QOS-AWARE SERVICE SELECTION FOR WEB SERVICE COMPOSITION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

RAHAT ABDYLDAEVA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF INFORMATION SYSTEMS

JUNE 2012

QOS-AWARE SERVICE SELECTION FOR WEB SERVICE COMPOSITION

Submitted by **RAHAT ABDYLDAEVA** in partial fulfillment of the requirements for the degree of **Master of Science in Information Systems, Middle East Technical University** by,

Prof. Dr. Nazife BAYKAL _____
Director, **Informatics Institute**

Prof. Dr. Yasemin YARDIMCI ÇETİN _____
Head of Department, **Information Systems**

Assist. Prof. Dr. Aysu BETİN CAN _____
Supervisor, **Information Systems, METU**

Assoc. Prof. Dr. Altan KOÇYİĞİT _____
Co-Supervisor, **Information Systems, METU**

Examining Committee Members:

Assoc. Prof. Dr. Pınar ŞENKUL _____
Computer Engineering, METU

Assist. Prof. Dr. Aysu BETİN CAN _____
Information Systems, METU

Assoc. Prof. Dr. Altan KOÇYİĞİT _____
Information Systems, METU

Assist. Prof. Dr. Pekin Erhan EREN _____
Information Systems, METU

Assist. Prof. Dr. Tuğba TAŞKAYA TEMİZEL _____
Information Systems, METU

Date: **13.06.2012**

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name : Rahat Abdyldaeva

Signature : _____

ABSTRACT

QoS-AWARE SERVICE SELECTION FOR WEB SERVICE COMPOSITION

Rahat, Abdyldaeva
M.S., Department of Information Systems
Supervisor: Assist. Prof. Dr. Aysu Betin Can
Co-Supervisor: Assoc. Prof. Dr. Altan Koçyiğit

June 2012, 63 pages

Composition of web services is one of the flexible and easiest approaches for creating composite services that fulfill complex tasks. Together with providing convenience in creation of new software applications, service composition has various challenges. One of them is the satisfaction of user-defined Quality of Service (QoS) requirements while selecting services for a composition. Load balancing issue is another challenge as uncontrolled workload may lead to violation of service providers' QoS declarations. This thesis work proposes a QoS aware method for optimum service composition while taking into account load balancing. M/M/C queuing model is utilized for the individual services to determine sojourn time distribution for possible compositions. Percentile of the execution time, price and availability are considered as QoS parameters. Proposed algorithm selects the optimum composition according to QoS constraints and utility provided by the services. The performance of the method is evaluated by custom simulation software and is compared to two other methods, random selection and average execution time-based optimal service selection.

Key words: Web Service Composition, Quality of Service, Queuing Theory.

ÖZ

Rahat, Abdyldaeva
Yüksek Lisans, Bilişim Sistemleri Bölümü
Tez Yöneticisi: Yard. Doç. Dr. Aysu Betin Can
Ortak Tez Yöneticisi: Doç. Dr. Altan Koçyiğit

Haziran 2012, 63 sayfa

Web servislerinin birleşimi birleşik servisler yaratmak için esnek ve en basit yaklaşımlardan biridir. Servis birleşimi, yeni yazılım uygulamaları yaratılmasını sağlamasıyla birlikte, birçok çözülmesi gereken sorun içermektedir. Bunlardan biri birleşim için servis seçerken kullanıcı tarafından tanımlanmış servis kalitesi gereksinimlerinin karşılanmasıdır. Yük paylaşımı konusu ise control dışı iş yükünün servis sağlayıcının ilan ettiği servis kalitesinin ihlal edilmesine yol açabilmesi nedeniyle bir diğer problemdir. Bu tez yük paylaşımını da dikkate alan servis kalitesi farkındalıklı bir en iyi servis birleşimi yöntemi önermektedir. Olası birleşimlerin sistemde kalış zamanını dağılımını belirlemek amacıyla her bir servis için M/M/C kuyruk modeli kullanılmıştır. Servis kalitesi parametreleri olarak yürütüm zamanının yüzdelik dilimi, fiyat ve yararlanılırdık dikkate alınmıştır. Önerilen algoritma servis kalitesi kısıtlarına ve servisler tarafından sağlanan faydaya göre en iyi birleşimi seçmektedir. Metodun başarımı bir özel benzetim yazılımı ile değerlendirilmiş ve rastgele seçim ve ortalama yürütüm zamanına dayanan en iyi servis seçimi yöntemlerinin başarımı ile karşılaştırılmıştır.

Anahtar Sözcükler: Web Servisi Birleşimi, Servis Kalitesi, Kuyruk Kuramı.

To My Grandparents

To My Parents

To My Husband

and

To My Little Sunny-Son

ACKNOWLEDGMENTS

With a great pleasure, I would like to express my sincere gratitude to Dr. Altan KOÇYİĞİT for his patience, encouragement, continuous support and guidance throughout the thesis. I greatly appreciate his enormous contribution in completing this study and in improving my knowledge about the considered research field as well as about issues related to the research, such as academic writing, queuing theory and handling experiments.

I would like to thank my supervisor Dr. Aysu Betin Can for her support throughout the study.

I owe much to the faculty and staff of Informatics Institute for providing me high quality education during my master studies.

I would like to thank Turkish Government for providing me a scholarship, which made my education in Turkey possible.

I would like to thank all my friends for their friendship that support me during all this time. Special thanks go to Nazgul and Eldar who always present me a lot of joy and invaluable advices.

I would like to express my deepest gratitude to my parents, for their endless love, support and faith in me. I really appreciate help and understanding provided by my parents-in law. To my husband, Alisher, I present special thanks for his love and continuously encouraging me in my endeavors. I am truly grateful to my son, Aman, for his patient waiting for his mom for more than a half year.

TABLE OF CONTENTS

LIST OF TABLES

TABLE

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

AP	Admission Probability
Av	Availability
AvA	Average Availability
AvET	Average Execution Time
AvP	Average Price
AvU	Average Utility
DR	Delayed Requests
ET	Execution Time
MST	Mean Sojourn Time-based method
P	Price
QoS	Quality of Service
RND	Random method
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SOC	Service-Oriented Computing
Spc	services per composition
STP	Sojourn Time (pth percentile)-based method
U	Utility
UDDI	Universal Description, Discovery and Integration infrastructure
WSDL	Web Service Description Language

CHAPTER 1

INTRODUCTION

In this thesis, we propose a new approach for web service selection and composition to provide end-to-end QoS guarantees requested by clients.

In recent years, the number of applications based on Service-Oriented Architecture (SOA) principles is increased. Applications are created in areas such as telecommunication industry [1], health sector [2], business sector [3], and education management services [4]. This tendency is motivated by the numerous advantages provided by the Service-Oriented Computing (SOC), namely, *rapid* development of complex, scalable distributed applications from the simple components called services which are *self-descriptive*, *independent*, *loosely coupled*, *technologically neutral* and *available over the network* [5].

Web services are successful implementation of SOA concepts as they have aforementioned properties and are described, published, and discovered based on Extensible Markup Language (XML) artifacts. Core standards supporting web services are Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL) and Universal Description, Discovery and Integration infrastructure (UDDI). Approach proposed in this thesis is related to the web services, and we use “web service” and “service” terms interchangeably.

Services can be used individually or in the scope of *composite* services. A composite service is a combination of services and is used to fulfill complex business tasks. Service selection for a composition can be manual or automatic [5]. In manual service selection, a service requestor chooses necessary services manually from the list of discovered services, while automatic selection means selection of services

according to specified needs of the client. This thesis work considers automatic service selection mechanism.

A particular functionality can be offered by several services, as with the increase in demand on web services, the number of services providing the similar functionality increases [35]. In such situations, selection of services is based on their other attributes such as response time, availability, monetary cost, etc. Service selection according to its quality is important issue, as it defines the fulfillment of non-functional requirements and the success of service execution. So, QoS-based web service selection has become one of the key challenges in creation of applications by composing services. This is assured by the numerous studies on developing approaches that consider QoS-aware service selections and composition. Some of such approaches are presented in Chapter 2.

Section 2.4 reveals several shortcomings of the existing approaches. Based on those, possible improvements that could be done in the state-of-the-art service selection approaches based on QoS-constraints are defined. One of the possible improvements could be considering the stochastic nature of QoS values that depend on the service request arrivals and resources available on the servers providing the services. With this improvement some performance attributes such as execution time of the services can be predicted. Another improvement is linked to the necessity of balancing of load among the available resources. Therefore, general requirements for our proposed service selection methods are:

1. Service selection and composition approach shall take into account stochastic nature of QoS attributes such as execution time, availability, etc. In this thesis work, we primarily focus on execution time.
2. Service selection and composition approach shall distribute the load of among the service providers properly.

In order to satisfy these requirements, we developed two broker-based service selection methods. We use the queuing theory for predicting the execution time of applications on service providers. For this purpose we collect request arrival and execution time statistics on individual services and these statistics are collected and used by the broker in service selection.

Two different service selection methods based on above described general features are proposed:

1. Sojourn Time pth-percentile (STP) method: Service selection for composition ensures requirements for minimum price, maximum availability and specified percentile of a given execution time.
2. Mean Sojourn Time (MST) method: Service selection for composition satisfies required minimum price, maximum availability, and mean execution time.

In order to make performance comparisons we also consider a simple service selection for composition method based on random selection (RND method). Main focus of the thesis is on the first method, which is expected to provide the best performance.

General characteristics of the proposed approaches:

1. All methods employ a broker-based architecture;
2. We consider multiple QoS-constraints: execution time, price and availability;
3. Methods provide global selection of services, i.e. provides end-to-end provision of QoS;
4. We only consider service composition plan with sequential flow structure.

The content of the thesis is as follows: this chapter briefly explains the advantages of Service-Oriented Architecture, web services, motivation and the content of this thesis. In Chapter 2, service selection and composition approaches in the literature are reviewed. Chapter 3 introduces STP and MST methods, and RND method used for performance evaluation is described. Theoretical background of STP and MST is given in this chapter. Chapter 4 presents performance evaluation of the proposed approaches by computer simulations. Simulation scenarios are described and sets of QoS parameters are given. Then, results of simulations are presented and discussed. Chapter 5 gives conclusion of the study, limitations and issues for future consideration.

CHAPTER 2

LITERATURE REVIEW

Service-Oriented Architecture is an architectural style that enables the use of loosely coupled, reusable, platform and language independent software components called services, to perform some tasks. One of the realizations of services is web services. Web services are developed using any contemporary programming language and XML, are described by WSDL (Web Service Description Language), communicate using standard protocol such as SOAP (Simple Object Access Protocol) and discovered by UDDI.

Briefly, SOAP is a communication protocol that enables message exchange between service requestor and service provider in a distributed environment [5]. WSDL is an abstract language that describes service in terms of its functional and non-functional properties. UDDI provides a standard platform for registering services by service providers and discovering services by service requestors.

2.1 Brief overview of QoS-aware Service Selection and Composition Architectures, Taxonomy and Strategies

There are review papers on QoS-based service selection approaches, proposed QoS-based service selection architectures, taxonomy of service selection techniques [34] and service selection strategies [35]. This review is based on the proposed templates for describing existing service selection and composition architectures and strategies, but taxonomy of service selection techniques is modified, particularly following techniques are discussed explicitly: Decision Making-based approaches, Multiple Criteria-based approaches, Single Criterion-based approaches, Multiple Constraints-based approaches, Single Constraint-based

approaches, Heuristic approaches, approaches used Queuing theory and approaches providing Load Balancing. Figure 2.1 demonstrates modified taxonomy of service selection techniques.

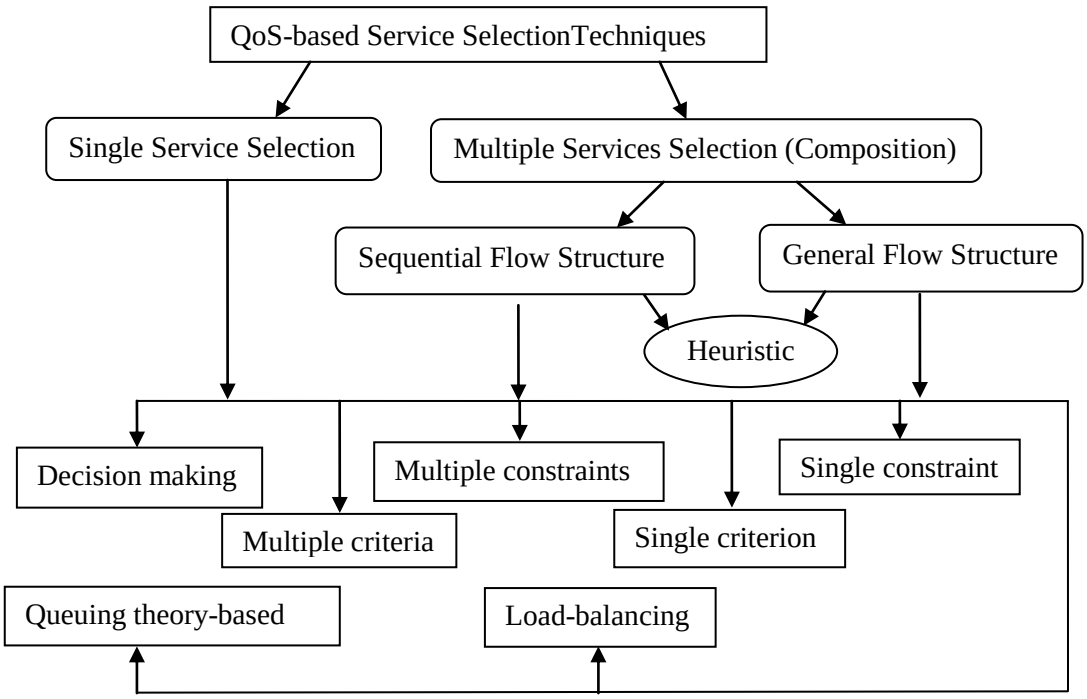


Figure 2.1 Modified Taxonomy of QoS-aware Service Selection Techniques

QoS-aware service composition architectures can be divided into three categories [29]: Augmented Architecture (a), Broker Architecture (b) and Hybrid Architecture (c). In (a), UDDI information is augmented by QoS properties, i.e. QoS information added to WSDL, OWL-S or to other standards. In (b), broker (middleware) provides discovering, selection, composition functionalities according to QoS values of services. In (c), augmented standard protocol is used along with broker architectural

Table 2.1 Architectural styles and existing approaches

Architectural styles	**Approaches**
Augmented Architecture	[6]
Broker Architecture	[9], [10], [11], [14], [15], [16], [19], [20], [21], [27], [28], [29], [32], [33], [37], [38], [39], [40], [42], [45], [46], [48], [49], [50], [51], [53], [54]
Hybrid Architecture	[8], [13], [17], [18], [41], [43], [44], [47], [52]

style for providing operations related to service selection and composition. Table 2.1 points out architectural styles leveraged by surveyed approaches. Service selection strategies according to QoS can be local selection (a), global selection (b), and mixed

strategy (c) [35]. Local selection (a) of services considers each service separately. This approach consumes low computational cost, but at the end of selection, global QoS (user preferences) can be not reached. This strategy usually used for selecting single service for a given task. Global selection (b) is necessary for composite services as this strategy chooses services according to QoS constraints defined by the user, i.e. ensures fulfilling end-to-end constraints. Global selection problem for service composition is NP-hard, and it can be considered as 0/1 Knapsack problem or Resource Constrained Project Scheduling problem [36]. Mixed strategy (c) combines local and global selection strategies in order to leverage advantages of each strategy. Table 2.2 demonstrates approaches that are solved using aforementioned strategies.

