

# DYNAMIC SELECTION OF REDUNDANT WEB SERVICES

A Thesis Submitted to the  
College of Graduate Studies and Research  
in Partial Fulfillment of the Requirements  
for the degree of Master of Science  
in the Department of Computer Science  
University of Saskatchewan  
Saskatoon

By  
Svetlana Slavova

©Svetlana Slavova, August 2007. All rights reserved.

# PERMISSION TO USE

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science  
176 Thorvaldson Building  
110 Science Place  
University of Saskatchewan  
Saskatoon, Saskatchewan  
Canada  
S7N 5C9

# ABSTRACT

In the domain of Web Services, it is not uncommon to find redundant services that provide functionalities to the clients. Services with the same functionality can be clustered into a group of redundant services. Respectively, if a service offers different functionalities, it belongs to more than one group. Having various Web Services that are able to handle the client's request suggests the necessity of a mechanism that selects the most appropriate Web Service at a given moment of time.

This thesis presents an approach, Virtual Web Services Layer, for dynamic service selection based on virtualization on the server side. It helps managing redundant services in a transparent manner as well as allows adding services to the system at run-time. In addition, the layer assures a level of security since the consumers do not have direct access to the Web Services.

Several selection techniques are applied to increase the performance of the system in terms of load-balancing, dependability, or execution time. The results of the experiments show which selection techniques are appropriate when different QoS criteria of the services are known and how the correctness of this information influences on the decision-making process.

## ACKNOWLEDGEMENTS

I would like to thank my supervisors, Dr. Ralph Deters and Dr. Julita Vassileva, for their supervision and support during my M. Sc. program. Their help is greatly appreciated.

In addition, I would like to thank the other members of my thesis committee: Dr. Gord McCalla, Dr. Derek Eager, Dr. Mark Keil, and Dr. Denard Lynch.

Finally, I would like to thank my parents for their support and love all the time.

To my family.

# CONTENTS

<b>Permission to Use</b>	<b>i</b>
<b>Abstract</b>	<b>ii</b>
<b>Acknowledgements</b>	<b>iii</b>
<b>Contents</b>	<b>v</b>
<b>List of Tables</b>	<b>vii</b>
<b>List of Figures</b>	<b>ix</b>
<b>List of Abbreviations</b>	<b>xi</b>
<b>1 Introduction and Problem definition</b>	<b>1</b>
1.1 Web Services selection . . . . .	2
1.2 Problem definition . . . . .	3
<b>2 Related Work</b>	<b>7</b>
2.1 Selection criteria . . . . .	8
2.2 Reasoning mechanism . . . . .	11
2.3 Selection techniques . . . . .	16
2.4 Enterprise Service Bus . . . . .	18
2.5 Virtualization . . . . .	20
2.6 Summary . . . . .	22
<b>3 Virtual Web Services Layer</b>	<b>23</b>
3.1 Main concepts . . . . .	24
3.2 Architecture overview . . . . .	24
3.3 Example . . . . .	29
3.4 Design implications . . . . .	29
3.4.1 Multiple reasoning mechanisms . . . . .	30
3.4.2 Augmented reasoning mechanism . . . . .	30
3.4.3 Data consistency . . . . .	31
<b>4 Evaluation</b>	<b>32</b>
4.1 Phase 1: VWSL prototype . . . . .	33
4.1.1 Description . . . . .	33
4.1.2 Results . . . . .	33
4.2 Phase 2: Clients' Load Generator and Simulator . . . . .	34
4.2.1 Description . . . . .	34
4.2.2 Experiment setups . . . . .	35
4.2.3 Results . . . . .	36
4.2.4 Phase 1 and phase 2: Conclusions . . . . .	37
4.3 Phase 3: VWSL simulation . . . . .	38
4.3.1 Description . . . . .	38
4.3.2 Phase 3: Main focus . . . . .	43
4.3.3 Experiment setups . . . . .	43
<b>5 Simulation</b>	<b>63</b>

5.1	AnyLogic Enterprise Library . . . . .	64
5.2	Simulation design of the proposed architecture . . . . .	64
5.2.1	Load Generator . . . . .	64
5.2.2	Virtual Web Services Layer . . . . .	67
5.2.3	Web Services . . . . .	71
5.2.4	Requests disposer - object <i>Load Sink</i> . . . . .	71
5.3	Summary . . . . .	72
<b>6</b>	<b>Virtual Web Services Layer Simulation Results</b>	<b>73</b>
6.1	Experiments results . . . . .	73
6.2	Conclusions . . . . .	84
<b>7</b>	<b>Conclusions and Future work</b>	<b>92</b>
7.1	Conclusions . . . . .	92
7.2	Future work . . . . .	94
<b>A</b>	<b>VWSL prototype</b>	<b>100</b>
A.1	Manager of the VWSL prototype . . . . .	100
A.2	Automatically generated Virtual Web Service in Java . . . . .	100
<b>B</b>	<b>Clients Load Generator and Simulator</b>	<b>103</b>

# LIST OF TABLES

2.1	QoS Classifications . . . . .	9
2.2	Selection criteria for dynamic service selection . . . . .	10
2.3	Architectures Comparison . . . . .	15
4.1	Clients' Load Generator and Simulator: experiment results based on RPC communication style (Axis 1) . . . . .	36
4.2	Clients' Load Generator and Simulator: experiment results based on document communication style (Axis 2) . . . . .	37
4.3	Experiments workload . . . . .	40
4.4	Inverting the boolean value of a QoS criterion, depending on the accuracy of the model	42
4.5	Web Services of the simulation . . . . .	47
4.6	Simulation runs . . . . .	47
4.7	Experiment setup: Scenario A, Experiment 1 . . . . .	48
4.8	Experiment setup: Scenario A, Experiment 2 . . . . .	49
4.9	Experiment setup: Scenario A, Experiment 3 . . . . .	50
4.10	Experiment setup: Scenario B1, Experiment 1 . . . . .	52
4.11	Experiment setup: Scenario B1, Experiment 2 . . . . .	52
4.12	Experiment setup: Scenario B1, Experiment 3 . . . . .	53
4.13	Experiment setup: Scenario B1, Experiment 4 . . . . .	53
4.14	Experiment setup: Scenario B1, Experiment 5 . . . . .	54
4.15	Experiment setup: Scenario B1, Experiment 6 . . . . .	54
4.16	Experiment setup: Scenario B1, Experiment 7 . . . . .	55
4.17	Experiment setup: Scenario B1, Experiment 8 . . . . .	55
4.18	Experiment setup: Scenario B1, Experiment 9 . . . . .	56
4.19	Experiment setup: Scenario B1, Experiment 10 . . . . .	56
4.20	Experiment setup: Scenario B2, Experiment 1 . . . . .	58
4.21	Experiment setup: Scenario B2, Experiment 2 . . . . .	58
4.22	Experiment setup: Scenario B2, Experiment 3 . . . . .	59
4.23	Experiment setup: Scenario B2, Experiment 4 . . . . .	59
4.24	Experiment setup: Scenario B2, Experiment 5 . . . . .	60
4.25	Experiment setup: Scenario B2, Experiment 6 . . . . .	60
4.26	Experiment setup: Scenario B2, Experiment 7 . . . . .	61
4.27	Experiment setup: Scenario B2, Experiment 8 . . . . .	61
4.28	Experiment setup: Scenario B2, Experiment 9 . . . . .	62
4.29	Experiment setup: Scenario B2, Experiment 10 . . . . .	62
5.1	Enterprise Library objects [1] used in the simulation . . . . .	65
5.2	Enterprise Library - other components used for the VWSL simulation . . . . .	66
6.1	Simulation results: Ideal setting (Scenario A, Experiment 1) . . . . .	73
6.2	Simulation results: Availability 75% (Scenario A, Experiment 2) . . . . .	75
6.3	Simulation results: Availability 50% (Scenario A, Experiment 3) . . . . .	75
6.4	Simulation results: Availability 100%, random selection for half of the Web Services (Scenario B1, Experiment 1) . . . . .	76
6.5	Simulation results: Availability 75%, random selection for half of the Web Services (Scenario B1, Experiment 2) . . . . .	76
6.6	Simulation results: Availability 50%, random selection for half of the Web Services (Scenario B1, Experiment 3) . . . . .	77
6.7	Simulation results: Availability 75%, Accuracy 100%, the fastest service selection for half of the Web Services (Scenario B1, Experiment 4) . . . . .	77



