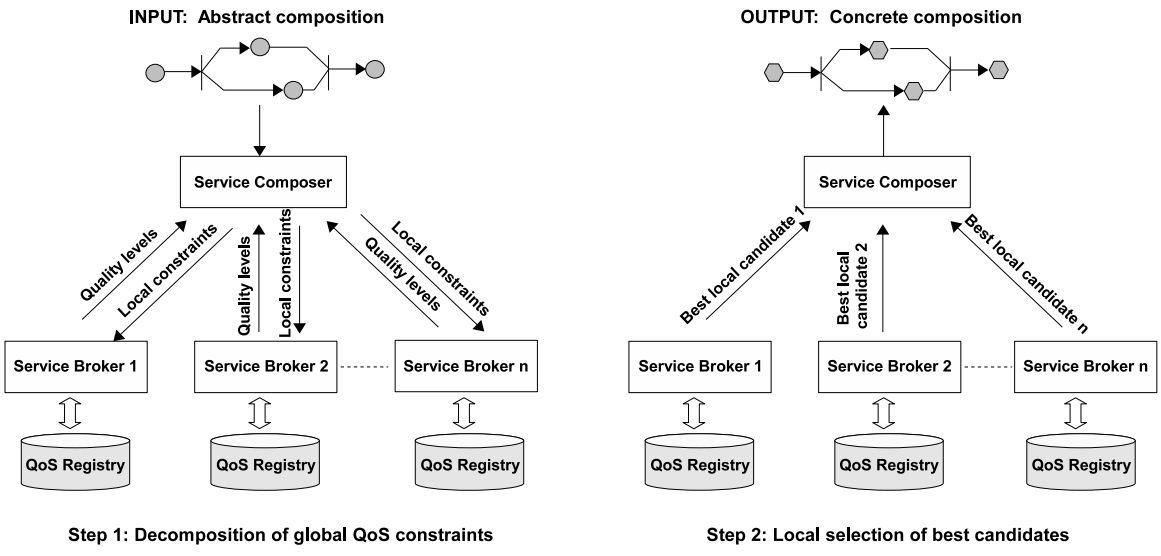


Figure 3.4 Distributed QoS-aware Service Selection

3.3.1 Decomposition of Global QoS Constraints

In the first phase of the hybrid approach, each QoS global constraint c' is decomposed into a set of n local constraints $c_1, \ldots, c_n$ (n is the number of abstract service classes in the composite service). The local constraints serve as conservative upper bounds, such that the satisfaction of local constraints guarantees the satisfaction of global constraints.

A naive decomposition method would be to divide each global constraint c' equally into n local constraints such that: $c_j = c'/n$, $1 \leq j \leq n$. However, as different service classes can have different QoS levels, a more sophisticated decomposition algorithm is required.

To this end, we analyze the QoS data of the available web services and extract a set of discrete values that better represent the various QoS levels in the collection, which we call *quality levels*. The decomposition of global QoS constraints is then performed by mapping them to the local quality levels. For example, given a set of candidate web services and their execution prices, we create a list of price levels for that service class. The global constraint on total execution price is then mapped to the appropriate price level of each service class.

In the following we first present our method for extracting local quality levels. After that, we describe how we formulate the QoS constraints decomposition problem as an optimization problem and use Integer Linear Programming to solve it.

Extracting Quality Levels

The goal of this step is to determine a small set of discrete QoS values that represent a collection of services. Figure 3.5 gives an overview of this method. The method takes for each attribute $q_k \in Q$ as input the QoS values of all l services in a certain service class S_j and outputs a set of d discrete values $QL_{jk} = \{q^1_{jk}, \ldots, q^d_{jk}\}$ such that:

$$Qmin(j,k) \leq q^1_{jk} \leq \ldots \leq q^d_{jk} \leq Qmax(j,k).$$

In this thesis we use a simple and effective method for selecting the quality levels, which we describe in Algorithm 1. This algorithm is executed for each QoS attribute $q_k \in Q$ separately. The first step in this algorithm is to sort the candidate services in class S_j based on their respective q_k value. Then the minimum and maximum values are directly added to the set of quality levels. Next, the rest of the sorted set of

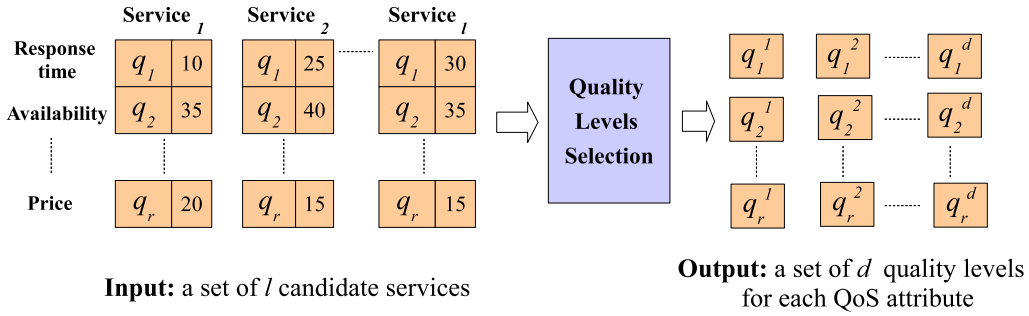


Figure 3.5 Quality Level Selection- Overview

services is divided into d sub-sets. From each sub-set we randomly select one sample service and use its QoS value as a quality level.

Algorithm 1 SelectQualityLevels(S_j, q_k, d)

Input : S_j: a set of l candidate services of a certain class, q_k: the QoS attribute to be considered, d: the required number of quality levels

Output : QL_{jk}: a set of d QoS values

1: $QL_{jk} \leftarrow \{\}$
2: $S_j \leftarrow sort(S_j, q_k)$ (sort the services w.r.t. q_k)
3: Let $q_{jk}^{min} \leftarrow S_j[1]$, $q_{jk}^{max} \leftarrow S_j[l]$
4: $QL_{jk} \leftarrow QL_{jk} \bigcup \{q_{jk}^{min}, q_{jk}^{max}\}$ (add the min and max values to the list of QoS levels)
5: $index \leftarrow 2$
6: $offset \leftarrow l - 2/d$
7: **for** $z = 1$ to d **do**
8: $S_z \leftarrow \{s_i | i \in [index, index + offset - 1]\}$ (divide the set of services int d sub-sets)
9: $s_z \leftarrow randomSelection(S_z)$ (randomly select one service from each sub-set)
10: $q_{jk}^z \leftarrow q_k(s_z)$ (use the QoS value of the selected service as a QoS level)
11: $QL_{jk} \leftarrow QL_{jk} \bigcup \{q_{jk}^z\}$
12: $index \leftarrow index + offset$
13: **end forreturn** QL_{jk}

We further explain this method by an illustrating example, which is depicted in Figure 3.6. In this example, there are 32 candidate services in one service class. The values of a certain QoS attribute q are shown for each service (e.g. $q(s_1) = 1, q(s_{10}) = 4$ and so on). The goal is to determine d QoS levels for this QoS attribute that better represent the whole set of services ($d = 5$ in this example). The first step is to sort the whole list according to the value of q. Next, the minimum and maximum values

are selected directly and the rest of the whole set is divided into 5 equal sub-sets (e.g. $s_2 - s_7, s_8 - s_{13}, s_{14} - s_{19}, s_{20} - s_{25}$ and $s_{26} - s_{31}$). Finally, one service is selected randomly from each sub-set and its value is used as a QoS level.

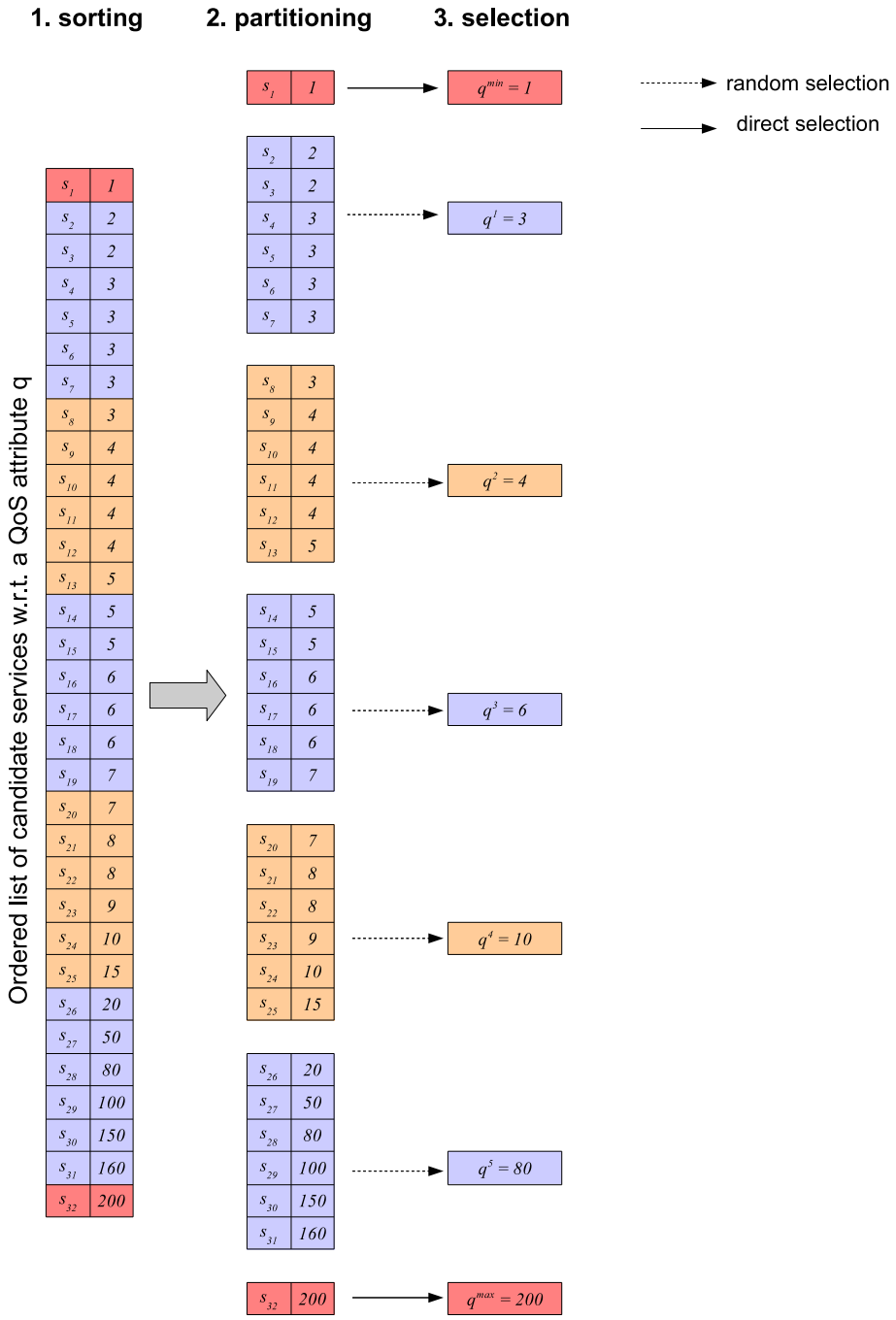


Figure 3.6 Quality Level Selection - Example

Note, that we do not remove duplicate values (i.e. services with the same QoS value). Therefore, the more frequent a given value is, the higher the probability that it is selected as a quality level. This ensures that the selection method takes into account the distribution of the QoS values in the collection. Figure 3.7 depicts the distribution of the QoS values in this example. The horizontal dashed lines represent the selected QoS level. We observe that most of the selected levels lie in the range [1, 10] (i.e. q^1, q^2, q^3 and q^4), which conforms with the fact that the QoS value of the majority of the services also lie in this range.

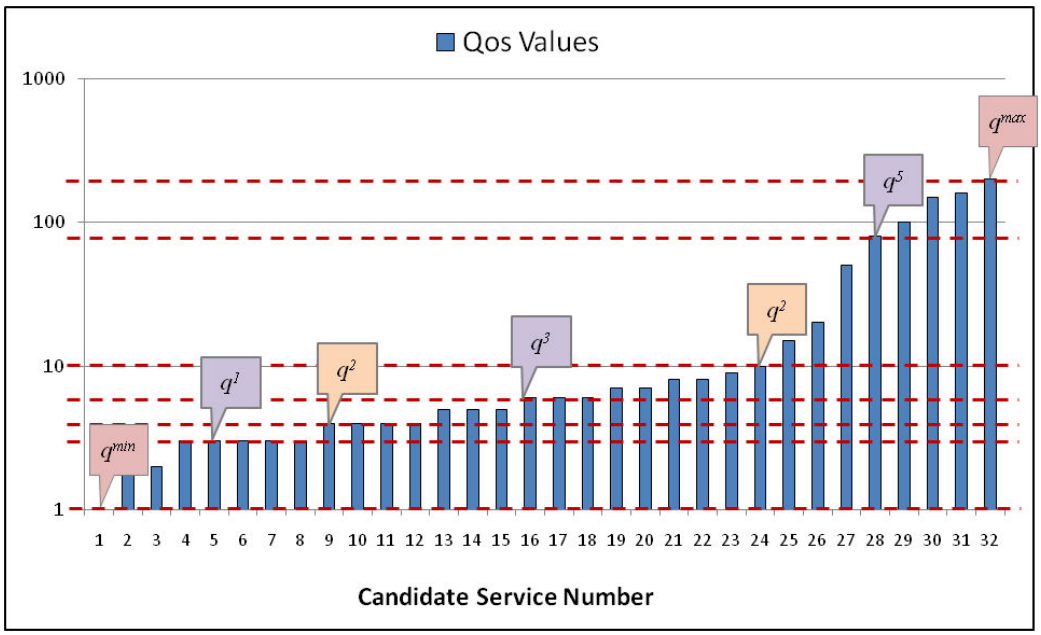


Figure 3.7 Distribution of the QoS data in the Quality Level Selection Example

Intuitively we can infer that the smaller the number of quality levels d, the faster the search for a mapping between global constraints and local quality levels will be. However, we can also infer that there is a trade-off between performance and optimality with respect to the selected number of selected local quality levels. Experimental results, which we present in Section 3.5.2 confirm this conclusion. The optimal number of local quality levels to be used (i.e. d) depends heavily on the data set as well as on the user's constraints. In some scenarios with too many very tight constraints, the decomposition of global constraints into local constraints might fail when using a small number of quality levels. Therefore, finding the optimal value of d is a very

difficult task. To handle this issue, we propose an iterative method that starts always with a small number of quality levels (like 10 levels) and iteratively duplicates the number of quality levels when needed (i.e. if no feasible decomposition of the global constraints is found). This process continues until a solution is found or d reaches a certain limit. According to our performance analysis (see Section 3.3.3), the hybrid approach outperforms the global optimization approach as long as $d << \frac{l}{m}$, where l is the (average) number of candidate services in a service class and m is the number of QoS constraints. Therefore, the iterative method is applied as long as the number of QoS levels does not exceed this limit. In the extreme cases, where d reaches the maximum number of $\frac{l}{m}$ before a solution is found, the process is stopped and the global optimization method is applied. However, the results of the experimental evaluation, which we conducted on different data sets that include some real-world data set as well as some synthetic datasets that represent the extreme cases of QoS distributions (correlated, anti-correlated and independent distributions) have shown that in average a solution is found after less than 4 iterations (see Section 3.5.2).

Constraint Decomposition as an Optimization Problem

Given a global QoS constraint c'_k for a composite service $CS_{abstract} = S_1, ..., S_n$, and a set of d local quality levels of the respective QoS attribute $QL_{jk} = v^1_{jk} \dots v^d_{jk}$ for each service class S_j, the goal of the constraint decomposition is to select an "'appropriate"' quality level v_{jk} from each service class such that the aggregation of the selected levels satisfy the global constraint.

To avoid discarding any service candidate that might be part of a feasible composition, the decomposition method needs to ensure that the local constraints are not more restrictive than needed. In other words, it is required that the local constraints are relaxed as much as possible while not violating the global constraints. Therefore, we model the QoS constraint decomposition problem as an optimization problem. The goal of this optimization problem is to find a set of local constraints for each service class that cover as many as possible service candidates, while their aggregation does not violate any of the global constraints.

We model this optimization problem as a Integer Linear Program (ILP) [NW88] and use ILP solving techniques to find the best mapping of global constraints to local quality levels. The objective function of the ILP is to minimize the number of discarded candidate services, when selecting local quality levels. Furthermore,

we want to eventually maximize the overall utility value of the selected services. Therefore, we assign each quality level v^z_{jk} a weight p^z_{jk} between 0 and 1, which estimates the benefit of using this quality level as a local constraint. This value is computed as follows. First, we compute $h(v^z_{jk})$, i.e. the number of candidate services that would qualify if v^z_{jk} was used as local constraint. Second, we calculate the utility value of each service candidate in the service class using the utility function (3.3) and determine $g(v^z_{jk})$, i.e. the highest utility value that can be obtained by considering these qualified services. Finally, p^z_{jk} can be calculated as

$$p^z_{jk} = \frac{h(v^z_{jk})}{l} \cdot \frac{g(v^z_{jk})}{u_{max}} \tag{3.8}$$

where l is the total number of service candidates of service class S_j, and u_{max} is the highest utility value that can be obtained for this class by considering all service candidates.

In the following we describe the formulation of the constraints decomposition problem as a Integer Linear Program. For the sake of simplicity, we consider here only the sequential composition pattern. In next section, we will show in details how other composition patterns can be handled.

A binary decision variable x^z_{jk} is used for each local quality level v^z_{jk} such that $x^z_{jk} = 1$ if v^z_{jk} is selected as a local constraint for the QoS attribute q_k at the service class S_j, and $x^z_{jk} = 0$ otherwise.

Therefore, we use the following allocation constraints in the model:

$$\forall j, \forall k : \sum_{z=1}^{d} x^z_{jk} = 1 \quad , 1 \leq j \leq n \ , 1 \leq k \leq m \tag{3.9}$$

Unlike the ILP model in the exact solution, i.e. the global optimization approach described in [ZBD+03, ZBN+04, AP05, AP07], our ILP model has much less number of variables (i.e. the quality levels instead of actual service candidates) and can be, therefore, solved much faster. The total number of variables in the model equals to $n \cdot m \cdot d$, i.e. it is independent of the number of service candidates. By keeping the number of quality levels d satisfies $m \cdot d \leq l$ we can ensure that the size of our ILP model is smaller than the size of the model used in the exact solution (where the number of decision variables is $n \cdot l$), thus can scale better with respect to the number of available web services.

The objective function of our ILP model is to maximize the p value (as defined in 3.8) of the selected local constraints to minimize the number of discarded feasible selections and maximize the expected utility value. Therefore, the objective function can be expressed as follows:

$$\text{maximize} \prod_{j=1}^{n}\prod_{k=1}^{m} p_{jk}^{z} \quad , 1 \leq z \leq d \tag{3.10}$$

We use the logarithmic function to linearize (3.10) in order to be able to use it in the ILP model:

$$\text{maximize} \sum_{j=1}^{n}\sum_{k=1}^{m}\sum_{z=1}^{d} ln(p_{jk}^{z}) * x_{jk}^{z} \tag{3.11}$$

In order to ensure that the aggregation of the selected levels satisfy the global constraint, we need to add corresponding constraints into the created ILP model. As the ILP model only supports linear constraints, nonlinear aggregation functions (e.g. multiplication, and minimum functions) need to be transformed into linear constraints.

To this end, we add the following constraint to the model for each QoS attribute that can be aggregated using a summation relation:

$$\sum_{j=1}^{n}(\sum_{z=1}^{d} v_{jk}^{z} \cdot x_{jk}^{z}) \leq c_{k}' \quad , 1 \leq k \leq m \tag{3.12}$$

For QoS attributes with a product aggregation function we use the logarithmic function to transform the product relation to a summation relation. We write the constraint as follows:

$$\sum_{j=1}^{n}(\sum_{z=1}^{d} \ln(v_{jk}^{z}) \cdot x_{jk}^{z}) \leq \ln(c_{k}') \quad , 1 \leq k \leq m \tag{3.13}$$

For QoS attributes with a minimum aggregation function we add one constraint for each component service:

$$\forall j : \sum_{z=1}^{d} v_{jk}^{z} \cdot x_{jk}^{z} \leq c_{k}' \quad , 1 \leq k \leq m \tag{3.14}$$

By solving this model using any ILP solver, we get a set of local quality levels. These quality levels are then sent to the service brokers to use them as local thresholds when performing local selection.

Handling Complex Composition Models

Recall that the ILP model formulation we described in the previous section assumes a sequential composition model. This assumption was made to simplify the description of the proposed ILP formulation. However, in many practical applications, the composition structure can be very complex involving different types of constructs, like for example, conditional branching or multiple parallel execution paths. In order to be able to formulate the ILP model as described in the previous section, we reduce arbitrary composition structures into a sequential one by replacing each of the loop, parallel and conditional constructs by a single *virtual* service. The local constraints, which are assigned to a virtual service after solving the constraint decomposition problem, serve as global constraints for the services it represents.

Consider the example shown in Figure 3.8. This is an example of a complex composition structure that involves both sequential and parallel executions of services. It can be transformed into an equivalent sequential structure in two steps as shown in Figure 3.8. In the first step we replace the sequence S_2 and S_3 by a virtual service S'. In the second step we replace the parallel construct involving S' and S_4 by a virtual service S''. The resultant structure is a sequence of services S_1, S'' and S_5. Applying the ILP formulation steps from previous section we can decompose global QoS constraints into local constraints for S_1, S'' and S_5. The local constraints of S'' are then used as global constraints for the services S' and S_4. By further decomposing these constraints we obtain local constraints for S' and S_4. Finally, by decomposing the constraints of S' we obtain local constraints for S_2 and S_3.

For substituting a set of random variables (i.e. service classes in a construct) with a single random variable (i.e. a virtual service class) we need to define the domain of the new random variable (i.e. quality levels) based on the domains of the replaced variables (i.e. quality levels of the replaced service classes). For this purpose, we use the same QoS aggregation functions, which are used to estimate the end-to-end QoS values of a given construct (see Table 3.1).

Given a set of service classes in a composition construct $S = \{S_1, ..., S_n\}$, and a set of quality levels $QL_{jk} = \{v_{kj}^1, ..., v_{kj}^d\}$ for each $S_j \in S$ and $q_k \in Q$, we define the quality levels of the virtual service class S' that substitutes for S as follows:

$$QL_{kS'} = \{v_{kS'} | v_{kS'}^z = F_k{}_{j=1}^{n}(v_{kj}^z) \wedge p_{kS'}^z = \min_{j=1}^{n}(p_{kj}^z), 1 \leq z \leq d\}$$

with the function F_k denoting the aggregation function of the k-th QoS attribute.