Table 2.2 Service Selection Strategies and approaches

Service Selection Strategies	Approaches
Local Selection	[28], [41], [42], [43], [44], [45], [46], [51]
Global Selection	[8], [10], [11], [13], [16], [40], [50]
Mixed Strategy	[9], [14], [15], [17], [18], [19], [37], [38], [49]

2.2 QoS-aware Service Composition Approaches.

QoS-aware service selection and composition approaches consider different qualities of service. Brief information about quality attributes that are taken into account by service selection and composition mechanisms is given in Table 2.3.

Single service selection mechanisms mostly use *Multiple Criteria Decision Making (MCDM)* [43], [44], [45], [46], [54]. In MCDM, service with the highest score is chosen. First step of MCDM is normalization of finite set of QoS values. The goal of the normalization process is to make different kinds of QoS values to be considered uniformly and to be in range [0, 1]. Next step in MCDM is to score services according to their QoS and weights defined by the user. Sum of the weights is equal to 1. The score of a service is the sum of QoS multiplied by their weights. MCDM is an effective and easy-to-implement approach, but it doesn't consider QoS constraints. QoS-aware service selection approaches use MCDM as an addition to their main frameworks or methods. Zhen et al. [44] propose Web service selection based on information about context and QoS. Information is taken from context and QoS ontologies. First, services are chosen according to context information, such as user's social role, computing device, network device, operation system, etc. After selection based on context, MCDM is used for defining best service among selected

services. [45], [46] describe a QoS-based Web Service Selection Model (WSSM-Q). WSSM-Q is a broker that accomplishes following procedures: defines QoS model, collects QoS information and stores in its QoS DB, and performs service selection [46].

Table 2.3 QoS considered by service selection approaches

Quality of Service	Approaches
Response Time	[8], [9], [17], [19], [28], [29], [38], [40], [43], [44], [45], [46], [50], [53]
Execution Time/Duration	[10], [11], [14], [16], [29], [37], [41], [45], [51]
Availability	[9], [10], [11], [12], [16], [17], [19], [37],[39], [41], [43], [45], [46], [48], [49], [50], [54]
Price	[9], [10], [11], [14], [16], [17], [37], [38], [39], [40], [41], [42], [43], [44], [45], [46], [48], [49], [50], [54]
Throughput	[17], [43], [45]
Latency	[13], [46], [53], [54]
Security	[13], [19]
Reliability	[13], [17], [19], [20], [38], [40], [41], [43], [44], [45], [48]
Trustworthiness	[43], [49]
Scalability	[43]
Integrity	[43]
Interoperability	[43]
Stability	[43]
Robustness	[43]
Reputation	[9], [14], [36], [39], [40], [41], [44], [48]
Successful execution rate	[14], [37], [39], [42], [49]
Capacity	[43], [54]
Network delay	[21], [50]
Successful completion	[9]
Transaction	[13], [14], [15]
Composability	[48]
Fidelity	[41]
Fault-tolerance	[30]
QoS stated in general	[6], [18], [47]

Service selection consists of two steps: first, all candidate services are ranked. If there is specific requirements for the QoS and if there are services that don't fulfill those requirements, they are eliminated from the list. Second step is choosing the best service using MCDM. [45] extends [46] by replacing ranking process with threshold application. Thus, time for first step of selection is reduced. Fuzzy *MCDM* [54] is an approach that uses fundamentals of fuzzy set theory and models service selection process as FMDCM. FMDCM manages two kinds of information about QoS: subjective and objective weights. Subjective weight is defined by users using preference values for evaluating QoS. Users evaluate QoS that cannot be evaluated

quantitatively (robustness, security, etc). Objective weight helps to judge consistency using entropy concept. Then, objective and subjective weights are synthesized into one weight according to synthetic parameter. Gained weights are scored and the best scored service is chosen. Negotiable and non-negotiable measurements of QoS are introduced in [39] and their combination gives "credibility" of service. Non-negotiable measurement is independent value from provider and taken from historical log of service and it is used for the first step of selection. Service is discarded if it's non-negotiable value less than user *constraint*. Using formula that uses maximum, minimum values of overall of particular QoS, credibility of non-negotiable values are defined. Negotiable values of QoS are determined by provider and by applying TOPSIS (Technique for order performance by similarity to ideal solution) credibility of negotiable measurements is determined. TOPSIS principle is to measure distance from solution to variants of solution, and it is claimed that positive QoS values have the shortest distance from the solution (or requirements) whereas negative QoS values have the longest distance from the solution (or requirements). Credibility values of non-negotiable and negotiable QoS are aggregated using weighting of each QoS. Sum of weighted QoS determines the score and service with the highest scored QoS is chosen. Extended semantic description of QoS imported into OWL-S and service selection using MCDM and TOPSIS are described in [43]. Semantic ontology of QoS is extended by introducing three levels of QoS description: upper level (general characteristics such as name, attribute, measurement, etc), middle level (domain-independent QoS values) and lower level (domain-dependent QoS values). Service selection is executed using MCDM's normalization formula and utility formula of TOPSIS. Service with the highest utility is chosen at the end. WS-QoS Framework [47] is proposed for monitoring and selection of services. This is an alternative concept to UDDI concept. WS-QoS has its own XML schema for defining QoS ontology, requirements definition and other information related to service selection. Service selection is fulfilled in Web Service Broker (WSB) that chooses services according to QoS information which is placed in WSB. After appropriate services are chosen, user invokes services through SOAP which handles QoS information in its header.

Multiple Services Selection or *Composition of services* is an approach that selects more than one service for accomplishing given task. Service composition

problem with *multiple global QoS constraints* is an NP hard problem. Some works solve such problem by different *heuristic* methods. *Local* and *global selection* strategies employed in [37]. First, global constraint is heuristically subdivided into local constraints according to the number of service classes. Second, local constraints are satisfied by using *MCDM*. Combinatorial and graph-based heuristic approaches are discussed in [10], [11]. Problem of service selection is modeled as MMKP (Multi-choice, Multi-dimension 0-1 Knapsack Problem). Combinatorial algorithm is called WS_HEU, modified heuristic HEU approach from [12]. Modified heuristic approach excludes services that are infeasible at the beginning of the algorithm. Method heuristically optimizes utility while minimizing constraints. The second algorithm is based on directed acyclic graph, where nodes are utilities and paths contains constraints. The problem of service selection is turned into problem of MCSP (Multi-Constrained Shortest Path). By limiting number of paths, authors propose heuristic MCSP-K. Both algorithms outperform traditional selection algorithms. WS-HEU is extended [16] by adding a rule library that manages the excluding of too bad services for composition. Another *heuristic* approach is proposed by Liu et.al. [9]. First, Multiple QoS values are normalized and scored by *MCDM* method. Service selection without global selection criteria is executed by greedy algorithm which chooses service with the highest score for each task. When there are *one* or *more global constraints*, the problem of service selection is considered as Multi-Choice Knapsack problem. *Heuristic* algorithm is based on convex hull. The idea is that each service set for one task is mapped to two-coordinated system, where x-axis is resource consumption of the service, and y-axis is its QoS score. Each point has convex hull that is constructed by algorithms such as Graham-scan and Quick-hull. Frontiers of the convex hull are calculated and used to define the segment between the highest and lowest points. This segment is used as heuristic information. In [17] an approach for selecting and composing services dynamically based on the set of QoS is proposed. Service discovery is based on semantic-based approach offered by Mokhtar et.al. [7]. The selection is achieved by *heuristic* algorithm based on clustering technique known as K-Means. The technique groups services according to their QoS and according to the local classification; algorithm finds the near-optimal solution (global selection).

RuGQoS [8] – is a service composition system based on breadth first algorithm and uses syntactic and semantic descriptions of services. Components of RuGQoS are XML parser, composite engine based on BFS algorithm and BPEL code generator. Priority queue is filled with services according to their QoS. Breadth First Search algorithm is exploited using priority queue and composition is gradually extended by adding appropriate services. Algorithm always finds composition if it exists, or returns no solution. Result is sent to BPEL that combines and executes the composition.

DSD [6] tries to overcome following limits of existing web service technologies: WSDL doesn't fully provide semantic information about services (like constraints, behavior), UDDI makes leveraging services difficult, and BPEL does not provide dynamism for composing services. Next prototypes are proposed: DIRE (Distributed Registry), SCENE (Service Composition Execution Environment) and Dynamo (dynamic monitoring). DIRE is the prototype that enables different registries to be connected (UDDI, ebXML). SCENE is the prototype of composition mechanism that extracts BPEL's possibilities by including preferences and *constraints.* There are Selection preferences in SCENE. It is used for selecting services by employing some limits (for example cost of the service). The description is published on DIRE or on other standards. SCENE has two parts: business logic in BPEL and rules part. The latter part is responsible for binding service according to constraints and preferences. Dynamo monitors composition of services in SCENE. Dynamo, SCENE and BPEL are interconnected through Service Execution Bus.

Authors propose an approach to face one of the challenges of Ambient Systems [13] which is about designing system that can be reconfigured according to user goals at runtime taking into account QoS requirements. Services are modeled as OWL-S process model, tasks are modeled as abstract OWL-S process model, and QoS are expressed in the form of arithmetic *constraints.* Authors of [18] propose service discovery and service selection model based on policies, such as user, application, environment and resource policies. Service composition is modeled as Directed Acyclic graph. Authors of [19] propose service composition model based on QoS and QoE. QoE is quality metric defined by user. Composite service is represented as graph; its edges are required parameters. Service selection is made according to ranking of services in terms of QoS and QoE. Service composition

middleware in [20] takes into account pervasive computing attributes. SM looks for available service in service table that is situated in the middleware. If necessary service exists then service execution begins. If there is no needed service, then service discovery is initiated. If discovery phase finds several nodes that provide same services, all of them are participated in service integration. In service integration phase clusters are created depending on the number of nodes. Clusters are ranked according to their QoS and labeled as Cp –primary cluster and Cs – secondary cluster. Service compositions in these clusters are processed in parallel. Primary cluster is leveraged and if it fails, secondary cluster takes primary cluster's place. This approach is toward to reliability of service composition execution. Importance of successful completion of service composition is discussed and approach directed to transactional composition of services is proposed in [14], [15]. Service selection consists of two steps: selection according to behavioral properties of services and to QoS. Behavioral property defines dependencies among services, i.e. how one service in the composition influence another service. Services that can make successful composition are selected. Then, *MCDM* is applied in order to define the best services in each service class. In [14] behavioral patterns of individual services are described, while [15] proposes also behavioral dependencies among composites. *Sequential* and *general flow* composites are considered. Pan and Mao [41] consider service discovery as multi-agent problem and service selection as planning problem. Agents are considered as services and implemented in JADE (Java Agent Development Environment), OWL-S ontologies translated into Planning Domain Description Language (PDDL), and service selection is executed using *MCDM*. Structural Equation Modeling Service Selection (SEMSS) [42] algorithm selects services according to predicted QoS defined by historical data. First SEM is used to model the relationship between user concerns and QoS. User concern is forecast by using structural equation for time T+1. Goodness-Fit Selection (GFS) uses this concern for defining goodness-of-fit index for each service. Using goodness-of-fit index of current time and adjusted goodness-of-fit index for T+1, GFI is defined. Service with maximum value of GFI is selected. Paper [49] extends GFS algorithm by introducing traversal structure that splits workflow into blocks. For each block best service is defined with the help of GFS and then aggregation structure along traversal tree checks the end-to-end constraint satisfaction. *MCDM* is

used to select services [48] for compositions if there are no *QoS constraints*. Branch-and-Bound algorithm is applied for service composition with *multiple QoS-constraints* if there is small number of services. When the number of services is large, piecewise reduction of service consideration is applied. Performances of exhaustive search algorithm, dynamic programming algorithm and Pisinger's algorithm for selecting services with end-to-end QoS constraints are compared [50]. Pisinger's algorithm is the fastest among these algorithms for service choosing for composition with *multiple QoS-constraints*. Two approaches dedicated to the solution of *multi-constrained* service composition problems are Multi-objective Ant Colony Optimization (MOACO) [38] and Max-Min Ant System [40]. Overview of approaches according to taxonomy of service selection and composition is given in Table 2.4.

Table 2.4 Arrangement of service selection approaches according to taxonomy

Taxonomic features	**Approaches**
Single Service Selection	[30], [39], [43], [44], [45], [46], [47], [51], [52], [53], [54]
Composition	[8], [9], [10], [11], [13], [14], [15], [16], [17], [18], [19], [20], [27], [28], [29], [32], [33], [37], [38], [40], [41], [42], [48], [49], [50]
Sequential Flow	[8], [9], [13], [16], [19], [28], [37], [38], [41], [42], [49]
Graph-based (General Flow)	[10], [11], [14], [15], [17], [18], [32], [33], [40], [48], [49]
Heuristic Approach	[9], [10], [11], [16], [17], [37]
Single Criteria	[28], [29], [51]
Multiple Criteria	[8], [14], [15], [19], [30], [37], [39], [41], [42], [43], [44], [45], [46], [48], [49], [52], [53], [54]
Decision Making	[9], [14], [15], [37], [39], [41], [42], [43], [44], [45], [46], [47], [49], [54]
Single Constraint-based	
Multiple Constraints-based	[8], [9], [10], [11], [13], [16], [17], [18], [32], [33], [37], [38], [39], [40], [46], [47], [50]
Queuing Theory-based	[21], [29], [51], [52], [53]
Load-Balancing Approach	[27], [28], [29], [30], [31], [32], [33]

2.3 Load balancing algorithms

Load balancing is a methodology for distributing workload among available resources [23]. These resources can be computers, servers, network links, central processing units, memory and other resources. Load balancing is applied to avoid overloads, decrease response time, increase throughput, optimally utilize resources [22], and to ensure stability [23].

Local scheduling and global scheduling are approaches for distributing jobs among processors [23]. Local scheduling is performed by operating system, where workload is divided into time slices for processing. Global scheduling is about

deciding where to execute task. It can be static and dynamic. In static scheduling, job is assigned to the resources at the beginning of processing and it cannot be reassigned to other processing units. Dynamic scheduling is sensitive to the load of overall system all time, and in case when some resources are overloaded, their jobs can be redistributed among their more idle counterparts.