6.8	Simulation results: Availability 50%, Accuracy 75%, the fastest service selection for half of the Web Services (Scenario B1, Experiment 5)	78
6.9	Simulation results: Availability 75%, Accuracy 100%, load balancing technique for half of the Web Services (Scenario B1, Experiment 6)	78
6.10	Simulation results: Availability 50%, Accuracy 75%, load balancing technique for half of the Web Services (Scenario B1, Experiment 7)	78
6.11	Simulation results: Availability 75%, Accuracy 100%, more accurate selection for half of the Web Services (Scenario B1, Experiment 8)	79
6.12	Simulation results: Availability 50%, Accuracy 75%, more accurate selection for half of the Web Services (Scenario B1, Experiment 9)	79
6.13	Simulation results: Availability 50%, Accuracy 50%, more accurate selection for half of the Web Services (Scenario B1, Experiment 10)	80
6.14	Simulation results: Availability 100%, random selection for all Web Services (Scenario B2, Experiment 1)	80
6.15	Simulation results: Availability 75%, random selection for all Web Services (Scenario B2, Experiment 2)	81
6.16	Simulation results: Availability 50%, random selection for all Web Services (Scenario B2, Experiment 3); * - extended queue	81
6.17	Simulation results: Availability 75%, Accuracy 100%, the fastest service selection for all Web Services (Scenario B2, Experiment 4)	82
6.18	Simulation results: Availability 50%, Accuracy 75%, the fastest service selection for all Web Services (Scenario B2, Experiment 5)	82
6.19	Simulation results: Availability 75%, Accuracy 100%, load balancing technique for all Web Services (Scenario B2, Experiment 6)	83
6.20	Simulation results: Availability 50%, Accuracy 75%, load balancing technique for all Web Services (Scenario B2, Experiment 7); * - extended queue	83
6.21	Simulation results: Availability 75%, Accuracy 100%, more accurate selection technique for all Web Services (Scenario B2, Experiment 8)	84
6.22	Simulation results: Availability 50%, Accuracy 75%, more accurate selection technique for all Web Services (Scenario B2, Experiment 9); * - extended queue	85
6.23	Simulation results: Availability 50%, Accuracy 50%, more accurate selection technique for all Web Services (Scenario B2, Experiment 10)	85
6.24	Simulation results: Selection Techniques Analysis	85

# LIST OF FIGURES

1.1	RPC Style vs. Document Style . . . . .	2
1.2	Web Services Conceptual Model . . . . .	2
1.3	Clustering Web Services into groups by their functionality . . . . .	4
1.4	Group of redundant Web Services . . . . .	4
1.5	Web Service as a "black box" . . . . .	4
1.6	Reasoning Mechanism . . . . .	6
2.1	Overview of the architecture proposed by Liu, Ngu, and Zeng in [2] . . . . .	13
2.2	Overview of the architecture proposed by Day and Deters in [3] . . . . .	14
2.3	Overview of the architecture proposed by Padovitz, Krishnaswamy, and Loke in [4] . . . . .	14
2.4	Overview of the architecture proposed by Maximilien and Singh in [5] . . . . .	15
2.5	Evaluation function proposed by Day and Deters in [3] . . . . .	17
2.6	QoS Ontology proposed by Day and Deters in [3] . . . . .	19
2.7	SOA Architecture . . . . .	19
2.8	SOA and ESB . . . . .	19
2.9	ESB infrastructure given in [6] . . . . .	19
3.1	Client — Group of redundant Web Services . . . . .	23
3.2	Client — VWSL — Group of redundant Web Services . . . . .	25
3.3	Interaction steps: Client — VWSL — Group of redundant Web Services . . . . .	25
3.4	Virtual Web Services Layer . . . . .	26
3.5	Reasoning Mechanism . . . . .	27
3.6	Reasoning Mechanism - selection criteria, model, and selection technique . . . . .	27
3.7	Virtual Web Services Layer - Main Elements . . . . .	28
3.8	Example: Virtual Web Services Layer . . . . .	29
4.1	Clients' Load Generator and Simulator for Web Services . . . . .	35
4.2	Evaluation scenarios A and B . . . . .	38
4.3	Model, represented as a black box with a level of accuracy . . . . .	40
4.4	Example: Execution time intervals, depending on the model accuracy . . . . .	41
4.5	Inverting the boolean value of a QoS criterion, depending on the accuracy of the model . . . . .	41
4.6	Workload levels of the simulation . . . . .	47
5.1	AnyLogic: Hierarchy of active objects presented in [7] . . . . .	63
5.2	Simulation: Clients — VWSL — Web Services . . . . .	66
5.3	Simulation: Load Generator . . . . .	67
5.4	Simulation: Virtual Web Services Layer . . . . .	67
5.5	Simulation: Virtual Web Service . . . . .	68
5.6	Simulation: Object <i>Policy</i> of the Virtual Web Services Layer . . . . .	70
5.7	Simulation: Object <i>Redirect</i> of the Virtual Web Services Layer . . . . .	70
5.8	Simulation: Web Service . . . . .	72
5.9	Simulation: Requests disposer . . . . .	72
6.1	Simulation results: Arrival rate distributions . . . . .	74
6.2	Simulation results: Request size distributions . . . . .	74
6.3	System's Throughput, Low level of Web Services' availability and models accuracy . . . . .	88
6.4	System's Throughput, High level of Web Services' availability and models accuracy . . . . .	88
7.1	Future work: Composite Web Services and Virtual Web Services Layers . . . . .	95
A.1	Virtual Web Services Manager GUI . . . . .	101

A.2	Example of a generated Virtual Web Service in Java . . . . .	102
B.1	Clients Load Generator and Simulator: Main window . . . . .	104
B.2	Clients Load Generator and Simulator: Create single client's behavior . . . . .	104
B.3	Clients Load Generator and Simulator: Simulate multiple clients' behavior . . . . .	105
B.4	Clients Load Generator and Simulator: Log file, created during the clients' simulation	105

## LIST OF ABBREVIATIONS

ACID	Atomicity, Consistency, Isolation, and Durability
API	Application Program Interface
DB	Database
DBMS	Database Management System
ESB	Enterprise Service Bus
HTTP	HyperText Transfer Protocol
LRT	Long Running Transactions
MAS	Multi-Agent System
OGSA	Open Grid Services Architecture
QoS	Quality of Service
RPC	Remote Procedure Call
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
UDDI	Universal Description Discovery and Integration
URL	Uniform Resource Locator
VS	Virtualized Service
VWS	Virtual Web Service
VWSL	Virtual Web Services Layer
VWSM	Virtual Web Services Manager
WS	Web Service
WSDL	Web Services Description Language
XML	Extensible Markup Language

# CHAPTER 1

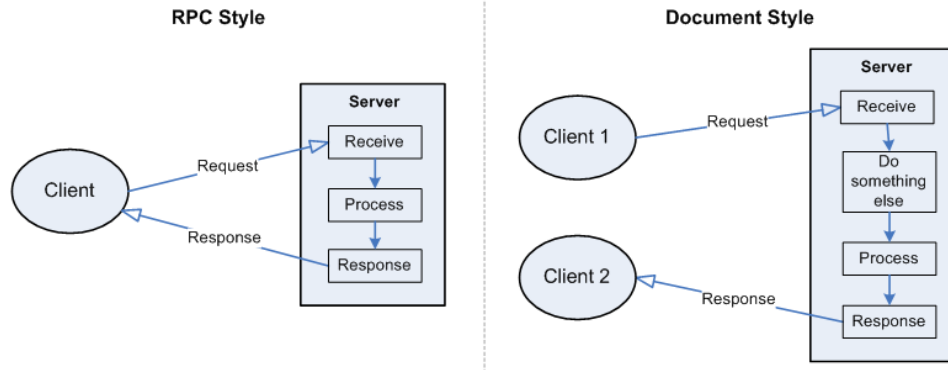
## INTRODUCTION AND PROBLEM DEFINITION

The Service-Oriented Architecture (SOA) allows the development of modular components that can be dynamically discovered and incorporated into platform and language independent applications, using just-in-time integration [8], [9], [10]. The Web Services is the most popular technology which evolves from the SOA paradigm. These services encapsulate different behavior, hide the implementation details, and furthermore, can be “described, published, located, and invoked” over the network [8]. They enable interoperability and inter-process communication, based on standards such as HTTP, XML, Simple Object Access Protocol (SOAP), Web Service Definition Language (WSDL), and Universal Description Discovery and Integration (UDDI) [11].

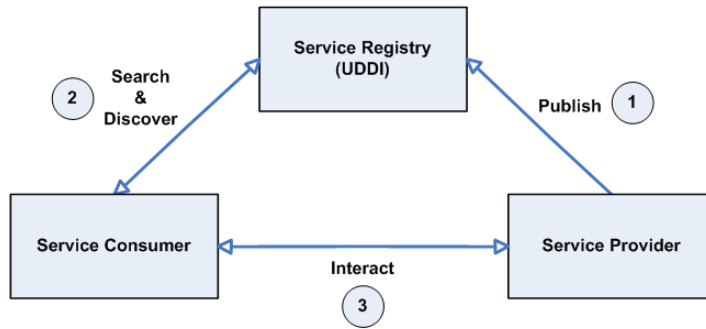
Web Services are described by Web Service Definition Language in XML format [11]. The Web Service description hides the implementation details, but at the same time, gives enough information that is necessary for the service interaction (such as endpoint, location, methods, their input and output parameters, and data types). The XML messaging is realized via SOAP. The standard allows exchanging data among applications in a decentralized, distributed, and heterogeneous environment using HTTP [12]. Each SOAP message is presented as an envelope with two sections - a header and a body. The header contains information about the message itself, and the body consists of data that has to be transferred to the recipient. These standards make the Web Services independent of any programming language, hardware and software platform.