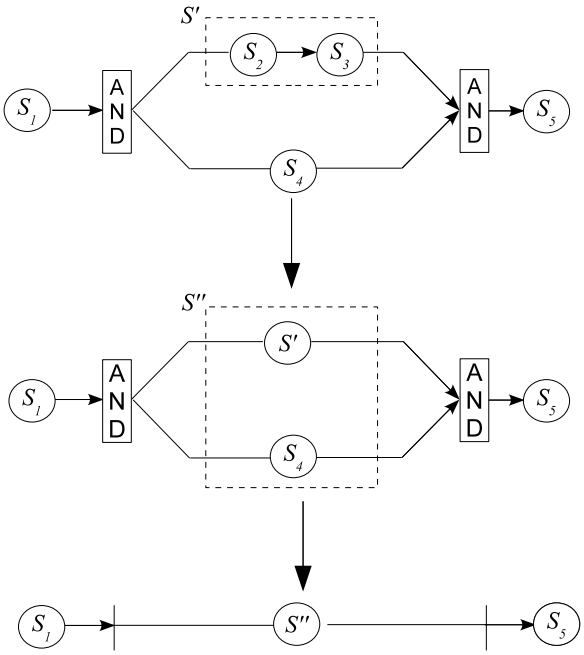


Figure 3.8 Transforming Complex Composition Patterns into Sequential Patterns

In other words, the i-th quality level of S' is defined by aggregating the i-th quality level of each service s in the construct, and its weight p is set to the minimum weight of the aggregated levels.

3.3.2 Local Selection

Using the method described in previous section for decomposing global QoS constraints, the end-to-end Service Level Agreement between the provider and consumers of the composite web service can be automatically decomposed into separate bilateral SLA's between the composite service's provider and each of the service brokers. Hence, the actual selection of services is carried out locally by the service brokers based on the constraints specified in the bilateral SLA's. Service brokers match the specified QoS constraints with the QoS information of the candidate services and return a reference to the best available web service at the request time. More specifically, the local constraints are used as upper bounds for the QoS values of the candidate services. A list of qualified services is created and sorted by their utility values.

The use of (3.3) for this purpose is not appropriate for the following reason. This utility function compares the distance $Qmax(j,k) - q_k(s_{ji})$ between the quality value

of a service candidate s_{ji} and the local maximum value in its class S_j with the distance $Qmax(j,k) - Qmin(j,k)$ between the local minimum and maximum values. This scaling approach can be biased by local properties leading to local optima instead of global optima. Therefore, we compare the distance $Qmax(j,k) - q_k(s_{ji})$ with the distance between the maximum and minimum overall quality values: $Qmax'(k) - Qmin'(k)$. This scaling method ensures that the evaluation of service candidates is globally valid, which is important for guiding local selection in order to avoid local optimum. The scaling process is then followed by a weighting process for representing user's over the different QoS attributes. We compute the utility $U(s_{ji})$ of the i-th service candidate in class S_j as

$$U(s_{ji}) = \sum_{k=1}^{r} \frac{Qmax(j,k) - q_k(s_{ji})}{Qmax'(k) - Qmin'(k)} \cdot w_k \tag{3.15}$$

with $w_k \in \mathbb{R}_0^+$ and $\sum_{k=1}^{r} w_k = 1$ being the weight of q_k to represent user's priorities. Service brokers sort the candidate services based on this utility value and select the service with the highest value for the composition.

At run-time, in case of failure to deliver the promised Quality of Service at the composition level (e.g. due to changes in the QoS of some of the selected services or because some services become unavailable) it is sufficient to perform local selection to find a replacement for the failed services only, as opposite to the global optimization approach, where a re-optimization of the whole composition is required. In the hybrid approach, the brokers of the failed services can quickly and independently respond to such changes by updating the list of candidate services and replacing the failed service by the top service on the list that fulfills the local SLA.

3.3.3 Performance Analysis

As discussed earlier in Section 3.1, the QoS-based service composition problem is a combinatorial problem, which can be modeled as a Multiple-choice Multi-dimensional Knapsack Problem (MMKP) [MT90]. The MMKP is known to be NP-hard [MT90], which means that any exact algorithm to solve this problem is expected to have an exponential time complexity with respect to the size of the problem. There are three factors that determine the size of the composition problem: 1) the number of global QoS constraints m, 2) the number of different service classes n, and 3) the number of candidate services per class $l_i, 1 \leq i \leq n$. For the sake of simplicity and without

loss of generality we assume that the number of candidate services is equal for all classes, i.e. $l_1 = l_2 = ... = l_n = l$. As m and n can be assumed to be small for most of the real life scenarios, while the number of functional equivalent services is growing rapidly and is expected to grow even faster in the future with the proliferation of the Software as a Service business model on the web, the focus of this study is on the scalability of the service selection methods with respect to the number of candidate web services.

Existing global optimization solutions model the service selection problem as a standard Integer Linear Program (ILP). The worst case time complexity of ILP solvers using the Simplex method and Branch and Bound algorithms is an exponential function with respect to the problem size (i.e. n, l and m) [Mar03]. Therefore, ILP based solutions are only applicable for small size composition problems, where the number of service candidates l is very limited. In addition, a quick response to changes in the QoS values of the selected services is not possible as the global optimization approach requires re-considering all possible combinations for satisfying the end-to-end constraints.

In our hybrid approach, we use Integer Linear Programming to solve part of the problem, namely, the decomposition of the global QoS constraints into local ones. The actual selection of services, however, is done using distributed local selection strategy, which is very efficient and scalable. The local utility computation for service candidates has a linear complexity with respect to the number of service candidates, i.e. $O(l)$. As service brokers can perform the local selection in parallel, the total time complexity of this step is not affected by the number of service classes, hence, the complexity of the second step remains $O(l)$.

The time complexity of our approach is dominated by the time complexity of the constraint decomposition part. The number of decision variables in our ILP model is $n \cdot m \cdot d$, where d is the number of quality levels. As a result, the time complexity of our approach is independent on the number of candidate web services, which makes it more scalable than existing solutions that rely solely on "pure" global optimization. By selecting a small number of quality levels d with $1 < d << \frac{l}{m}$ we ensure that the size of the ILP is much smaller than the size of the ILP model used in the global optimization approaches in [LNZ04, ZBN+04, AP07], and hence, can be solved much faster. The results of our extensive experimental evaluation, which we show in next section confirm these finding.

3.4 The Skyline Approach

Motivated by the fact that the selection of services for QoS-aware service composition is inherently a multi-criteria decision-making problem, and inspired by the success of the skyline query model [BKS01] in solving such problems for databases, we introduce in this section our skyline-based approach for efficient QoS-aware service composition.

By considering dominance relationships between web services based on their QoS values, we observe that only those services that belong to the skyline [BKS01], i.e. are not dominated by any other functionally-equivalent service, are valid candidates for the composition. All non-skyline services can be, therefore, safely pruned from the search space, thus, speeding up the selection process.

However, it is realistic to assume that specific QoS parameters are typically anti-correlated (e.g. execution time and price), which results in a large number of skyline services. To overcome this problem, we propose a method for clustering skyline services and selecting a small set of representatives to consider for the composition.

In Section 3.3 we presented the hybrid approach for addressing the QoS-aware service composition problem, and in Section 3.3.1 a greedy method for extracting QoS levels from the QoS information of service candidates was presented. However, the proposed method deals with each QoS dimension independently and does not take potential dependencies and correlations among these dimensions into account. In some scenarios with very constrained QoS requirements, this leads to very restrictive decompositions of the global constraints to local constraints that cannot be satisfied by any of the service candidates, although a solution may actually exist. In this section we propose a new skyline-based method for extracting QoS levels, which always leads to a feasible decomposition of end-to-end constraints.

In addition, from the service provider perspective, we describe how we can exploit the information about the skyline services to provide a clear distinction whether a given service is a promising candidate or not. In the latter case, we provide a strategy that proposes which QoS parameters of the service should be improved and how, so that it becomes more competitive, i.e. it is no longer dominated by other services.

3.4.1 Skyline Services

Given a set of points in a d-dimensional space, a skyline query [BKS01] selects those points that are not dominated by any other point. A point P_i is said to dominate

another point P_j, if P_i is better than or equal to P_j in *all* dimensions and strictly better in *at least one* dimension. Intuitively, a skyline query selects the "best" or most "interesting" points with respect to *all* dimensions. In this work, we define and exploit dominance relationships between services based on their QoS attributes. This is used to identify services in a service class that are dominated by other services in the same class. These services can then be pruned, hence reducing the number of combinations to be considered during service composition.

Definition 3. (Dominance) Consider a service class S, and two services $x, y \in S$, characterized by a set of Q of QoS attributes. x *dominates* y, denoted as $x \prec y$, iff x is as good or better than y in all parameters in Q and better in at least one parameter in Q, i.e. $\forall k \in [1, |Q|] : q_k(x) \leq q_k(y)$ and $\exists k \in [1, |Q|] : q_k(x) < q_k(y)$.

Definition 4. (Skyline Services) The skyline of a service class S, denoted by SL_S, comprises the set of those services in S that are not dominated by any other service, i.e., $SL_S = \{x \in S | \neg \exists y \in S : y \prec x\}$. We refer to these services as the *skyline services* of S.

Figure 3.9 shows an example of skyline services of a certain service class. Each service is described by two QoS parameters, namely execution time and price. Hence, the services are represented as points in the 2-dimensional space, with the coordinates of each point corresponding to the values of the service in these two parameters. We can observe that the service a belongs to the skyline, because it is not dominated by any other service, i.e. there is no other service that offers both shorter execution time *and* lower price than a. The same holds for the services b, c, d and e, which are also on the skyline. On the other hand, service f is not contained in the skyline, because it is dominated by the services b, c and d.

3.4.2 Composing the Skyline Services

As discussed earlier in Section 3.1, the problem of QoS-based service composition can be formulated as a constraint optimization problem, and ILP techniques [NW88] can be employed [ZBN+04, AP07] to solve it. However, as the number of variables in this model depends on the number of candidate services, it may only be solved efficiently for small instances. To cope with this limitation, we propose pruning all non-skyline services from the ILP model in order to keep its size as small as possible. By focusing

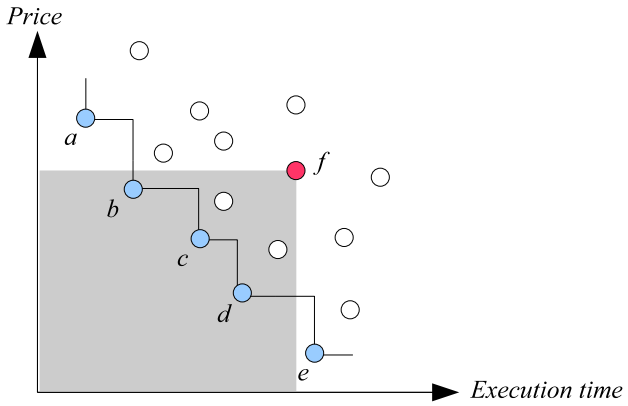


Figure 3.9 Example of Skyline Services

only on the skyline services of each service class, we speed up the selection process, while still being able to find the optimal selection, as formally shown below.

Lemma 1. Each service s_i in the optimal selection of services (as defined in Definition 2) for a web service composition $CS = \{s_1, \ldots, s_n\}$ belongs to the skyline of the corresponding class S_i, i.e. $\forall s_i \in CS : s_i \in SL_{S_i}$.

Proof 1. Let s_i be a service that is part of CS and does not belong to the skyline of its class S_i. Then, according to the definitions for service skyline and service dominance (see Section 3.4.1), there exists another service s'_i that belongs to the skyline of S_i and dominates s_i, i.e. s'_i is better than (or equal to) s_i in *all* considered QoS parameters. Let CS' be the composite service that is derived by CS by substituting s_i with s'_i. CS' also satisfies the request, in terms of the delivered functionality, since the two services s_i and s'_i belong to the same class S_i. Moreover, given that the QoS aggregation functions (see Table 3.1) are monotone, i.e. higher (lower) values produce a higher (lower) overall result, CS' also satisfies the constraints of the request. In addition, given that the utility function is also monotone, CS' will have a higher overall utility than CS. Hence, CS' is a better solution than CS for this request.

According to Lemma 1, it is sufficient to focus on the skyline services of each service class to find the optimal selection of services. By discarding non-skyline services from the search space, we can improve the efficiency of the QoS-based service selection algorithms.

However, the size of the skyline set can significantly vary for each dataset, as it strongly depends on the distribution of the QoS data and correlations between the

different QoS parameters. Figure 3.10 shows an example of 3 types of datasets in the 2-dimensional space: (a) in the independent dataset, the values on the two QoS dimensions are independent to each other; (b) in the correlated dataset, a service that is good in one dimension is also good in the other dimension; (c) in the anti-correlated dataset there is a clear trade-off between the two dimensions. The number of skyline services is relatively small in correlated datasets, large in anti-correlated and medium in independent ones.

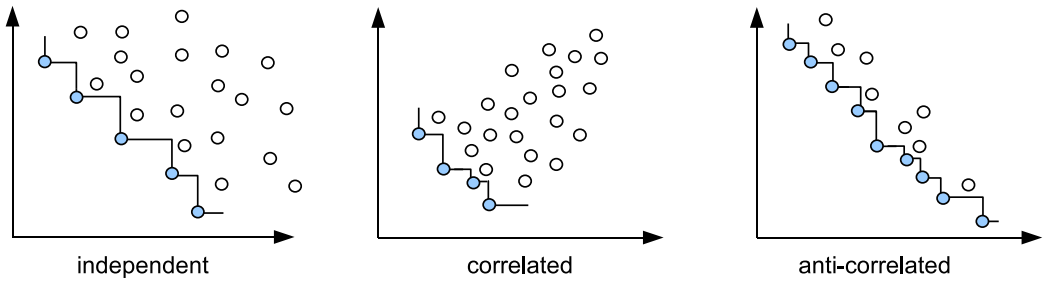


Figure 3.10 Skyline of Different Dataset Types

In the following section we present a method for handling the cases, where the number of skyline services is too large.

3.4.3 Representative Skyline Services

There has been a lot of work done in the literature for controlling the size of skylines in order to address one or some of the drawbacks of conventional skyline compution methods [JHE04, CJT+06a, CJT+06b, BGL07, BGS07, XZT08]. While the work presented in [JHE04] aims at *increasing* the size of the returned skyline (by adding close neighbors of skyline objects to the skyline) in order to satisfy more users in case of limited number of skyline objects, [CJT+06a] and [CJT+06b] propose a relaxed definition of the dominance relation in order to *decrease* the number of skyline objects in highly anti-correlated datasets. A more generic solution was presented in [XZT08] that allows both increasing and decreasing the size of the skyline. In [BGL07, BGS07] the authors focus on a more realistic and more complicated variation of the skyline computation problem, namely computing skylines on partial order domains, where users can have preferences on non-numerical attributes that do not have a total ordering. In such scenarios the returned skyline can be very large and the computation can be very expensive. The authors propose in [BGL07] an incremental computation of

the skyline by exploiting the input of the user on the different attributes and thus step by step prune more and more items from the results set. In [BGS07] the authors relax the Pareto semantics and introduce the concept of weak Pareto dominance relation, which yields an efficient computation of the skylines.

More recently, the concept of *representative skyline services* has been proposed to solve this problem [LYZZ07, TDLP09]. The idea is to determine a small set of skyline objects that best represent the whole skyline set. In [LYZZ07] the authors define the set of representative skyline objects as the minimum set of objects that dominate all other non-skyline objects. In [TDLP09] the propose a distance-based definition of the representative skyline that minimizes the distance between each non-representative skyline object and its nearest representative.

However, the main challenge that arises in the web service selection scenario is how to identify a set of representative skyline services that best represent all trade-offs of the various QoS parameters, so that it is possible to find a solution that satisfies the constraints and *at the same time* has a high utility score (recall our problem formulation in Section 3.2.6). The aforementioned general methods for reducing the number of skyline objects can only capture one aspect of the problem, namely representing the different trade-offs, but do not necessarily return services that maximize the utility value. Here, our goal is to select a set of representative skyline services, providing different trade-offs for the various QoS parameters, and use this reduced set as input for the ILP model, whose objective function is to maximize the overall utility value (after satisfying all user constraints).

Selecting representative skyline services also involves essentially a trade-off regarding the number of representatives to be selected: the number of representative services should be large enough to allow finding a solution when performing the to the composition request, but also small enough to allow for efficient computation.

In the following, we present a method for selecting representative skyline services in order to address the situation where the number of skyline services K of a certain service class S is too large and thus cannot be handled efficiently.

To address this challenge, we use divisive clustering. The main idea is to cluster the skyline services into k clusters with $k = 2, 4, 8, 16, ..., K$, where K is the number of skyline services of a certain class S, and select one representative service from each cluster. In our case, we select as representative the service with the best utility value. In particular, we build a tree structure of representatives, as shown in the

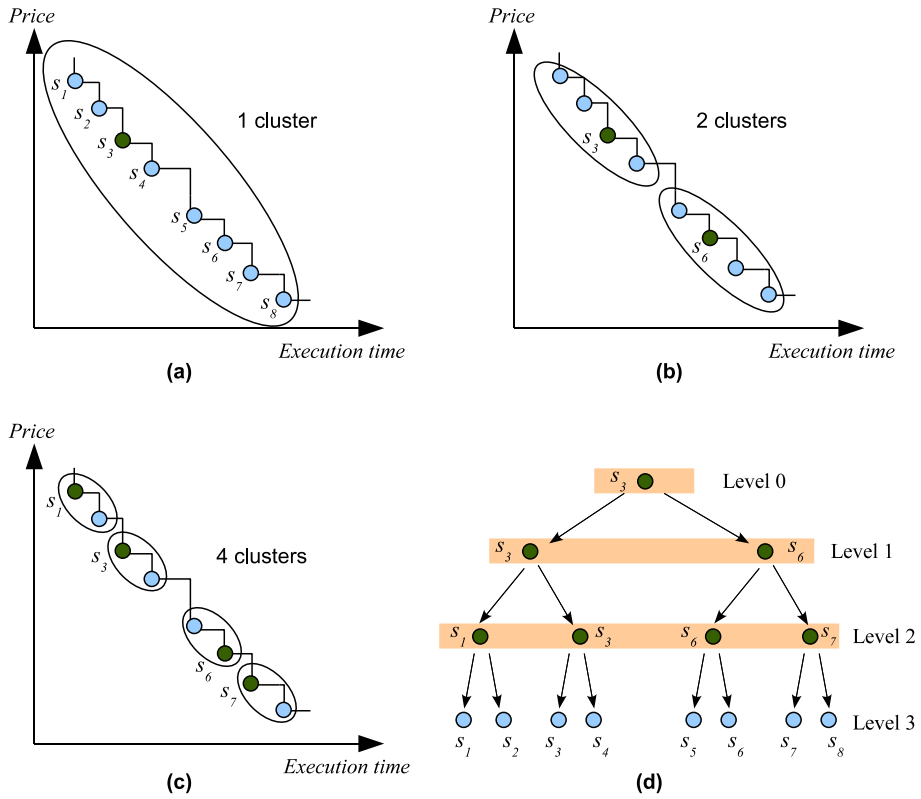


Figure 3.11 Selecting Representatives via Clustering of Skyline Services

example of Figure 3.11. Each leaf node of this tree corresponds to one of the skyline services in SL, whereas the root and intermediate nodes correspond to the selected representatives of the created clusters.

We use the well-known *k-means* clustering algorithm [Llo82] for building the representatives tree, as described in Algorithm 2. The algorithm takes as input the skyline set SL of class S and returns a binary tree structure of representative services. The algorithm starts by determining the root s, which is the service with maximum utility value in SL. The algorithm then clusters SL into two sub-clusters $CLS[0]$ and $CLS[1]$ and adds the representatives of these two sub-clusters to the children list of s. The process is then repeated for each sub-cluster until no further clustering is possible (i.e. until the size of new created clusters is lower than 2).

At run-time, when a service composition request is processed, we start the search from the root node of the tree, i.e. we first consider only the top representative service of each class (e.g. service s_3 for class S in the example). These selected representatives are inserted into the mixed integer program and the optimization problem is solved.

Algorithm 2 BuildRepresentativesTree(SL)

Input : a set of skyline services SL

Output : a tree of representatives with service s as a root

```
1: s ← maxUtilityService(SL)
2: CLS ← KMeansCluster(SL, 2)
3: for i = 1 to 2 do
4:     if (CLS[i].size > 2) then
5:         C ← BuildRepresentativesTree(CLS[i])
6:     else
7:         C ← CLS[i]
8:     end if
9:     s.addChild(C)
10: end for
11: return s
```

In the case that no solution is found using the given representatives, we proceed to the next level, taking two representatives for each class (s_3 and s_6 for class S in the example). This process is repeated until a solution is found or until the lowest level of the tree, which consists of all skyline services, is reached. In the latter case, it is guaranteed that a solution will be found (if one exists), and this solution is the optimal solution according to Lemma 1. However, if a solution is found earlier, i.e. before reaching the skyline level, we proceed by examining those services that are descendants of the selected representatives for further optimization. This expanding of the search space is continued until no further optimization in terms of utility value is achieved, or the skyline level is reached.

3.4.4 Improving Service Competitiveness

As described previously, focusing on skyline services can be useful for improving the performance and scalability of QoS-aware service selection methods. As non-skyline services are filtered out early and cannot be in the result set of any request, regardless of the given QoS requirements or preferences, it is important for service providers to know whether their services are in the skyline, given their current QoS levels. Even more importantly, if this is not the case, providers should be guided in determining which QoS levels of their services should be improved and how, in order to become

skyline services. Such information is valuable for service providers to analyze the position of their services in the market compared to other competing services.

To address this issue, we present an algorithm that proposes how to improve the competitiveness of non-skyline services. Clearly, there are various modifications that can lead a non-skyline service to the skyline. Our goal is to identify the minimum improvement in each QoS dimension that is required in order to bring a non-skyline service into a position where it is not dominated by any other service, thus becoming part of the skyline.