We consider static (Round Robin (R), Randomized (R), Central Manager Algorithm (CMA), and Threshold (T)) and dynamic (Central Queue Algorithm (CQA) and Local Queue Algorithm (LQA)) load balancing approaches. RR [24] answers to DNS requests by several servers' IP addresses in round robin sequence. Round robin sequence is a cyclic sequence: when IP addresses in the list are all have jobs, the next coming job is assigned to the first IP address on the list and so on. Advantage of this approach is its easy implementation. It shows good performance when workloads are equal. Disadvantages are uneven distribution in case of not equal jobs arriving and if the server is unavailable, method still send request to it, i.e. it doesn't sense the server conditions. R [25] is similar to RR, the difference is it chooses servers in random manner. It also has advantage such as simplicity and disadvantage such as poor performance when jobs are not equal. CMA [23] uses one central processing unit as a distributer of workload according to the loads on different servers. This information is updated by servers when their loads change: they inform about changes to central processor. This approach shows good performance but inter-process communication can be a bottleneck. T [23] assumes that each processor maintains its resources and resources are scored as under-loaded, medium and overloaded. Jobs are processed locally until server becomes overloaded. When it happens, overloaded processing unit tries to find remote under-loaded or medium loaded server. If it finds such server, the job is sent to that server. If all servers are overloaded, the overloaded server processes job locally. Advantage of the method is limited inter-processor communication. Disadvantages are the unavailability to distribute a load according different processors capabilities and assigned jobs. This means that different servers have different possibilities: one server's overloaded state varies from other servers' such states. And sometimes job can be held in some servers well enough even if the server is overloaded, but because it has overloaded state, the job cannot be sent to this server. In CQA [26] queue manager maintains a cyclic FIFO queue in main host. New jobs are inserted to the

queue. Resources send request to fulfill the job to the queue manager. Queue manager sends job to the requester. If there is no activity in the queue, the request is buffered. If new activity arrives and there is no responded request, first activity from the list is removed and new activity is put there. LQA [26] is about managing local queues of servers. Local queue is managed by using latest-job-arrived policy. Load balance index is evaluated and according to the index load distribution is executed. Besides load balance index, in LQA, resource balance index is used. Load balancing is executed according to resource and load imbalance. Despite the fact its better performance with comparison to other baseline algorithms, it is less effective than CQA, as local queue manager is not able to see other queues in case if its load is low.

Following parameters are used to compare these six approaches [23]: Overload Rejection (OR), Fault Tolerant (FT), Forecasting Accuracy (FA), Stability (S), Cooperative (C), Process Migration (PM), and Resource Utilization (RU). OR assumes, that when processor is not able to handle the job, it has rejection state. When load is decreased this state is not active. This property is supported by only dynamic algorithms. FT is an availability to control failures by supporting continuous processing even some resource fails. CMA from static algorithms group and dynamic algorithms provide this feature. FA is supported by static algorithms better than by dynamic ones, as static approaches have more accurate and static approximation data at the beginning of the process than dynamic approaches have. Static approaches are more stable with comparison to dynamic approaches. C is a characteristic that determine the relationship between processors. Dynamic approaches CMA, and FT enable processors be aware of each other. PM is about transferring jobs of one processor to other processors. Only LQA has this feature. RU characteristic is about how effective the processor resource is used according to the requests. LQA is a leader among approaches on effective utilizing resources.

2.3.1 Load balancing algorithms for service composition

Load balancing in service composition mechanisms should be treated differently than load balancing for traditional systems [27]. The reason is that load balancing in service composition is performed for set of services situated in different nodes, while in web server load balancing we deal with one server-mirror [27]. Some load balance approaches for service selection are adapted from existing approaches

[29],[31] and some approaches are designed specifically for service composition [27],[28],[30],[32], [33]. Load balancing algorithms are either node-based load balancing or path based load balancing [27].

In [28], response time-based load balancing algorithm is proposed (RTLB). RTLB algorithm is based on service load and node load. Service load/weight is the average response time of services replicas. Node load depends on the sets of services running on the node. Services that are running on nodes can be simple or complex. Node load is equal to the sum of product of weights of services and number of these services instances in the node. Each node maintains three tables: routing table, weight table, and load table. When the node receives from client a request consisting of several services, algorithm looks to load table and defines the node with the lowest load for specific service. This is performed for requested services until done. Service instances are then composed and processed. When service is executed, node table is updates by adding weight of services to its overall load, and when the processing ends it is released. [29] considers Round Robin (RR), Static Lottery (SL), Shortest Queue (SQ), and Dynamic Lottery (DL) approaches. These methods are considered in the scope of service selection process. Service selection module that executes these algorithms is resided in Composition Execution Engine (CEE) such as BPEL. Service selection module uses service invocation data for evaluating response time and pending time of services. Performance model based on queuing theory is used for describing the overall execution time of services. According to Kendall's notation, elementary queue is described through A/B/k. A is the distribution of inter-arrival time of request, B is the distribution of service execution time, k is the number of server. Distributions of inter-arrival time and service execution time can be exponential distribution or Poisson arrivals, two-phase hyper-exponential distribution, Erlang distribution with k phases. This queuing model helps to define what have influenced response time: 1) execution time which can be affected by third-party load; and 2) inter-arrival time which can be affected by invocation requests. Distribution of inter-arrival time depends on selection algorithm. When it is probabilistic approach, inter-arrival time is exponentially distributed. When it is not probabilistic approach, it is non-exponential time. Execution time is hyper-exponentially distributed. Algorithms RR and SL are load-oblivious while SQ and DL are load-aware. SL's queuing model is M/H2/1. It has the worst performance

among considered algorithms. RR's queuing model is En/H2/1. As invocation time is En, this algorithm is faster than SL. It shows good performances when state limit is high, but when state limit is low load-aware algorithms are better. SQ shows very good performances when there are stateless services. It cannot be optimal algorithm for stateful service selections. DL is not enough good for stateless algorithms but shows very good results for stateful services, because it is able to monitor average response time of services. In [30] service discovery and invocation approach is proposed which takes into account load balancing among devices and considers fault-tolerance. Two lists are managed by service discovery mechanism: device list and service list. Requested service is searched in service list, and if such service exists, devices that can execute this service are defined. Then, the loads of devices are taken into consideration based on the queue of the device. Device with the lowest device is assigned for the request. Fault tolerance is provided by migration support from device to device in case of device or service failures. The approach [27] is based on path selection load balancing. Path has cost that is defined by taking inversion of difference of maximum loads and current loads of two connected nodes and then they are summarized. Dijkstra'a algorithm is applied for defining the shortest and cheapest path. After services are defined along the path, Cluster Manager is responsible for running algorithms that chooses specific service instances. As service instances are chosen they add load to the node. Update of current load information is executed with the help of piggybacking mechanism. This mechanism updates load information only upon request is done, and this helps to reduce load oscillation. Sometimes the algorithm generates long path that affects overall performance in bad manner. For solving this problem, no-op factor is added to count horizontal path. This factor is a bridge between different cluster, and this enables path to go horizontally and reduce the path length.

In [31], load-balancing approach is proposed for server initiated connections cases. This approach implies that there should be a server between client and processing server. This server acts as a dispatcher that takes requests from clients, validates it and sends them to queue. Processing servers takes requests from the queue. This two-tier architecture for load-balancing is discussed in a case study, SMS Gateway, which acts in between SMSC and the Internet Applications. DSCiPC(or SeSCo (Seamless Service Composition)**,** [32] represents tasks and

services as directed acyclic graph, which provides efficient discovering and matching services with tasks. The LATCH protocol is proposed for load balancing of workloads among devices. It is achieved through dividing devices into the levels according to their resources. Powerful resources assist less powerful nodes by taking part of responsibilities such as service discovering and invocation. ReSCo [33] defines trustworthy compositions and nodes in dynamic systems. ReSCo uses SeSCo [12] for defining available compositions. Service compositions are evaluated as successful or unsuccessful according to experiences and evaluated data is stored in data base. Database is organized in scalar form so it doesn't take much space. ReSCo uses SeSCo [32] for defining available compositions. ReSCo then looks into the database, and calculates paths between selected services. The paths and nodes are chosen randomly, as the best path can have traffic. Load balancing is supported in this manner.

2.3.2 Queuing Theory-based Service Selection Approaches

In dynamic service selection systems, services are requested in stochastic way. Queuing theory concepts enable to control such a system by compromising between different costs of services and the costs related to waiting for the service. That's why queuing theory is leveraged by service selection methods [51], [53]. Broker-based approach in [51] uses service scheduling model of M/M/k. Dispatcher algorithm resided in middleware D3D_Serv gets information about Expected Waiting Time (EWT) of all providers, chooses provider with minimum EWT value and puts a request to this provider's queue. EWT is calculated by dividing arrival rate to service rate which is multiplied by the subtraction of service rate and arrival rate. Arrival rate is the difference of requests number and elapse time. Queue is refreshed when feedback is received or when request arrives and there is no place in queue. This approach is compared with SSA-based service selection and random service selection method. Results show that D3D_Serv is better than random selection but worse than SSA-based selection in terms of performance. Another queue-based broker system is proposed by Badidi et.al. [53]. They modeled servers as M/M/1 service providers. Each server has several clusters of services. Requests are received by broker and broker dispatches requests to the providers. Dispatching of the requests is based on the results of evaluation of the overall system. In order to assess

the state of the system, system's throughput is measured in terms of arrival rate. The bound of throughput of overall system is the minimum value of arrival rate when system saturates. This value is used for decision making: when arrival rate reaches that bound, request is rejected, because requests cannot be processed successfully. Dependence of response time on arrival rate is discussed. As a conclusion, we say that service centers processes requests successfully until arrival rate reaches the saturation level of system. This approach is extended [52] by proposing semantic QoS model that can be leveraged by the service center systems instead of WSDL. The reason of replacing WSDL is that many requests are similar in syntactic way but differ semantically. For solving this problem, semantic QoS model is constructed, which clarifies requests at the beginning of service selection, in order to fulfill user requests correctly. Theoretic approach for allocating server in time delay cloud computing systems is proposed in [21]. Various Customers, Heterogeneous Servers (VCHS) queuing model is offered where services are various and servers have different performance. Before allocating a job to the server, weights of the servers are calculated according to the number of jobs in servers queue. Minimum-weighted server is assigned for the job. VCHS model is tested with the assumption that there is no time delay and with time delay that has independent conditional distribution random variables (uniform, normal, logarithmic normal and exponential). Compared results show that mean waiting time difference between non delay time system and delay time system exists.

2.4 Discussion

Large amount of studies considered in this chapter showed that service selection and composition are important issues. If to pay attention to the multiple constraints-based service selection and composition methods, one can see that all those methods considers services with constant QoS characteristics. Such kinds of approaches fail when the number of requests increases, because selection method chooses services according to their advertised QoS-values, despite the reason that service's QoS characteristics are changed due to the load. Some QoS values that are sensible to the amount of requests arriving per unit time are response time, execution time, availability, etc. It has been seen that several studies leveraged queuing theory for facing the stochastic nature of service request arrivals and considered expected

waiting time, throughput, and server's capacity. The main limitation of these approaches is that they didn't consider multiple constraints-based service compositions. New method of service selection and composition that can overcome limitations of multiple constraints-based and queue theory-based approaches is proposed in this thesis and the method is introduced in Chapter 3.

CHAPTER 3

SERVICE SELECTION ALGORITHMS

In this chapter, we consider service compositions and propose QoS-based service selection algorithms. Proposed approaches are based on multi-constrained service selection across service providers offering similar services with different qualities. Service selection is fulfilled by using the QoS and utilities of the services to satisfy QoS constraints specified for the applications. The QoS parameters considered are monetary price, execution time and availability. We suppose that the price and availability information are encoded in WSDL, whereas execution time is computed by using request arrival and service time statistics. Moreover, execution time is not taken into account in its pure view: particular percentile of the execution time is considered [55]. Such consideration of execution time will influence to the quality of overall service selection and composition process in terms of service admissibility and delay time.

3.1 Application Model

In this thesis, we consider applications that are compositions of services that fulfill some task. Number of services in the composition varies and depends on the given task, i.e. different application types have different number of services. Flow structure of a composition can have a general or a sequential flow. In this thesis, we only consider sequential flow structure. Figure 3.1 demonstrates a possible service composition for applications. Each service (S) for the application could be selected among services that are defined under particular service type (ST). However, it must be ensured the satisfaction of QoS-constraints and among the possible compositions, one with the highest utility should be chosen. The QoS parameters considered,

service selection and execution model and how the service selection is performed according to the QoS constraints are discussed in the following sections.

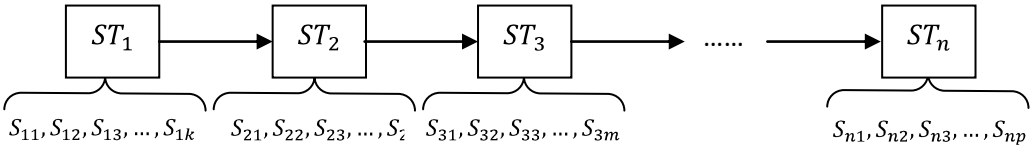


Figure 3.1 Sequential-flow structure of service composition

3.2 QoS Parameters

Following QoS parameters [1,2] are taken into account when services are selected for composition:

- Execution time– is the interval of time between sending the request to the first service and receiving response from the last service. It is the sum of the execution times of the services in the composition.
- Price – is the amount of money that should be paid for using the services. It is the sum of the prices of services in the composition.
- Availability – is the probability of the application completed without any failure. It is the multiplication of availabilities of individual services.

Table 3.1 Aggregate functions for sequential control-flow composition

Quality of Service	**Formula**
Execution time	$\sum_{i=1}^{n} ET(s_i)$
Price	$\sum_{i=1}^{n} P(s_i)$
Availability	$\prod_{i=1}^{n} A(s_i)$

We assumed that the Price and Availability information for a service is specified in its WSDL, whereas Execution Time is predicted by using request arrival and service time statistics for each service. The formulas in Table 3.1 are used for determining overall QoS of a composition which consists of n services s_i, i=1..n[17].

In Table 3.1, ET(s), P(s), and A(s) stands for execution time, price, and availability of service s, respectively.

3.3 Utility model

Each service has a utility, a value that measures profit provided by the service. In this thesis, we used the utility function in [17] to select the best among the compositions satisfying the QoS criteria. The utility for a service s_i is computed in two steps as follows [17]:

1. Normalization of QoS values q=[q_j]=[ET, Price, Av], that is transforming them into values between [0, 1]:

For negative (i.e., ET and Price) QoS parameters (parameter that should be minimized):

$$q_{i,j} = \begin{cases} \frac{q_j^{max} - q_{i,j}}{q_j^{max} - q_j^{min}} & if\, q_j^{max} - q_j^{min} \neq 0 \\ 1 & else \end{cases} \quad \text{(Equation 3.1)}$$

For positive (i.e., Av) QoS parameters (parameters that should be maximized):

$$q_{i,j} = \begin{cases} \frac{q_{i,j} - q_j^{min}}{q_j^{max} - q_j^{min}} & if\ q_j^{max} - q_j^{min} \neq 0 \\ 1 & else \end{cases} \quad \text{(Equation 3.2)}$$

Where, q_j^{min} and q_j^{max} are the minimum and the maximum values of QoS parameter j across all services, $q_{i,j}$ is the value of QoS parameter j for service s_i.

2. Utility of service s_i is calculated as follows:

$$U_i = \sum_{j=1}^{m} q_{i,j} / n \quad \text{(Equation 3.3)}$$

where *m* is the number of QoS parameters considered.