There are two forms of client-server communication - remote procedure call (RPC) and document style [13]. The first one represents a request-response interaction that allows synchronous data exchange. The consumer sends a SOAP message in XML format which contains the call to a specific service and waits for the response. When the request is received by the server, it can be mapped directly to an object that handles the call. Once the result is obtained, it is passed back to the requester. In contrast, the document style messages are “self-contained business documents” that cannot be mapped directly to an object [13]. Usually some pre-processing of the document must be done first when the message is received. This model represents asynchronous data exchange. It is even possible that another client receives the result. The architectural difference between the two messaging styles is presented in figure 1.1.

Web Services can encapsulate a specific task or can be designed as a composition of other



**Figure 1.1:** RPC Style vs. Document Style



**Figure 1.2:** Web Services Conceptual Model

services, representing a complex aggregation. The Web Services conceptual model describes the process of publishing, finding, and binding as well as introduces three participants - service provider, service registry, and service requester [12]. The provider advertises the Web Service and sends its WSDL description to the UDDI directory. The registry offers APIs for publishing and searching of Web Services as well as contains information about all advertised ones. A potential consumer, that could be even another Web Service, queries the registry in order to find a provider that meets its needs and preferences. As a result, the directory returns a list of services to the client that selects a server and starts the interaction. The whole process is shown in figure 1.2.

## 1.1 Web Services selection

According to the Web Services conceptual model (figure 1.2), the appropriate component is chosen among all Web Services that provide a particular functionality. The number of these interchangeable units varies from several to hundreds. In order to deal with distinguishable providers, a set of metrics for service evaluation and comparison should be used [2]. These metrics form an aggregated concept

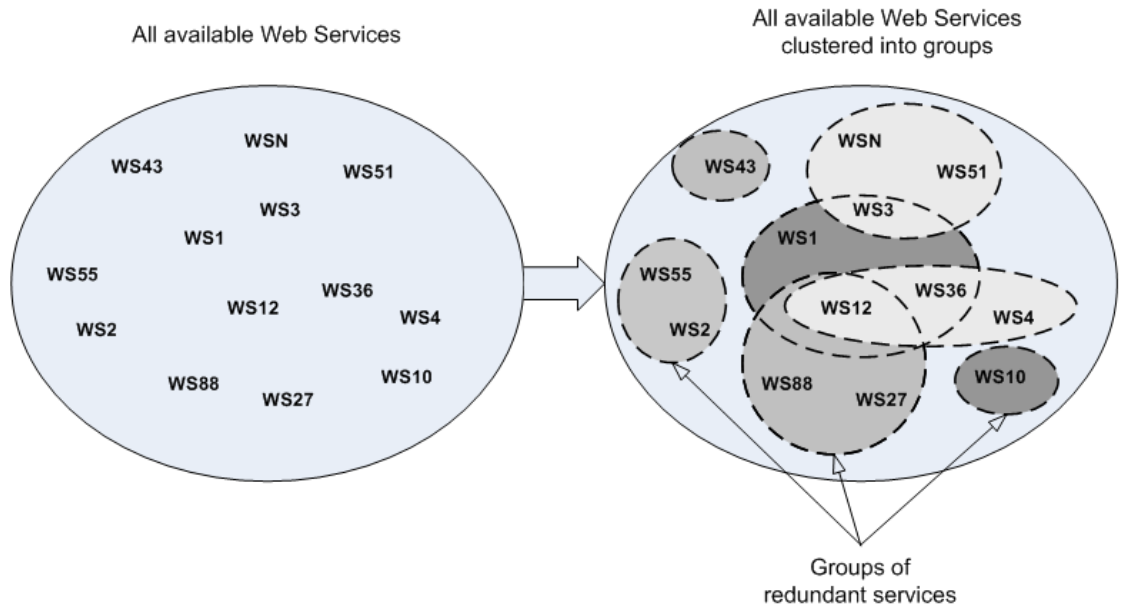
- Quality of Service (QoS), and are used during the selection process of the Web Services. The QoS criteria can be divided into two groups - generic and domain specific. The first group includes criteria that are applicable to all Web Services such as price, execution duration, dependability, failure rate, trust and reputation. The second group refers to characteristics, related to services of a particular field, that can be seen as business criteria. These measures vary from one domain to another. For example, when dealing with online booking, there could be different penalty charges for booking cancellations, according to the selected airline company. When referring to bank accounts, the monthly fees and the interest rates are important metrics and should be taken into consideration at the moment of choosing the most beneficial banking plan.

Once the list of services, offering the same functionality, is available and the criteria are specified, a set of steps (instructions, rules) for Web Service selection should be followed during the decision-making process, in order to determine the component to handle the client's request. Various sets of instructions represent different selection techniques. According to their location, the selection techniques could be on the client or on the server side. In the first case, each consumer should have an additional unit - a reasoning mechanism that decides which component will process the request. The standard Web Services conceptual model (figure 1.2) is based on this type of service selection. An alternative approach has an essential advantage - the appropriate Web Service is chosen in a manner that is transparent to the clients. This method does not require an additional decision-making mechanism for each consumer; just one - on the server side, and thus, does not augment the complexity of the client's logic and does not consume more of the client's resources. On the other hand, the centralized decision-making mechanism can be seen as a single point of failure, which is a potential disadvantage of the approach.

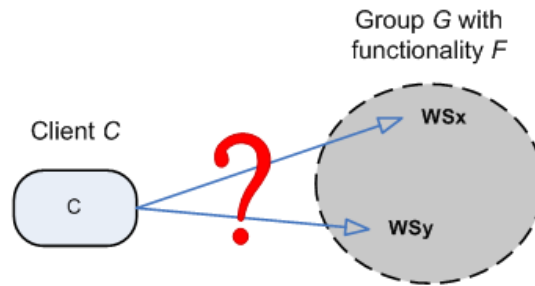
## 1.2 Problem definition

The domain of Web Services assumes redundant components that can perform one or more specific tasks. Usually, the components are provided by different parties and implement different business logic. These redundant services can be clustered into groups, according to the functionality they provide. Furthermore, if a service offers more than one operation, it will belong to more than one group respectively. Figure 1.3 shows how the redundant Web Services can be organized regarding their purpose. Within a cluster, the components are considered interchangeable. For example, given a group  $G$  that contains two Web Services:  $WS_x$  and  $WS_y$ , and each of them offers the same functionality  $F$  (figure 1.4). Then either of the services of the group is able to handle the client's request.

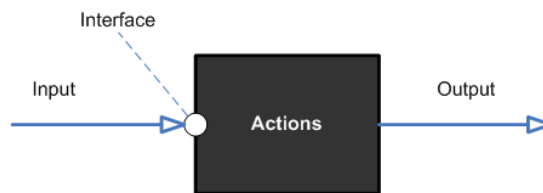
Having a group of interchangeable services gives the opportunity to operate with components that are written in different programming languages, run on various platforms, implement diverse



**Figure 1.3:** Clustering Web Services into groups by their functionality



**Figure 1.4:** Group of redundant Web Services



**Figure 1.5:** Web Service as a "black box"



algorithms, and at the same time provide the same functionality and public interfaces described by the WSDL. From a client's point of view, the services look like "black boxes" that have input, perform a particular behavior, and as a result give some output (figure 1.5).

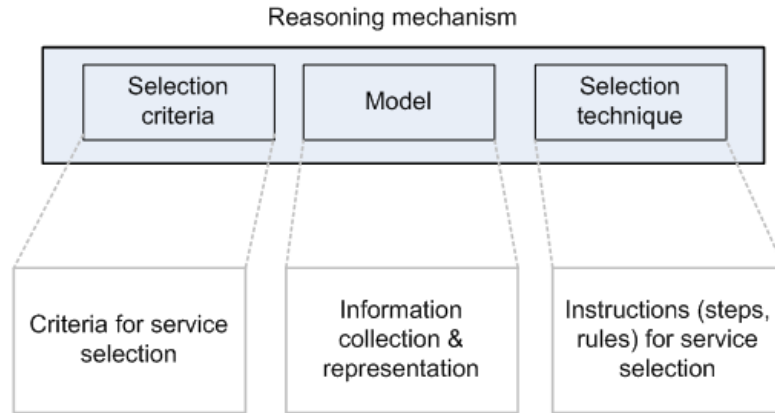
In order to make a distinction between the services which provide the same functionality, selection criteria should be used. They help evaluate the Web Services within a group and choose the component that matches the needs and the preferences of the consumers, while taking into account the abilities of the providers. Web Services can be ranked by the Quality of Service (QoS) they offer. QoS can be seen as an aggregated measure of generic criteria such as availability, reliability, failure rate, trust and reputation, response time, price, and network load and domain specific features [2].