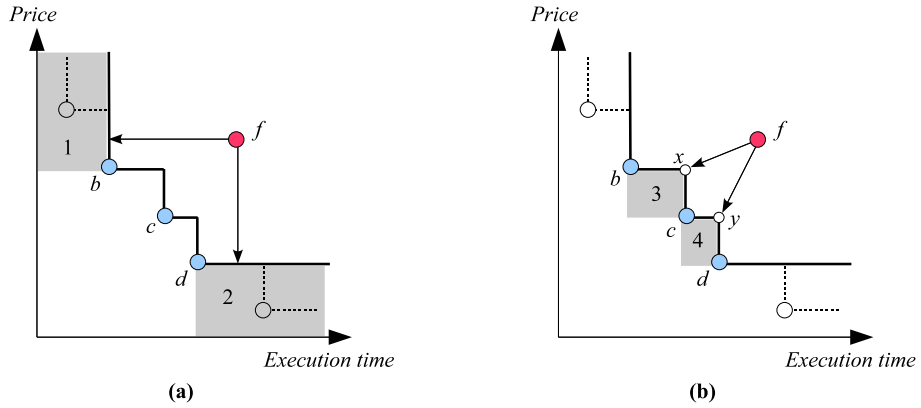


Figure 3.12 Measuring the Distance to the Skyline

Consider the example in Figure 3.12, where service f is dominated by the skyline services b, c and d. According to Definition (3), this means that each of these services are better or equal to f in all QoS dimensions and strictly better than f in at least one QoS dimension. In order to improve the competitiveness of f, the provider must ensure that it is not dominated by any other service. To achieve this, it is sufficient to make f better than each of its dominating services in (at least) one QoS dimension. By analyzing the skyline structure in Figure 3.12, we can identify four partitions of the 2-dimensional space, in which f can fulfill this requirement. The first two partitions are shown in Figure 3.12-a, and can be reached by improving only one of the QoS-dimensions, while the other two are shown in Figure 3.12-b, and can be reached by improving both QoS dimensions at the same time. We call each of these partitions a *no-dominance* partition for this service. A service in any of these partitions is incomparable with all the skyline services, as it is not dominated by any of them nor is dominating any of them.

Improving the QoS of provided services to a certain level, typically incurs some

cost. For example, reducing the execution time of the service might require using faster servers or more CPU computation power, if the service is running on the cloud. Thus, service providers would be interested in determining the best (set of) QoS dimension(s) to optimize, while minimizing the required cost. We assume that the cost of improving any QoS dimension increases monotonically in the sense that more improvement always implies more cost. We use the *weighted euclidean distance* for estimating the cost of moving a service s in the QoS multi-dimension space from its current position to a new position s':

$$d(s, s') = \sqrt{\sum_{i=1}^{|Q|} w_i (q_i(s) - q_i(s'))^2} \tag{3.16}$$

The weight w is specified by the service provider to express his preferences over the QoS dimensions. Higher weight implies higher cost for improving the corresponding dimension.

In order to minimize the cost of improving the service position in the QoS multi-dimensional space, we first need to identify the no-dominance partitions. Then, we measure the distance from the service to be improved to each of these partitions using Equation 3.16, and we select the one with the minimum distance.

Algorithm 3 locates the no-dominance partitions that can be reached by improving only one QoS dimension. It takes as input a non-skyline service s and the list of skyline services SL of the corresponding class, and it returns a list $I = \{p_1, \ldots, p_{|Q|}\}$, where each entry p_i denotes the improvement required in the i-th QoS dimension for the service to become part of the skyline (keeping all the other dimensions fixed).

Algorithm 3 OneDimImprovements(s, SL)

Input : a service s, the set of skyline services $SL$of its class

Output : a list I of the required amount of improvement for each single dimension

1: $DS \leftarrow \{r \in SL : r \succ s\}$
2: **for all** $q_i \in Q$ **do**
3: $\quad I[i] \leftarrow \max_{r \in DS} |r^{q_i} - s^{q_i}|$
4: **end for**
5: **return** I

Algorithm 4 locates the coordinates of the maximum corner (i.e. top-right) of each no-dominance partition (e.g. the points x and y in Figure 3.12-b). Modifying

the QoS values of the service to values that are slightly better than the values of one of these points, ensures that it is not dominated by the skyline services. The algorithm takes as input a non-skyline service s and the list of skyline services SL of the corresponding class, and suggests a new position s' that can be reached with minimum cost, in order to make s not dominated by any other services. First, the algorithm computes the list DS of services dominating s. Then, DS is sorted for each QoS dimension separately. The coordinates of the maximum corners are determined by taking the maximum QoS values of each two subsequent services in each sorted list. For example, the coordinates of the maximum corners x and y in Figure 3.12-b, are determined by sorting the dominating services b, c and d by execution time and then taking the maximum price and execution time of the services b and c. This process is repeated for each other dimension and only new discovered points are added to the list M. Finally, Equation 3.16 is used to estimate the cost of moving s to any of the positions listed in M and the position with minimum cost is returned.

Algorithm 4 MultiDimImprovements(s, SL)

Input : a service s, the set of skyline services SL of its class

Output : a new not dominated position s', with minimum improvement cost

1: $DS \leftarrow \{r \in SL : r \succ s\}$
2: $M \leftarrow \{\}$
3: **for all** $q_i \in Q$ **do**
4: $\quad DS_i \leftarrow DS.sortBy(q_i)$
5: $\quad$ **for** $j = 1$ to $DS.size - 1$ **do**
6: $\quad\quad s_j \leftarrow DS_i[j]$
7: $\quad\quad s_{j+1} \leftarrow DS_i[j+1]$
8: $\quad\quad m \leftarrow newQoSVector$
9: $\quad\quad$ **for all** $q_k \in Q$ **do**
10: $\quad\quad\quad q_k(m) \leftarrow \max(q_k(s_j), q_k(s_{j+1}))$
11: $\quad\quad$ **end for**
12: $\quad\quad M \leftarrow M \cup m$
13: $\quad$ **end for**
14: **end for**
15: **return** $s' = agr \min_{m \in M} d(s, m)$

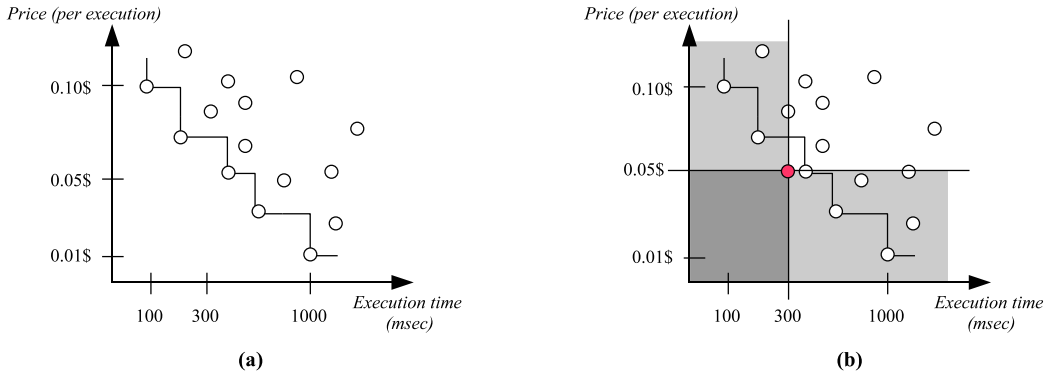


Figure 3.13 Example of Unsuccessful Constraints Decomposition

3.4.5 Skyline-based Decomposition of QoS Constraints

In Section 3.3, we proposed a hybrid approach for QoS-based web service composition, using ILP for decomposing the end-to-end SLA into bilateral SLA's with the involved service brokers. The variables in the ILP model of the hybrid approach represent the local QoS levels of each service class rather than the actual service candidates, making it more scalable to the number of services than the global optimization approach.

However, the greedy method presented in Section 3.3.1 for extracting local quality levels and the mapping of global constraints into these local quality levels does not take into account potential correlations and dependencies among the different QoS attributes. Therefore, it is possible that the local constraints in the resulted bilateral SLA cannot be fulfilled by any of the candidate services. In other words, the intersection of the subsets of services that satisfy each of the local constraints is an empty set. Consider the example shown in Figure 3.13. This is a two dimensional QoS space that represents the collection of candidate web services in a certain service class. The values indicated on the X and Y axis in Figure 3.13-a represent the quality levels of the respective QoS attribute. Figure 3.13-b shows an example of an unsuccessful decomposition of the global constraints. In this example the global constraints on price and execution time are mapped to the local quality levels 0.05$ and 300 msec respectively. This mapping is considered unsuccessful as none of the candidate services can fulfill *both* of the local constraints.

To overcome this limitation we present in the following a new method for extracting local quality levels and decomposing global QoS constraints. A quality level in the new method is in the form of a QoS vector, which translates to one point in the

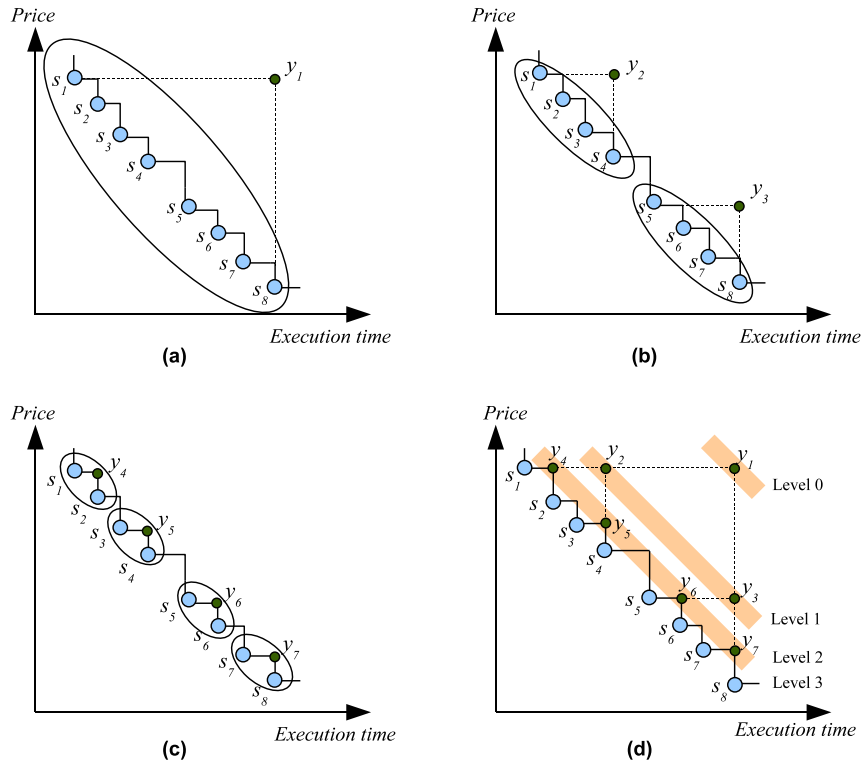


Figure 3.14 Extracting Quality Levels via Clustering of Skyline Services

multidimensional QoS space. To ensure that the set of services that satisfy a quality level is not empty, we need to make sure that its position in the QoS space is either on or above the skyline. We use Algorithm 5 for this purpose. The main idea is similar to the representatives selection method described earlier. First, we determine the skyline services of each service class, and we recursively cluster them using the k-means clustering algorithm. However, instead of selecting one representative service from each sub-cluster, we create a virtual point in the QoS multidimensional space, whose coordinates are calculated as the maximum (i.e. worst) QoS values in the sub-cluster, as illustrated in the example of Figure 3.14. The virtual point y_1 in Figure 3.14-a has the maximum execution time and maximum price of all skyline services, i.e the execution time of service s_8 and the price of service s_1.

Hence, we use the created points (y_1 to y_7 in the example) to represent the various QoS levels of the service class. We also assign each of the QoS levels a utility value, which is the best utility value that can be obtained by any of the services of the corresponding sub-cluster. We then use ILP to map each of the end-to-end constraints into one of the local QoS levels (i.e. one of the virtual points) of each class in the

Algorithm 5 SelectQoSLevels(SL)

Input : a set of skyline services SL

Output : a tree of QoS levels with y as a root

1: $y \leftarrow newQoSLevel$
2: **for all** $q_i \in Q$ **do**
3: $\quad q_i(y) \leftarrow \max q_i(s), \forall s \in SL$
4: **end for**
5: $y.utility \leftarrow maxUtilityValue(SL)$
6: $CLS \leftarrow KMeansCluster(SL, 2)$
7: **for** $i = 1$ to 2 **do**
8: $\quad$ **if** $(CLS[i].size > 2)$ **then**
9: $\quad\quad C \leftarrow SelectQoSLevels(CLS[i])$
10: $\quad$ **else**
11: $\quad\quad C \leftarrow CLS[i]$
12: $\quad$ **end if**
13: $\quad y.addChild(C)$
14: **end for**
15: **return** y

composition problem. A binary decision variable x_{ij} is used for each local QoS level y_{ij} such that $x_{ij} = 1$ if y_{ij} is selected as a local constraint for the service class S_j, and $x_{ij} = 0$ otherwise. Thus, we reformulate the ILP model presented in 3.3.1 as follows:

$$\text{maximize} \quad \sum_{j=1}^{n} \sum_{i=1}^{l} U(y_{ij}) \cdot x_{ij} \tag{3.17}$$

subject to the global QoS constraints

$$\sum_{j=1}^{n} \sum_{i=1}^{l} q_k(y_{ij}) \cdot x_{ij} \leq c'_k, 1 \leq k \leq m \tag{3.18}$$

while satisfying the allocation constraints on the decision variables as

$$\sum_{i=1}^{l} x_{ij} = 1, 1 \leq j \leq n. \tag{3.19}$$

where the number of variables l equals the number of QoS level in each service class. We solve this ILP model for $l = 1, 2, 4....K$, where K is the total number of

skyline services. In the given example, this corresponds to the levels from 0 to 3 of the QoS levels tree in Figure 3.14-d. The process stops when a solution is found, i.e. a mapping of all end-to-end constraints to local QoS levels is found. In the worst case, the process will continue until the lowest level is reached. In this case, each skyline service represents a local QoS level, and the problem becomes similar to the original global optimization problem we discussed earlier. According to Lemma 1, if a solution to the original problem exists, a successful decomposition of the end-to-end constraints will be found. In other words, it is guaranteed that the set of services that satisfy the obtained local constraints is always a non-empty set.

3.5 Experimental Evaluation

The aim of this evaluation is to validate our hypothesis that our approximate solutions achieve close-to-optimal results with a much lower computation time compared to the exact solutions as proposed by [LNZ04, ZBN+04, AP07]. For this purpose, we have conducted extensive simulations to evaluate the performance of the proposed QoS-aware service selection algorithms, which we describe in this section.

3.5.1 Experimental Setup

In our evaluation we experimented with two types of datasets: real and synthetic datasets. The first is the publicly available dataset QWS[4], which comprises measurements of 9 QoS attributes for 2500 real-world web services. These services were collected from public sources on the Web, including UDDI registries, search engines and service portals, and their QoS values were measured using commercial benchmark tools. Table 3.2 lists the QoS attributes in this dataset and gives a brief description of each attribute.

In order to make sure that the results of our experiments are not biased by the used QWS dataset, we also experimented with three synthetically generated datasets with larger number of services and different distributions. For this purpose, we used a publicly available synthetic generator[5] to obtain three different datasets: a) a correlated dataset (cQoS), in which the values of the QoS parameters are positively

[4]http://www.uoguelph.ca/ qmahmoud/qws/index.html/
[5]http://randdataset.projects.postgresql.org/

Table 3.2 QoS attributes in the QWS dataset

QoS Attribute	Description	Unit
Response Time	Time taken to send a request and receive a response	msec
Availability	Number of successful invocations/total invocations	percent
Throughput	Total number of invocations for a given period of time	invocations/sec
Successability	Number of response/number of request messages	percent
Reliability	Ratio of the number of error messages to total messages	percent
Compliance	To which extent a WSDL document follows the WSDL spec.	percent
Best Practices	To which extent a web service follows the Web Services Interoperability (WS-I) Basic Profile	percent
Latency	Time the server takes to process a given request	msec
Documentation	Measure of documentation (i.e. description tags) in WSDL	percent

correlated, b) an anti-correlated (aQoS) dataset, in which the values of the QoS parameters are negatively correlated, and c) an independent dataset (iQoS), in which the QoS values are randomly set. Each dataset comprises 10K QoS vectors, and each vector represents the 9 QoS attributes of one web service.

We used the open source Mixed Integer Programming solver *lpsolve* version 5.5 [MB] for solving the ILP model in both approaches. The experiments were conducted on a HP ProLiant DL380 G3 machine with 2 Intel Xeon 2.80GHz processors and 6 GB RAM. The machine is running under Linux (CentOS release 5) and Java 1.6.

3.5.2 Evaluation of the Hybrid Approach

For the purpose of this evaluation, we considered a scenario, where a composite application comprises services from n different service classes (n varies in our experiments between 10 and 50 classes). Users of the composite application submit a set of numerical constraints on (a sub-set of) the 9 QoS attributes shown in Table 3.2. The goal of this evaluation is to measure how fast the hybrid approach can find the best service from each service class to instantiate the composite application, while meeting the given QoS constraints and maximizing the overall utility value (U') as given in formula 3.4.

Thus, we randomly partitioned each of the aforementioned datasets into n sub-

sets. Each sub-set represents the service candidates of one service class. We then created several vectors of up to 9 random values to represent the users end-to-end QoS constraints. Each constraints vector corresponds to one composition request.

We solved each composition request using the following methods:

- *Global*: this is the global optimization method [LNZ04, AP07] with all service candidates represented in the ILP model. This method returns the optimal selection of services, and therefore is used as a baseline in our experiments.
- *Hybrid*: this is our proposed method in this paper for combining global optimization with local selection, based on the concept of constraint decomposition.
- *WS-HEU*: this is the heuristic method proposed in [YZL07a], which is a modification of the original heuristic M-HEU [KLMA02] for solving multi-choice multi-dimensional Knapsack problems.

We then recorded the required computation time (we report here the average of 50 executions with the same parameters) by each of the aforementioned methods to return the selection of services. Recall that the service selection problem has been defined as an optimization problem, whose goal is to maximize the objective function (i.e. the utility function in our scenario). Therefore, we also record the overall utility value of the selected services by each method. In order to evaluate the optimality of the returned solution by one of the heuristic solutions (i.e. *Hybrid* or *WS-HEU*), we measure how close is its utility value u to the utility value of the optimal solution u_{max} obtained by the exact method (i.e. the *Global* method), as follows:

$$optimality = u/u_{max}$$

Effect of Number of Quality Levels

Before evaluating the performance and optimality of the *Hybrid* approach, in the first experiment we investigate the effect of the chosen number of quality levels d in the *Hybrid* approach. For this purpose we created a test case of a composition request with 5 service classes with 500 candidate services per class. We then randomly created a vector of end-to-end QoS constraints. We solved the selection problem using both the *Global* approach and the *Hybrid* approach, while in the *Hybrid* approach we solved the problem several times, each time with a different number of quality levels:

i.e. 10, 20, 30, 40 and 50 quality levels. The results of this experiment are shown in Figure 3.15. To the left side of this figure a performance comparison of the two methods is shown. We notice that the *Hybrid* method is much faster than the *Global* method, while still able to achieve a very close-to-optimal results as shown on the second chart to the right side of Figure 3.15. We also notice that using more quality levels improves the optimality if the selection, but on the other hand, it also imposes more overhead in terms of computation time.

Hence, there is a trade-off between performance and optimality with respect to the chosen number of quality levels. Moreover, in some cases with very constrained composition requests, the decomposition of global constraints into local constraints might fail when using a small number of quality levels. To handle this issue, we apply the iterative method, which we described in Section 3.3.1. The iterative method starts always with a small number of quality levels (10 levels in this case) and if no feasible decomposition of the global constraints is found, the number of QoS levels is iteratively duplicated. The method continues until a solution is found. This approach is applied in the following experiments, and the practice shows that in almost all cases, with all different types of datasets, a solution is found after a few number of iterations (up to 3 iterations, i.e the number of levels used are 10, 20 or 40 levels).

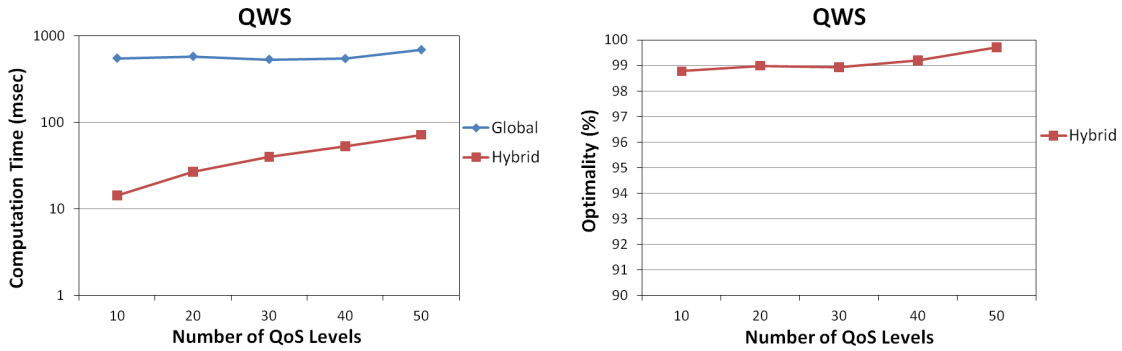


Figure 3.15 Performance and Optimality vs. Number of QoS Levels

Performance Comparison

- **Performance vs. Number of Candidate Services**: In Figure 3.16 we compare the performance of our *hybrid* approach with the performance of the *Global* and *WS-HEU* approaches with respect to the number of candidate services. The graphs show the measured average computation time of each of the three selection methods.

The number of service candidates per class l in this experiment varies between 100 to 1000 services per class, while the number of service classes n is set to 10 classes and the number of QoS constraints to 5 end-to-end constraints.

We observe that, in general, increasing the number of candidate services, increases the computation time of the *Hybrid* approach very slowly (if at all) compared to the *Global* and *WS-HEU*, which makes our solution more scalable to the number of available web services. Although slightly slower than *WS-HEU* with the correlated dataset, our *Hybrid* approach remains significantly faster than *WS-HEU* in the other datasets, and far faster than the *Global* approach in all datasets.