3.4 Broker Model

In our model, a Broker is responsible for selecting services for an application request sent by a client and executing the application by calling those services. Figure 3.2 illustrates the application execution scenario considered. For each application

request, possible compositions are evaluated against QoS-constraints, and if there is no composition that satisfies the QoS-constraints, the application request is rejected. Broker employs an exhaustive search algorithm to find the possible compositions and select the highest utility composition among them. As mentioned in the previous section, one of the considered QoS parameters is the execution time of the application and it is predicted in the broker model according to queuing theory. The role of queuing theory is calculating of execution time (ET) on each service according to request arrival, service execution statistics for the service and the number of processors on the server providing the service. It is assumed that these statistics are kept by each server and periodically sent to the broker. In particular, *M/M/c* queuing model is leveraged, where two *M* is stands for Markov Arrival and Service processes. We assumed that arrivals follow a Poisson distribution and service times are distributed exponentially and *c* is the number of processors on the server. We assumed that the length of the queue is infinite in each server.

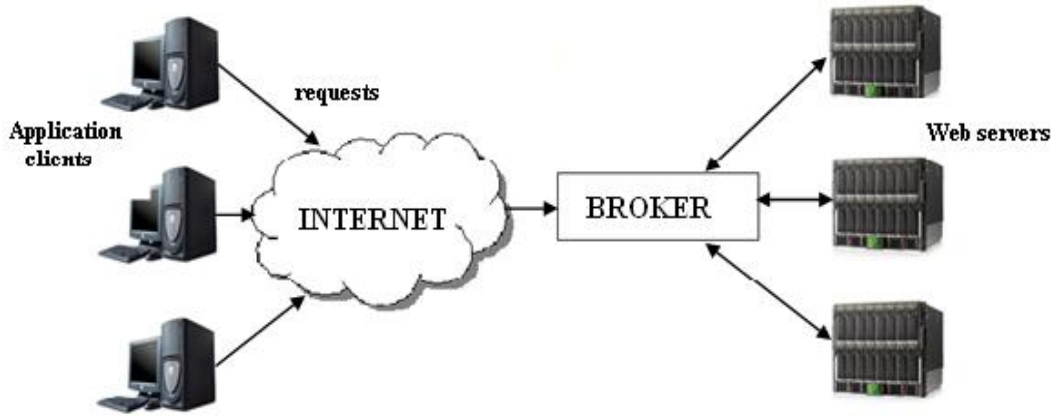


Figure 3.2 Broker model that provides QoS-aware service selection and composition mechanism

According to the Figure 3.2, application clients request services for different application types from the broker over the Internet. These requests include QoS requirements for their web service compositions and the type of services to be called for the application. Broker has necessary information about available services offered by each provider. This information includes the service types, service QoS characteristics such as price and availability, the number of processors on the server providing the service, and service execution rate. The request arrival rate is updated

by each provider in regular base. Broker uses queuing theory for computing execution time on the servers according to the information about services. Then, possible service compositions are created and checked for the QoS requirements. If the requirements are satisfied by at least one composition, the application is admitted and execution starts. If QoS-constraints aren't satisfied by any composition, the request is denied.

3.5 Service Selection Algorithms

Two service selection methods are proposed in the thesis work:

1. Sojourn Time (pth percentile)-based (STP) Method. This service selection method selects services based on price, availability, and percentile of the execution time.

2. Mean Sojourn Time-based (MST) Method. Service selection is based on price, availability and mean sojourn time.

In addition to these algorithms, we define another algorithm just to make performance comparisons.

3. Random Selection (RS) Method. This method selects services randomly.

In *STP Method* and *MST Method*, optimal service selection is achieved by maximizing utility of composition that satisfies the QoS requirements whereas in RND method QoS requirements are not taken into account.

In Chapter 4, the performances of these three different service selection methods are compared in terms of QoS satisfaction, utility provision, service admissibility, and delay.

3.5.1 Sojourn Time (pth percentile)-based service selection

In this method, the QoS constraint for the execution time is formulated as follows: select services for the composition that provide overall execution time less than t seconds for p% [55] of similar requests (i.e., the same application). Let T be the random variable for execution time of the similar requests. Therefore, the QoS constraint for the execution time is:

$$Prob\ T \leq t\ = p/100 \qquad \text{(Equation 3.4)}$$

For a selected service, the probability in (Equation 3.4) can be found by using the distribution of sojourn times of customers in M/M/c queue as [56]:

$$Prob\ T \leq t\ = 1 - \frac{P_w}{1-c(1-\rho)} e^{-c\mu\ 1-\rho\ t} - (1 - \frac{P_w}{1-c\ 1-\rho}) e^{-\mu t} \quad \text{(Equation 3.5)}$$

Where P_w is:

$$P_w = \frac{c\rho^c}{c!}(\ 1-\rho\ \ \ \sum_{n=0}^{c-1} \frac{c\rho^n}{n=0} + \frac{(c\rho)^c}{c!})^{-1} \quad \text{(Equation 3.6)}$$

Parameters used in (Equation 3.5) and (Equation 3.6) are given in Table 3.2.

Table 3.2 Notation used in Execution Time prediction using M/M/c queuing model

Parameter	**Description**
T	Random variable representing the execution time
t	Execution time constraint
c	Number of processors on the server providing the service
λ	Request arrival rate
μ	Service execution rate
ρ	Fraction of time the server is busy and defined as λ/(c*μ)

Let T be a random variable for the overall sojourn time for a composition. T will be equal to the sum of random variables of sojourn time of each service in the composition as:

$$T = T_1 + T_2 + \cdots + Tn \quad \text{(Equation 3.7)}$$

Therefore, the cumulative distribution function [57] for the composition will be

$$F_T\ t\ = Prob\ T < t\ = Prob\{T_1 + T_2 + \cdots + Tn < t\} \quad \text{(Equation 3.8)}$$

The above function can be found in terms of number of processors, arrival and service rates of the services in the composition. In order to find the cumulative distribution function for the composition we can use individual cumulative distribution functions for each service. By using Equation 3.5 cumulative distribution function of sojourn time for each service can be found as:

$$F_{T_1}\ t\ = P\ T_1 \leq t\ = 1 - \frac{P_{w1}}{1-c_1\ 1-\rho_1} e^{-c_1\mu_1\ 1-\rho_1\ t} -\ \ 1 - \frac{P_{w1}}{1-c_1\ 1-p_1}\ \ e^{\mu_1 t}$$

$$F_{T_2}\ t\ = P\ T_2 \le t\ = 1 - \frac{P_{w2}}{1 - c_2\ 1 - \rho_2} e^{-c_2\mu_2\ 1-\rho_2\ t} - \ 1 - \frac{P_{w2}}{1 - c_2\ 1 - p_2}\ \ e^{\mu_2 t}$$

…… (Equation 3.9)

$$F_{T_n}\ t\ = P\ T_n \le t\ = 1 - \frac{P_{wn}}{1 - c_n\ 1 - \rho_n} e^{-c_n\mu_n\ 1-\rho_n\ t} - \ 1 - \frac{P_{wn}}{1 - c_n\ 1 - p_n}\ \ e^{\mu_n t}$$

The corresponding probability density functions can be computed by finding the derivative of cumulative distribution functions as:

$$f_{T_1}\ t\ = \frac{c_1\mu_1(1-\rho_1)P_{w1}}{1 - c_1(1-\rho_1)} e^{-c_1\mu_1(1-\rho_1)t} + \mu_1(1 - \frac{P_{w1}}{1 - c_1(1-\rho_1)})e^{-\mu_1 t}$$

$$f_{T_2}\ t\ = \frac{c_2\mu_2(1-\rho_2)P_{w2}}{1 - c_2(1-\rho_2)} e^{-c_2\mu_2(1-\rho_2)t} + \mu_2(1 - \frac{P_{w2}}{1 - c_2(1-\rho_2)})e^{-\mu_2 t}$$

…… (Equation 3.10)

$$f_{T_n}\ t\ = \frac{c_n\mu_n(1-\rho_n)P_{wn}}{1 - c_n(1-\rho_n)} e^{-c_n\mu_n(1-\rho_n)t} + \mu_n(1 - \frac{P_{wn}}{1 - c_n(1-\rho_n)})e^{-\mu_n t}$$

Suppose $\frac{c\mu(1-\rho)P_w}{1-c(1-\rho)} = n_1$, $c\mu\ 1 - \rho\ = a_1, \mu\ \ 1 - \frac{P_w}{1-c\ 1-\rho}\ = n_2$, and $\mu = a_2$, thus we have:

$$f_{T_1}\ t\ = n_{11}e^{-a_{11}t} + n_{12}e^{-a_{12}t}$$

$$f_{T_2}\ t\ = n_{21}e^{-a_{21}t} + n_{22}e^{-a_{22}t}$$

…… (Equation 3.11)

$$f_{T_n}\ t\ = n_{n1}e^{-a_{n1}t} + n_{n2}e^{-a_{n2}t}$$

The random variable T is the sum of the random variables T_i, i=1..n. Therefore, probability density function of T can be found by n-fold convolution of probability density functions of T_i. In order to simplify the convolution we can take the Laplace transforms of these probability density functions as [22]:

$$\mathcal{F}_{T_1}\ s\ = n_{11}\frac{1}{s + a_{11}} + n_{12}\frac{1}{s + a_{12}}$$

$$\mathcal{F}_{T_2}(s) = n_{21}\frac{1}{s+a_{21}} + n_{22}\frac{1}{s+a_{22}}$$

…….……..…… (Equation 3.12)

$$\mathcal{F}_{T_n}(s) = n_{n1}\frac{1}{s+a_{n1}} + n_{n2}\frac{1}{s+a_{n2}}$$

By using the identity for the Laplace Transform of convolution [22] we get:

$$\mathcal{L}(f_{T1}(t) * f_{T2}(t) * \ldots * f_{Tn}(t)) = \mathcal{L}(f_{T1}(t)) \cdot \mathcal{L}(f_{T2}(t)) \cdot \ldots \cdot \mathcal{L}(f_{Tn}(t)) =$$

$$= \left(\frac{n_{11}}{s+a_{11}} + \frac{n_{12}}{s+a_{12}}\right) \cdot \left(\frac{n_{21}}{s+a_{21}} + \frac{n_{22}}{s+a_{22}}\right) \cdot \ldots \cdot \left(\frac{n_{n1}}{s+a_{n1}} + \frac{n_{n2}}{s+a_{n2}}\right) \quad \text{(Equation 3.13)}$$

(Equation 3.13) is expanded using partial fraction expansion method as explained in Appendix A. Then by taking inverse Laplace transform of the result (sum of the inverse Laplace transform of each fraction) and integrating from 0 to t gives the cumulative distribution function for the sojourn time of the composition. Then, by computing the $F_T(t)$, we can find the percentile of the sojourn time t for the composition. If the percentile is greater than the requirement (we take 0.95 which leads to 95th percentile) the composition is feasible for the application.

As a special case, when the number of processors on a server is equal to 1, Equation 3.9 for c=1 becomes:

$$F_T(t) = P(T \leq t) = 1 - e^{-\mu(1-\rho)t} \quad \text{(Equation 3.14)}$$

The corresponding probability density function is:

$$f_T(t) = \mu(1-\rho)e^{-\mu(1-\rho)t} \quad \text{(Equation 3.15)}$$

Expression (3.15) can be rewritten as in (3.16) by denoting $n = \mu(1-\rho)$ as:

$$f_T(t) = ne^{-nt} \quad \text{(Equation 3.16)}$$

As in expression (3.12) we use Laplace Transform formula as:

$$f_T(t) = n\frac{1}{s+n} \quad \text{(Equation 3.17)}$$

(Equation 3.17) takes place in convolution of functions as in (3.13) and contributes to the formation of expression that further decomposed using partial fraction expansion.

3.5.2 MST – Mean Sojourn Time-based Service Selection

This method considers total price, availability and average execution time of services in the composition. Average execution time of a service s_i can be predicted as follows:

$$E\ T\ = \frac{1}{\mu} + P_w \frac{1}{1-\rho} \frac{1}{c\mu} \quad \text{(Equation 3.18)}$$

Parameters used in this formula can be found in Table 3.2 and Equation 3.6. After finding average execution time of each service, average total execution time of the composition can be found as the sum of individual execution times.

3.5.3 RND – Randomly Selection Method

In this method services for composition are selected without taking into account QoS requirements. A composition among the possible compositions is selected randomly without taking into account QoS requirements and the utilities.

CHAPTER 4

PERFORMANCE EVALUATION

This chapter presents the performance evaluation of proposed service selection algorithms. The simulation software used, scenarios simulated and performance evaluation of the proposed methods are given in the following sections.

4.1 Simulation Software

To demonstrate performance of service selection and composition mechanisms described in Chapter 3, custom event-driven simulation software is developed. The simulator is implemented in the Java programming language and it simulates broker,

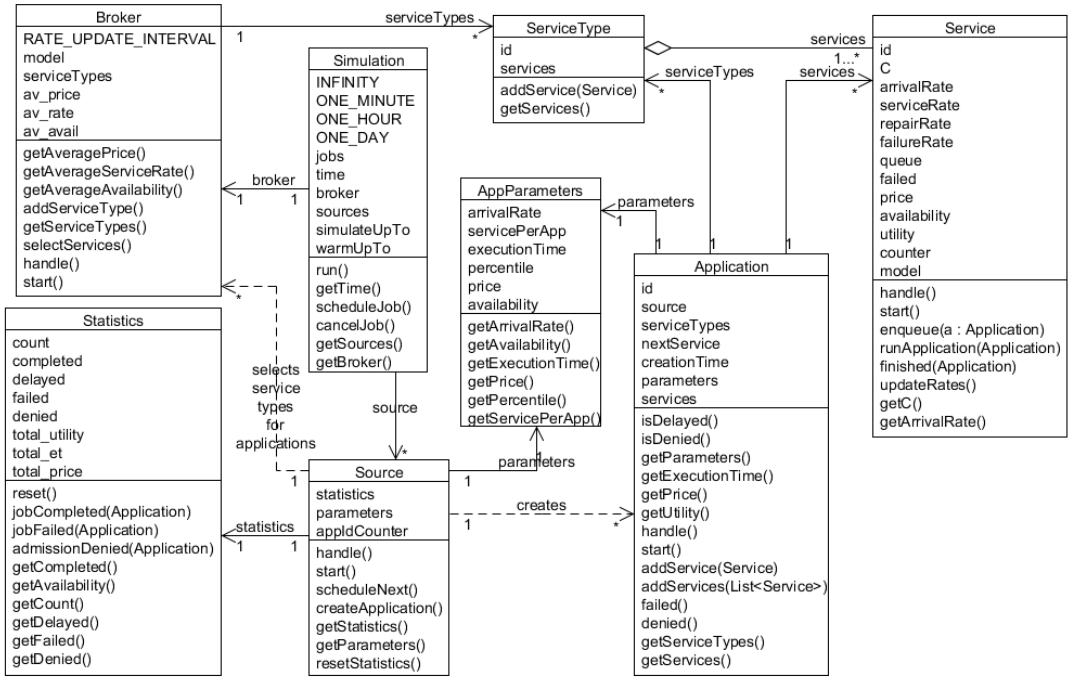


Figure 4.1 Partial class diagram for the simulation software

clients and the service providers. Figure 4.1 depicts the partial class diagram of developed simulation software which includes the noteworthy classes constituting

the simulator. At the beginning of each simulation, broker, sources with application parameters (i.e., QoS parameters), duration of the simulation and duration of warm up period are initialized. *Broker* initializes service providers with randomly assigned QoS parameters such as number of processors on the server, service rate, prices of services, and availability. The parameters for the services used in the experiments are chosen randomly from the values given in Table 4.1. As the table indicates, 10 service types are created in the experiments. For the number of services for each service type is randomly selected from the set {5, 7, 9}. Then three integer random numbers, r1, r2, r3 are generated in the range [0,2] and the values at index positions r1, r2, and r3 are used from the corresponding sets for service rate, number of processors, and availability, respectively, for each service.