The QoS criteria help in choosing a component to handle the client's request. The selection of the provider is a key point. The choice is crucial for both the server and the requester. For example, if the provider's capacity is reached, an additional request will cause overloading which will influence the performance of the Web Service. The service's overhead reflects on the response times of all requests that are being processed. It is even possible that the augmentation of the execution duration is of significant importance for the client too. An essential question arises: *How to manage redundant Web Services? (1)*

In order to choose a Web Service from a list of redundant services using specific criteria, a reasoning mechanism is needed. It is a decision-making unit that is responsible for the selection of the appropriate component at a given moment of time. It could be located on the client side or on the server side. A reasoning mechanism on the client side increases the consumer's complexity. Moreover, such a unit is required for each client and consumes additional resources. On the other hand, it is possible that different clients have different knowledge about the redundant services, which can reflect on the correct estimation of the consumers regarding the appropriate service. However, if the reasoning mechanism is located on the server side, the consumers will represent thin clients. The key advantage of this scenario is that just one decision-making mechanism is required and the service selection is hidden to the clients. The question (1) above can be extended to: *How to manage redundant Web Services on the server side?*

The reasoning mechanism is responsible for the selection of a Web Service at a particular moment of time. In order to distinguish one service from another using the specified criteria, this unit requires a set of instructions that help evaluate each component and choose the most appropriate one respectively. A set of instructions can be seen as a selection technique. Figure 1.6 shows the main parts of a reasoning mechanism - criteria, model, and selection technique. The model collects information about the participants of the client-server interaction as well as represents it as aggregated measures. Different selection techniques can implement various business logics in order to make a decision. Another key question appears: *What selection techniques should be used? (2)*

The reasoning mechanism takes a decision regarding which Web Service to handle the client's



**Figure 1.6:** Reasoning Mechanism

request. This decision could be static or dynamic [14]. The static approach for publishing, finding, and binding of Web Services requires all end points to be known at design time of the system. The dynamic service invocation is a more flexible technique, since it enables the Web Service selection to be done during the execution time of the application. The main advantage of this method is that it offers more flexibility and adaptability to processes that change continuously in the dynamic environment.

The goal of this work is to develop an approach for dynamic and transparent service selection and to evaluate the proposed architecture in terms of what selection techniques should be applied depending on the QoS of the services as well as the correctness of this information, in order to assure dependability, better response time, load-balancing, or a high level of overall performance. The main focus is on the following questions:

1. *How to manage redundant Web Services on the server side?*
2. *What selection techniques should be used in order to assure transparent and dynamic selection of redundant Web Services?*

The remainder of the thesis is organized as follows: Chapter 2 is an overview of the related work; Chapter 3 discusses the proposed approach - Virtual Web Services Layer. The evaluation of the VWSL architecture is discussed in chapter 4. A simulation environment and a design of a simulation model are presented in chapter 5 and the results of the experiments are discussed in chapter 6. Chapter 7 presents the conclusions of the thesis and some potential future directions.

## CHAPTER 2

### RELATED WORK

Web Services are running in a distributed, dynamic, and unreliable environment, which makes them vulnerable to faults and failures. It is difficult for these components alone to provide a high level of dependability. Dependability is an essential feature of every software system that includes *availability*, *reliability*, *safety*, and *security* [15]:

- *Availability*. Guarantees that the system is up and running but does not assure the correctness of the result;
- *Reliability*. Offers available and properly working components that can get an accurate outcome for a specified amount of time;
- *Safety*. Guarantees that there are no crucial consequences on the environment and the clients of the system;
- *Security*. Assures that only authorized consumers have access to the system and guarantees the system integrity.

There are different techniques for achieving dependability and assuring fault-tolerance respectively. A fault-tolerant application can continue working even in the presence of failure in one or more of its components [15]. Duplication is a common method that protects against deviation of the expected work of engineering systems. It provides a variety of identical or similar components with the same functionality that could be used in parallel or as a backup. This widely used technique for protecting against faults can be realized via *object replication*, *object redundancy*, and/or *object diversity*:

- *Replication*. The replicated system operates with all copies in parallel and the final result depends on the outcome of the majority replicas;
- *Redundancy*. The redundant system consists of several equal units and only one of them is used at a given moment, i.e. only one replica of a replicate group is active;
- *Diversity*. The third type of duplication provides multiple components with different implementation that are executed simultaneously.

The classical research focuses on improving fault-tolerance by adding replicas to the system artificially. These replicas are tightly-coupled and have a point of control. However, the redundancy of web services arises naturally in the domain since it consists of redundant services that are dynamic, distributed, loosely-coupled, and provided by different parties. This does not allow these services to have a single point of control. Therefore, other techniques for managing redundant services should be considered.

This chapter focuses on different approaches that are used by researchers in order to deal with autonomous and loosely-coupled components. It is organized as follows:

- Section 2.1 (Selection criteria) discusses the key QoS metrics that are taken into account in the literature;
- Section 2.2 (Reasoning mechanism) presents various approaches for dynamic selection of loosely-coupled components, as well as their strengths and weaknesses;
- Section 2.3 (Selection techniques) focuses on selection techniques that can be applied in the reasoning mechanism;
- Section 2.4 (Enterprise Service Bus) presents an infrastructure for dynamic service invocation;
- Section 2.5 (Virtualization) discusses a virtualized service-oriented architecture for dynamic service selection;
- Section 2.6 (Summary) presents a summary of the discussed literature.

## 2.1 Selection criteria

Quality of Service is an aggregated metric for describing characteristics of systems in areas, such as networks and distributed systems [16]. Ran [16] discusses that although international quality standards are available for some domains (such as Software Engineering), for others, including Web Services, there is no consensus about the QoS measures that should be taken into account. Researchers propose various ways for grouping the QoS metrics into different categories. Liu, Ngu, and Zeng [2] present the QoS measures as generic or domain specific. The first group includes features, common for every service, such as price, execution time, dependability, and failure rate, whereas the second group refers to criteria, specific to a particular domain. In addition, the paper provides another grouping of the QoS metrics - deterministic and non-deterministic. The deterministic group includes features that are known at the time when the service is invoked (such as price), whereas the non-deterministic group consists of criteria that are uncertain when the Web Service is called (such as response time). Maximilien and Singh [17] divide the QoS attributes into objective and subjective. The former include QoS features such as availability, reliability, and response

QoS Classification		
Category type	QoS category	Examples
According to the area of application of the criteria	Generic	Price, execution time, dependability, failure rate, etc.
	Domain-specific	Penalty rate
According to the knowledge known about the services	Deterministic	Price
	Non-deterministic	Response time
According to the way of gathering of the criteria	Objective	Availability, reliability, response time
	Subjective	Clients' experience
According to the type of the criteria	Run-time related	Execution time, latency, throughput
	Transaction support related	Atomicity, consistency, isolation, durability, LRT
	Management and price related	Regulations, expected/available features, cost, etc.
	Security related	Access, privacy, data encryption, etc.

**Table 2.1:** QoS Classifications

time, and the latter refers to the clients' experience. Ran [16] proposes another classification of the QoS parameters - run-time related, transaction support related, management and price related, and security related. The run-time category contains features that can be evaluated dynamically, such as scalability, capacity, performance (execution time, latency, and throughput), availability, reliability, error rate, degree of correctness, and error handling. Transaction-related QoS include the characteristics of the ACID transactions (atomicity, consistency, isolation, and durability) as well as long running transactions (LRT). The third category, management and cost related QoS, consists of criteria related to standards, regulations, expected features/available features, cost as well as updates in the services. The last group represents security related QoS, such as access related features, privacy, and data encryption. Table 2.1 summarizes the QoS classification.

There are different approaches described in the literature for dynamic selection of Web Services that take into account the QoS criteria discussed above. Maximilien and Singh [5] propose a multi-agent based architecture and the use of the Semantic Web in order to select the best service according to the consumers' preferences. In the paper, trust and reputation are taken into account during the decision process. Liu, Ngu, and Zeng [2] consider these features in their proposed approach as well. In addition, other criteria are discussed too - execution price, duration, transactions support, compensation and penalty rate. The authors of [2] suggest an open, fair, and dynamic framework that evaluates the QoS of the available Web Services by using clients' feedback and monitoring.

QoS Criteria	Papers					
	Maximilien & Singh	Liu, Ngu & Zeng	Day & Deters	Ran	Ludwig & Reyhani	Padovitz, Krishnaswamy & Loke
Availability	No	No	Yes	Yes	Yes	Yes
Reliability	No	No	Yes	Yes	Yes	Yes
Scalability	No	No	Yes	Yes	No	No
Execution time	No	Yes	Yes	Yes	Yes	Yes
Trust & Reputation	Yes	Yes	No	No	Yes	No
Network & Machine Load	No	No	Yes	No	No	Yes
Execution price	No	Yes	No	Yes	Yes	No
Transactions	No	Yes	No	Yes	No	No
Compensation rate	No	Yes	No	No	No	No
Penalty rate	No	Yes	No	No	No	No
Security	No	No	No	Yes	No	No
Clients' experience / feedback	No	Yes	Yes	No	Yes	No

**Table 2.2:** Selection criteria for dynamic service selection

The solution, suggested by Day and Deters in [3] offers another technique that deals with Web Services selection - a Semantic Web approach that takes into consideration generic criteria - availability, reliability, and execution time of the services, and as in [2], relies on the past experience of different consumers. Padovitz, Krishnaswamy, and Loke [4] talk about availability, reliability, execution time, and service load as criteria that should be taken into consideration when selecting Web Services at run-time.