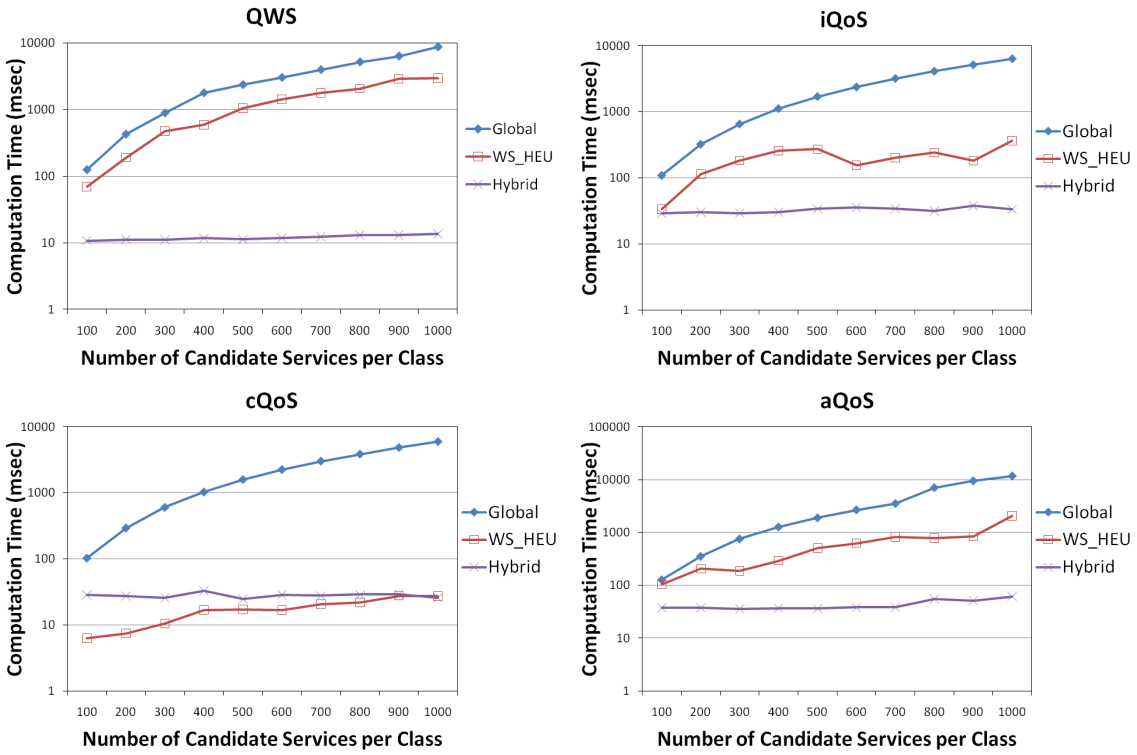


Figure 3.16 Performance vs. Number of Candidate Services

The reason for this scalable behavior of the *Hybrid* approach is that the decomposition of the global constraints into local ones using ILP is not directly affected by the number of available services, rather by the distribution of the QoS values among them. Consequently, increasing the number of available services does not necessarily lead to increasing the computation time as in the *Global* approach (where the number of variables to consider in the ILP increases) or in the *WS-HEU* algorithm (where the number of options also increases). Instead, increasing the number of

services in the *Hybrid* approach affects only the process of extracting quality levels. When new candidate web services become available (or some existing services become not available) the set of quality levels need to be updated accordingly. Since extracting and updating the set of quality levels of each service class can be carried out offline by the responsible service broker, this step has no direct impact on the performance of the service selection algorithm at run-time.

- **Performance vs. Number of Service Classes**: In this experiment we evaluate the scalability of the three approaches with respect to the number of service classes n in the composition. For this purpose, we fixed the number of candidate services per class l to 500 and the number of end-to-end constraints to 5 constraints, while varying the number of service classes in each composition instance between 10 and 50. The results of this experiment shown in Figure 3.17 indicate that the performance of all three methods degrade as the number of service classes increases. However, the *Hybrid* approach still outperforms the *Global* approach in all datasets. Again, we observe that the *Hybrid* approach outperforms the *WS-HEU* approach in all datasets except the correlated dataset, where it performs slightly slower than *WS-HEU*.

Optimality Comparison

The results presented so far have shown that the *Hybrid* outperforms the *Global* approach in terms of computation time. This improvement in the performance is due to the fact that the *Hybrid* approach is an approximate approach, i.e. it cannot guarantee finding the optimal selection of services. Therefore, it is important to evaluate how good are the results obtained by this approximation method compared to the exact method that guarantees returning the optimal selection. As the objective function of the optimization problem in hand is to maximize the overall utility value of the selected services according to Definition 2 of the optimal solution, in this evaluation we compare the utility value u of the selected services by the *Hybrid* approach with the maximum utility value u_{max} obtained by *Global* approach. Ideally, the quality of the obtained selection could be evaluated by computing the utility value of all the feasible selections and analyzing the distribution of these values. The smaller the distance between u and u_{max} is (and the farther the distance between u and the worst value u_{min} is), the better the selection is. However, computing the

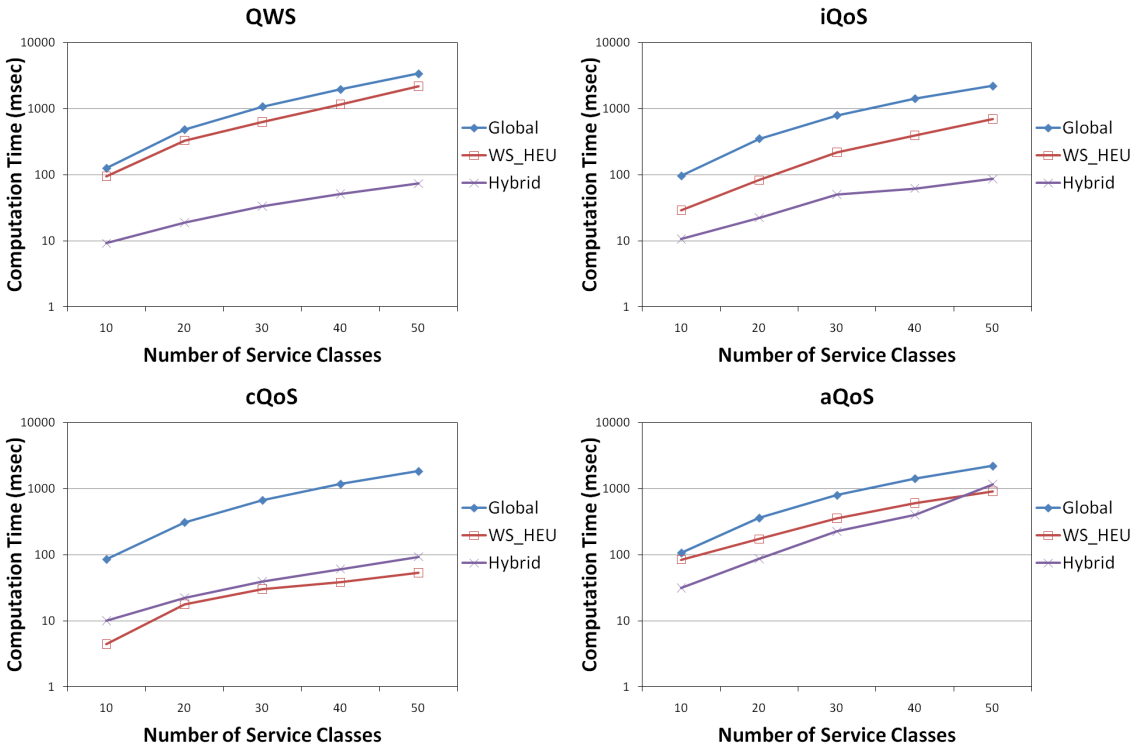


Figure 3.17 Performance vs. Number of Service Classes

utility value of all possible selections is very expensive in terms of time, especially when the number of possible combinations to consider is huge.

Therefore, we conducted a small experiment with small number of services (10 service classes, 100 service per class) and measured only the utility value u of the selection obtained by the *Hybrid* approach, the maximum utility value u_{max} of the optimal selection and the minimum utility value u_{min} of the worst solution (which we get using the *Global* approach after changing the objective function of the optimizer from maximizing to minimizing the utility value). We then computed the optimality of the selection obtained by the *Hybrid* approach and the optimality of the worst solution as follows:

$$optimality(Hybrid) = u/u_{max}$$

$$optimality(WorstCase) = u_{min}/u_{max}$$

The measured optimality of 100 instances of the selection problem are shown in Figure 3.18. This experiment shows that the selections obtained by the *Hybrid* ap-

proach are very close to the optimal selection and far enough from the worst selections. In the following larger experiments we will focus only on the distance to the optimal selection and thus only compare value u of the obtained selections with u_{max}.

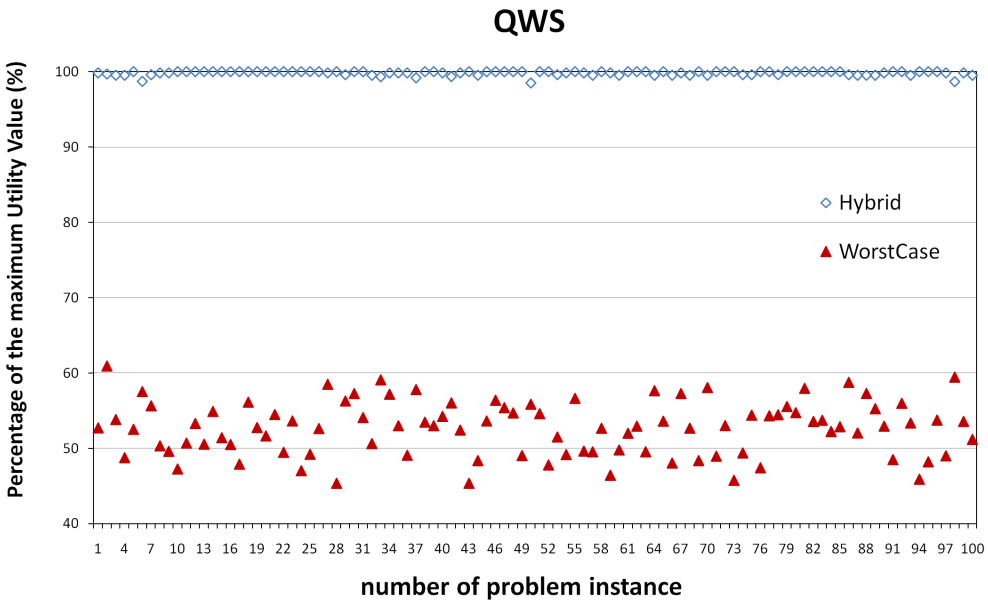


Figure 3.18 Optimality Measurement

- **Optimality vs. Number of Candidate Services**: Figure 3.19 shows the measured optimality of the *Hybrid* and *WS-HEU* approaches with different datasets and a varying number of candidate services. The results indicate that the *Hybrid* approach was able to achieve very close-to-optimal results (always above 98%). The results also show that the *WS-HEU* approach was able to achieve even a higher optimality than the *Hybrid* approach.

- **Optimality vs. Number of Service Classes**: Figure 3.20 shows the measured optimality of the *Hybrid* and *WS-HEU* approaches with different datasets and a varying number of service classes. Again, we observe that the *Hybrid* approach was able to achieve very close-to-optimal results in all cases (above 98% in average), while the optimality achieved by the *WS-HEU* was still higher than the *Hybrid* approach.

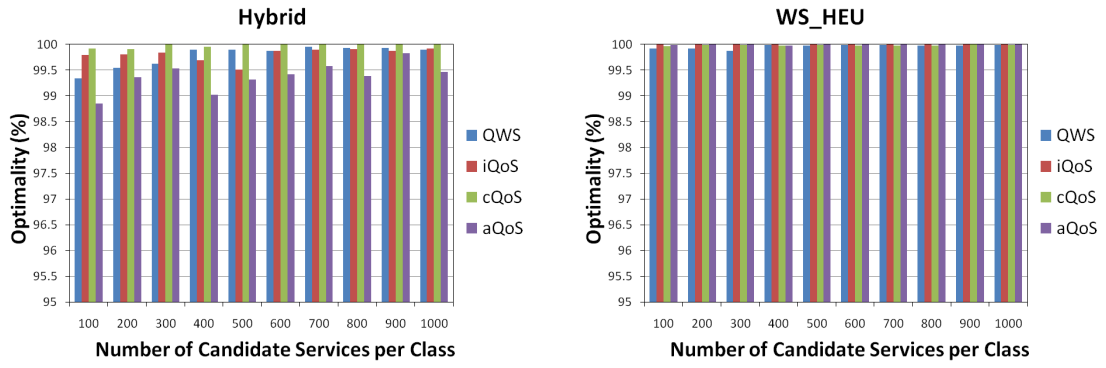


Figure 3.19 Optimality vs. Number of Candidate Services

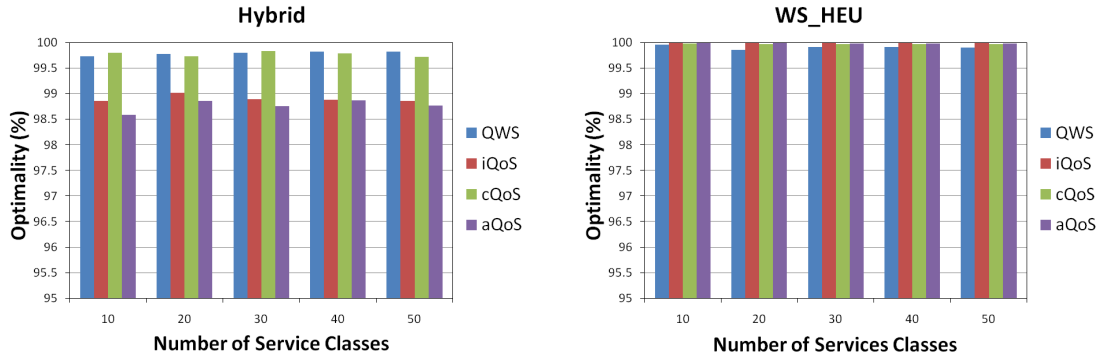


Figure 3.20 Optimality vs. Number of Service Classes

Communication Cost Comparison

In this experiment we investigate the communication overhead of deploying either of the service selection methods (i.e. *Hybrid* and *WS-HEU*) in a distributed setting. We measure the overhead in terms of the number of messages that need to be exchanged between the composer and the distributed service brokers. Without loss of generality, in this experiment we assume that each service class is managed by a separate service broker. In the results shown in Figure 3.21 we see that while the number of exchanged messages in the *Hybrid* approach (for obtaining local quality levels) remains very limited, the number of exchanged messages in the *WS-HEU* method is much higher and is constantly increasing. The results of this experiment prove our hypothesis that the *WS-HEU* method is not suitable for distributed environments. This is due to the fact that *WS-HEU* optimizes the service selection by undertaking several iterations of downgrading and upgrading of local selections (i.e. replacements of already selected

services) until no further optimization is possible. This process requires extensive message exchange with the service brokers in each round, which in a distributed environment can lead to a high communication cost.

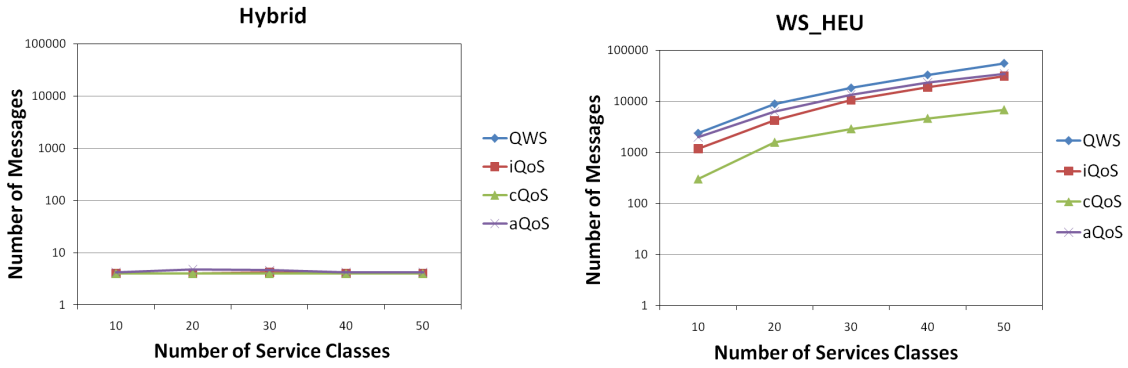


Figure 3.21 Communication Cost vs. Number of Service Classes

Summary of the Results

The results of the experimental evaluation have shown that the *Hybrid* approach significantly outperforms the *Global* approach in terms of computation time, while being able to achieve close-to-optimal results. The results have also shown that the *Hybrid* approach is more scalable than the *WS-HEU* approach with respect to the number of candidate services and service classes in the composition, except for strong correlated datasets, where the *WS-HEU* approach performs slightly better than the *Hybrid* approach. In addition, we observe that the *WS-HEU* approach is able to achieve higher optimality in most of the cases. On the other hand, our *Hybrid* approach imposes much less communication overhead in comparison with *WS-HEU* when applied in a fully distributed setting, while it is still able to achieve a reasonable level of optimality (above 98% in average). This makes the *Hybrid* approach fits better to the open and distributed environment of web services, where central brokerage of all service classes is not possible. It also fits well to the SLA decomposition scenario, where the goal is to find a feasible decomposition of the composition's SLA to local bilateral SLA's with the involved service brokers, rather than directly selecting concrete services for the composition. Both the *Global* and *WS-HEU* approaches are not appropriate for this scenario, as they can only be used for finding a concrete service for each service class in the composition. The *Hybrid* approach, on the other hand, can be used for both scenarios.

3.5.3 Evaluation of the Skyline Approach

In this section, we present the results of our experimental evaluation of the skyline approach we presented in Section 3.4. In this part of the evaluation we compare between the following QoS-aware service composition methods:

- *Global*: this method is the standard global optimization approach with all candidate services represented in the ILP model.
- *GlobalSykline*: this method is similar to the *Global* method, except that only skyline services are considered in the ILP model.
- *SkylineRep*: this method uses representative skyline services as described in Section 3.4.3.
- *Hybrid*: this is the *Hybrid* approach we presented in Section 3.3, which maps end-to-end constraints into local QoS levels.
- *HybridSkyline*: this is the modified version of the *Hybrid* approach, which uses a skyline-based method for determining local QoS levels as described in Section 3.3.1.

The aim of this evaluation is to measure the improvement in performance and success rate achieved by applying the skyline-based solutions. For this purpose, we compare between the performance of the *Global* method and the performance of the *GlobalSkyline* and *SkylineRep* methods in terms of the execution time required to find a solution, and compare between the success rate of the *Hybrid* method with the success rate of the *HybridSkyline* method. We experimented with the same four datasets we described in the previous section, i.e. the real dataset (QWS), the correlated (cQoS), anti-correlated (aQoS) and independent dataset (iQoS).

Performance Comparison

For this evaluation we considered a scenario, where a composite application involves services from 10 different service classes. Therefore, we randomly partitioned each of the aforementioned datasets into 10 service classes. We then created 100 vectors of 5 random values to represent the users end-to-end QoS constraints. Each vector corresponds to one composition request, for which one concrete service needs to be selected from each class, such that all end-to-end constraints are satisfied, while the

overall utility value is maximized. We solved each composition request using the different service selection methods and measured the average computation time of each method. To evaluate the scalability of these methods against the number of available candidate services, we repeated the experiment with different number of candidate services varying between 100 and 1000 service per class.

The results of this experiment are presented in Figure 3.22. Comparing the performance of *Global* and *GlobalSkyline* methods, we can observe that a significant gain is achieved when non-skyline services are pruned.

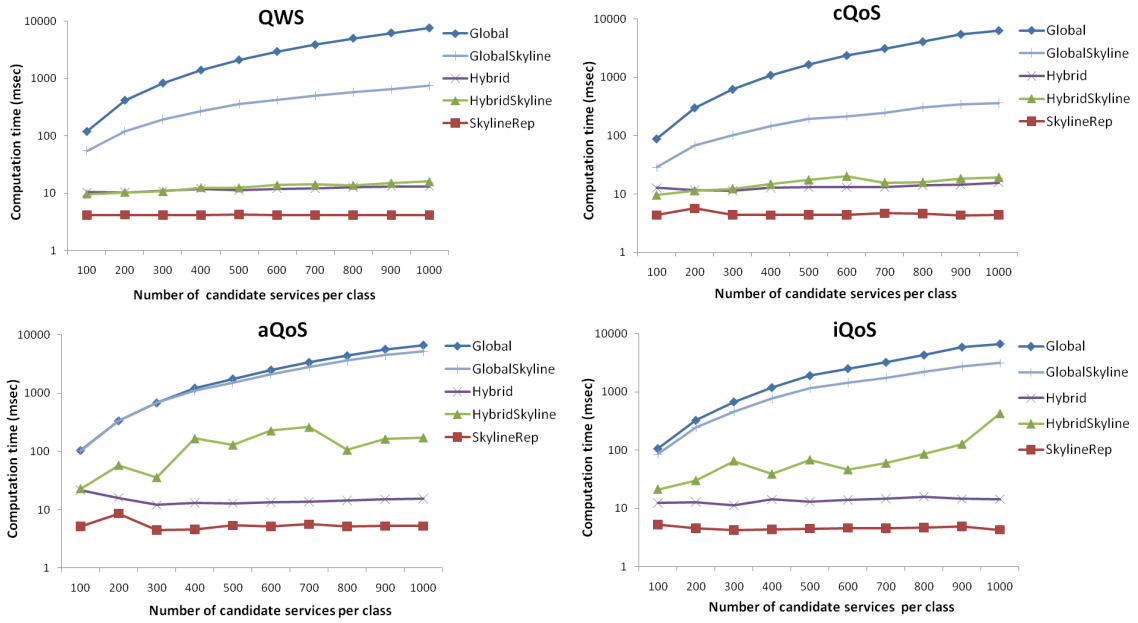


Figure 3.22 Performance vs. Number of Service Candidates

However, as expected, this gain in performance differs for the different datasets, based on the size of the skyline, with the lowest gain being recorded for the anti-correlated dataset. On the other hand, the *SkylineRep* method clearly outperforms all other methods, which shows that we can cope effectively with this limitation by using skyline representatives as described in Section 3.4.3. In general, the performance of the *HybridSkyline* method is comparable with the performance of the *Hybrid* method as long as the size of the skyline is not very large (see the performance of both methods with the QWS and correlated datasets). Although less efficient than the original *Hybrid* method with the independent and anti-correlated datasets, the *HybridSkyline* method still outperforms the *Global* method with more than an order of magnitude

gain in performance. Moreover, the *HybridSkyline* method outperforms the *Hybrid* method in terms of success rate as we will see in the next subsection.

Optimality Comparison

In order to evaluate the quality of the obtained results in the previous experiment, we also measured the optimality of the returned selection by comparing its overall utility value (u) with the overall utility value ($u_{optimal}$) of the optimal selection obtained by the *Global* approach, i,e.:

$$optimality = u/u_{optimal}$$

The measured optimality of the *SkylineRep*, *Hybrid* and *HybridSkyline* methods are shown in Figure 3.23. The results show that the *SkylineRep*. The optimality achieved by the *HybridSkyline* method was in all cases above 90%, althogh still lower than the optimality of the *Hybrid* method with some datasets such as the anti-correlated dataset. On the other hand, with the real dataset QWS, the *HybridSkyline* method was able to achieve about 97% optimality.