Table 4.1 Input data used for creating services

Name	Abbreviation	Values
Service rate	μ	{8; 10; 12}
Number of service types	N_{st}	10
Number of processors per server	C	{1; 2; 4}
Availability	Av	{0.99; 0.98; 0.97}
Services per service type	N_s	{5; 7; 9}

Price for using a service is calculated according to the following expression:

$$Price = (r1 + 1) + (r2 + 1) + (r3 + 1) \qquad \text{(Equation 4.1)}$$

The randomly generated service types and services used in the experiments are presented in Appendix B. The same set of service types and services is used in the experiments.

After service types and their services are created, simulation starts. During the simulation, the broker updates arrival rate statistics for the services in each second. The arrival rate for each service is computed by means of the moving average method and the number of arrivals in the last 10 seconds is counted for this purpose. *Source* is responsible for creation of applications (i.e., compositions) and determining service types for applications. In the simulations, there may be one or more sources that create requests at different arrival rates. A source always creates an

application consisting of the same number of service types. The service types utilized in an application is randomly chosen from the set of available service types.

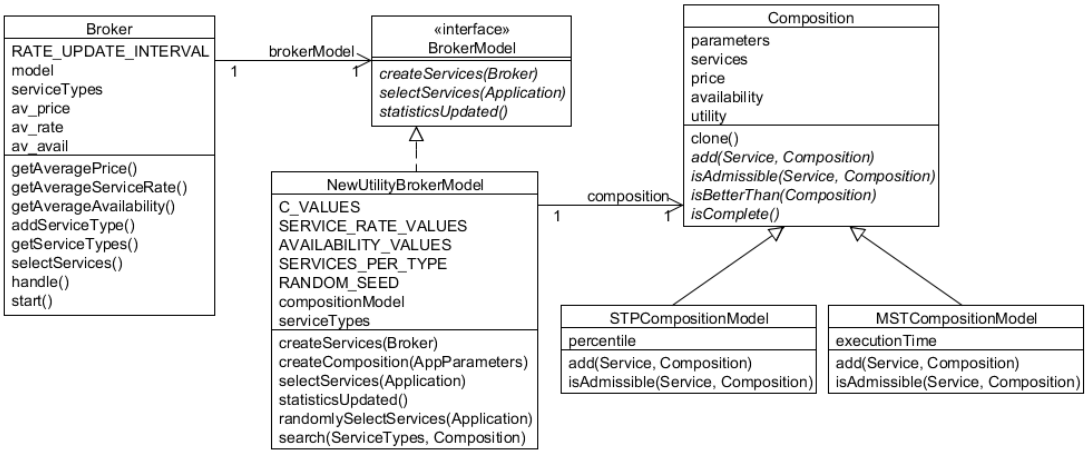


Figure 4.2 The Partial Class Diagram of broker, broker models and composition models

The number of services per application is defined while creating the service. Each source also has predefined QoS requirements such as average execution time, percentile of execution time, maximum price, and minimum availability which are set by the constructor of the source class. When a request (i.e., an application) is created by a source, the request is sent to the broker and broker handles service selection for the composition, applying the methods presented in Chapter 3. The utility classes used for this purpose is presented in Figure 4.2. After the service selection completed the request is handled and the statistics are updated accordingly.

4.2 Defining QoS requirements

The same set of QoS requirements are used for the compositions created by the same source. The QoS constraints are set for a source according to the following formulas. In the simulations we create sources for 3, 4, and/or 5 services/application. In the formulas, the average values for each QoS parameter across all services are used. Therefore, scale is used for adjusting QoS requirements, which is supposed to contribute in demonstrating methods' advantages more precisely.

$$ET_c = scale * \ 3 * \frac{(\text{spc})}{\mu_{av}} \qquad \text{(Equation 4.2)}$$

$$Price_c = scale * \ \text{spc} \ * Price_{av} + \ \text{spc} \qquad \text{(Equation 4.3)}$$

$$Av_c = 1 - scale * (1 - Av_{av})^{(\text{spc})} \quad \text{(Equation 4.4)}$$

Data used in these formulas are given in Table 4.2. These values are the average values for the services presented in Appendix B.

Table 4.2 Data used to calculate QoS constraints

Name	Abbreviation	Values
# of services per composition	spc	{3, 4, 5}
Scale for adjusting QoS constraints	scale	0.95
Average service rate	μ_{av}	10.2857
Average price	$Price_{av}$	6.1428
Average Availability	Av_{av}	0.9794

4.3 Validation of Simulation Model

In order to check a reliability of results provided by the developed simulator, we perform a set of experiments. In this section, test results for percentile of execution time of the composition and overall availability of the services in the composition is presented. The results obtained from the simulations and the expected results which are computed analytically are compared. For this purpose, a special broker model is designed for creating services and applications. Simulation duration is one day where one hour is spent for the warm up and the rest of the time is spent for collecting statistics. Input data for creating services are given in the Table 4.3:

Table 4.3 Input data used for validation tests

Name	Abbreviation	Values
Number of service types	NST	10
Number of services per service type	NSST	5
Number of services per application	NSA	4
Number of processors per service provider	C	3
Service rate	μ	1
Price	P	1

According to M/M/C queuing model the load per processor is,

$$\rho = \frac{\lambda_s}{\mu * C} \quad \text{(Equation 4.5)}$$

Where $\lambda_s$is service arrival rate, C is the number of processors and μ is the service execution rate.

Service arrival rate is determined as following:

$$\lambda_s = \frac{\lambda_t * NSA}{NSST * NST} \quad \text{(Equation 4.6)}$$

Where λ_t is the number of service calls/sec and other abbreviations can be found in Table 4.3. Hence, equation 4.6 can be rewritten:

$$\lambda_t = \frac{\rho * C * \mu * NSST * NST}{NSA} \quad \text{(Equation 4.7)}$$

Variables in Equation 4.7 are given in Table 4.3 and Equation 4.5.

Execution time threshold is determined according to:

$$ET = \frac{2}{\mu} NSA \quad \text{(Equation 4.8)}$$

Value of percentile of ET (pET) per composition provided by simulation is calculated as following:

$$pET = 100 * (1 - \frac{d}{c}) \quad \text{(Equation 4.9)}$$

Where d is the number of delayed requests and c is the number of completed requests.

Analytical value of pET can be found for selected ET (Please refer to Equation 4.8) and using method described in Section 3.5.1.

Availability provided by the simulation is defined as: Number of failed requests/ Number of generated requests. Analytical value of availability can be found as:

$$Av_a = 1 - Av^{(\text{spc})} \quad \text{(Equation 4.10)}$$

Where Av is a constraint value for availability and spc stands for the number of services per composition.

Validation Test1 is dedicated to testing analytical pET and experimental pET. For this, all data except ρ are constant, including Availability, which is equal to 1. Results of the tests are given in Table 4.4. As seen in Table 4.4, difference between analytical and experimental values of pET is very small: the biggest difference is about ~0.16% at $\rho = 0.8$.

Table 4.4 Results of Validation Test1

ρ	Analyticalresults	Simulationresults
0.5	92.3626	92.3886
0.6	87.8461	87.7803
0.7	76.943	76.9367
0.8	52.2636	52.3488

Validation Test2. Analytical and experimental values of availability are measured. This time, all data except Av are constant including $\rho = 0.5$. Results of this test are depicted in Figure 4.3.

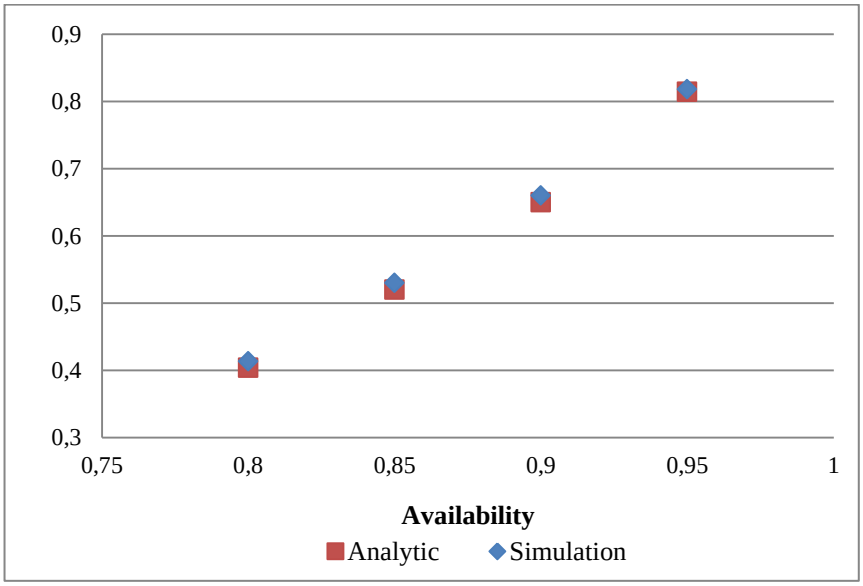


Figure 4.3 Result of Validation Test2.

Figure 4.3 shows results of Validation Test2, which demonstrates that difference between analytical and experimental values of availability is very small: the largest difference is about ~0.01 when availability equals to 0.85 and 0.9. Based on two validation tests, it can be concluded that simulation model is able to provide reliable results for evaluating the proposed service selection and composition models.

4.4 Experimental scenarios

Two scenarios are considered for testing proposed approaches:

Scenario1. Service for selection and composition are requested simultaneously by three sources during one day. Total arrival rate, λ_t=number of service calls/sec, is split to each source in according to:

$$\lambda_i = \frac{\lambda_t}{n * spc_i} \qquad \text{(Equation 4.11)}$$

Where λ_I is the number of application requests generated by source i, n is the number of sources. QoS requirements, arrival rates and number of services for applications used in this scenario are shown in Table 4.5.

Table 4.5 Settings for the Scenario1

Source	spc_i	ET	Percentile	Price	Availability	Arrival Rates Simulated			
						λ_t=20	λ_t=40	λ_t=60	λ_t=80
1	3	0.8312	95	17.5071	0.9425	2.22	4.44	6.67	8.89
2	4	1.1083	95	23.3428	0.9242	1.67	3.33	5	6.67
3	5	1.3854	95	29.1785	0.9062	1.33	2.67	4	3.33

Scenario2. Service selection and composition are requested by one source during one day. QoS requirements are the same as for Scenario1. Arrival rate and number of services for application are given in Table 4.6.

Table 4.6 Settings for the Scenario 2

Simulation Set	# of Services (spc_i)	ET	Percentile	Price	Availability	Arrival Rates Simulated			
						λ_t=20	λ_t=40	λ_t=60	λ_t=80
I	4	1.1083	95	23.3428	0.9242	5	10	5	20
II	5	1.3854	95	29.1785	0.9062	4	8	12	16

4.5 Simulation Results

In order to evaluate each methods performance, statistical data such as percentage of delayed compositions, admission probability, average ET, average availability, and average price are measured. Each method is simulated for an entire

Table 4.7 Statistical data collected in simulations

Subject of the measurement	Measuring method
Percentage of delayed compositions	100 * # of delayed requests/ # of completed requests
Admission probability	1.0 - # of denied requests/ # of total requests
Average Utility	total Utility / # of completed requests
Average ET	total ET/ # of completed requests
Average price	total Price paid / # of completed requests
Average availability	# of failed requests/ # of generated requests

day of 24 hours of simulation running, one hour is used for warming-up and statistics are collected for the remaining 23 hours. Table 4.7 demonstrates which statistics are collected. In this table, delayed requests are the requests whose execution time exceeds the average execution time specified. In some cases, it is not possible to find a composition which satisfies QoS constraints with STP and MST algorithms. In such cases, those requests are denied.

4.5.1 Simulation results of Scenario1

Tables 4.8, 4.9, and 4.10 present the results of simulations for Scenario1. As it is described in the previous section, Scenario1 is executed with three sources, with spc=3, spc=4 and spc=5. Above mentioned tables contain results of simulation for each source respectfully.

Table 4.8 Results of simulation for applications with spc=3

λ_t	Method	# of services/app =3						% of successfully completed requests
		DR (%)	AP	AvU	AvET	AvP	AvAv	
20	STP	2.79	0.85	2.23	0.35	16.6	0.9515	82.63
	MST	3.76	0.86	2.24	0.36	16.59	0.9646	82.77
	RND	1.35	1	1.41	0.3	**18.57**	**0.9375**	98.65
40	STP	3.88	0.81	2.16	0.37	16.7	0.9436	77.86
	MST	**7.44**	0.86	2.19	0.43	16.6	0.9486	79.6
	RND	1.45	1	1.41	0.3	**18.56**	0.9511	98.55
60	STP	4.77	0.78	2.09	0.38	16.75	0.9568	74.28
	MST	**12.16**	0.84	2.12	0.49	16.63	0.961	73.79
	RND	1.53	1	1.41	0.31	**18.57**	**0.9417**	98.47
80	STP	**5.71**	0.75	2.02	0.4	16.78	0.9439	70.72
	MST	**17.35**	0.84	2.05	0.55	16.64	0.9505	69.43
	RND	1.65	1	1.42	0.31	**18.58**	**0.9372**	98.35

*Delayed requests (%) (DR (%)).*With the increase of arrival rate, even though the average execution time constraint is satisfied with MST, the percentage of applications which experience execution time higher than average execution time constraint increases as the overall arrival rate increases. As seen from the Table 4.8 percentage of delayed requests by MST method reaches to 17.35% when total arrival rate equals to 80. This situation is repeated in Tables 4.9 and 4.10. This is due to the fact, that MST tends to select the same services repeatedly, making them very busy and so, the completion of the application takes a long time. STP method satisfies 5%

constraint (i.e., 95th percentile) in all simulations except one of cases, when spc=3 and arrival rate is 80 (Please refer to Table 4.8). Such violations of the the given constraint can be explained with the fact that, the number of services per application is too little for compensating a large execution time experienced in one of the services. In some cases, the arrival rate to a service may be very high and this may continue until when the next arrival rate measurement is sent to the broker. That is, the current arrival rate used in computations may not match with the actual arrival rate for some time period during which queue builds up in the service and large execution times are experienced. RND method composes services randomly, and so, almost no service is selected too regularly for making requests delayed. All three tables demonstrate the low level of delayed requests provided by RND method.