Ran [16] extends the standard Web Services conceptual model (figure 1.2), by adding a new component to the current architecture - a QoS component. This unit contains QoS metrics of the published services, such as scalability, performance (response time, latency, and throughput), reliability, and availability.

Ludwig and Reyhani [18] apply QoS metrics in Grid computing in order to assure dynamic service selection. They take into consideration execution duration, execution price, reputation, reliability, and availability. The services are ranked according to the QoS criteria and the service is chosen by a matchmaking or a heuristic algorithm.

When dealing with dynamic components like Web Services, it is hard to observe all of their possible features. The researchers focus mostly on some generic criteria, since they can be applied to any service. Availability, reliability, and response time are the most popular ones, as they provide an overview of the services and at the same time they can be evaluated relatively easily. A summary of the reviewed literature that uses criteria for dynamic service selection, is shown in table 2.2. The papers are compared according to some important criteria of the distributed applications, which have been taken into account in the discussed works.

## 2.2 Reasoning mechanism

The selection criteria are one aspect of the decision-making process. Another aspect is the mechanism for Web Service selection. Liu, Ngu, and Zeng [2] propose a framework that evaluates services according to some generic (price, response time, and reputation) and domain specific criteria (business related - transactions support, compensation and penalty rates). The architecture is flexible and allows extension of the QoS parameters according to the characteristics of the system. The service related attributes are given by the providers, evaluated by the consumers via monitoring, or gathered through clients' feedbacks, and are stored in a database. The reasoning mechanism in this approach computes the QoS of the Web Services, ranks them, and selects the most appropriate one. The bottleneck of the approach is the dependency on the consumers to give regular feedback about their past experience with the Web Services. An overview of the architecture is shown in figure 2.1.

In [3], it is the client's responsibility to detect the most appropriate service, depending on its configuration and requirements. An overview of the architecture is shown in figure 2.2. It is based on the Semantic Web and uses a service evaluation function and an expert system during the Web Services selection process. However, it does not provide service transparency, since the consumers decide which Web Service should be invoked. In addition, it relies on the clients participation to report experience with the providers. Furthermore, each client requires a reasoning mechanism which augments its complexity. In some cases, the approach can be valuable, because the consumers would be able to choose the best Web Service themselves, according to their needs and preferences, but in other cases the clients cannot be involved in this process.

Another approach for Web Services selection is presented by Padovitz, Krishnaswamy, and Loke in [4]. An overview of the architecture is shown in figure 2.3. The client chooses at run-time which service to handle the request. In order to make a decision, service related information must be available to the consumer. The authors present three techniques for gathering this data:

- *Remote procedure calls (RPC) method.* It consists of the following steps:
  1. The client sends a call (a remote procedure call) to all hosts asking for different information about the services. The requested data depends on particular criteria and preferences of the consumers;
  2. Each host sends back a response that contains knowledge about its services;
  3. The client evaluates the received data.

The disadvantage of this method for gathering information about the services is the large number of request-response messages that are exchanged between the clients and the providers. It leads to an augmentation of the system's overhead when the amount of hosts increases.

Furthermore, if the Web Services are compositions of services, the client will not be able to collect any information in depth about the internal services.

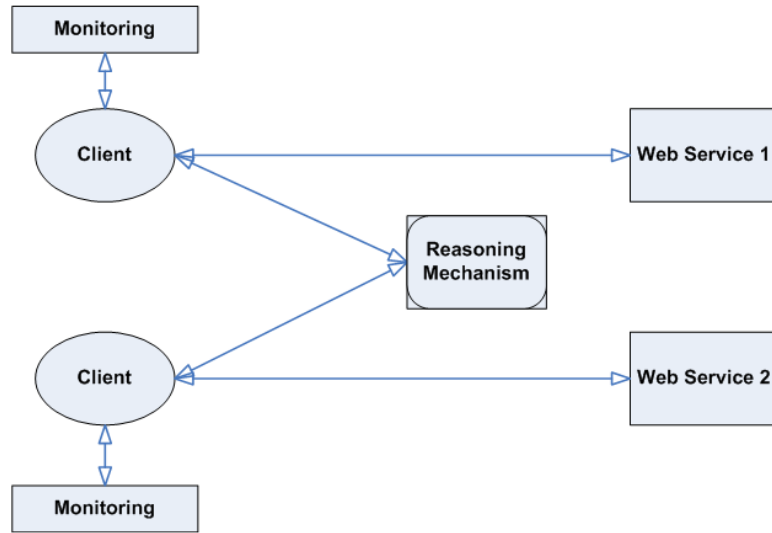
- *Mobile agents.* The mobile agents method is a more flexible approach that allows the client to control the behavior of the agent in depth when dealing with composite Web Services. The mobile agent, sent to the host by the consumer, queries the server about the services it provides, and if necessary, goes to another host on a deeper level to get some additional information. This method for collecting data is appropriate in less reliable environments because the remote communication is significantly reduced. However, it is more complex, since the mobile agents need to react to the changing environment and need to be able to reach back to the client using even another route.
- *Circulating mobile agents.* This approach is beneficial when up-to-date information about the services is needed regularly. The idea is the following: There are several agents in the system circulating from host to host. They gather information about the Web Services and report it periodically to the client. In this case, some data could be collected without being evaluated if a fresher update comes to the requester.

The reasoning approach in [4] as well as the mechanisms provided in [2] and [3] require the clients' participation - to gather run-time information related to the services, to share information about the interactions with the services, to evaluate the QoS of the Web Services, and even to take a decision which service matches their needs and preferences. These techniques require each consumer to contain an intelligent mechanism which makes it more complex. On another hand, if the clients are responsible for the decision process, it might lead to a situation where multiple consumers have chosen the same provider and service overload has occurred. Furthermore, if the requesters have a local knowledge about the services, it is even possible that the chosen components are not the most appropriate ones. This can happen if the clients collect information only about their interactions with the services and does not have access to all interactions in the system.

A multi-agent approach for dynamic service selection that does not increase the consumers' complexity is presented in [5] by Maximilien and Singh. An overview of the architecture is shown in figure 2.4. The authors propose an architecture that consists of interacting agents. Every Web Service has an autonomous agent attached to it, which offers the same interface as the service. This interface allows the consumer and the service agent to communicate. In addition, the agents interact and share information through agencies. The agencies contain data about the interactions between the clients and the services which is used during the Web Services selection process. Furthermore, the reasoning mechanism of the approach is based on ontology and matching algorithm and is used to match semantically clients to services, in a manner transparent to the consumer.

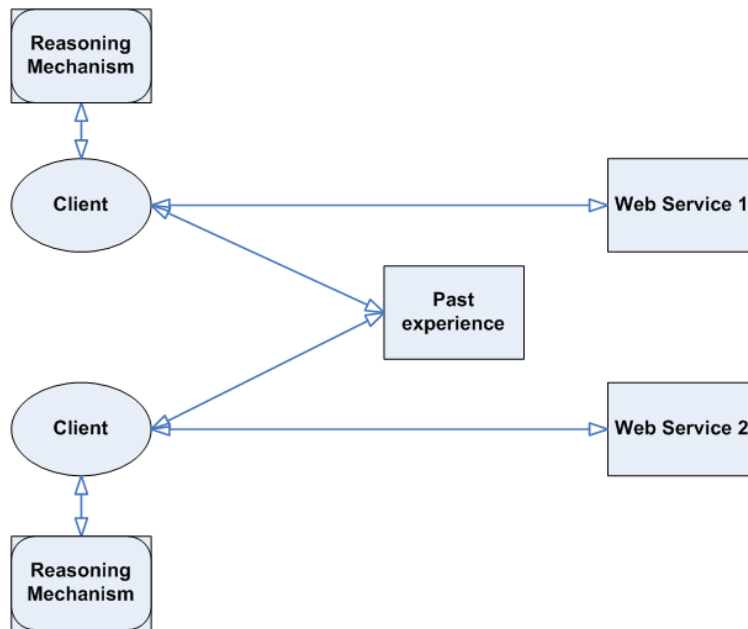
The discussed approaches are presented in table 2.3. A review of different selection techniques



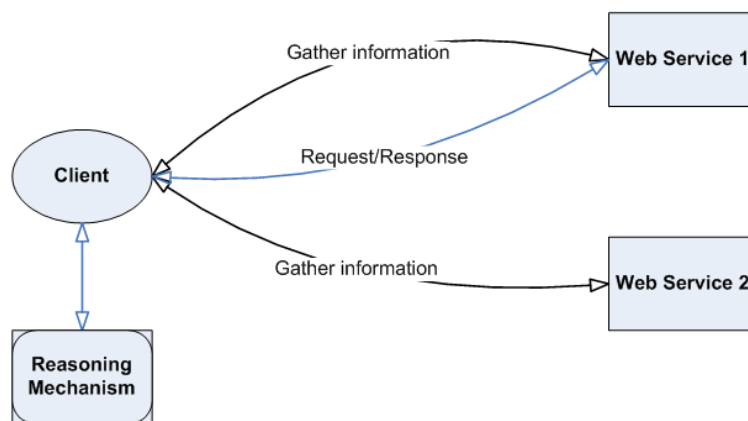


**Figure 2.1:** Overview of the architecture proposed by Liu, Ngu, and Zeng in [2]

is provided in the next section.