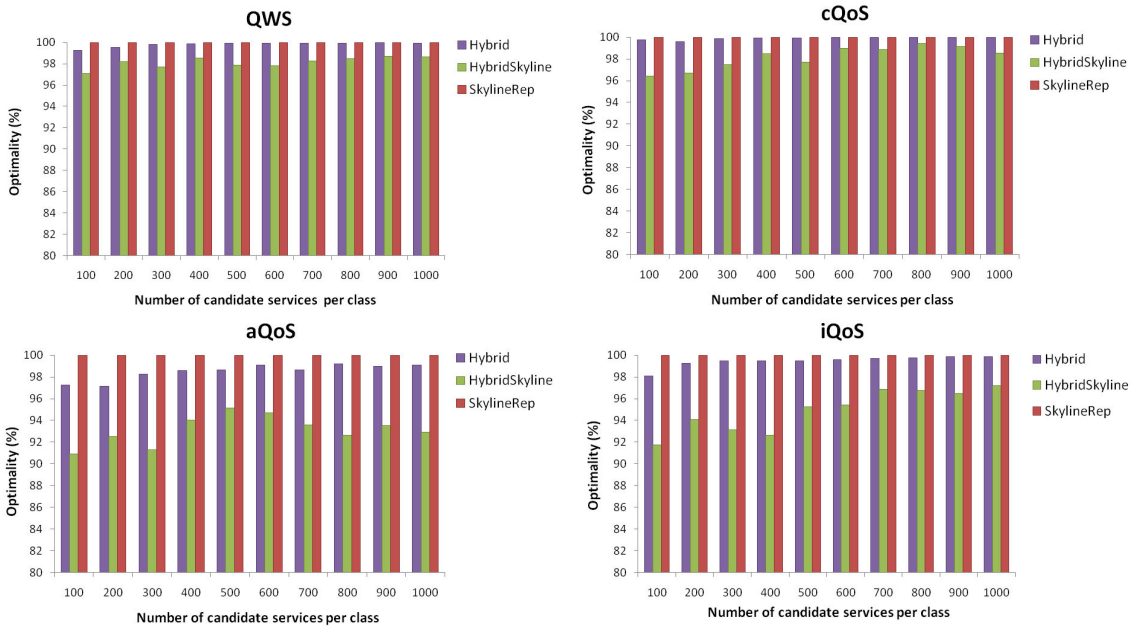


Figure 3.23 Optimality vs. Number of Service Candidates

Success Rate Comparison

Clearly, the number of feasible selections for a given composition request decreases as the number of end-to-end QoS constraints increases. This can affect the performance of all methods as more computation time is required to find a solution. More specifically, with very constrained problems the probability that the iterative algorithm of *SkylineRep* and *HybridSkyline* will need to go through more iterations until a solution is found increases. In this experiment, we measured the performance of the different methods with respect to the number of end-to-end QoS constraints. For this purpose, we fixed the number of service candidates per class to 500 services, and we varied the number of QoS constraints from 1 to 9 (notice that the total number of QoS parameters in the QWS dataset is 9). In addition, we measured the success rate, i.e., the percentage of scenarios where a solution is found, if one exists. In Figure 3.24, Figure 3.25, Figure 3.26 and Figure 3.27 we show the results of this experiment with the different datasets. We observe that *SkylineRep* clearly outperforms all other approaches. We also observe that the computation time of *Hybrid* and *HybridSkyline* methods increases as the number of QoS constraints increases. In general, the *Hybrid* approach is less scalable to the number of constraints compared to the *HybridSkyline*. As shown in the right-hand side of each of the figures, *SkylineRep* and *HybridSklyine* always find a solution. This is because *SkylineRep* and *HybridSklyine* iteratively expand the search space by examining more representative services or local QoS levels,respectively, until a solution is found or until the whole set of skyline services has been examined. In the latter case, a solution is guaranteed to be found (if one exists) according to Lemma 1. On the other hand, the success rate of the *Hybrid* method degrades significantly as the difficulty of the composition problem increases. The reason for this behavior is that the *Hybrid* method decomposes each of the end-to-end constraints independently, which in such difficult composition problems may results in a combination of local constraints that cannot be satisfied by any candidate service.

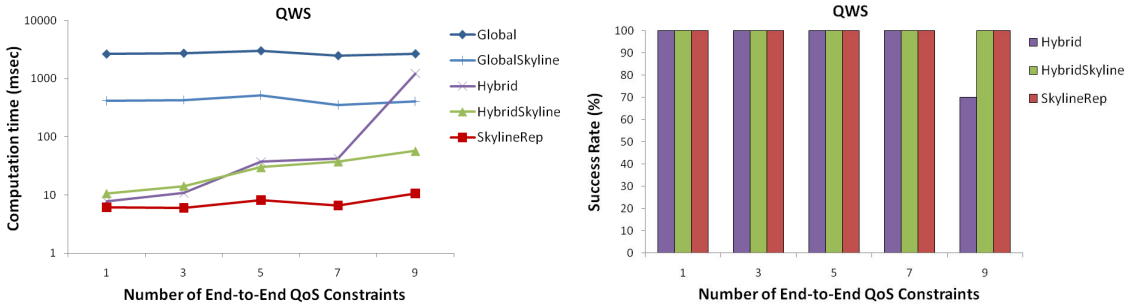


Figure 3.24 Performance and Success Rate vs. QoS Constraints - QWS

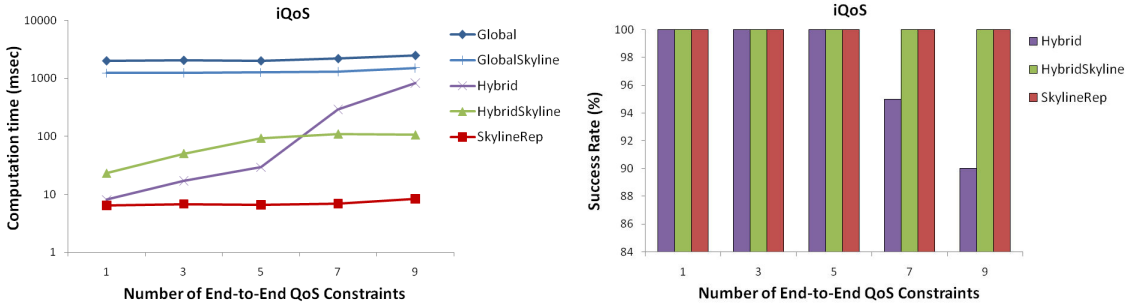


Figure 3.25 Performance and Success Rate vs. QoS Constraints - iQoS

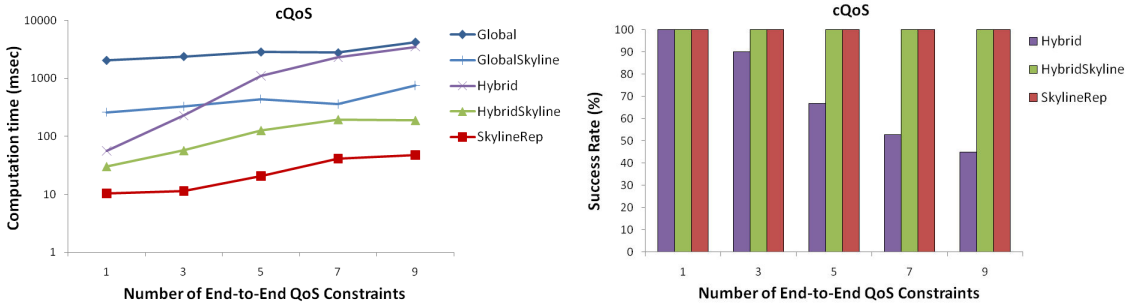


Figure 3.26 Performance and Success Rate vs. QoS Constraints - cQoS

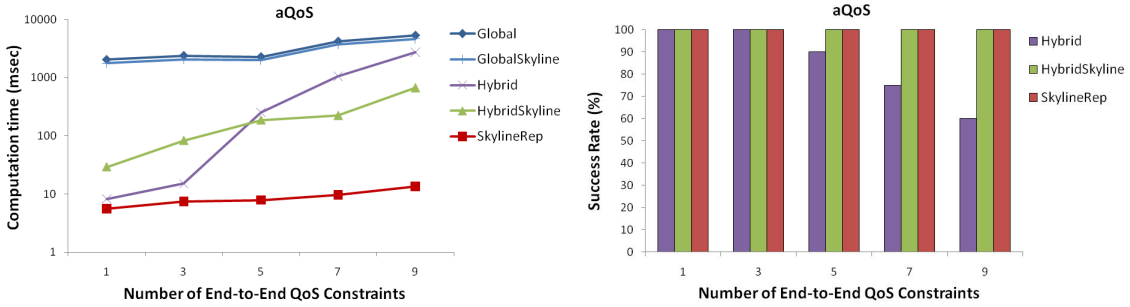


Figure 3.27 Performance and Success Rate vs. QoS Constraints - aQoS

4 Decentralized Concurrency Control for Transactional Web Services

In this chapter we address the problem of ensuring reliable transactional management of web service transactions. In particular, we focus on maintaining consistency of concurrent invocations of web services in the context of independent business processes. We start in Section 4.1 by introducing to the problem and describing a motivating scenario. In Section 4.2 we provide a formal description of web service transactions and transactional dependencies. We also show how a multi-level transaction model can be adopted for supporting concurrency control on the web services' level and propose an extension to the current web service transactions' framework to implement this model. In Section 4.3 we introduce an optimistic and fully decentralized concurrency protocol for web service transactions that can be employed in the proposed architecture. Algorithms for handling global transactional are also presented in this section. Finally, in Section 4.4 we describe our experimental evaluation of the proposed solutions and present the results of this evaluation. The presented solutions in this chapter have been published in [ADN06], [ABDN07] and [ADBN09].

4.1 Introduction

Web service-based business-to-consumer (B2C) and business-to-business (B2B) applications usually involve invocations of services running on different heterogeneous back-end systems managed by autonomous service providers. A key requirement for successful web service-based business applications is to ensure reliable execution of their processes with respect to the partners' transactional requirements. A reli-

able transaction processing should provide the illusion that each transactional process executes as if no other process were executing concurrently (serializability) and as if there were no failures (recoverability) [AAA+96]. In contrast to traditional ACID transaction models, which assume short lived transactions, transactional web service-based processes are usually long-running processes (in the order of hours or even days). Therefore, strict isolation requirements to guarantee serializability of distributed transactions have to be relaxed. A bank service provider, for example, would not accept locking its local resources (customer accounts) on behalf of some client application for unbounded time.

Advanced Transaction Models (ATM) have been proposed in the literature to address the new requirements of advanced applications (see [Elm92] for a comprehensive overview). The Open Nested Transaction Model [Gra88] was widely adopted by industry (e.g. [Comg, Come, Comf]) and academia (e.g. [AAA+96, Pap03, BPG05, KHC+05]) for web service transactions. The main feature of this transaction model is the possibility to relax the isolation property by exploiting semantic properties of operations, which allows participants to commit independently (thus, preserving autonomy). The concept of compensation plays a major role in this model to "'repair"' the semantic effects of already completed activities in the case of failures or transaction abort requests. Many commercial companies nowadays (e.g., Amazon.com) that provide transactions without isolation in their online services, also provide semantic compensation mechanisms (usually in the form of canceling an order within a given time limit). However, in the open and dynamic web service environment, business transactions enter and exit the system independently. Under isolation relaxation transactional dependencies can emerge among independent business processes, which need to be taken into account when compensation is required in order to avoid inconsistency problems. Such transactional dependencies are currently overlooked in the web service transaction models and standards.

Motivating Scenario

The following example demonstrates the problem of maintaining global consistency in the presence of concurrent service invocations and motivates the need for concurrency control at the web services level. In the example shown in Figure 4.1 two independent business processes (*Process 1* and *Process 2*) concurrently access the web service of some online banking system. The interface of this online banking web service

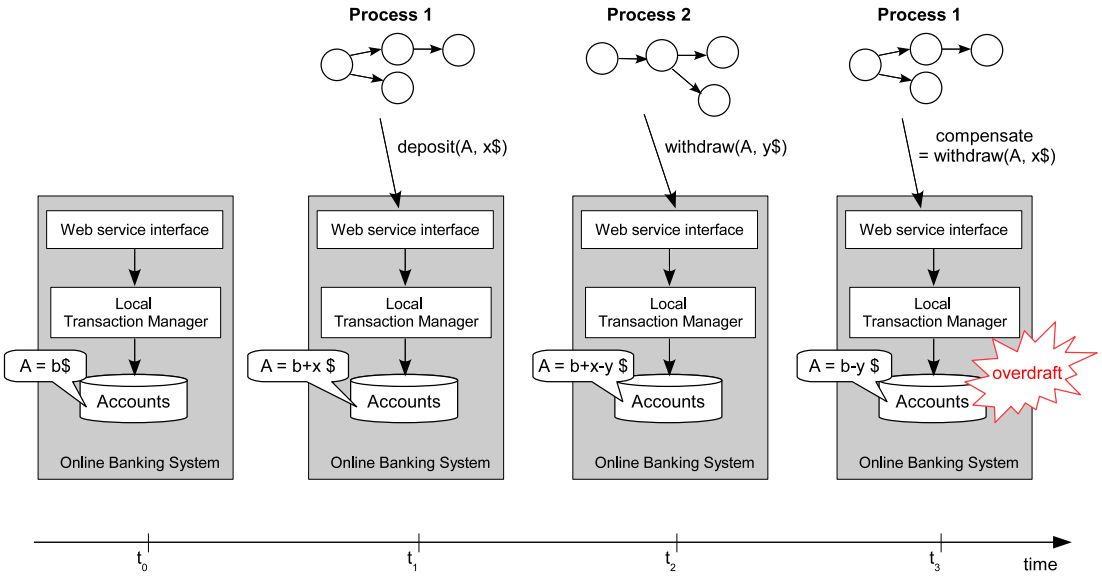


Figure 4.1 Example of a Transactional Dependency Between Two Processes

provides operations for transferring money from or to bank acoounts. Both processes update the same account (account A) through two subsequent calls to the deposit and withdraw operations respectively.

Assume that the initial balance of account A at time t_0 is $b\$$ and that the bank does not allow overdrafts (i.e. $A\$ > 0$ must always be true). *Process 1* invokes the web service at time t_1, requesting depositing account A with $x\$$. The balance of account A is now updated to $(b + x)\$$. Due to the isolation relaxation policy, the new balance is immediately made visible to other processes, even before *Process 1* has been completed. At time t_2 *Process 2* invokes the web service requesting a Withdraw operation of $y\$$ $(y > b)$ from the same account. Accordingly, the balance of account A is updated to $(b + x - y)\$$. Note that without *Process 1*'s deposit operation being successfully executed, the withdrawal operation of *Process 2* cannot be accepted by the online banking system as it would lead to an overdraft. Assume now that later, due to the failure of some activities of *Process 1*, its coordinator decides to cancel the whole process and issues a compensation request of its previous deposit operation. *Process 1*'s compensation request is received by the banking system at time t_3. The compensation is done by a semantically reverse operation (withdraw in this case) on account A with the same amount of $x\$$. However, such operation is not allowed by the banking system as this would lead to an overdraft $(b - y < 0)$.

This scenario points to the following problems:

1. Transactional dependencies can occur dynamically between autonomous processes due to concurrent access to transactional web services.

2. The transactional processes are usually coordinated by independent coordinators. The transactional dependencies are therefore invisible to the coordinators.

3. With the absence of a mechanism for detecting and managing such dependencies among concurrent invocations to transactional web services, dependency conflicts such as the one described in this scenario can lead to inconsistent overall outcome of the executed processes.

These problems raise the need for a concurrency control mechanism for transactional web services in order to ensure reliable execution of web service-based business processes. In the following sections we will show how the current solutions for web service transactional management can be extended to enable concurrency control. We will also present and compare two distributed solutions for handling global dependency cycles: the edge chasing approach for detecting global cycles at commit time, and the pre-scheduling approach for avoiding global dependency cycles.

4.2 Transaction-aware Architecture

The current standard for transactional management of long-running and loosely-coupled web service business transactions is the WS-BusinessActivity specifications [Comf]. The theoretical foundation of the WS-BusinessActivity is based on the open nested transaction model [Gra88, Mos81]. Transactions in this model can form a tree (of arbitrary height) of *sub-transactions*. The sub-transactions may commit independently of each other without having to wait for the root transaction to commit. In case of a sub-transaction failure, the client driving this business process may decide whether the overall transaction should abort or simply ignore the failed sub-transaction. The open nested transaction model, and hence, the WS-BusinessActivity coordination type also relaxes the isolation property. It permits disclosing intermediate results by autonomous participants instead of locking local resources until the end of the (global) transaction. In the case of transaction abort, the effects of already committed sub-transactions are undone by means of compensating sub-transactions. However, the assumption that all service operations can always be compensated is not realistic.

When the number of transactions having access to intermediate results increases, the compensation of some operations becomes either too expensive or even impossible.

Supporting concurrency control for the WS-BusinessActivity is challenging for the following reasons. First, business activities are usually long-running, which yields locking-based solutions (e.g. 2PL protocol) unacceptable. Second, participants of a business activity are loosely-coupled and highly autonomous, which makes solutions based on centralized concurrency control (e.g. global serialization graph testing protocol) inapplicable.

4.2.1 A Multi-level Transaction Model

In this section we introduce an architecture that supports web service concurrency control in a modular way. The architecture distinguishes between service-level and resource-level concurrency control. Figure 4.2 shows the conceptual view of this multi-layered architecture. Resource-level transactional conflicts (e.g. select/update queries to a DBMS) are managed by the Resource-level Concurrency Control component (e.g., the transaction manager of the DBMS). Transactional dependencies between interleaving processes, caused by service-level semantic conflicts (e.g. deposit/withdraw conflicts), are managed by the Service-level Concurrency Control component. This separation allows supporting any back-end system and concurrency control protocol (e.g. 2PL, multiversion, etc.) [WV01].

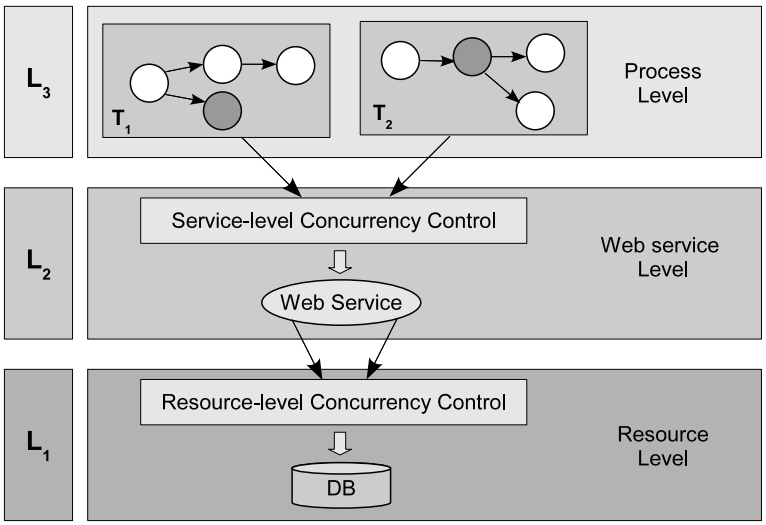


Figure 4.2 The Multilayered architecture

The proposed architecture can be elegantly modeled with the multilevel nested transaction model from [Wei86, Wei91]. This model has a sound theoretical foundation and fits well to multi-layered architectures where each layer has its own level-specific semantics of the set of operations. The model is a special case of the open nested transaction model with the requirement that all leaf nodes in a transaction tree have the same distance to the root. The nodes in a transaction tree correspond to operations at particular levels of abstraction, where the edges represent the implementation of each operation at level L_i by a sequence of operations at the next lower level $L - i - 1$ (for $i = 1, , n$ in bottom-up order of a n-level system). In the architecture shown in Figure 4.2 we have a 3-level system (L_1= resource level, L_2=service level and L_3=process level). To map it onto our scenario, at the process level (L_3) there can be a set of business processes that invoke the online banking service in the context of some business activities. In the service level (L_2) we have the web service interface, which provides an access to the customers' accounts for online banking. For the sake of simplicity, we assume that each activity on the process level is mapped to one web service operation (e.g. a deposit/withdrawal operation). The resource level (L_1) is a level of some database where the customers' accounts are managed. Each web service operation is mapped to a database transaction on the resource level. The consistency of the overall system can only be guaranteed, if the produced schedule at each level is guaranteed to be serializable (i.e. equivalent to some serial execution of the involved transactions) [Wei86]. We restrict our focus on process-level transactions with web service level operations as elementary operations of these transactions. The correctness of access to low level resources is then left to the resource-level transaction manager (e.g., a DBMS).

Transactional Dependencies

There is a dependency relation between two process-level transactions T_1 and T_2 if the outcome of T_2 is influenced by the outcome of T_1. We formally define this relation s follows:

Definition 5. (Transactional Dependency)

There is a transactional dependency between two transactions T_1 and T_2, from level L_3 if there are two activities (service operations) $a_1 \in T_1$ and $a_2 \in T_2$ from level L_2 such that:

- the failure of a_1 causes the failure of a_2, or
- the success of a_1 causes the failure of a_2.

We refer to T_1 as the dominant transaction and to T_1 as the dependent transaction. In our online banking scenario, *Process 2* is dependent on *Process 1*. The transactional dependency relation is analogous to the semantic conflict relation in database transactions terminology [Wei91]. Ensuring consistency of business transactions requires tracking these dependencies and handling them appropriately. This can be achieved by building and maintaining the so-called dependency graph (analogous to the serialization graph in databases) and ensuring it contains no cycles.

Dependency Graphs

A dependency graph is a directed graph where the nodes represent transactions and the edges represent transactional dependencies between them. Each edge points from the dependent transaction to the dominant one. Dependency graphs are updated whenever a new transaction enters or leaves the system. A transaction with no outgoing edges (i.e., it has no dominants) is said to have an *exclusive* lock on the shared resources. All its dependent transactions are said to have *shared* locks on these resources. Figure 4.3 depicts an example of a dependency graph composed of three active transactions. In this example, transactions T_1 and T_2 have exclusive locks, whereas T_3 has shared locks with T_1 and T_2. The direction of the edges indicates that the outcome of T_3 depends on the outcome of both T_1 and T_2. Therefore, a concurrency control mechanism is required to detect such dependencies and ensure that T_3 does not leave the system before T_1 and T_2 successfully terminate.

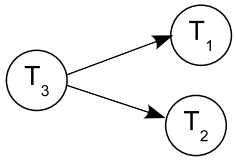


Figure 4.3 Example of a dependency graph

Global Consistency through Local Guarantees

In the distributed and open environment of web services, independent business process transactions can co-exist without knowing about each other. Under the absence

of a global transaction manager, maintaining the global dependency graph is therefore not feasible, and a distributed management of the transactional dependencies is required. To this end, we refer to the results of extensive research in the databases field (see [WV01] for a comprehensive overview). It has been shown that global consistency can be achieved through strong local guarantees: i.e., by ensuring that each local transaction schedule satisfies either the Rigorousness (RG) or the Commit-Order preservation (CO) criteria [WV01].

The Rigorousness criterion requires isolating the intermediate results of active transactions until the termination (commit or abort) of all preceding transactions. As a result of this strict policy, the concurrency level of the system decreases significantly, which negatively influences the overall performance. A de facto standard implementation of a rigorous concurrency control strategy is the well-known (strict) 2PL protocol.

On the other hand, the Commit-Order preservation criterion permits accessing data processed by active transactions under the constraint that the commit order of any two semantically conflicting transactions must preserve their execution order. This practically means delaying the commit of transactions in conflict till after the commitment of their preceding transactions. Commit-order preserving schedules therefore allow higher level of concurrency than rigorous schedules, which leads to better performance and higher overall throughput. However, this gain in performance does not come without costs. Commit-order preserving schedules run under the risk that distributed transactions might get blocked by other relatively longer transactions at commit time which can lead to unacceptable waiting times. In this paper, we adopt the Commit-Order preserving policy for concurrency control and propose solutions to cope with the aforementioned limitations.