Table 4.9 Results of simulation for applications with spc=4

λ_t	Method	# of services/app =4						% of successfully completed requests
		DR (%)	AP	AvU	AvET	AvP	AvAv	
20	STP	1.49	0.97	3.03	0.46	22.4	0.9272	95.55
	MST	2.04	0.97	3.03	0.48	22.4	0.9456	95.02
	RND	0.66	1	1.88	0.4	**24.76**	**0.9176**	99.34
40	STP	2.45	0.96	2.96	0.49	22.50	**0.92**	93.65
	MST	**5.19**	0.97	2.97	0.56	22.41	0.9204	91.97
	RND	0.7	1	1.88	0.41	**24.75**	0.9353	99.3
60	STP	3.37	0.95	2.88	0.52	22.58	0.9353	91.8
	MST	**9.76**	0.97	2.87	0.65	22.44	0.9403	87.53
	RND	0.75	1	1.88	0.41	**24.77**	**0.9238**	99.25
80	STP	4.3	0.94	2.8	0.55	22.6	**0.9142**	89.96
	MST	**15.26**	0.96	2.8	0.73	22.47	0.9246	81.35
	RND	0.82	1	1.89	0.41	**24.78**	**0.9169**	99.18

Admission probability (AP). The only method admitting all received application request is RND method, in which there is no constraints in service selection. With the increase of arrival rates, admission probabilities of STP and MST methods are decreasing. This especially true for the first source with spc=3. First reason is explained with the fact that, when the amount of requests is large, more services become busy and so more requests are denied. The second reason is little number of services for application makes hard to compensate a large delay in one of the services. It is seen from the Table 4.8, that STP has worse admissibility than

MST method. But if to evaluate the overall percentage of successfully completed requests, which is determined by multiplicating delayed requests and admission probability, it is seen that STP and MST have almost the same performance (Please refer to Table 4.8). Admission probabilities occurred in two other sources are in a satisfactory level. It leads to the increase of percentage of successfully completed applications. Table 4.9 and Table 4.10 show that this measurement is better for STP than for MST, because STP provides delay time of requests under the constraint.

Table 4.10 Results of simulation for applications with spc=5

λ_t	Method	# of services/app =5						% of successfully completed requests
		DR (%)	AP	AvU	AvET	AvP	AvAv	
20	STP	0.87	0.99	3.86	0.57	28.21	**0.9042**	98.14
	MST	1.18	0.99	3.85	0.59	28.19	0.9325	97.83
	RND	0.29	1	2.36	0.5	**30.96**	**0.8984**	99.71
40	STP	1.73	0.99	3.8	0.63	28.27	**0.8935**	97.29
	MST	3.69	0.99	3.77	0.69	28.2	0.8997	95.35
	RND	0.36	1	2.36	0.51	**30.9**	0.9193	99.64
60	STP	2.55	0.99	3.73	0.67	28.35	0.9193	96.48
	MST	**8.1**	0.99	3.65	0.81	28.24	0.9238	90.98
	RND	0.36	1	2.36	0.51	**30.95**	**0.9054**	99.64
80	STP	3.48	0.99	3.65	0.7	28.41	**0.8989**	95.55
	MST	**13.92**	0.99	3.54	0.92	28.27	0.9043	85.22
	RND	0.41	1	2.36	0.52	**30.96**	**0.8976**	99.59

Utility. Utilities provided by STP and MST are almost equal in simulations for all sources. RND method provides much lower utilities than other two methods (Please see Tables 4.8, 4.9 and 4.10).

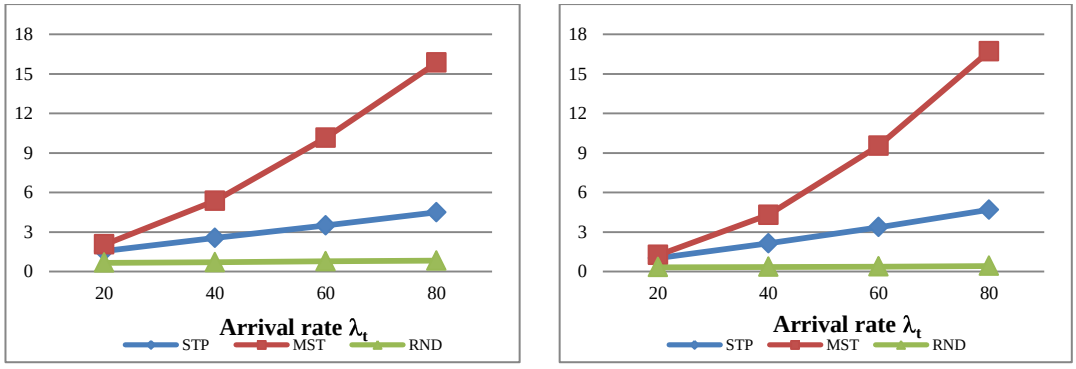
QoS constraints. Requirement for ET is fulfilled by all three methods. STP and MST always satisfy price constraint and RND always violates it (Please refer to Table 4.8, Table 4.9 and Table 4.10). Availability constraint is fulfilled by MST in all simulations. As RND doesn't take into account QoS values, availability is violated in many experiments. STP method doesn't provide required availability in several simulations. This is probably due to the short duration of simulation. As the failures are rare events, short simulation duration reduces the confidence level of the statistics for measuring availability. Therefore, much longer simulations are necessary to get more precise results.

4.5.2 Simulation results of Scenario2

Outcomes of simulations for Scenario2 are given in Figures 4.4, 4.5, 4.6 and 4.7.

As described in Section 4.1.3 simulations are held with single source: with spc=4 or spc=5. In figures, (a) depicts simulations with four services per application and (b) depicts simulations with five services per application.

*Delayed requests (%).*Figures 4.4 (a) and (b) depict percentage of delayed requests with respect to arrival rates. Figures show that MST method's delayed requests drastically increased with the increase of arrival rate. STP and RND keep percentage of delayed requests in the 5% window. This satisfies the 95th percentile requirement for execution times.



(a) (b)

Figure 4.4 Percentage of delayed applications with respect to arrival rates

Admission probability. Table 4.11 shows the admission probabilities of the requests.

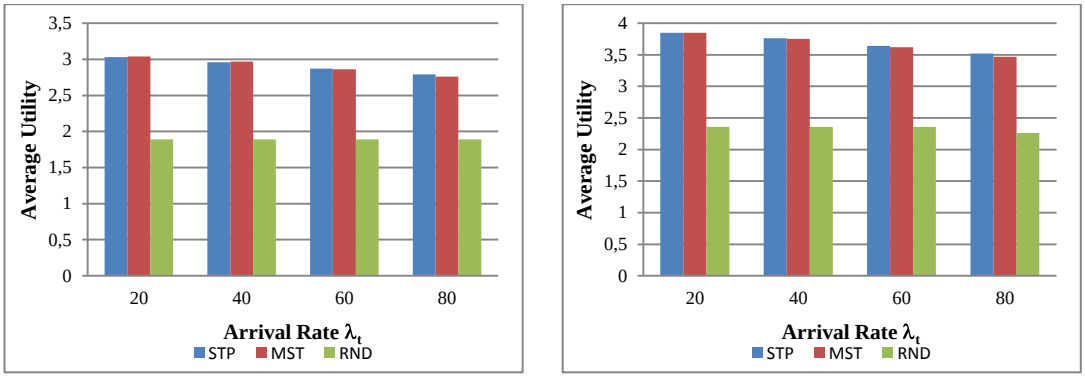
Table 4.11 Admission probabilities provided by considered methods

	$\lambda_t$for spc=4				$\lambda_t$for spc=5			
Method	20	40	60	80	20	40	60	80
STP	0.9728	0.9624	0.9518	0.9422	0.9997	0.9985	0.997	0.9956
MST	0.9761	0.974	0.9684	0.9627	0.9999	0.9993	0.9979	0.9963
RND	1	1	1	1	1	1	1	1

As Table 4.11 shows, all methods' admissibility capabilities are at the satisfactory level. Since QoS constraints are not taken into account, RND method's admission probability is equal to 1 in all cases. With MST and STP, the admission probabilities decrease as the load increases. This is mainly due to the fact that, under heavy load, execution time constraints may not be satisfied when busiest services are

required by an application. The results also indicate that admission probabilities with STP are lower than that of MST. This is an expected result as the execution time constraints with STP are much tighter than that of MST.

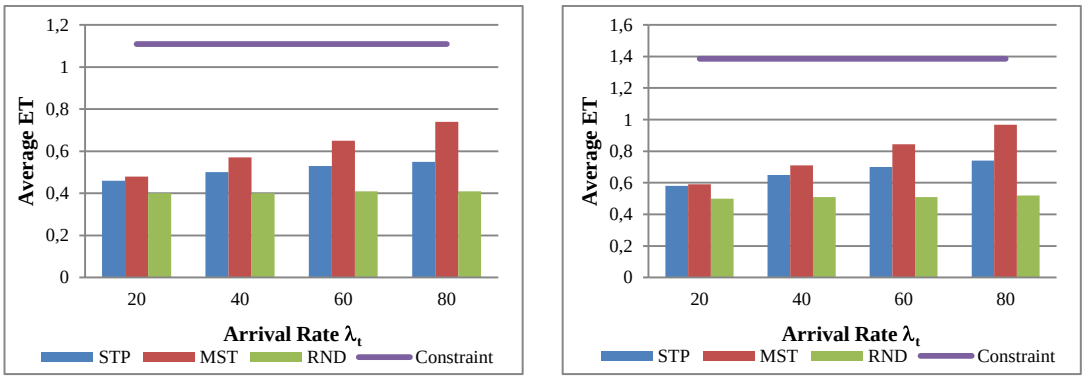
Utility. Figure 4.5 demonstrates that average utilities provided by MST and STP are almost equal as in simulations of Scenario1. Random method produces the lowest utility as it does not consider service utilities in service selection.



(a) (b)

Figure 4.5 Average utility of compositions with respect to arrival rates.

QoS constraints. Figure 4.6 depicts performances of considered methods according to ET constraints. As it is seen from the figure, ET requirement is satisfied by all three methods. The best execution times are obtained with RND method. The reason for this is that load is evenly distributed to all services with RND method and therefore load per service is lower with RND method.

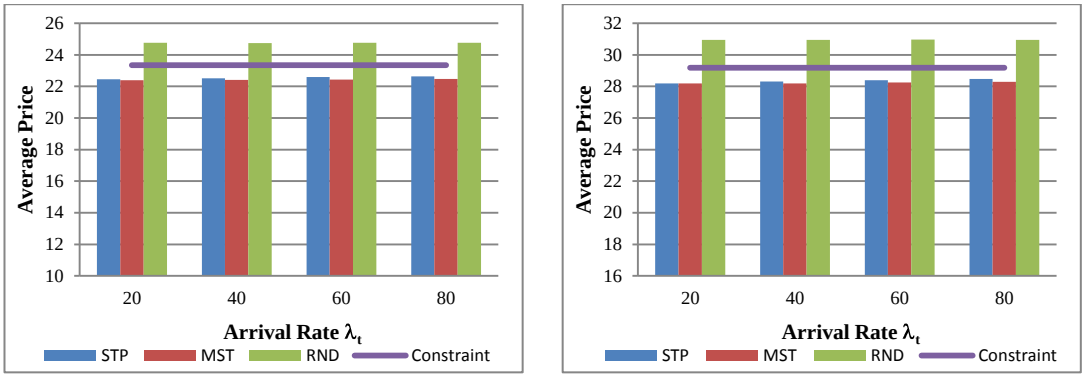


(a) (b)

Figure 4.6 Average ET of compositions with respect to arrival rates

However, services are selected according to QoS requirements with MST and STP. Therefore, some of the services will be used more frequently than the others. This leads to higher average execution times with MST and STP methods compared to RND method.

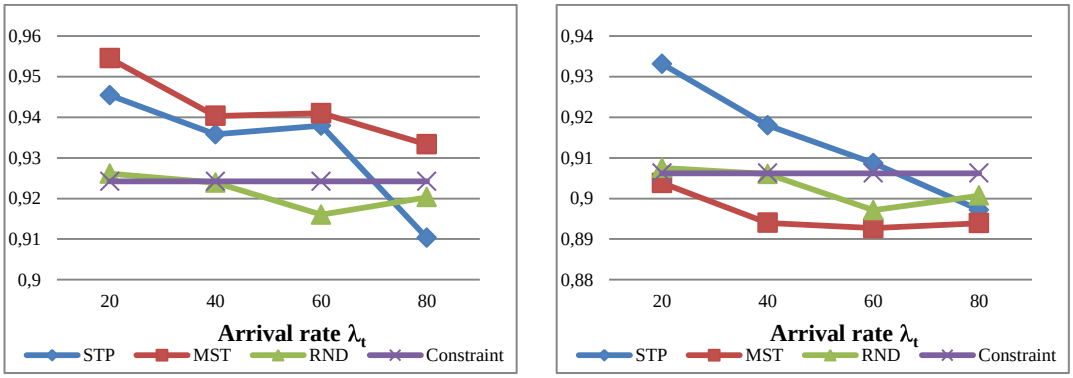
Figure 4.7 depicts average price of compositions completed by methods. As in results of Scenario 1, STP and MST satisfy price constraint while RND always violates it.



(a) (b)

Figure 4.7 Average prices of compositions with respect to arrival rates.

Figure 4.8 shows availability requirements fulfillment by STP, MST and Random. When requests have spc=4, only MST fully provides satisfactory availability. STP method violates constraint when the arrival rate equals 80. When requests have spc=5 for compositions, MST violates availability constraint in all simulation sets. Here, relatively good results are offered by STP, as it violates availability only when the arrival rate is 80. RND shows the same results in both set of simulations: availability decreases when arrival rates are 40 and 80.However, in all cases, under low load, MST and STP satisfy the availability constraint and as the load increases the availability requirement are not satisfied. Note that the services are randomly selected with RND algorithm, hence the results for RND are independent of the QoS constraints and if the availability constraint were higher, it would not be able to satisfy the constraint. On the other hand, MST and STP algorithms try to meet the QoS constraints and the simulation results show that the availability requirements are almost satisfied with these algorithms. The simulation results below the constraint might be related to the short simulation duration (which is only one day) and we can expect that with longer simulation durations independent of the load offered the availability constraint are satisfied with STP and MST.



(a) (b)

Figure 4.8 Average availabilities of compositions with respect to arrival rates.

CHAPTER 5

CONCLUSION

5.1 Summary

With the increasing demand on application creation by leveraging web services, needs for developing optimum service selection and composition has become very crucial. There are several important issues in service selection and composition. One of them is QoS-aware service selection for web services composition. This is an important issue and this study deals with this problem.

In this thesis, two broker-based service selection and composition methods, STP and MST are proposed. The QoS parameters that are taken into account are execution time, price and availability. The approaches leverage queuing theory for estimating execution time of compositions and load balancing among available services. For this purpose request arrival and execution time statistics are collected by each service provider and measured average arrival rate and average execution time values are periodically sent to the broker. Price and availability parameters for each service are assumed to be taken from WSDL documents. The exhaustive search algorithm is employed for selecting the services for the compositions.

Chapter 4 presents the results of performance evaluation with computer simulations. The performances of proposed approaches are compared with each other and with the simple random selection method (RND). Each method has several advantages and disadvantages. Based on them, it can be decided which of the approaches is the most suitable for particular cases and in what circumstances they should be applied.