**Figure 2.2:** Overview of the architecture proposed by Day and Deters in [3]



**Figure 2.3:** Overview of the architecture proposed by Padovitz, Krishnaswamy, and Loke in [4]

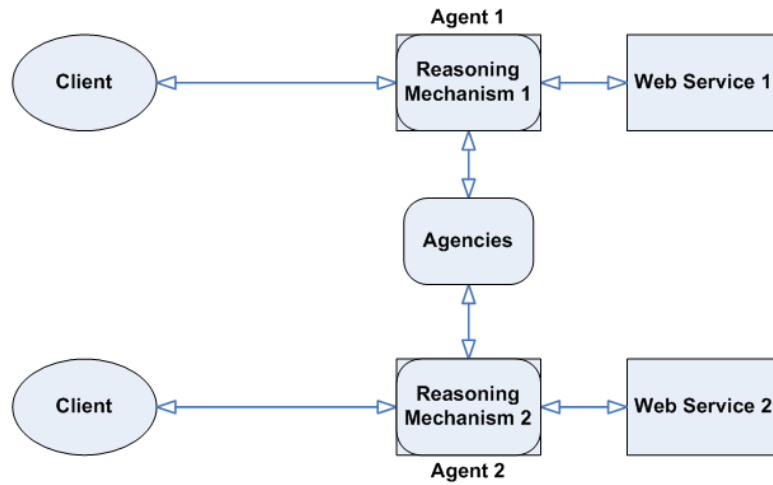


Figure 2.4: Overview of the architecture proposed by Maximilien and Singh in [5]

	Papers			
	Liu, Ngu & Zeng	Day & Deters	Maximilien & Singh	Padovitz, Krishnaswamy & Loke
<b>Dynamic Service Selection</b>	Yes	Yes	Yes	Yes
<b>Selection transparency</b>	No	No	Yes	No
<b>Clients involved in QoS computation</b>	No	Yes	No	Yes
<b>Clients involved in Web Services selection</b>	No	Yes	No	Yes
<b>Clients give feedback</b>	Yes	Yes	No	No
<b>Reasoning mechanism</b>	- QoS computation - QoS ranking - DB storage	- Semantic Web - Services evaluation function - Expert systems - DB storage	- Agents - Semantic Web - Matching algorithm - DB storage	- RPC / - Mobile agents - Ranking - DB storage

Table 2.3: Architectures Comparison

## 2.3 Selection techniques

A selection technique consists of a set of instructions (steps, rules) that should be followed in order to select a Web Service at run-time, depending on the information that is available to the system. Selection techniques can be divided into several groups, according to their nature as well as to the information that they use about the services. Usually, the approaches are combined together to achieve a more powerful reasoning mechanism. However, in this work the groups of selection techniques are reviewed separately in order to distinguish the principles of the techniques from one another:

- *Selection technique that is not based on any information about the services.* If there is no knowledge about the behavior of the Web Services, random service selection or round-robin techniques can be applied. This is the simplest technique since it does not depend on any QoS metrics of the services. In [19], Fedoruk and Deters apply this technique for improving fault-tolerance in multi-agent systems (MAS) - the first found agent of a group of redundant agents is selected.
- *Selection technique that assures system dependability.* This approach takes into consideration the availability of redundant Web Services. If a service is not available at a given moment of time, it cannot be selected. This technique relies on one QoS metric - services availability. If this criterion is known for all Web Services of the system, a high level of dependability can be achieved.
- *Selection technique that assures better response time.* This approach relies on the speed of the Web Services. It assures that the fastest component is chosen every time.
- *Selection technique that assures a high level of dependability and better response time.* This selection technique is a combination of the previous two techniques. It selects the fastest available service at the moment.
- *Selection technique that assures load balancing.* “Load balancing uses multiple hardware devices to spread work around and thereby speed performance” [20]. When dealing with groups of redundant Web Services, this approach refers to the concept of balancing the clients’ requests between the interchangeable services. The load of the components must be known in order to select the appropriate provider. There are various ways for keeping track of the servers’ load - by monitoring the number of the calls that have been handled by each component, or as discussed in [21], the load can be detected according to the execution time that each service provides when processing a particular request. However, if the load of the services is not available, a simple round-robin technique can be used, i.e. no criteria are taken into account.



**Figure 2.5:** Evaluation function proposed by Day and Deters in [3]

- *Selection techniques that take into account past and current information about the services* ([22], [4], [3]):
  - *Parallel invocation.* All Web Services of a particular group are invoked simultaneously. The result, received from the service which answers first, is considered. The responses, received from the other services, are ignored;
  - *Probe.* A “probe” request is sent simultaneously to all Web Services of a particular group. The service that answers first is selected for the next interaction;
  - *Best last.* The service with the lowest response time of the last interaction is selected. This technique requires history information about the client-service interactions to be collected;
  - *Best median.* This technique takes into account the last  $k$  client-service interactions for each Web Service of the group. The selected service has the median response time among the considered services;
  - *Best state information.* At run-time, each component is questioned for its state information, such as response time, availability, and QoS. Depending on the obtained responses and the requirements of the client, the most appropriate Web Service is selected.
  - *Evaluation function.* Day and Deters [3] use an evaluation function in order to determine the most appropriate service at run-time. The function takes as input the data about the redundant components, estimates the information, and returns “the best service” as a result (figure 2.5). The authors present two functions:
    - \* Evaluation function that is based on weights in order to distinguish the services according to availability, reliability, and the execution time. The function could be extended to take into account trust and reputation as well as the geographic location of the components.
    - \* Evaluation function that is based on more data - QoS about the services (availability, reliability, and execution time) and information about the client (such as CPU in use, memory in use, and running processes).

- *Semantic selection technique.* This is a complex approach, based on the Semantic Web, that takes into consideration several QoS metrics. Semantic Web is a way for describing the meaning of the data in a machine-understandable manner [23]. Different semantic markup languages, schemas, and ontologies have been developed (such as OWL-S, DAML+OIL, and RDF), in order to make the computers able to understand the data in a way that is close to human perception. The schemas describe how the information should be presented, whereas the ontologies show the relationships between the terms.

QoS ontology is used in [5], [24], and [3]. Maximilien and Singh ([5], [24]) present a QoS specification in XML format that is able to match Web Services to clients, using a matching algorithm. The QoS of the services and the requirements of the consumers are described with an ontology, containing the following key characteristics:

- Qualities of the services, such as response time, availability, reliability, failure rate, etc;
- Relationships between the QoS parameters. The relationships show the direction and the strength of the qualities. The direction could be *opposite*, *parallel*, *independent*, or *unknown*; whereas the strength could be defined as *weak*, *mild*, *strong*, or *none*. For example, the relationship between the execution time and the throughput can be seen as negative and strong;
- Aggregate quality represents a combination of qualities that can be grouped as a separate quality.

Day and Deters [3] use expert systems and a much simpler ontology during the Web Services selection process. Their ontology represents a hierarchy of QoS parameters, shown in figure 2.6. The authors divide the QoS into two groups - criteria specific to the service, and criteria specific to the request. The former refers to generic (availability and reliability) and domain-oriented attributes, whereas the latter relates to time needed by the service to handle the request.

## 2.4 Enterprise Service Bus

According to the Service-Oriented Architecture (figure 2.7 [6]), heterogeneous systems are able to communicate in a transparent manner using middleware. Enterprise Service Bus (ESB) [6] is an infrastructure that allows such communication (figure 2.8, figure 2.9). It consists of a central point of control and a distributed infrastructure (run-time engines). The run-time engines are responsible for data transformation, routing, security, etc.