4.2.2 Extending the Web Service Transaction Framework

In Section 2.3.2 we presented the standard framework for web service transactions, and its associated industrial specifications: WS-Coordination [Comg], WS-AtomicTransaction [Come] and WS-BusinessActivity [Comf] for managing and coordinating distributed web service transactions. We also pointed to the limitation of this framework in handling transactional dependencies that occur due to the relaxed isolation among concurrent transactions in the WS-BusinessActivity specification. In this section we introduce our proposal for extending the current web service transactions'

framework in order to solve this problem.

WS-Scheduler

We extend the current web service transactions' framework by introducing the WS-Scheduler to implement the service-level concurrency control according to the the multi-level transactions model we presented in Section 4.2. The WS-Scheduler resides on the web service provider's side and is responsible for managing concurrent instances of the WS-Coordination (and WS-BusinessActivity) protocol. Figure 4.4 shows how the WS-Scheduler is integrated into the standard framework.

The WS-Scheduler maintains a list of active participants, and the transaction contexts they are involved in. This can be easily implemented as part of the invocation mechanism. As every service request within a global transaction is associated with the coordination context according to the WS-Coordination standard [Comg], the context is extracted from the received message and a new participant is created. The control is then transferred to the Scheduling Service of the WS-Scheduler, which (on behalf of the created participant) registers itself as a participant of the given context. This is done by invoking the registration service at the given coordinator address. The Scheduling Service then checks if there are transactional dependencies between the requested service operation and previously executed operations of active transactions. The WS-Scheduler maintains a dependency graph and decides (based on the deployed concurrency control mechanism) when transactions are allowed to commit their activities and leave the system. This ensures that commit and compensation requests are handled consistently. In the following sections we describe in detail how potential transactional conflicts are detected and handled.

In the extended framework all coordination (response) messages from the coordinator are received and processed by the WS-Scheduler before forwarding them to the participant. For example, when a commit message is received from the coordinator, the WS-Scheduler has to check first whether this transaction is allowed to commit before forwarding this message to the respective participant. Consequently, all the states of the web services in the different contexts (e.g. completing, compensating, aborted etc) as defined by the WS-BusinessActivity specification are kept by the WS-Scheduler. Similarly, all responses by the participants to the coordination messages are sent through the WS-Scheduler, and the WS-Scheduler updates the dependency graph accordingly.

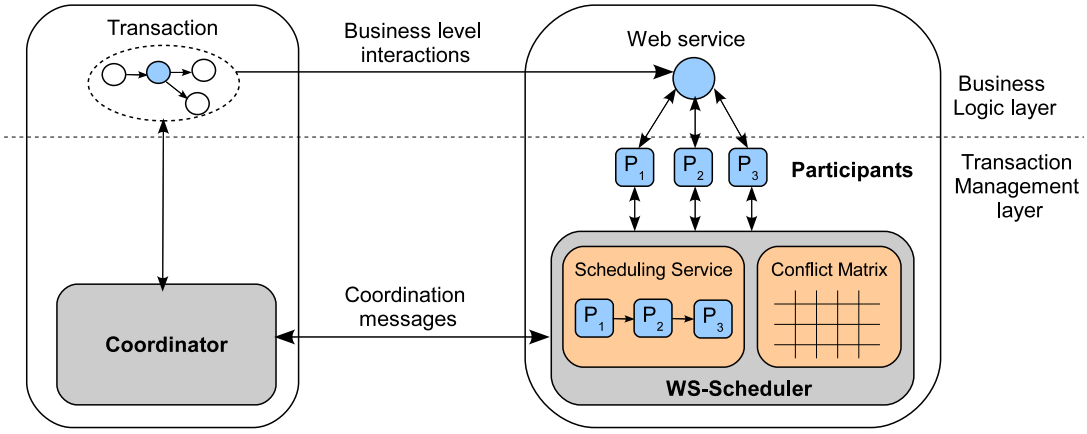
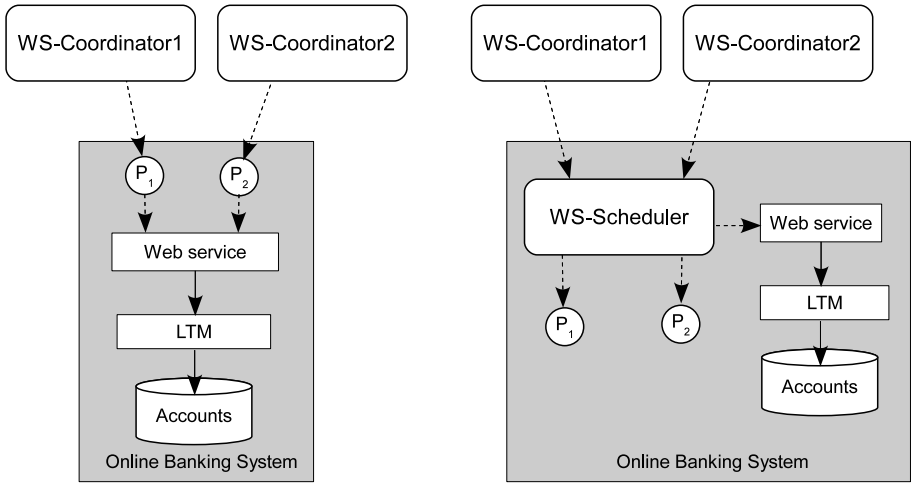


Figure 4.4 Extended web service transaction framework

In Figure 4.5 we refer back to the use case scenario of Section 4.1. The two processes *Process 1* and *Process 2* access the online banking service within two independent contexts. The two contexts are coordinated by two autonomous coordinators *WS-Coordinator 1* and *WS-Coordinator 2* and represented locally by two participants P_1 and P_2 respectively (Figure 4.5-a). Potential transactional dependencies between the two contexts cannot be detected as they are not visible to the two coordinators. With the deployment of the WS-Scheduler as shown in Figure 4.5-b, such dependencies can be easily detected and appropriately handled. Upon the invocation of the deposit operation by *Process 1* a new node (P_1) is added to the transactional dependency graph to represent the instance of the service, which is participating in the context of *Process 1*. Another node (P_2) is also added to the graph as *Process 2* invokes the withdrawal operation of the service, and an edge between the two instances (P_1 and P_2) is added to the graph to represent the dependency relation between the two processes. Keeping this linkage between the two instances and using a concurrency control mechansim (as we will describe later in this chapter), enables the WS-Scheduler to ensure that conflict between the two processes such as the one described in the motivating scenario not occur.

Detection of Transactional Dependencies

The conflict matrix, which is used by the WS-Scheduler at run-time to detect potential transactional dependencies between two subsequent service invocations, is built at design-time by the service provider based on his knowledge about the implemen-



(a): Using the current framework (b): Using the extended framework

Figure 4.5 The Online Banking Service with the extended framework

tation of the services. The conflict matrix is an $N \times N$ matrix, where N is the total number of operations that can be invoked via web service calls. The conflicts can be defined based on the semantics of these operations (i.e. based on their behavior and effects) to reflect their execution commutativity relations [Wei91]. Two operations do semantically conflict if they do not commute, i.e. if changing the order of their execution results in different final state. Consider our Banking web service example, and assume that the service has three operations: *deposit*(A, x\$), *withdraw*(A, x\$) and *getBalance*(A). A deposit (or withdraw) operation performs a credit (or debit) action on the requested account A with the specified amount of money (i.e. x\$). A getBalance operation returns the current balance of the specified account and writes the returned value into a log record. According to this functional description a withdraw operation would semantically conflict with a deposit operation if it was called while the transaction that invoked the deposit operation is still active (i.e. neither committed nor aborted). On the other hand, invoking the deposit operation after a withdraw operation can be tolerated. The operation getBalance does not conflict with any other operation in this example.

However, this definition of semantic conflicts has its limitation: decisions about conflicts are made independently on the accessed resources and their status at run time. Therefore, we extend the commutativity-based conflict definition to capture the dynamic nature of semantic conflict relations. We use conflict predicates, which

can be defined by the service provider at design time and evaluated at request time to detect any transactional conflicts. The conflict predicate takes input parameters (e.g. account number) and the current status of the targeted resources (i.e. current balance) as parameters and returns either a *TRUE* (i.e. conflicting) or *FALSE* (i.e. not conflicting). For example, in our Banking web service example, a conflict predicate for the deposit(A_1, x_1)/withdraw(A_2, x_2) operations would check if the two operations access the same bank account (i.e. if $A_1 == A_2$), and also compare the requested amount of money (x_2) with the last committed balance ($b - x_1$). If the requested amount of money is greater than the current balance a conflict is detected and the predicate evaluates to *TRUE* and it evaluates to *FALSE* otherwise. The conflict predicate for state-independently non-conflicting operations always returns a *TRUE*, while the conflict predicate for state-independently conflicting operations always returns a *FALSE*. Table 4.1 gives an example of a conflict matrix for the Banking web service including the conflict predicates. In this paper we assume that semantic conflict matrices are built and updated by the service provider and made accessible to the WS-Scheduler.

Request operation	**Last executed operation**		
	withdraw (A_1, x_1)	deposit(A_1, x_1)	getBalance(A_1)
withdraw (A_2, x_2)	*FALSE*	$A_1 == A_2 \wedge x_2 > b - x_1$	*FALSE*
deposit(A_2, x_2)	*FALSE*	*FALSE*	*FALSE*
getBalance(A_2)	*FALSE*	*FALSE*	*FALSE*

Table 4.1 Conflict Matrix of the Banking Web Service

4.3 Concurrency Control For Web Services

Extending the standard Web service transactions framework with WS-Schedulers enables the detection of transactional dependencies between concurrent processes. However, once detected, these dependencies need to be handled appropriately to ensure consistent outcome of the dependent processes. In this section we present a concurrency control mechanism for this purpose. We introduce a concurrency control protocol, which is a distributed variant of the conventional Serialization Graph Testing

protocol [WS84] and implements the Commit-Order preservation policy [WV01]. We also present a distributed solution for handling global dependency cycles.

4.3.1 Optimistic Decentralized SGT Protocol

The proposed protocol is a distributed variant of the original Serialization Graph Testing protocol (SGT) [WS84]. The SGT protocol maintains a graph representation of the transactional conflicts among active transactions, called serialization graph (i.e. dependency graph in our case). The global serializability of the concurrent transactions is guaranteed by ensuring that the graph always remains acyclic. We implement a distributed variant of the SGT protocol, in which every WS-Scheduler maintains a local view of the global dependency graph. The local sub-graphs capture dependency relations among transactions that have active invocations to local services. Each WS-Scheduler ensures that its local dependency sub-graph remains acyclic. WS-Schedulers applying the Commit-Order preserving policy to control the commit order of concurrent transactions. As discussed earlier in Section 4.2.1, this is an optimistic concurrency control policy, in which concurrent access to local services is accepted immediately, while a consistency check is made at the commit time. The consistency of transactions' outcome is ensured by the WS-Schedulers by applying the following two rules:

1. A transaction is only allowed to commit after all its dominant transactions have committed.

2. If a transaction aborts or compensates its local activities, the local activities of its dependent transactions are compensated as well.

As a consequent of the first rule, any commit request issued by a dependent transaction is delayed until all its dominants have committed. Therefore, we add a new state, i.e. the waiting state, to the defined states of the WS-BusinessActivity specification [Comf]. An extended version of the abstract state diagram of the WS-BusinessActivity's BusinessActivityWithCoordinatorCompletion protocol including the new waiting state is shown in Figure 4.6. In addition to the defined message types, we add the message *WAIT* in our protocol to inform the coordinators that their commit request hast to be delayed due to consistency reasons. The WS-Scheduler keeps track of the current state of all concurrent participants and their transactional

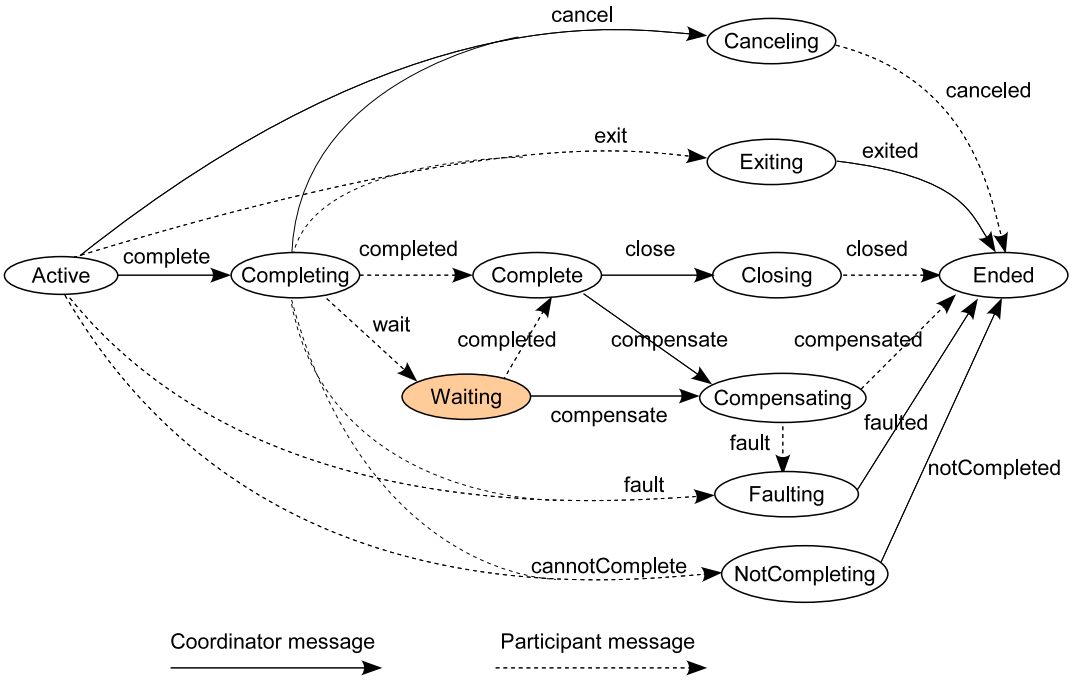
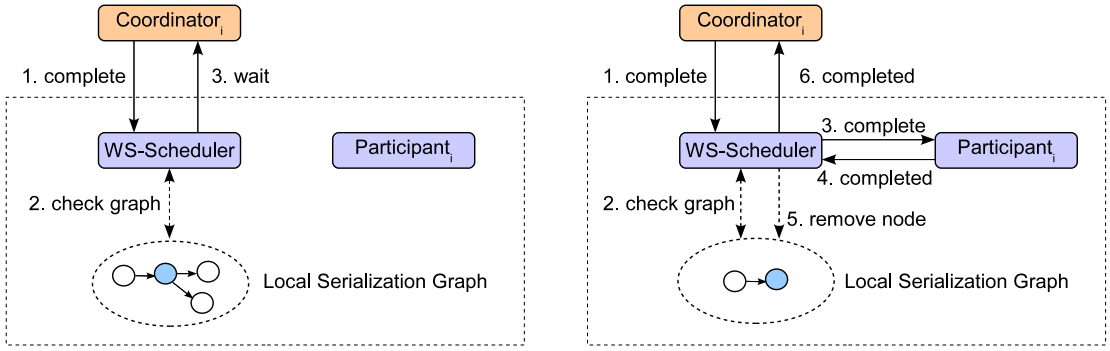


Figure 4.6 Abstract state diagram of BusinessAgreementWithCoordinator-Completion with the new waiting state

contexts. As soon as all dominants of a waiting transaction commit, the delayed commit request is forwarded to the web service and the transaction's coordinator is informed by a *COMPLETED* message.

The example shown in Figure 4.7 shows how the WS-Scheduler controls the commitment of transactions based on the local dependency graph. The commit requests (i.e. *COMPLETE* messages) from transaction coordinators are received by the WS-Scheduler on behalf of the participants. The WS-Scheduler then checks his local dependency graph to determine whether this transaction has outgoing edges. In Figure 4.7-a the WS-Scheduler finds some outgoing edges of the transaction node (i.e. the transaction is holding a shared lock) and decides to delay the commitment of this transaction until all its dominants terminate. The *COMPLETE* message is not forwarded to the participant and a *WAIT* message is sent to the coordinator. This is important to avoid conflicting situations like the one given in the Online Banking Service scenario from Section 4.1. The dependency relation between the deposit and withdrawal operations can be defined in the conflict matrix at design time. Using this information, the WS-Scheduler can detect at run time the dependency between *Process 1* and *Process 2*. Accordingly, an edge from *Process 2* to *Process 1* is added



(a): delayed commit (b): immediate commit

Figure 4.7 WS-Scheduler decides upon commit requests

to the local SGT sub-graph. The WS-Scheduler delays the commitment of *Process 2* until *Process 1* terminates. This ensures that *Process 1* can compensate its deposit operation safely when required. The compensation of *Process 1*'s deposit operations triggers the compensation of *Process 2*'s withdrawal operation automatically, in order to preserve the overall consistency of the system. Figure 4.7-b depicts another situation, in which the WS-Scheduler finds that the transaction node in the sub-graph has no outgoing edges (i.e. holding an exclusive lock). The commit request is accepted and forwarded to the corresponding participant immediately. Upon receiving the completed message from the participant, the WS-Scheduler removes the node from its graph and forwards the *COMPLETED* message to the coordinator.

WS-Scheduler Protocol

The WS-Scheduler protocol is presented in Algorithm 6. The WS-Scheduler maintains the SGT local sub-graph, a list of local web services and a list of active participants and their transactional contexts including their current states. The WS-Scheduler takes the Conflict Matrix as an input and uses this matrix for detecting potential dependencies at run time.

Lines 16-33 in Algorithm 7 describe how the WS-Scheduler updates its SGT sub-graph upon receiving new service invocation requests. A new node is added to the graph if the invoking transaction (i.e. process) has not yet been added to the graph. Using the Conflict Matrix, the WS-Scheduler detects potential dependencies and adds a new edge from the invoking transaction's node to every conflicting node. The edges indicate that the new transaction is not allowed to commit before all its dominant

transactions terminate. The WS-Scheduler then checks if the local dependency graph remains acyclic after adding the new edges. If a cycle is found in the graph, the new service invocation request is rejected and the new added node and all its outgoing edges are removed from the graph.

Lines 35-50 in Algorithm 8 describe how the WS-Scheduler responds to coordination messages from transaction coordinators. When a *COMPLETE* message is received (i.e. a commit request) the WS-Scheduler forwards the message to the corresponding participant only if the transaction does not have any dominants (i.e. its node does not have any outgoing edges). Upon receiving the *COMPENSATE* message from a transaction coordinator the WS-Scheduler triggers the compensation of all its dependent transactions before forwarding the message to the participant.

Lines 52-61 in Algorithm 9 describe how the WS-Scheduler responds to internal messages from the participants. All messages are forwarded immediately to the corresponding transaction coordinator. In addition, after forwarding a *CANCELED* or *CLOSED* or *COMPENSATED* message, the WS-Scheduler removes the transaction node as well as the calling participant from the dependency graph and the participants list respectively as this indicates that the transaction has terminated.

Algorithm 6 WS-Scheduler Protocol - Main Procedure

Input :
$SG = \{\}$ // local serialization graph
$S = \{s_1, ..., s_n\}$ // list of local services
$CM = n \times nMatrix$ // Conflict Matrix
$P = \{\}$ // list of active participants

```
1:  while (true) do
2:      if (newRequestReceived(T_t, s_j)) then
3:          // process new request from transaction T_i for service s_j
4:          execute Algorithm 7
5:      end if
6:      if (externalMessageReceived(m, T_i)) then
7:          // process message m received from coordinator of T_i
8:          execute Algorithm 8
9:      end if
10:     if (internalMessageReceived(m, P_i)) then
11:         // process message m received from participant P_i
12:         execute Algorithm 9
13:     end if
14: end while
```

Algorithm 7 WS-Scheduler Protocol - Part I

15: // processing new request from transaction T_i for service s_j
16: **if** ($\neg$ SG.contains(T_i)) **then**
17: SG.addNode(T_i)
18: P_i = createParticipant(T_i)
19: $P+ = P_i$
20: SG.setStatus($T_i, ACTIVE$)
21: **end if**
22: D = getDominants(CM, SG, s_j)
23: **for all** T_k such that $T_k \in D$ **do**
24: SG.addLink(T_i, T_k)
25: **end for**
26: **if** (SG.isCyclic) **then**
27: rejectRequest(T_i, s_j)
28: SG.removeNode(T_i)
29: $P- = P_i$
30: sendMessage($T_i, CANNOTCOMPLETE$)
31: **else**
32: execute(s_j)
33: **end if**

Algorithm 8 WS-Scheduler Protocol - Part II

34: // processing message m from coordinator of T_i
35: **Switch** (m)
36: **case** COMPLETE:
37: **if** ($\neg$SG.hasDominant(T_i)) **then**
38: forward(m, P_i)
39: **else**
40: sendMessage(T_i, WAIT)
41: **end if**
42: **EndCase**
43: **case** COMPESATE:
44: compensateDependentOf(T_i)
45: forward(m, P_i)
46: **EndCase**
47: **case** CANCEL or CLOSE:
48: forward(m, P_i)
49: **EndCase**
50: **EndSwitch**

Algorithm 9 WS-Scheduler Protocol - Part III

```
51: // processing message m from participant P_i
52: Switch (m)
53:     case COMPLETED:
54:         forward(m, T_i)
55:     EndCase
56:     case CANCELED or CLOSED or COMPENSATED:
57:         forward(m, T_i)
58:         SG.remove(T_i)
59:         P− = P_i
60:     EndCase
61: EndSwitch
```

4.3.2 Handling Global Dependency Cycles

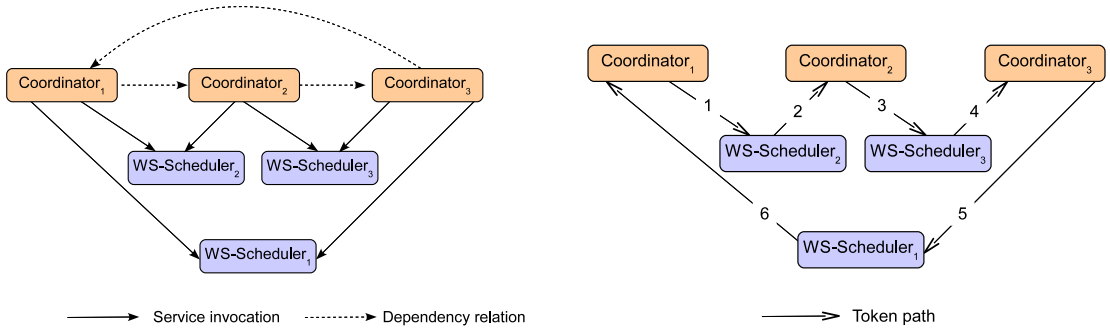
By preserving the commit order of transactions, WS-Schedulers can guarantee consistency of their accessed data. However, the distributed implementation of the commit differing policy has a side effect, namely global waiting cycles. These cycles occur as a result of dependency cycles that are neither visible to WS-Schedulers, nor to WS-Coordinators. Figure 4.8-a depicts an example of such dependency cycles. In the worst case, such cycles can lead to having the transactions waiting for ever.