The results reveal that STP method handles the requests without *over-constrained delay* in all simulated cases. *Admission probability* with the STP method is satisfactory with the number of services per composition equal to 4 and 5, but

when the number of services per composition is 3, admission probability gets lower. Offered average utility provided by STP is nearly equal to the average utility provided by MST method. STP also performs well in fulfilling *QoS requirements* for ET, and price. According to simulation results availability requirements are fulfilled not in all simulation testbeds, however this might be related to the simulation duration. That is, longer simulations might show that all requirements are satisfied with this method.

MST method is good in providing end-to-end QoS requirements. Chapter 4 showed that this method satisfies QoS constraints in all simulation cases for multiple-source case, but in single source cases availability constraint isn't fulfilled. As it is discussed in previous paragraph, it could be caused by the short simulation duration. Despite the fact that MST almost always good at providing QoS requirements, a large proportion of completed applications experience execution time that is larger than the average execution time constraint. This is especially true when the load is high. Admission probability of MST is satisfactory in simulation cases with single source. In simulation cases with triple sources, MST method's admission probability becomes low for source with spc=3. Overall percentage of successfully completed applications provided by MST is lower than that of STP. Evaluation of MST showed that this method provides good performance only when amount of requests per unit time is small.

RND method demonstrates the best performance in terms of *delayed requests* and *admission probability* but it always provide low utility compared to other methods. As expected, tight price and tight availability constraints will always be violated with RND method as the QoS constraints and utility are not taken into account in service selection.

The results obtained from simulations confirmed the assumption that STP is the method that exhibits the best performance in service selection and composition, at least for the selected simulation scenarios.

5.2 Limitations of the Study and Future Works

Availability performance of the methods couldn't be clearly demonstrated in Chapter 4. This is mainly related to the duration of the simulations. In each

simulation, only one day was simulated. The longer simulations would provide more accurate results.

In this thesis, we use exhaustive search over all possible service compositions. However, this slows down the service selection process. As a future work other selection algorithms, such as heuristic approaches can be applied instead of the exhaustive search algorithm.

We applied proposed methods to service plans that only have sequential flow structure. Real world applications mostly constructed more complexly, and described with general flow structure that includes AND, OR, XOR and loops. So, the composition module of the proposed approaches should consider general flow structure too. This is another issue that is considered as a future work for improving the applicability of the proposed methods.

In our methods, execution time is estimated without considering network delay. Moreover, our methods do not take into account the time spent for numerous interactions between broker and providers for interchanging data about services, and between broker and requestors for receiving requests. A mechanism to incorporate network delay into our methods can improve the methods' accuracy and make them more attractive for applying in real world situations.

REFERENCES

[1] D.Griffin and D. Pesch, “A survey on web services in telecommunications," *IEEE Commun. Mag.,* vol. 45, no. 7, pp. 28-35, July 2007.

[2] F. Kart, L. E. Moser, and P. M. Melliar-Smith, “Building a distributed e-healthcare system using SOA," *IT Professional,* vol. 10, no. 2, pp. 24-30, 2008.

[3] Q Chen and Y. Wang, “Enterprise Collaborative Business Systems Based on Web Services Technology”, 2010 International Conference on E-Business and E-Government, *IEEE Computer Society,* pp.34-37, 2010.

[4] S.Caballe, “On the Advantages of Using Web & Grid Services for the Development of Collaborative Learning Management Systems”, in Proceedings of the First International Conference on Complex, Intelligent and Software Intensive Systems (CISIS'07), *IEEE Computer Society,* 2007.

[5] M.P. Papazoglou “Web Services: principles and technology”, *Addison-Wesley Longman,* 752 pages, 2008.

[6] L. Baresi, E. Di Nitto, C, Ghezzi, S. Guinea, “A Framework for the Deployment of Adaptable Web Service Compositions,”, *Service Oriented Computing and Applications,* vol. 1, no. 1, pp. 75–91, 2007.

[7] S. B. Mokhtar, D. Preuveneers, N. Georgantas, V. Issarny, and Y. Berbers. “EASY: Efficient semAntic Service discoverY in pervasive computing environments with QoS and context support”, *J. Syst. Softw.,* 81(5), pp.785-808, 2008.

[8] M. Aiello, Elie el Khoury, A. Lazovik, and P. Ratelband, “Optimal QoS-Aware Web Service Composition”, Conference on Commerce and Enterprise Computing, *IEEE Computer Society,* pp.491-494, 2009.

[9] D. Lui, Z.Shao, C. Yu, G. Fan, “A Heuristic QoS-Aware Service Selection Approach to Web Service Composition”, *IEEE Computer Society,* pp.1184-1189, 2009

[10] T. Yu, K.-J. Lin, “Service Selection Algorithms for Composing Complex Services with Multiple QoS Constraints.”, *ICSOC,* pp.130-143, 2005.

[11] T. Yu, Y.Zhang, and K.-J.Lin, "Efficient algorithms for Web services selection with end-to-end QoS constraints", *ACM Trans. Web 1, 1, Article 6*, pp.26, 2007.

[12] S. Khan, K.F. Li, E.G. Manning, M.Akbar, "Solving the knapsack problem for adaptive multimedia systems", *StudiaInformaticaUniversalis,* vol. 2, no. 1, pp. 157-178, Sept. 2002.

[13] S. Mokhtar, J.Lui, N.Georgantas, V. Issarny, "QoS-Aware Dynamic Service Composition in Ambient Intelligence Environments," Proc. 20th IEEE/ACM Int'l Conf. Automated Software Eng., *ACM Press,* pp. 317–320,2005

[14] J.E. Haddad, M. Manouvrier, G. Ramirez, M. Rukoz, "QoS-Driven Selection of Web Services for Transactional Composition", *IEEE Computer Society,* pp.653-660, 2008.

[15] J.E. Haddad, M. Manouvrier, M. Rukoz, "T-QoS: Transactional and QoS-aware Selection Algorithm for Automatic Web Service Composition", *IEEE Transactions on Services Computing,* Vol.3, No1, January-March, 2010.

[16] Z. Chen, Q.Yao, "A Framework for QoS-Aware Web Service Composition in Pervasive Computing Environment", *IEEE,* pp. 1011-1016, 2008.

[17] N.B Mabrouk, S. Beauche, E. Kuznetsova, N.Georgantas, V. Issarny, "QoS-aware Service Composition in Dynamic Service Oriented Environments", *MIDDLEWARE,Lecture Notes in Computer Science,* Volume 5896, pp.123-142, 2009.

[18] B. Zhang, Y. Shi and X. Xin, "A Policy-Driven Service Composition Method for Adaptation in Pervasive Computing Environment", *THE COMPUTER JOURNAL,* Vol. 53 No. 2, pp.152-165, 2009. Available at: http://comjnl.oxfordjournals.org/

[19] K.Tari, Y. Amirat, A.Chibani, A.Yachir, A. Mellouk, "Context-aware Dynamic Service Composition in Ubiquitous Environment", *IEEE,* 2010.

[20] P.Kumaran, R.Shriram, "Service Composition Middleware for Pervasive Computing", *IEEE,* pp.25-28, 2011.

[21] T. Kusaka, T. Okuda, T. Ideguchi, X. Tian, "Queuing Theoretic Approach To Server Allocation Problem In Time-delay Cloud Computing Systems", Proceedings of the 2011 23rd International Tele-traffic Congress: Students Poster Session Paper, *ITC,* pp.310-311, 2011.

[22] www.wikipedia.org.

[23] S. Sharma, S. Singh, and M. Sharma, “Performance Analysis of Load Balancing Algorithms”, World Academy of Science, Engineering and Technology, *IEEE,* 2008.

[24] Z. Xu, R. Huang, "Performance Study of Load Balancing Algorithms in Distributed Web Server Systems", CS213 Parallel and Distributed Processing Project Report, In TR, CS213 Univ. of California,Riverside.,Available at: http://www.cs.ucr.edu/~bhuyan/CS213/load_balancing.ps.

[25] R. Motwani and P. Raghavan, “Randomized algorithms”, *ACM Computing Surveys (CSUR),* 28(1), pp.33-37, 1996.

[26] W. Leinberger, G. Karypis, and V. Kumar, "Load Balancing Across Near-Homogeneous Multi-Resource Servers", Heterogeneous Computing Workshop, 2000 (HCW 2000), Proceedings 9th, pp. 60-71, 2000.

[27] Bhaskaran Raman, Randy H. Katz, “Load Balancing and Stability Issues in Algorithms for Service Composition”, *INFOCOM 2003.Twenty-Second AnnualJoint Conference of the IEEE Computerand Communications. IEEE Societies* , vol.2, no.,pp. 1477- 1487 vol.2, 30 March-3 April 2003.

[28] C. Yan, M. Zhu, Y. Shi, “A Response Time-based Load Balancing Algorithm for Service Composition”,*Pervasive Computing and Applications, 2008. ICPCA 2008., Third International Conference on* , vol.1, no., pp.13-16, 6-8 Oct. 2008.

[29] M. Bjorkqvist, Lydia Y. Chen, Walter Binder, “Load-Balancing Dynamic Service Binding in Composition Execution Engine”, Asia-Pacific Services Computing Conference *(APSCC), 2010, IEEE Asia-Pacific* , vol., no., pp.67-74, 6-10 Dec. 2010.

[30] W. Xu, Y. Xin, and G. Lu, “A Lightweight, Fault-tolerant, Load Balancing Service Discovery and Invocation Algorithm for Pervasive Computing Environment”, The 3rd International Conference on Innovative Computing Information and Control (ICICIC'08), *IEEE,* 2008.

[31] R. Kumar, M. V. Ghatage, “Load Balancing of services with server initiated connections”, ICPWS, *IEEE,* pp. 254-257, 2005.

[32] S. Kalasapur, M. Kumar, and B.A. Shirazi, “Dynamic Service Composition in Pervasive Computing,” *IEEE Trans. Parallel and Distributed Systems*, vol. 18, no. 7, pp. 907–918, 2007.

[33] B. Lagesse, M.Kumar, M.Wright, “ReSCo: Reliable Service Composition Middleware Component for Pervasive Computing Environments”, Pervasive Computing and Communications Workshops (PERCOM Workshops), 8th IEEE International Conference*, IEEE*, pp.486-491, 2010.

[34] D'Mello D.A, Ananthanarayana V.S, "A Review of Quality of Service (QoS) Driven Dynamic Web Service Selection Techniques", 2010 5th International Conference on Industrial and Information Systems (ICIIS2010), Jul29-Aug 01, India, *IEEE,* pp.201-209, 2010.

[35] X. Han, Y. Liu, B. Xu, G. Zhang, "A Survey on Qos-Aware Dynamic Web Service Selection",*Wireless Communications, Networking and Mobile Computing (WiCOM), 2011 7th International Conference on* , vol., no., pp.1-5, 23-25 Sept. 2011.

[36] M.C. Jaeger, G. Muhl, S. Golze, "QoS-aware Composition of Web Services: A Look at Selection Algorithms", Proceedings of the IEEE International Conference on Web Services (ICWS'05), *IEEE Computer Society*, pp.1-2, 2005.

[37] Y. Zheng, H. Ni, H. Deng, L. Liu, "A Dynamic Web Services Selection Based on Decomposition of Global QoS Constraints",*Information Computing andTelecommunications (YC-ICT), 2010 IEEE Youth Conference on* , vol., no., pp.77-80, 28-30 Nov. 2010.

[38] Q, X. Fang, Peng, Q. Li, Y. Hu, "A Global QoS Optimizing Web Services Selection Algorithm based on MOACO for Dynamic Web Service Composition", 2009 International Forum on Information Technology and Applications, *IEEE Computer Society,* pp.37-42, 2009.

[39] L. Qi, R. Yang, W. Lin, X. Zhang, W. Dou, "A QoS-Aware Web Service Selection Method Based on Credibility Evaluation", 12th IEEE International Conference on High Performance Computing and Communications, *IEEE Computer Society*, pp.471-476, 2010.

[40] Z-Z. Liu, Z-J.Wang, X-F.Zhou, Y-S. Lou, L. Shang, "A New Algorithm for QoS-aware Composite Web Services Selection", *IntelligentSystemsand Applications (ISA), 2010 2nd International Workshop on* ,vol., no., pp.1-4, 22-23 May 2010.

[41] S-L. Pan, Q-J.Mao, "Semantic Web Service Composition Planner Agent with a QoS-aware Selection Model", 2009 International Conference on Web Information Systems and Mining, *IEEE Computer Society*, pp.325-331, 2009.

[42] M. Li, J. Huai, H. Guo, "An Adaptive Web Services Selection Method Based on the QoS Prediction Mechanism", 2009 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology – Workshops, IEEE Computer Society, pp.395-402, 2009.

[43] L-l. Qu, Y. Chen, "QoS Ontology Based Efficient Web Service Selection", 2009 International Conference on Management Science and Engineering (16th), September 14-16, 2009, Moscow, Russia, *IEEE,* pp.45-50, 2009.

[44] Z. Wang, J. Li, N. Zhou, Z. Lin, "Semantic Web Service Selection Based on Context and QoS", 2009 International Conference on Web Information Systems and Mining, *IEEE Computer Society,* pp.332-335, 2009.

[45] R. J. R. Raj, T. Sasipraba, "Web Service Selection Based on QoS Constraints", *Trendz in Information Sciences& Computing (TISC), 2010* , vol., no., pp.156-162, 17-19 Dec. 2010.

[46] S. Lui, S. Guo, X. Chen, M. Lan, "A QoS-based Web Service Selection Model", 2009 International Forum on Information Technology and Applications, *IEEE Computer Society,* pp.353-356, 2009.

[47] M. Tian, A. Gramm, H. Ritter, J. Schiller, "Efficient Selection and Monitoring of QoS-aware Web Services with the WS-QoS Framework", Proceedings of the IEEE, WIC/ACM International Conference on Web Intelligence (WI'04), *IEEE Computer Society,* 2004.

[48] G. Zhang, H. Zhang, Z. Wang, "A QoS-based web services selection method for dynamic web service composition", 2009 First International Workshop on Education Technology and Computer Science, *IEEE Computer Society,* pp.832-835, 2009.

[49] M. Li, T. Deng, H. Sun, H. Guo, X. Liu, "GOS: A Global Optimal Selection Approach for QoS-Aware Web Services Composition", 2010 Fifth IEEE International Symposium on Service-Oriented System Engineering, *IEEE Computer Society,* pp.7-14, 2010.

[50] T. Yu, K-J.Lin, "Service Selection Algorithms for Web Services with End-to-End QoS Constraints", Proceedings of the International Conference on E-Commerce Technology, IEEE Computer Society, pp.1-8, 2004.

[51] X. Wang, K. Yue, J.Z. Huang, A. Zhou, "Service Selection in Dynamic Demand-Driven Web Services", Proceedings of the IEEE International Conference on Web Services (ICWS'04), *IEEE Computer Society*, 2004.

[52] D. Celik, A. Elci, "Semantic QoS Model for Extended IOPE Matching and Composition of Web Services", Annual IEEE International Computer Software and Applications Conference, *IEEE Computer Society*, pp. 993-997, 2008.