The Enterprise Service Bus is a framework that is able to connect services, regardless of their operating system, programming language, and message formats [10], [25]. It is a “centralized, scal-

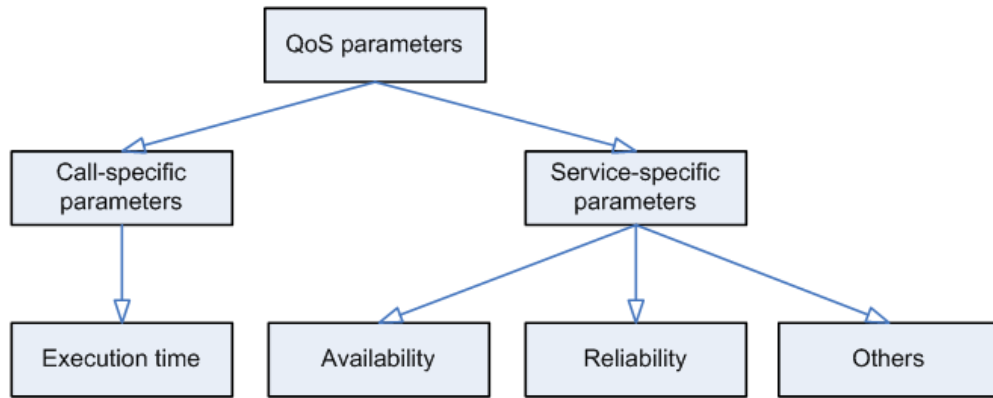


Figure 2.6: QoS Ontology proposed by Day and Deters in [3]



Figure 2.7: SOA Architecture



Figure 2.8: SOA and ESB

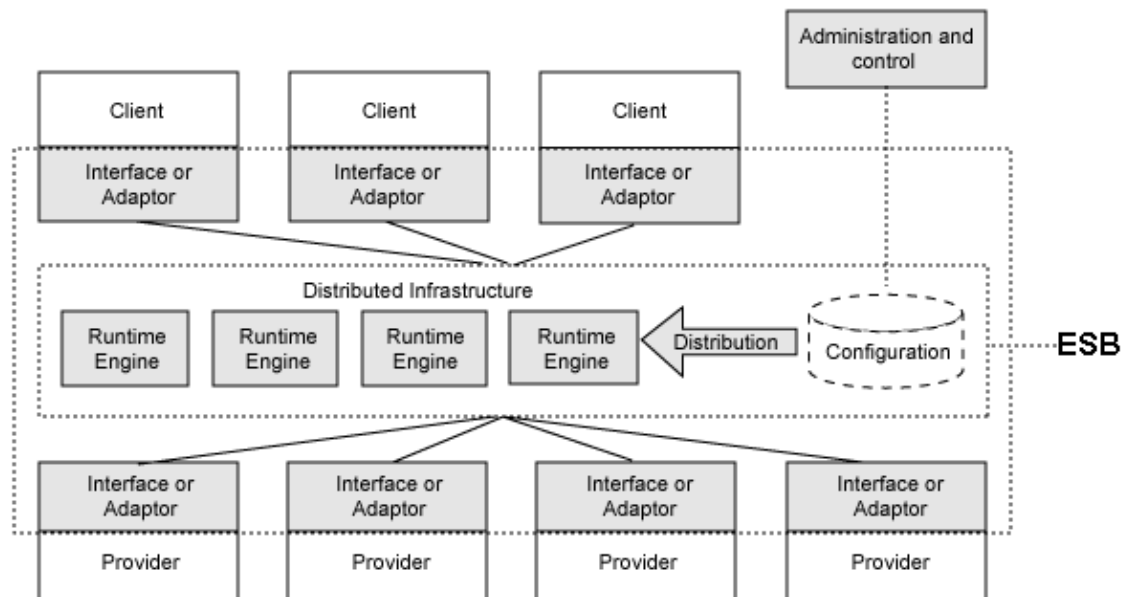


Figure 2.9: ESB infrastructure given in [6]

able, fault-tolerant, service-messaging” ([25]) infrastructure that allows the following characteristics ([6]):

- *Communication.* The message exchange is secure and can be realized synchronously and asynchronously; routes clients’ requests to the appropriate service providers;
- *Integration.* Supports DB, service mapping, different application server environments (such as .NET and J2EE), programming languages; heterogeneous services based on various protocols could communicate in a transparent manner;
- *Security.* Supports security standards as well as authentication, authorization, and confidentiality; deals with failures;
- *Message processing.* Responsible for data transformation, validation, mapping;
- *Interaction.* Responsible for the interactions between the services; Supports interchangeable services;
- *Transactions.* Supports transactions such as ACID and compensation;
- *QoS.* Takes into account QoS parameters such as availability, reliability, failure rate, throughput, etc;
- *Management.* Responsible for monitoring, logging, measuring data;
- *Reasoning.* Rules for service selection.

The Enterprise Service Bus provides the possibility of dynamic service selection and invocation that is transparent to the clients. The consumers do not have to deal with monitoring, QoS evaluation, and the whole decision process is hidden by the ESB. This is a possible solution to the problem of managing redundant services.

## 2.5 Virtualization

Grid is a technology that operates with a virtual architecture, called *virtual organization*, which is able to solve large-scale computational problems, using geographically distributed resources (such as computers, disk storages, and software programs), connected by a network [26]. “The vision behind the Grid is to supply computing and data resources over the Internet seamlessly, transparently and dynamically when needed.” [18]. The Grid technology and the Service-Oriented Architecture lead to the appearance of Open Grid Services Architecture (OGSA), which allows communication between heterogeneous resources (referred as services) through message exchange. The virtual organization of the services allows dynamic service invocation, which is transparent to the client.



A virtualized service-oriented architecture, based on Grid technology and SOA, is presented in [27], in order to facilitate selection of redundant services. The services that provide the same functionality form a *virtualized service* (VS), which is accessible by the client. The advantages of the approach are as follows:

- The service selection is done at run-time, according to the applied selection technique;
- A service of a VS can be replaced, without affecting the proper working of the system;
- Transparent service invocation;
- New services can be deployed on demand, in order to achieve load-balancing and to improve dependability.

This architecture is an applicable technique for managing redundant Web Services, but allows only limited system extensibility. The services are associated with one or more virtualized services at run-time, but all VSs have to be specified at design-time of the system. This does not allow adding virtualized services dynamically.

## 2.6 Summary

Redundancy of services can occur naturally or artificially. Duplication techniques add redundant components to the system artificially and allow control of the replicas. However, when the redundancy has occurred naturally in the domain (for example the domain of Web Services), the redundant components are different by their nature, behavior, and do not have any points of control.

Web Services can be ranked by the quality of service (QoS) they provide. QoS can be seen as an aggregated measure of different features such as availability, reliability, failure rate, trust and reputation, response time, price, and network load. A key issue is which QoS criteria should be taken into account during the decision-making process. According to [2], availability, reliability, and response time of the services are the most typical QoS parameters. They can be applied to every service, can be monitored and evaluated relatively easy, and at the same time they provide an overall overview of the services.

Different reasoning mechanisms for service selection are discussed in the literature. They could be located on the client or on the server side. If the reasoning mechanism is on the client side, it is very likely that one Web Service is used more frequently than others. This may affect the performance of the service and does not allow load balancing techniques to be applied easily. Furthermore, if the client has only local knowledge about the services, the most appropriate components may not be selected every time.

A reasoning mechanism on the server side is a much more powerful technique. It enables dynamic and transparent selection of services. The consumers represent thin clients and the whole business logic is hidden to the requesters.

The approach, provided in this thesis, shows how to manage redundant Web Services on the server side. Its architecture is described in the next chapter.

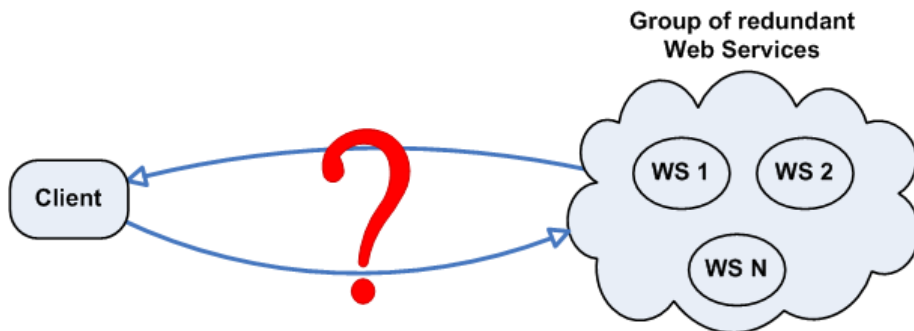
# CHAPTER 3

## VIRTUAL WEB SERVICES LAYER

In the domain of Web Services, it is not uncommon to find redundant services that provide functionalities to the clients. According to the Web Services conceptual model (discussed in chapter 1), the client receives a list of services from the UDDI, selects one, and starts an interaction with the service to process the request (figure 3.1).

As seen in chapter 2, service selection is an important process and various techniques have been proposed. In this chapter, another approach for dynamic service selection and invocation is introduced, which has the following advantages in comparison with previous approaches:

- It provides location and replication transparency of the Web Services;
- It hides the system's complexity from the clients;
- It provides a transparent service selection from the client's point of view;
- It assures a level of security, since the clients do not have direct access to the Web Services;
- As in [27], it is based on the virtualization of Web Services, but in addition, it allows adding virtual services to the system at run-time;
- The architecture is flexible, since various reasoning mechanisms can be applied, without modifying the virtual services;
- Load-balancing and a high level of dependability can be achieved.