Definition 6. (Waiting Cycle) A waiting cycle is a dependency cycle involving k transactions $T_1, , T_k, k > 1$, such that: 1) T_i depends on T_{i+1} for $1 \leq i \leq k-1$ 2) T_k depends on T_1 3) T_i is ready to commit, $1 \leq i \leq k$

According to this definition, we consider only dependency cycles in which all involved transactions have already reached the ready-to-commit state. In the following we present two methods for for handling global waiting cycles in a fully distributed manner: the Edge Chasing approach and Pre-Scheduling approach.

Edge Chasing Approach

In order to detect global waiting cycles in the absence of global knowledge we need a distributed solution that leverages local knowledge of the involved transactions. The edge chasing method [Kna87], which was designed for detecting deadlocks in distributed databases, has been proposed in [CJK+05] and [HST05] for this purpose. The transaction coordinators maintain local versions of the dependency graph and update it based on the conflict information they receive from the web service



(a): Example of a global dependency cycle

(b): Edge chasing-based detection

Figure 4.8 Dependency cycle detection using Edge Chasing

providers. Conflicts are then resolved by direct communication among the conflicting transactions. Although successful in detecting global waiting cycles, this approach suffers from several problems. First, the exchanged dependency information can reveal some business relations and activities, which might be confidential. Therefore the assumption that each transaction initiator would like to exchange these information is unrealistic. Second, handing over the concurrency control and maintaining transactional dependency graphs to the transactions' coordinators is not practical, since it requires extensive communication between independent coordinators in order to keep their versions of the graphs up-to-date.

In contrast to this approach, our solution separates the roles of the transaction coordinators (commitment protocol) and transaction schedulers (concurrency control protocol). Maintaining the dependency graph by a WS-Scheduler does not require any extra communication with other transactions than those who are actually using the services managed by this WS-Scheduler.

We apply the edge chasing algorithm in a way that avoids direct communication between independent transactions as in the example shown in Figure 4.8-b. The cycle detection process can be started by a WS-Coordinator upon the receipt of a *WAIT* message. The WS-Coordinator creates a unique token (e.g. using the transaction id and client IP address). We call this token a *WaitingCycleCheck*. The token is then sent to the WS-Scheduler that sent the *WAIT* message. The WS-Scheduler forwards the token to the sender's dominants (according to its local dependency graph). Each of the receiving WS-Coordinators in turn checks the status of its web services and responds as follows. If the WS-Coordinator has no services in the waiting state, it replies by sending a *NoWaitingCycle*. Otherwise, the WS-Coordinator propagates

the *WaitingCycleCheck* token to the (WS-Schedulers of the) waiting web services. This policy ensures that the token is only propagated when all involved transactions are in a waiting state and ready to commit. This is an important condition that adheres to Definition 6 and is useful for the cycle resolution as we will see later. As a result of the cycle detection process, the initiator of this check either receives a *NoWaitingCycle* token or its own *WaitingCycleCheck* token. While the former case indicates that some of the dominant transactions are still busy, which means that the WS-Coordinator has to wait, the latter case indicates the existence of a waiting cycle, which needs to be resolved. Conventionally, waiting cycles are resolved by means of either a complete or partial roll back of (some of) the involved transactions [Kna87]. A victim selection policy is usually applied to select the transactions to be restarted. Note that in the context of Web Services-based business transactions restarting already completed transactions can reduce the performance dramatically. Therefore, we use a forward cycle resolution policy instead. Based on the strong condition that WS-Coordinators forward the *WaitingCycleCheck* token only if they are ready to commit, we allow WS-Coordinators to commit their activities once a waiting cycle is detected, as long as all involved transactions are ready to commit. The readiness to commit is implicitly confirmed by the transaction coordinators when forwarding the *WaitingCycleCheck* token instead of responding with a *NoWaitingCycle* token. As soon as a WS-Coordinator receives his own *WaitingCycleCheck* token, it knows that t is involved in a dependency cycle and that all involved transactions are ready to commit. By committing and closing own activities, the dependency cycle is resolved and other transactions can safely commit as well.

Pre-Scheduling Approach

In contrast to the edge chasing approach, pre-scheduling of transactions solves the waiting cycle's problem without high communication costs. This is useful for environments where the probability of getting into transactional conflicts is very high. For example, when some web services are heavily used by many concurrent business transactions. In such case, using the edge chasing approach would raise a high communication cost for detecting and resolving global dependency cycles. The basic idea of the pre-scheduling approach is to impose some time constraints on using Web services and communicate this information with service requesters. By checking the time constraints of all Web services in a transaction, WS-Coordinators detect timing

conflicts that can lead to blocking the transaction at commit time and handle them appropriately prior the actual execution of the transaction.

Recall the concept of exclusive and shared locks from Section 4.2. A transaction that does not depend on the outcome of any other transaction is said to have an exclusive lock. This means that this transaction can commit its local changes immediately without any delay. A transaction holding a shared lock, on the other hand, is not allowed to commit until all its dominant transactions terminate and its shared lock is upgraded to an exclusive lock. In the pre-scheduling approach, WS-Schedulers impose time constraints on holding exclusive locks. Any timing conflict between the participating Web services can lead to blocking the transaction at commit time.

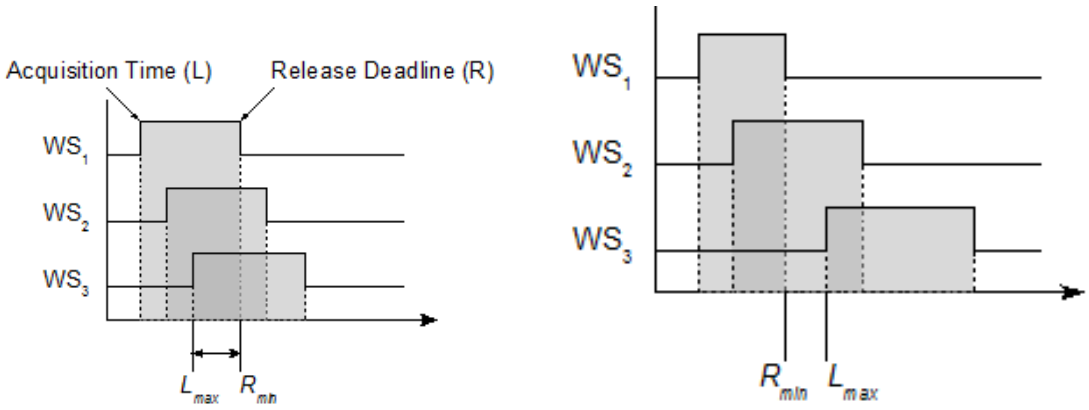
In the following we give a formal definition of a transaction schedule.

Definition 7. (Transaction Schedule) Let T be a business transaction composed of n Web services $WS_1, ..., WS_n$. Let L_i be the expected time for acquiring the exclusive lock of WS_i and R_i be the deadline for releasing this exclusive lock. A transaction schedule S of T is a schedule in which L_i and R_i are defined for all $WS_i \in T$, $1 \leq i \leq n$.

Both the expected time for acquiring an exclusive lock of a Web service and the deadline for releasing the lock are specified by the corresponding WS-Scheduler. The WS-Schedulers specify these time constraints based on the concurrency policy of the service provider, the current status of the local SGT sub-graph and statistical information about the expected service execution duration (i.e. mean and standard deviation values). WS-Schedulers provide the necessary interface for WS-Coordinators (e.g. via a Web service interface) to inquiry about these time constraints when required. In the following we give a formal definition of blocking and nonblocking schedules.

Definition 8. (Blocking vs. Nonblocking Schedule) Let $T = \{WS_1, ..., WS_n\}$ be a business transaction. Let $L_{max} = \max_{i=1}^{n}(L_i)$, be the latest exclusive lock acquisition time and $R_{min} = \min_{j=1}^{n}(R_j)$, be the earliest deadline for releasing an exclusive lock in a schedule S of T. Schedule S is nonblocking iff: $R_{min} > L_{max}$ and is blocking otherwise.

In other words, a transaction schedule is blocking if the acquisition of the required exclusive locks cannot be synchronized, i.e. not all needed exclusive locks can be acquired before the earliest release deadline. The synchronization of the exclusive locks acquisition ensures that all web services are able to commit at the same time. Con-



(a): conflict-free schedule (b): conflict between WS_1 and WS_3

Figure 4.9 Avoiding Dependency Cycles via Pre-Scheduling

sequently, the invoking business transaction is not going to be blocked (i.e. delayed) by any of the participating WS-Schedulers at commit time.

We explain this further in the following example. Figure 4.9 illustrates two example schedules for a transaction with three Web services. The schedule in Figure 4.9-a is an example of a nonblocking schedule. We see that the exclusive lock holding spans (i.e. time span between lock acquisition L and lock release R) of all web services are overlapping, i.e. locks can be acquired (and released) before the earliest provider's deadline R_{min} s expired. Recall that these lock holding spans are given by the participating WS-Schedulers based on the current status of their local dependency graphs. Hence, the overlap between the exclusive lock holding spans implies that the transaction's node in all distributed dependency sub-graphs has no outgoing edges at the period between L_{max} and R_{min}. The WS-Coordinator can safely commit and close its transaction at all sites within this period without any delay. This also implies that the transaction is not involved in any global dependency cycle.

The schedule shown in Figures 4.9-b, on the other hand, is an example of a blocking schedule. There is a conflict between the lock holding spans of WS_1 and WS_3 as the activities of WS_3 cannot be committed before L_{max}, which is later than the deadline R_{min} imposed by WS_1. When such a blocking schedule is detected, WS-Coordinators have to handle this by re-scheduling the execution of the transaction at a later point of time that does not lead to the same conflict as described in Algorithm 10.

This pre-scheduling algorithm is executed by the WS-Coordinator prior the actual execution of the business transaction. First, the WS-Coordinator inquiries about

the timing constraints for acquiring and releasing the exclusive locks of each of the involved services (Lines 1-5 in Algorithm 10). The queries are sent to the scheduling service of the corresponding WS-Schedulers. Each WS-Scheduler responds to this query by checking its local dependency graph and makeing an offer for the WS-Coordinator based on the current load at the moment of the request (i.e. based on the list of active transactions). The WS-Coordinator then checks these constraints to detect any timing conflict (Lines 6-11 in Algorithm 10). There is a conflict if the lock holding spans of all the involved services do not overlap. The WS-Schedulers verifies this by comparing L_{max} and R_min, i.e. the latest acquisition time and the earliest release deadline respectively. There is an overlap between all lock holding spans if $L_{max} < R_min$, i.e. all locks can be acquired, before the earliest deadline expires. Otherwise, a conflict is detected and the WS-Coordinator repeats the previous steps in randomly set time intervals until a conflict-free (i.e. nonblocking) schedule is found. Once a conflict-free schedule is found, the WS-Coordinator uses the overlapping interval L_{max} to R_min for all web services in the transaction in order to synchronize their commit (Lines 12-15 in Algorithm 10). The WS-Coordinator communicates these values establishes a service Level Agreement (SLA) with all involved WS-Schedulers.

Algorithm 10 Transaction PreScheduling

Input :
$T = \{WS_1, ..., WS_n\}$ // a transaction composing n services
1: **for all** $WS_i \in T$ **do**
2: $Scheduler_i$ = WS-Scheduler of WS_i
3: $L_i = Scheduler_i$.getLockAcquisitionTime(WS_i)
4: $R_i = Scheduler_i$.getLockReleaseDeadline(WS_i)
5: **end for**
6: $L_{max} = \max_{i=1}^{n}(L_i)$
7: $R_{min} = \min_{i=1}^{n}(R_i)$
8: **if** $(L_{max} > R_{min})$ **then**
9: **wait** for a random amount of time
10: **goto** step
11: **end if**
12: **for all** $WS_i \in T$ **do**
13: $L_i = L_{max}$
14: $R_i = R_{min}$
15: **end for**

4.4 Experimental Evaluation

The purpose of this evaluation is to study the performance of our distributed and optimistic variant of the SGT protocol in comparison with the conventional distributed Two Phase Locking protocol (2PL). We experimented with both global cycle handling methods: the edge chasing method and the pre-scheduling method. The performance is measured in terms of average response time (as perceived by the business transaction's initiator) and overall throughput of the system (i.e. number of terminated transactions per second). Our hypothesis that we want to validate, is that our solution outperforms the distributed 2PL in terms of both criteria. For the purpose of this evaluation, we implemented a prototype for the WS-Coordination and the BusinessActivityWithCoordinatorCompletion protocol according to the BusinessActivity specifications [23]. On the service provider's side, we implemented the WS-Scheduler component and extended the participant's functions to be able to communicate with it. On the client-side, we extended the coordinator's functions to be able to support the cycle detection service as well as the pre-scheduling algorithm.

4.4.1 Experiment Settings

For experimental evaluation purposes we simulated the environment of concurrently running Web service-based processes. In each experiment we ran a number of concurrent transactions each of them is assigned to a coordinator. Every transaction is composed of a (randomly set) number of tasks; each of them can be accomplished by one of several alternative Web services from different providers. Every call to a service starts a new thread, which performs some transactional operations (read/write) on some local resources (text files). To simulate variant execution lengths of the Web services, we delay the return of the results by a randomly set amount of time following a Pareto distribution. Table 4.2 summarizes the different parameters of the simulation setup in our experiments. All experiments were conducted on a machine with a 2GHz Genuine Intel CPU, T2500 processor and 2GB RAM equipped with Microsoft Windows XP Professional Version 2002. The JVM used is J2SE 1.5.

In this experiment we measure the average response time and overall throughput of the concurrency control methods under different concurrency levels. We execute the transactions in several runs with a different number of alternative web services in each run. By varying the number of alternative web services (from 200 to 40),

number of concurrent transactions	100
number of web services per transaction	5 to 30
number of providers per service	40 to 200
service execution length	5 to 30 seconds
distribution of service execution length	Pareto
shape parameter of Pareto distribution	$\alpha = 3$
scale parameter of Pareto distribution	$\beta = 5$
service execution mean value	7.5 seconds
service execution standard deviation	4.3 seconds

Table 4.2 Simulation setup

the probability that transactions invoke the same web service increases, hence, the probability that transactional dependencies among them occur also increases.

In the experiments we compare between the following three methods:

- 2PL: the distributed Two Phase Locking protocol used in distributed database.
- DSGT_EdgeChasing: our proposed distributed Serialization Graph Testing protocol with edge chasing for handling global dependency cycles.
- DSGT_PreScheduling: our proposed distributed Serialization Graph Testing protocol with pre-scheduling for handling global dependency cycles.

4.4.2 Response Time vs. Concurrency Level

In Figure 4.10, we compare the average response time of the three methods, 2PL, DSGT_EdgeChasing and DSGT_PreScheduling. The measured response time of 2PL is the required time for acquiring all the requested locks. The response time of DSGT_EdgeChasing involves waiting times that are caused by the commit differing policy and the response time of DSGT_PreScheduling is the required time for finding a nonblocking schedule. The results shown in Figure 4.10 indicate that the average response time of all methods increases as the concurrency level increases (as the number of alternative services decreases). However, the response time of DSGT in both cases increases much slower than the response time of 2PL, which indicates that DSGT is much more efficient and scalable. We also notice that the response time of DSGT_PreScheduling is in average lower than the response time of

DSGT_EdgeChasing. This is because in the DSGT_PreScheduling transactions avoid unnecessary waiting times by ensuring that the transaction schedule is conflict-free prior the actual execution. In DSGT_EdgeChasing on the other hand, transactions start execution immediately and delay the dependency check till the commit time. This quite often leads some transactions being blocked by other dominant ones due to the Commit-order preserving policy.

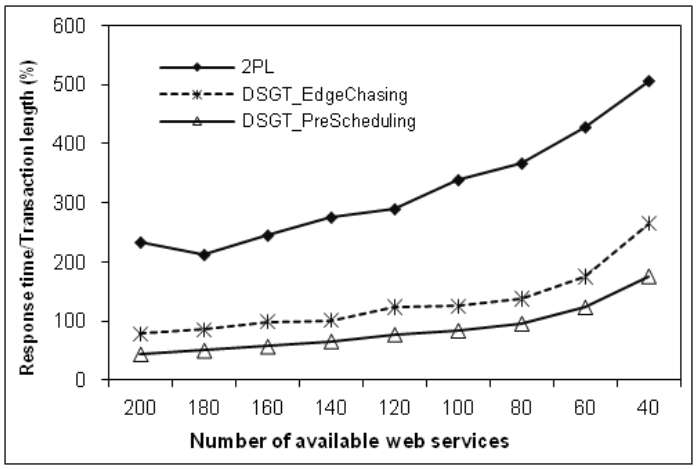


Figure 4.10 Response time versus concurrency level

4.4.3 Throughput vs. Concurrency Level

In this experiment we compare the overall throughput of the DSGT protocol and the conventional 2PL protocol under different concurrency levels. The overall throughput is measured by the number of terminated transactions per second. Figure 4.11 shows that DSGT_EdgeChasing and DSGT_PreScheduling in average have a much higher throughput than 2PL. We also observe that the throughput of DSGT_EdgeChasing and DSGT_PreScheduling decreases as the number of available web services decreases and, hence, the conflict probability increases. The throughput of DSGT_EdgeChasing however, decreases much faster than the DSGT_PreScheduling, as more and more transactions get delayed at commit time.

4.4.4 Communication Cost vs. Concurrency Level

The use of edge chasing algorithm for detecting global waiting cycles imposes some overhead in terms of communication cost for propagating the tokens. Similarly, the

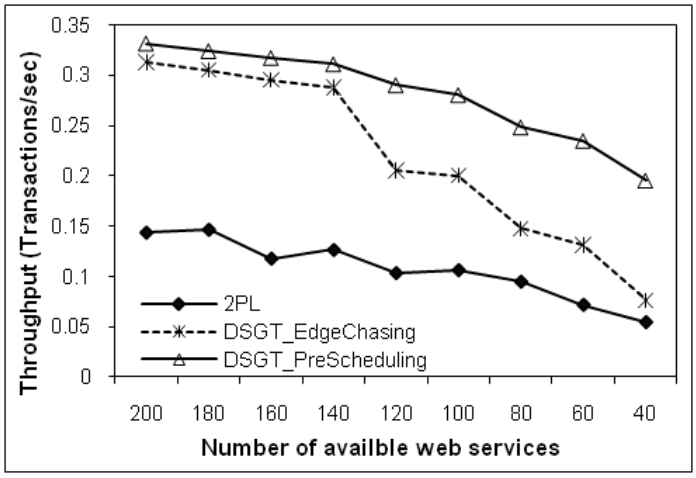


Figure 4.11 Throughput versus concurrency level

use of pre-scheduling to avoid dependency cycles requires communication with the involved WS-Schedules to detect potential timing conflicts. In the experiment shown in Figure 4.12 we measure the communication cost in terms of average number of messages exchanged between coordinators and schedulers. The number of messages increases dramatically with the edge chasing approach as the dependency conflicts among transactions increase. The communication overhead of pre-scheduling is in average much less than the overhead of the edge chasing approach.

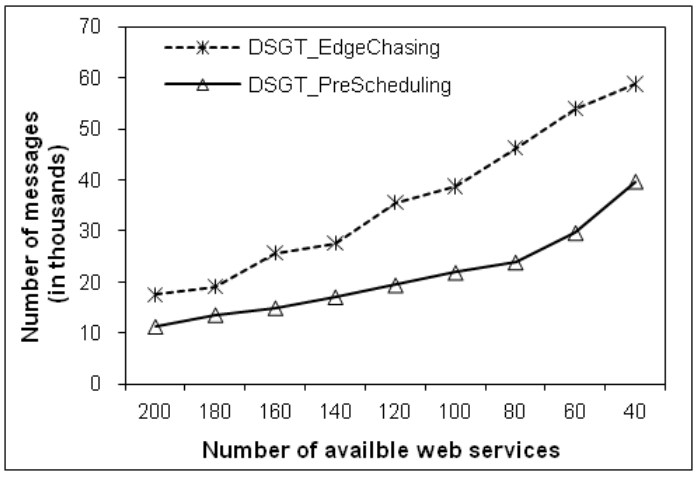


Figure 4.12 Communication cost comparison

4.4.5 Throughput vs. Transaction Complexity

In this experiment we study the impact of the transactions complexity (i.e. number of involved services) on the throughput of the applied concurrency method. Therefore, we repeated the experiment three times with the maximum number of composed services equals to: 10, 20 and 30 per transaction. The results shown in Figure 4.13 indicate that the throughput of all methods decreases when the number of involved services per transaction increases. However, in all cases, the throughput of the DSGT_EdgeChasing and DSGT_PreScheduling remains higher than the throughput of the 2PL protocol.

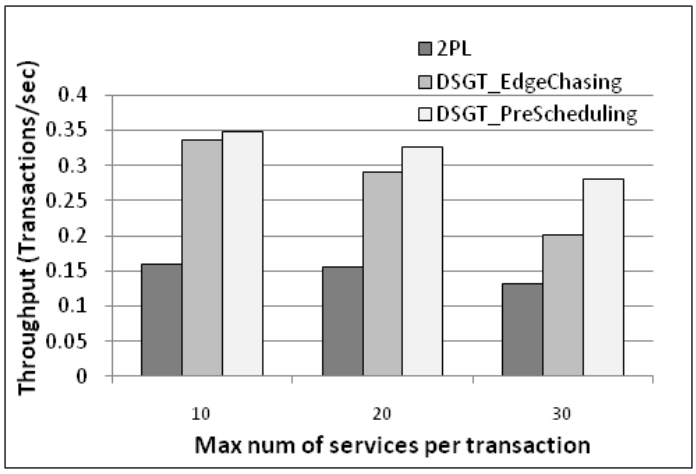
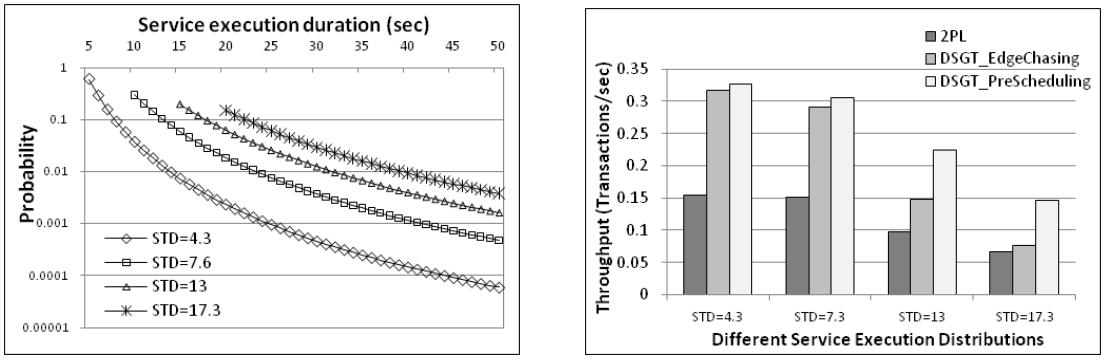


Figure 4.13 Throughput versus transaction complexity

4.4.6 Throughput vs. Service Execution Duration

In this experiment we study the impact of the service execution duration on the throughput of the applied concurrency method. For this purpose, we repeat the experiment with the standard deviation of the service duration distribution equals to: 4.3, 7.6, 13, 17.3 seconds (see Figure (a)). The results shown in Figure (b) indicate that the throughput of all methods decreases as the standard deviation increases. The throughput of the DSGT_EdgeChasing decreases much faster than the DSGT_PreScheduling. However, in all cases the throughput of the DSGT_PreScheduling remains higher than the throughput of the 2PL protocol in all cases.