[53] E. Badidi, L. Esmahi, M.A. Serhani, "A Queuing Model for Service Selection of Multi-classes QoS-aware Web Services", Proceedings of the Third European Conference on Web Services (ECOWS'05), *IEEE Computer Society*, 2005.

[54] P. Xiong, Y. Fan, "QoS-aware Web Service Selection by Synthetic Weight", Fourth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2007), *IEEE Computer Society,* 2007.

[55] D. A. Menaske. “QoS Issues in Web Services”, *IEEE Internet Computing,* pp.72-75, November-December, 2002.

[56] I. Adan and J. Resing, “Queuing Theory”, Department of Mathematics and Computer Science, Eindhoven University of Technology, 180 pages, February 28, 2002.

[57] S. M. Ross, “Introduction to Probability Models”, *Elsevier,* Ninth edition, 782 pages, 2007.

APPENDICES

APPENDIX A: Partial Fraction Expansion

Laplace Transform for n-fold convolution results in multiplication of Laplace transforms of individual terms. If the terms convoluted are exponentials, their Laplace transforms are fractions. In order to find the inverse Laplace transform of the product of fractions, the product can be decomposed using Partial Fraction Expansion and then inverse of entire product will be sum of the inverse Laplace transforms of the fractions which are exponentials. Partial fraction expansion for fractions with non-repeated and repeated real roots can be found as follows:

a) Partial Fraction Expansion for fractions with non-repeated roots.

In order to expand the partial fractions of the expression:

$$F\ s\ = \frac{mn}{s+a\ (s+b)} \qquad \text{(Equation A.1)}$$

as:

$$F\ s\ = \frac{mn}{s+a\ (s+b)} = \frac{A}{s+a} + \frac{B}{s+b} \qquad \text{(Equation A.2)}$$

By getting common denominators and summation, we have:

$$\frac{mn}{s+a\ (s+b)} = \frac{A\ s+b\ +B(s+a)}{s+a\ (s+b)} \qquad \text{(Equation A.3)}$$

So,

$$mn = A\ s + b\ + B(s + a) \qquad \text{(Equation A.4)}$$

Let $s = -a$, and we find out that $A = \frac{mn}{-a+b}$;

Let $s = -b$, and we find out that $B = \frac{mn}{-b+a}$.

Consider an example: $F\ s\ = \frac{3}{x+2\ (x-3)}$. This expression can be decomposed as following: $\frac{3}{x+2\ (x-3)} = \frac{A}{(x+2)} + \frac{B}{(x-3)}$. Define common denominator for decomposed part: $\frac{3}{x+2\ (x-3)} = \frac{A\ x-3\ +B(x+2)}{x+2\ (x-3)}$. So, $3 = A\ x-3\ + B(x+2)$, and by letting $x = -2$, we have $A = \frac{3}{-5}$. By letting $x = 3$, we have $B = \frac{3}{5}$.

As a result we have: $\frac{3}{x+2\ (x-3)} = -\frac{3}{5}\frac{1}{(x+2)} + \frac{3}{5}\frac{1}{(x-3)}$.

b) Partial Fraction Expansion for fractions with repeated real roots.

Summand of convolution expression can have the next form:

$$\frac{mn}{(s+a)^i(s+b)^j} \qquad \text{(Equation A.5)}$$

which is extended as following:

$$\sum_{q=1}^{i} \frac{A_q}{(s+a)^q} + \sum_{r=1}^{j} \frac{B^r}{(s+b)^r} \qquad \text{(Equation A.6)}$$

Multiplying both sides by $(s+a)^i$, we have:

$$\frac{mn}{(s+b)^j} = \sum_{q=1}^{i} A_q(s+a)^{i-q} + \sum_{r=1}^{j} \frac{B_r\ s+a\ ^i}{s+b\ ^r} \qquad \text{(Equation A.7)}$$

Where $$\sum_{q=1}^{i} A_q(s+a)^{i-q} = \sum_{p=0}^{i-1} A_p\ (s+a)^p \qquad \text{(Equation A.8)}$$

Letting $s = -a$, we find $A_0 = \frac{mn}{(b-a)^j}$.

In order to solve for other A_k coefficients we can differentiate the whole expression k times and let $s = -a$. Therefore A_k can be found as:

$$A_k = \frac{mn}{k!}\ \frac{\partial^k}{\partial s^k}\ s+b\ ^{-j}\Big|_{s=-a} \qquad \text{(Equation A.9)}$$

For B_r coefficients, the same procedure beginning from (Equation A.6) should be applied in terms of B_r.

The utility classes used in the simulator software for partial fraction expansion is given below. The partial fraction expansion is performed in multiply method of the FractionSum class.

Class Fraction:

```
package utils;
import java.util.Arrays;
public class Fraction implements Comparable {
double n[];
double a;
   // value == n[0] + n[1]/(x+a) + n[2]/(x+a)^2 + n[3]/(x+a)^3
public Fraction(double n[], double a) {
this.n = Arrays.copyOf(n, n.length);
this.a = a;
   }
public Fraction(double n, double a, int p) {
this.n = new double[p+1];
this.n[p] = n;
this.a = a;
   }
public Fraction(double n, double a) {
this.n = new double[]{0, n};
this.a = a;
   }
   @Override
public boolean equals(Object obj) {
return compareTo(obj) == 0;
   }
   @Override
public int hashCode() {
int hash = 7;
hash = 67 * hash + Arrays.hashCode(this.n);
hash = 67 * hash + (int) (Double.doubleToLongBits(this.a) ^
(Double.doubleToLongBits(this.a) >>> 32));
return hash;
   }
   @Override
publicintcompareTo(Object o) {
```

```
int result = -1;
if (o instanceof Fraction) {
        Fraction other = (Fraction) o;
result = Double.compare(a, other.a);
if (result == 0) {
if (n.length>other.n.length) {
result = 1;
            } else if (n.length<other.n.length) {
result = -1;
            } else {
for (inti = n.length - 1; i>= 0; i--) {
result = Double.compare(n[i], other.n[i]);
if (result != 0) {
break;
                    }
                }
            }
        }
    }
return result;
  }
public boolean isCompatible(Fraction other) {
return Double.compare(a, other.a) == 0;
  }
public void add(Fraction other) {
if (!isCompatible(other)) {
throw new RuntimeException("Fractions are not compatible!");
    }
if (n.length<other.n.length) {
        n = Arrays.copyOf(n, other.n.length);
    }
for (int i = 0; i<other.n.length; i++) {
n[i] += other.n[i];
    }
```

```
   }
public double getA() {
return a;
   }
public double[] getN() {
return n;
   }
}
```

Class FractionSum:

```
package utils;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
public class FractionSum {
   List<Fraction> fractions;
public FractionSum() {
fractions = new ArrayList<Fraction>();
   }
public List<Fraction>getFractions() {
return Collections.unmodifiableList(fractions);
   }
public FractionSum add(Fraction other) {
if (other != null) {
for (inti = 0; i<fractions.size(); i++) {
if (fractions.get(i).isCompatible(other)) {
fractions.get(i).add(other);
other = null;
break;
            }
         }
if (other != null) {
fractions.add(new Fraction(other.n, other.a));
         }
```

```
        }
return this;
    }
private static void filter(FractionSum s1, FractionSum s2) {
if (s1.fractions.size() > s2.fractions.size()) {
FractionSum s3 = s1;
            s1 = s2;
            s2 = s3;
        }
for (Fraction f : s1.getFractions()) {
for (Fraction g : s2.getFractions()) {
if ((Math.abs(f.a - g.a) / f.a) < 0.0001) {
f.a = g.a;
break;
                }
            }
        }
    }
public static FractionSum multiply(FractionSum s1, FractionSum s2) {
filter(s1, s2);
FractionSum result = new FractionSum();
for (int x = 0; x < s1.fractions.size(); x++) {
for (int y = 0; y < s2.fractions.size(); y++) {
                Fraction f1 = s1.fractions.get(x);
                Fraction f2 = s2.fractions.get(y);
double f10 = f1.n[0];
if (f10 != 0) {
double n[] = Arrays.copyOf(f2.n, f2.n.length);
for (int j = 0; j < f2.n.length; j++) {
n[j] *= f10;
                    }
result.add(new Fraction(n, f2.a));
                }
double f20 = f2.n[0];
```

```
if (f20 != 0) {
double n[] = Arrays.copyOf(f1.n, f1.n.length);
for (inti = 0; i< f1.n.length; i++) {
n[i] *= f10;
                }
result.add(new Fraction(n, f1.a));
            }
double a1 = f1.a;
double a2 = f2.a;
if (a1 == a2) {
for (inti = 1; i< f1.n.length; i++) {
for (int j = 1; j < f2.n.length; j++) {
double ni = f1.n[i];
double nj = f2.n[j];
result.add(new Fraction(ni * nj, a1, i + j));
                    }
                }
            } else {
for (inti = 1; i< f1.n.length; i++) {
for (int j = 1; j < f2.n.length; j++) {
double ni = f1.n[i];
double nj = f2.n[j];
double d = (-a1 + a2);
double r = Math.pow(d, j);
double n = ni * nj;
result.add(new Fraction(n / r, a1, i));
for (int k = i - 1; k > 0; k--) {
n *= 1 + (j - 1) / (i - k);
r *= -d;
result.add(new Fraction(n / r, a1, k));
                        }
                        d = (-a2 + a1);
                        r = Math.pow(d, i);
                        n = ni * nj;
```

```
result.add(new Fraction(n / r, a2, j));
for (int k = j - 1; k > 0; k--) {
n *= 1 + (i - 1) / (j - k);
r *= -d;
result.add(new Fraction(n / r, a2, k));
                        }
                    }
                }
            }
        }
return result;
   }
public double val(double x) {
double result = 0;
for (inti = 0; i<fractions.size(); i++) {
        Fraction f = fractions.get(i);
double d = 1;
for (int k = 0; k <f.n.length; k++) {
result += f.n[k] / d;
d *= x + f.a;
        }
    }
return result;
   }
   @Override
public String toString() {
StringBuilder sb = new StringBuilder();
for (inti = 0; i<fractions.size(); i++) {
        Fraction f = fractions.get(i);
for (int j = 0; j <f.n.length; j++) {
if (f.n[j] != 0) {
sb.append(" + ").append(f.n[j]);
if (j > 0) {
```

```
sb.append("/(x");
                    }
if (f.a> 0) {
sb.append("+");
                    }
sb.append(f.a).append(")");
if (j > 1) {
sb.append("^").append(j);
                    }
                }
            }
        }
return sb.substring(3);
    }
}
```

APPENDIX B: Simulation: Services

Services that were created and used in the simulations are given in Table B.1

Table B.1 List of services

Service Type	Service #	Number of Processors	Service Rate	Availability	Price
Service Type 0	Service0	2	12.0	0.97	6.0
	Service1	1	12.0	0.99	7.0
	Service2	2	10.0	0.99	7.0
	Service3	4	8.0	0.97	5.0
	Service4	4	8.0	0.98	6.0
Service Type 1	Service0	1	12.0	0.99	7.0
	Service1	1	10.0	0.97	4.0
	Service2	2	12.0	0.97	6.0
	Service3	1	12.0	0.97	5.0
	Service4	4	8.0	0.97	5.0
	Service5	2	8.0	0.99	6.0
	Service6	2	12.0	0.97	6.0
Service Type 2	Service0	4	12.0	0.97	7.0
	Service1	1	10.0	0.97	4.0
	Service2	4	12.0	0.99	9.0
	Service3	1	12.0	0.99	7.0
	Service4	4	12.0	0.98	8.0
	Service5	1	10.0	0.99	6.0
	Service6	2	8.0	0.98	5.0

Table B.1 (continued)

Service Type 3	Service0	4	8.0	0.97	5.0
	Service1	2	12.0	0.98	7.0
	Service2	1	12.0	0.98	6.0
	Service3	2	10.0	0.97	5.0
	Service4	4	12.0	0.98	8.0
	Service5	4	10.0	0.98	7.0
	Service6	1	10.0	0.99	6.0
Service Type 4	Service0	4	8.0	0.99	7.0
	Service1	2	12.0	0.99	8.0
	Service2	1	12.0	0.98	6.0
	Service3	4	12.0	0.97	7.0
	Service4	4	12.0	0.98	8.0
Service Type 5	Service0	2	12.0	0.98	7.0
	Service1	2	8.0	0.97	4.0
	Service2	1	8.0	0.99	5.0
	Service3	4	10.0	0.99	8.0
	Service4	1	8.0	0.99	5.0
	Service5	1	12.0	0.98	6.0
	Service6	1	8.0	0.98	4.0
Service Type 6	Service0	2	10.0	0.97	5.0
	Service1	4	8.0	0.99	7.0
	Service2	4	12.0	0.97	7.0
	Service3	1	8.0	0.99	5.0
	Service4	2	8.0	0.97	4.0
	Service5	1	12.0	0.98	6.0
	Service6	4	8.0	0.97	5.0
	Service7	1	10.0	0.99	6.0
	Service8	2	10.0	0.99	7.0

Table B.1 (continued)

Service Type 7	Service0	4	10.0	0.97	6.0
	Service1	2	12.0	0.99	8.0
	Service2	4	8.0	0.98	6.0
	Service3	4	12.0	0.99	9.0
	Service4	4	8.0	0.98	6.0
	Service5	2	12.0	0.97	6.0
	Service6	1	12.0	0.97	5.0
Service Type 8	Service0	2	12.0	0.98	7.0
	Service1	4	12.0	0.99	9.0
	Service2	2	12.0	0.97	6.0
	Service3	2	8.0	0.98	5.0
	Service4	2	8.0	0.98	5.0
	Service5	4	12.0	0.97	7.0
	Service6	1	12.0	0.97	5.0
	Service7	2	8.0	0.97	4.0
	Service8	4	12.0	0.97	7.0
ServiceType9	Service0	4	8.0	0.98	6.0
	Service1	1	10.0	0.99	6.0
	Service2	2	12.0	0.97	6.0
	Service3	2	8.0	0.98	5.0
	Service4	4	12.0	0.97	7.0
	Service5	2	8.0	0.99	6.0
	Service6	1	10.0	0.99	6.0

TEZ FOTOKOPİSİ İZİN FORMU

ENSTİTÜ

Fen Bilimleri Enstitüsü ☐

Sosyal Bilimler Enstitüsü ☐

Uygulamalı Matematik Enstitüsü ☐

Enformatik Enstitüsü ☒

Deniz Bilimleri Enstitüsü ☐

YAZARIN

Soyadı: Abdyldaeva
Adı : Rahat
Bölümü : Bilişim Sistemleri

TEZİN ADI (İngilizce): QoS-Aware Service Selection for Web Service Composition

TEZİN TÜRÜ : Yüksek Lisans ☒ Doktora ☐

1. Tezimin tamamından kaynak gösterilmek şartıyla fotokopi alınabilir. ☒
2. Tezimin içindekiler sayfası, özet, indeks sayfalarından ve/veya bir bölümünden kaynak gösterilmek şartıyla fotokopi alınabilir. ☐
3. Tezimden bir (1) yıl süreyle fotokopi alınamaz. ☐

TEZİN KÜTÜPHANEYE TESLİM TARİHİ:.....