(a): Distribution of web service execution

(b): Throughput

Figure 4.14 Throughput versus execution distribution

4.4.7 Summary of the Results

The results of the experimental evaluation have shown that the distributed serialization testing graph protocol outperforms the conventional distributed 2PL protocol in terms of overall throughput and average response time. The results have also shown that using edge chasing approach for detecting and resolving global dependency cycles performs well with low level of conflicts. The performance of this approach decreases as the conflict level increases, while its communication overhead increases significantly. Therefore, this method for handling global dependency cycles is only useful in small environments like enterprise-enterprise business transactions where the probability that concurrent transactions get into transactional conflict is not very high. The pre-scheduling method performs better than the edge chasing method in all cases even with high levels of conflict. The pre-scheduling method, therefore, fits well to open and dynamic environments, where the level of conflicts is not predictable. This method is also useful for business applications with tight time constraints. A disadvantage of the pre-scheduling method lies in the complexity of its implementation, as it requires extending the WS-Coordinator's and WS-Scheduler's functionality to schedule service invocations in a timely manner.

5 Conclusion and Future Work

5.1 Summary of Contribution

This thesis addresses two challenging research problems related to the non-functional aspects of web service composition.

The first research problem addressed in this thesis is the problem of efficient QoS-aware service selection for web service compositions with end-to-end QoS constraints. The contribution of this thesis is in developing approximate solutions for service selection that outperforms the exact solutions in terms of computation time while still able to achieve close-to-optimal results. A hybrid approach that combines global optimization with local selection techniques has been presented to solve this problem more efficiently than existing solutions that relies solely on global optimization solutions. The hybrid approach reformulate the QoS-aware service selection problem to a problem of end-to-end QoS constraints decomposition. By solving the latter problem, end-to-end QoS constraints are mapped into local constraints on the component service level, which in turn can be used by service brokers to find the best suitable service for the composition among a list of functionally-equivalent ones. A skyline based solution has been also presented to improve the scalability of the applied selection method by focusing on the set of skyline services (or a subset of it called representative skyline services) instead of the whole set of candidate services during the selection process. A skyline-based algorithms for decomposing QoS constraints has also been presented, which has been shown to perform better than the original greedy method used by the hybrid approach. In addition, a method for assisting service providers to improve the QoS of their services and, hence, their competitiveness with minimum

cost was presented. Extensive experimental evaluation of the proposed solutions with both real and synthetic datasets have been presented. The results of this evaluation have shown significant improvement by the proposed solutions in comparison with existing solutions.

The second research problem addressed in this thesis is the problem of ensuring reliable execution of transactional web services. The contribution of this thesis is in proposing an extension to the current web service transaction's framework that enables providing concurrency control on the web services' level in a modular way, by adopting a multi-level transaction model. In the new framework, the concept of WS-Scheduler was introduced for detecting and handling transactional dependencies among concurrent web service transactions. Moreover, an optimistic and fully decentralized concurrency control protocol was presented and complemented with two distributed methods for detecting and handling global transactional dependencies among concurrent web service transactions. Experimental evaluation of the performance of the proposed concurrency control protocol in terms of response time and throughput were presented. The results have shown that the proposed concurrency control mechanism outperforms the conventional distributed two phase locking protocol in terms of response time and throughput.

5.2 Outlook to Future Work

A possible continuation of the work presented in this thesis for QoS-aware service selection is to relax the assumptions made about the QoS data of web services. For example it would be interesting to consider the uncertainty of QoS data and develop a probabilistic model for estimating the expected QoS level based on some statistical data. Using such a probabilistic model, we would be able to select the services that are more likely to meet the requested QoS level.

Another direction for continuing this work could be to generalize the models and solutions we have presented to cover not only non-functional QoS criteria but also other criteria that are directly related to the functionality delivered by the service, like for example, the quality of the results delivered by a hotel search web service. Such generalization brings more challenges that need to be addressed such as defining a unified model for describing and evaluating functional QoS attributes.

A possible direction for continuing the work presented in this thesis for transactional web services could be to develop a cost model for assisting service providers in estimating the benefit/overhead of scheduling a service request. Such a model could help the service providers in making decisions about accepting or rejecting service requests in a way that maximizes the utilization of their resources.

A Curriculum Vitae

Mohammad Al-Rifai, born on March 8^{th} 1974, in Hodeidah, Yemen.

April 2006 -	Junior researcher and Ph.D student at Forschungszentrum L3S, Universität Hannover
Oct. 2005	Master of Science in Computer Science, Universität des Saarlandes, Saarbrcken Title of the thesis: *"Load Awareness in Flex"*
Aug. 1998	Bachelor of Science in Computer Engineering, University of Technology, Baghdad, Iraq
Oct. 1998 - Aug. 2001	Teaching assistant Hodeidah University, Yemen
Oct. 1998 - Aug. 2001	Teaching assistant Univeristy of Science and Technology, Hodeidah, Yemen

Bibliography

[AAA+96] Gustavo Alonso, Divyakant Agrawal, Amr El Abbadi, Mohan Kamath, Roger Günthör, and C. Mohan. Advanced transaction models in workflow contexts. In *ICDE '96: Proceedings of the Twelfth International Conference on Data Engineering*, pages 574–581, Washington, DC, USA, 1996. IEEE Computer Society.

[ABDN07] Mohammad Alrifai, Wolf-Tilo Balke, Peter Dolog, and Wolfgang Nejdl. Nonblocking scheduling for web service transactions. In *ECOWS*, pages 213–222, 2007.

[ACD+] Alain Andrieux, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Toshiyuki Nakata, Jim Pruyne, John Rofrano, Steve Tuecke, and Ming Xu. Web services agreement specification (ws-agreement), version 2005/09. Web page. http://www.ogf.org/documents/GFD.107.pdf.

[ACH98] Cristina Aurrecoechea, Andrew T. Campbell, and Linda Hauw. A survey of qos architectures. *Multimedia Systems*, 6(3):138–151, 1998.

[ADBN09] Mohammad Alrifai, Peter Dolog, Wolf-Tilo Balke, and Wolfgang Nejdl. Distributed management of concurrent web service transactions. *IEEE T. Services Computing*, 2(4):289–302, 2009.

[ADN06] Mohammad Alrifai, Peter Dolog, and Wolfgang Nejdl. Transactions concurrency control in web service environment. In *ECOWS*, pages 109–118, 2006.

[Alr08] Mohammad Alrifai. Distributed and scalable qos optimization for dynamic web service composition. In *ICSOC PhD Symposium*, 2008.

[AMM08] Eyhab Al-Masri and Qusay H. Mahmoud. Investigating web services on the world wide web. In *International World Wide Web Conference*, 2008.

[AP05] Danilo Ardagna and Barbara Pernici. Global and local qos constraints guarantee in web service selection. In *International Conference on Web Services (ICWS)*, 2005.

[AP07] Danilo Ardagna and Barbara Pernici. Adaptive service composition in flexible processes. *IEEE Trans. on Software Engineering*, 33(6):369–384, 2007.

[AR09] Mohammad Alrifai and Thomas Risse. Combining global optimization with local selection for efficient qos-aware service composition. In *International World Wide Web Conference*, pages 881–890, 2009.

[AR10] Mohammad Alrifai and Thomas Risse. Efficient qos-aware service composition. In Monique Calisti, Marius Walliser, Stefan Brantschen, Marc Herbstritt, Walter Binder, and Schahram Dustdar, editors, *Emerging Web Services Technology Volume III*, Whitestein Series in Software Agent Technologies and Autonomic Computing, pages 75–87. Birkhuser Basel, 2010.

[ARDN08] Mohammad Alrifai, Thomas Risse, Peter Dolog, and Wolfgang Nejdl. A scalable approach for qos-based web service selection. In *ICSOC Workshops*, pages 190–199, 2008.

[ARK+06] Md. Mostofa Akbar, M. Sohel Rahman, M. Kaykobad, E. G. Manning, and G. C. Shoja. Solving the multidimensional multiple-choice knapsack problem by constructing convex hulls. *Computers and Operations Research*, 33(5):1259–1273, 2006.

[ASR10] Mohammad Alrifai, Dimitrios Skoutas, and Thomas Risse. Selecting skyline services for qos-based web service composition. In *WWW*, pages 11–20, 2010.

[BEK+] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer. Simple object access protocol (soap) 1.1 - w3c note 08 may 2000. Web page. http://www.w3.org/TR/soap/.

[BGL07] Wolf-Tilo Balke, Ulrich Güntzer, and Christoph Lofi. Eliciting matters - controlling skyline sizes by incremental integration of user preferences. In *DASFAA*, pages 551–562, 2007.

[BGS07] Wolf-Tilo Balke, Ulrich Güntzer, and Wolf Siberski. Restricting skyline sizes using weak pareto dominance. *Inform., Forsch. Entwickl.*, 21(3-4):165–178, 2007.

[BHM+] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard. Web services architecture - w3c working group note 11 february 2004. Web page. http://www.w3.org/TR/ws-arch/#technology.

[BKS01] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. The skyline operator. In *International Conference on Data Engineering*, pages 421–430, 2001.

[BPG05] Sami Bhiri, Olivier Perrin, and Claude Godart. Ensuring required failure atomicity of composite web services. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 138–147, New York, NY, USA, 2005. ACM.

[BS04] A. Soydan Bilgin and Munindar P. Singh. A daml-based repository for qos-aware semantic web service selection. volume 0, page 368, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[BSND02] Boualem Benatallah, Quan Z. Sheng, Anne H. H. Ngu, and Marlon Dumas. Declarative composition and peer-to-peer provisioning of dynamic web services. In *International Conference on Data Engineering*, pages 297–308. IEEE Computer Society, 2002.

[CBB08] Sodki Chaari, Youakim Badr, and Frédérique Biennier. Enhancing web service selection by qos-based ontology and ws-policy. In *SAC '08: Pro-*

ceedings of the 2008 ACM symposium on Applied computing, pages 2426–2431, New York, NY, USA, 2008. ACM.

[CCMW] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web services description language (wsdl) 1.1 - w3c note 15 march 2001. Web page. http://www.w3.org/TR/wsdl.

[CJK+05] Seunglak Choi, Hyukjae Jang, Hangkyu Kim, Jungsook Kim, Su Myeon Kim, Junehwa Song, and Yoon-Joon Lee. Maintaining consistency under isolation relaxation of web services transactions. In *WISE*, pages 245–257, 2005.

[CJT+06a] Chee Yong Chan, H. V. Jagadish, Kian-Lee Tan, Anthony K. H. Tung, and Zhenjie Zhang. Finding k-dominant skylines in high dimensional space. In *SIGMOD Conference*, pages 503–514, 2006.

[CJT+06b] Chee Yong Chan, H. V. Jagadish, Kian-Lee Tan, Anthony K. H. Tung, and Zhenjie Zhang. On high dimensional skylines. In *EDBT*, pages 478–495, 2006.

[CKJ+08] Seunglak Choi, Hangkyu Kim, Hyukjae Jang, Jungsook Kim, Su Myeon Kim, Junehwa Song, and Yoon-Joon Lee. A framework for ensuring consistency of web services transactions. *Inf. Softw. Technol.*, 50(7-8):684–696, 2008.

[CLPZ09] K. Selçuk Candan, Wen-Syan Li, Thomas Phan, and Minqi Zhou. Frontiers in information and software as services. In *International Conference on Data Engineering*, pages 1761–1768, 2009.

[CN01] Yi Cui and Klara Nahrstedt. Supporting qos for ubiquitous multimedia service delivery. In *Proceedings of the ACM International Conference on Multimedia*, pages 461–462, 2001.

[Coma] OASIS UDDI Specification Technical Committee. Oasis universal description discovery and integration (uddi) version 3.0. Web page. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=uddi-spec.

[Comb] OASIS Web Services Business Process Execution Language (WS-BPEL) Technical Committee. Oasis web services business

process execution language (ws-bpel) version 2.0. Web page. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel.

[Comc] OASIS Web Services Reliable Messaging (WSRM) Technical Committee. Web services reliable messaging version 1.1. Web page. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrm.

[Comd] OASIS Web Services Security (WSS) Technical Committee. Web services security version 1.1. Web page. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss.

[Come] OASIS Web Services Transaction (WS-TX) Technical Committee. Web services atomic transaction (ws-atomictransaction) version 1.2. Web page. http://docs.oasis-open.org/ws-tx/wsat/2006/06.

[Comf] OASIS Web Services Transaction (WS-TX) Technical Committee. Web services business activity (ws-businessactivity) version 1.2. Web page. http://docs.oasis-open.org/ws-tx/wsba/2006/06.

[Comg] OASIS Web Services Transaction (WS-TX) Technical Committee. Web services coordination (ws-coordination) version 1.2. Web page. http://docs.oasis-open.org/ws-tx/wscoor/2006/06.

[CS01] Fabio Casati and Ming-Chien Shan. Dynamic and adaptive composition of e-services. *Information Systems*, 26(3):143–163, 2001.

[D'A06] Andrea D'Ambrogio. A model-driven wsdl extension for describing the qos ofweb services. In *ICWS*, pages 789–796, 2006.

[Elm92] Ahmed K. Elmagarmid, editor. *Database transaction models for advanced applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.

[GA04] Harumi Kuno Vijay Machiraju Gustavo Alonso, Fabio Casati. *Web services: concepts, architectures and applications*. Springer Verlag, 2004.

[Gra88] Jim Gray. The transaction concept: virtues and limitations. pages 140–150, 1988.

[Gro] European Semantic Systems Initiative WSML Working Group. Web service modeling language (wsml), w3c member submission 3 june 2005. Web page. http://www.w3.org/Submission/WSML/.

[GWW02] Michael Gillmann, Gerhard Weikum, and Wolfgang Wonner. Workflow management with service quality guarantees. In *Proceedings of the SIGMOD Conference*, pages 228–239, 2002.

[HST05] Klaus Haller, Heiko Schuldt, and Can Türker. Decentralized coordination of transactional processes in peer-to-peer environments. In *CIKM*, pages 28–35, 2005.

[JHE04] Wen Jin, Jiawei Han, and Martin Ester. Mining thick skylines over large databases. In *PKDD*, pages 255–266, 2004.

[Kha98] Md. Shahadatullah Khan. *Quality adaptation in a multisession multimedia system: model, algorithms, and architecture*. PhD thesis, Victoria, B.C., Canada, Canada, 1998. Adviser-Li, Kin F. and Adviser-Manning, Eric G.

[KHC+05] Dimka Karastoyanova, Alejandro Houspanossian, Mariano Cilia, Frank Leymann, and Alejandro Buchmann. Extending bpel for run time adaptability. In *EDOC '05: Proceedings of the Ninth IEEE International EDOC Enterprise Computing Conference*, pages 15–26, Washington, DC, USA, 2005. IEEE Computer Society.

[KLMA02] Shahadat Khan, Kin F. Li, Eric G. Manning, and Md. Mostofa Akbar. Solving the knapsack problem for adaptive multimedia systems. *Stud. Inform. Univ.*, 2(1):157–178, 2002.

[Kna87] Edgar Knapp. Deadlock detection in distributed databases. *ACM Computing Surveys (CSUR)*, 19(4):303–328, 1987.

[KP09] Kyriakos Kritikos and Dimitris Plexousakis. Mixed-integer programming for qos-based web service matchmaking. *IEEE T. Services Computing*, 2(2):122–139, 2009.

[LKD+] Heiko Ludwig, Alexander Keller, Asit Dan, Richard P. King, and Richard Franck. Web service level agreement

(wsla) language specification, version 1.0, 2003. Web page. http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf.

[Llo82] Stuart P. Lloyd. Least squares quantization in pcm. *IEEE Trans. on Information Theory*, 28:129–137, 1982.

[LNZ04] Yutu Liu, Anne H. H. Ngu, and Liangzhao Zeng. Qos computation and policing in dynamic web service selection. In *International World Wide Web Conference*, pages 66–73, 2004.

[LYSS07] Fei Li, Fangchun Yang, Kai Shuang, and Sen Su. Q-peer: A decentralized qos registry architecture for web services. In *International Conference on Service-oriented Computing*, pages 145–156, 2007.

[LYZZ07] Xuemin Lin, Yidong Yuan, Qing Zhang, and Ying Zhang. Selecting stars: The k most representative skyline operator. In *ICDE*, pages 86–95, 2007.

[Mar03] Istvn Maros. *Computational Techniques of the Simplex Method*. Springer, 2003.

[MB] Peter Notebaert Michel Berkelaar, Kjell Eikland. Open source (mixed-integer) linear programming system. Sourceforge. http://lpsolve.sourceforge.net/.

[MM06] Frederic Montagut and Refik Molva. Augmenting web services composition with transactional requirements. In *ICWS '06: Proceedings of the IEEE International Conference on Web Services*, pages 91–98, Washington, DC, USA, 2006. IEEE Computer Society.

[Mos81] E. B. Moss. Nested transactions: An approach to reliable distributed computing. Technical report, Cambridge, MA, USA, 1981.

[MRLD09] Anton Michlmayr, Florian Rosenberg, Philipp Leitner, and Schahram Dustdar. Comprehensive qos monitoring of web services and event-based sla violation detection. In *Proceedings of the 4th International Workshop on Middleware for Service Oriented Computing (MWSOC)*, pages 1–6, New York, NY, USA, 2009. ACM.

[MT90] Silvano Martello and Paolo Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., New York, NY, USA, 1990.

[NW88] George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience, New York, NY, USA, 1988.

[OVSH06] Nicole Oldham, Kunal Verma, Amit Sheth, and Farshad Hakimpour. Semantic ws-agreement partner selection. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 697–706, New York, NY, USA, 2006. ACM.

[Pap03] Michael P. Papazoglou. Web services and business transactions. *World Wide Web*, 6(1):49–91, 2003.

[RBW06] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.

[RS04] Jinghai Rao and Xiaomeng Su. A survey of automated web service composition methods. In *Semantic Web Services and Web Process Composition, First International Workshop, SWSWPC, San Diego, CA, USA, Revised Selected Papers*, pages 43–54, 2004.

[SDN08] Michael Schäfer, Peter Dolog, and Wolfgang Nejdl. An environment for flexible advanced compensations of web service transactions. *ACM Transactions on the Web*, 2(2):1–36, 2008.

[TDLP09] Yufei Tao, Ling Ding, Xuemin Lin, and Jian Pei. Distance-based representative skyline. In *ICDE*, pages 892–903, 2009.

[Tea] IBM Services Architecutre Team. Web services architecture overview - the next stage of evolution for e-business. Web page. http://www.ibm.com/developerworks/webservices/library/w-ovr/.

[TP02] Aphrodite Tsalgatidou and Thomi Pilioura. An overview of standards and related technology in web services. *Distrib. Parallel Databases*, 12(2-3):135–162, 2002.

[VOH+] Asir S Vedamuthu, David Orchard, Frederick Hirsch, Maryann Hondo, Prasad Yendluri, Toufic Boubez, and mit Yalinalp. Web services policy 1.5 - framework - w3c recommendation 04 september 2007. Web page. http://www.w3.org/TR/ws-policy/.

[Wei86] Gerhard Weikum. A theoretical foundation of multi-level concurrency control. In *PODS '86: Proceedings of the fifth ACM SIGACT-SIGMOD symposium on Principles of database systems*, pages 31–43, New York, NY, USA, 1986. ACM.

[Wei91] Gerhard Weikum. Principles and realization strategies of multilevel transaction management. *ACM Trans. Database Syst.*, 16(1):132–180, 1991.

[WS84] Gerhard Weikum and Hans-Jörg Schek. Architectural issues of transaction management in multi-layered systems. In *VLDB '84: Proceedings of the 10th International Conference on Very Large Data Bases*, pages 454–465, San Francisco, CA, USA, 1984. Morgan Kaufmann Publishers Inc.

[WV01] Gerhard Weikum and Gottfried Vossen. *Transactional information systems: theory, algorithms, and the practice of concurrency control and recovery*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[XZT08] Tian Xia, Donghui Zhang, and Yufei Tao. On skylining with flexible dominance relation. In *ICDE*, pages 1397–1399, 2008.

[YH95] K . Paul Yoon and Ching-Lai Hwang. *Multiple Attribute Decision Making: An Introduction (Quantitative Applications in the Social Sciences*. Sage Publications, 1995.

[YZL07a] Tao Yu, Yue Zhang, and Kwei-Jay Lin. Efficient algorithms for web services selection with end-to-end qos constraints. *ACM Trans. on the Web*, 1(1), 2007.

[YZL07b] Tao Yu, Yue Zhang, and Kwei-Jay Lin. Efficient algorithms for web services selection with end-to-end qos constraints. *TWEB*, 1(1), 2007.

[ZBD+03] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Quan Z. Sheng. Quality driven web services composition. In *International World Wide Web Conference*, pages 411–421, 2003.

[ZBN+04] Liangzhao Zeng, Boualem Benatallah, Anne H. H. Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. Qos-aware middleware for web services composition. *IEEE Trans. on Software Engineering*, 30(5):311–327, 2004.

[ZCL04] Chen Zhou, Liang-Tien Chia, and Bu-Sung Lee. Daml-qos ontology for web services. *International Conference on Web Services*, 0:472, 2004.

[ZLC07] Liangzhao Zeng, Hui Lei, and Henry Chang. Monitoring the qos for web services. In *Proceedings of the 5th international conference on Service-Oriented Computing (ICSOC)*, pages 132–144, Berlin, Heidelberg, 2007. Springer-Verlag.

[ZZL09] Yanlong Zhai, Jing Zhang, and Kwei-Jay Lin. Soa middleware support for service process reconfiguration with end-to-end qos constraints. In *ICWS*, pages 815–822, 2009.