**Figure 3.1:** Client — Group of redundant Web Services

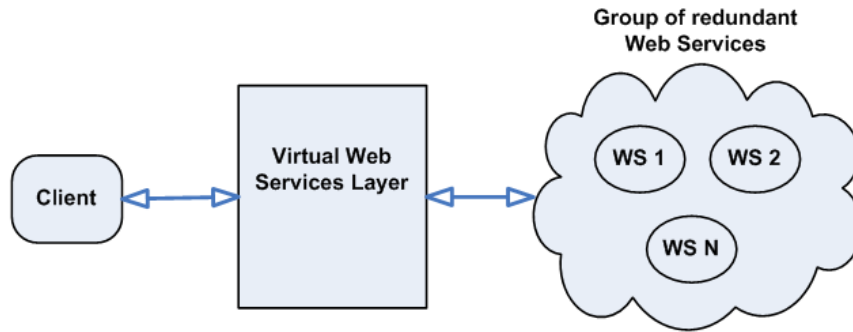
## 3.1 Main concepts

- **Definition 1.** *Web Service.* A component, written in a particular programming language and connected in a network, providing one or more functionalities through message exchange that follows the SOAP standard. The Web Service is identified by its service description (WSDL), which represents the name (URL) of the service as well as the service's interface - methods, input and output parameters.
- **Definition 2.** *Group of redundant Web Services.* Web Services, which provide the same functionality and are accessible via the same interface, but may implement different business logic, form a group of redundant services. If a Web Service offers more than one functionality, then it belongs to more than one group.
- **Definition 3.** *Virtual Web Service (VWS).* A Web Service, which represents a group of redundant services and provides the same interface as the Web Services, is considered a Virtual Web Service. When a client's request is received, it selects a Web Service from the group, according to the applied reasoning mechanism.
- **Definition 4.** *Virtual Web Services Layer (VWSL).* Architecture, located on the server side, which is responsible for the dynamic selection and invocation of Web Services. It consists of decentralized Virtual Web Services, a reasoning mechanism, and a Virtual Web Services Manager.
- **Definition 5.** *Virtual Web Services Manager (VWSM).* An application that manages the Virtual Web Services Layer at run-time. It creates Virtual Web Services, updates the groups of redundant services, and specifies the reasoning mechanism in use.

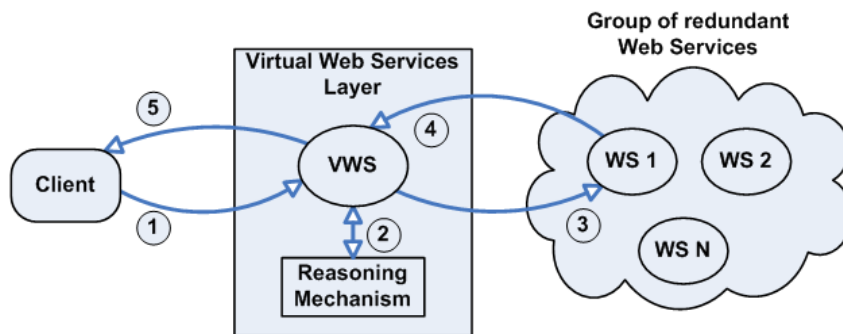
## 3.2 Architecture overview

The approach, presented in this thesis, offers a transparent way for managing redundant Web Services on the server side, by using dynamic service invocation. A Virtual Web Services Layer (figure 3.2) is introduced between the client that requires a particular functionality and the Web Services that provide that functionality. It is based on a reasoning mechanism that decides at run-time which component to invoke as a result of the consumer's call. The layer provides location and replication transparency of the services and hides the system's complexity from the clients. At the same time it assures a level of fault-tolerance which increases the availability and the reliability of the system.

Transparency is achieved by Virtual Web Services (VWSs) that are used as an entry point to the groups of redundant services. Each VWS is generated automatically when a new group is added



**Figure 3.2:** Client — VWSL — Group of redundant Web Services

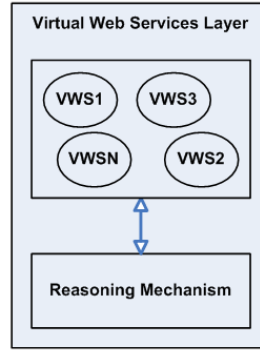


**Figure 3.3:** Interaction steps: Client — VWSL — Group of redundant Web Services

to the system. It provides the same interface (methods, input and output parameters, data types) as the real Web Services but does not implement the same business logic. Instead, the VWS uses a reasoning mechanism that determines which Web Service of the group is the most appropriate one, depending on the selection criteria and the applied selection technique.

Figure 3.3 shows the interaction between a client, the VWSL, and a group of redundant Web Services. The initiator sends a request to the Virtual Web Services Layer, more precisely, to a Virtual Web Service. From the client's perspective, the VWS is the service that handles the request. However, the VWS behaves as a smart proxy that selects at run-time the real Web Service and redirects the call to it. The proxy uses the reasoning mechanism of the layer in order to choose the most appropriate component of the redundant group. Once the service is selected, the VWS routes the consumer's request. When the operation is processed, the result is sent back to the Virtual Web Service. Then, the VWS transfers the response to the client.

The consumer and the Web Service never interact directly. From the client's point of view, the VWS is the component that handles the call, whereas from Web Service's perspective, the VWS behaves as a client.



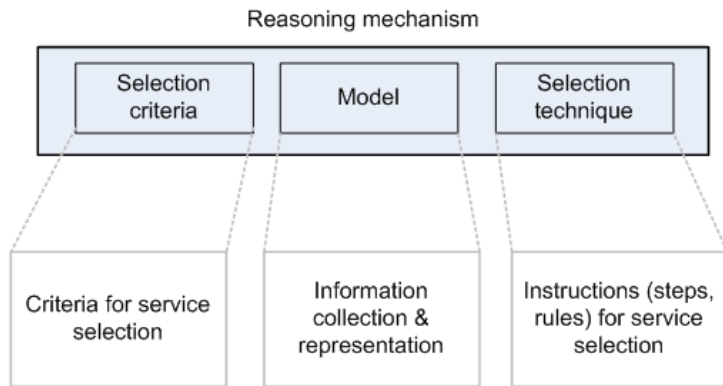
**Figure 3.4:** Virtual Web Services Layer

The Virtual Web Services Layer consists of logically independent units with a particular meaning and purpose. It contains two main elements (figure 3.4):

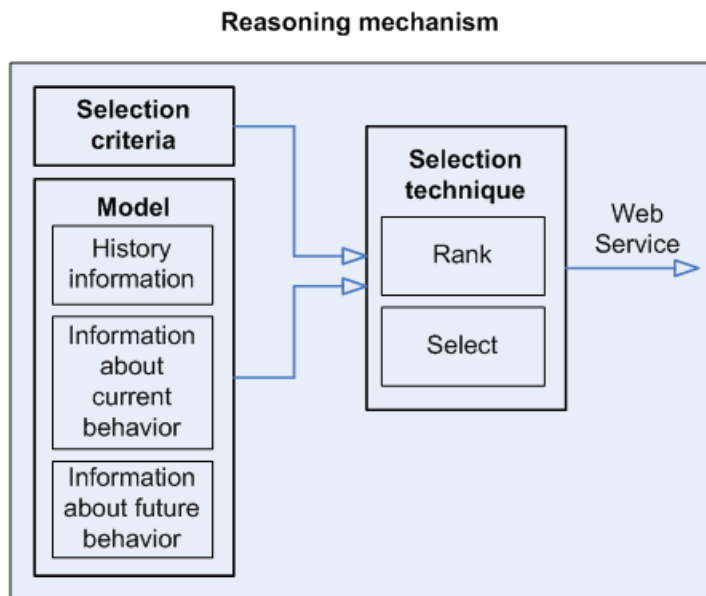
- *Virtual Web Services.* Their goal is to redirect the clients' requests, using a dynamic service invocation. The number of groups containing redundant Web Services is equal to the number of Virtual Web Services. A VWS is designed as a Web Service. It uses a reasoning mechanism in order to find the service to handle the client's call at a particular moment of time;
- *Reasoning mechanism.* This component, based on selection criteria, model, and selection technique (figure 3.5), is responsible for the dynamic service selection. The criteria specify which QoS parameters are taken into account during the decision-making process. The proposed architecture allows various generic and domain specific selection criteria to be used. The model is in charge of the collection of the QoS attributes and the interactions between the clients and the services. The model might represent data about past, present, and/or future behavior of the consumers and the providers. Knowing or calculating precisely the behavior of the clients and the services would allow a more accurate selection. The third element of the mechanism - the selection technique, specifies a set of instructions (steps, rules) that are used to rank the Web Services according to the criteria and to select the most appropriate service. The elements of the reasoning mechanism are shown in figure 3.6.

In order to design the VWSL as a flexible unit that can be managed easily, one more element is added to the layer - a Manager. It represents a user-friendly client-server application that consists of a Virtual Web Services Manager (VWSM) and a VWSM-Client. The former generates Virtual Web Services and manages the settings of the reasoning mechanism (such as selection criteria, models, and selection techniques). The latter is a client that specifies the behavior of the VWSM. The general architecture of the Virtual Web Services Layer and the relations between the elements are presented in figure 3.7.

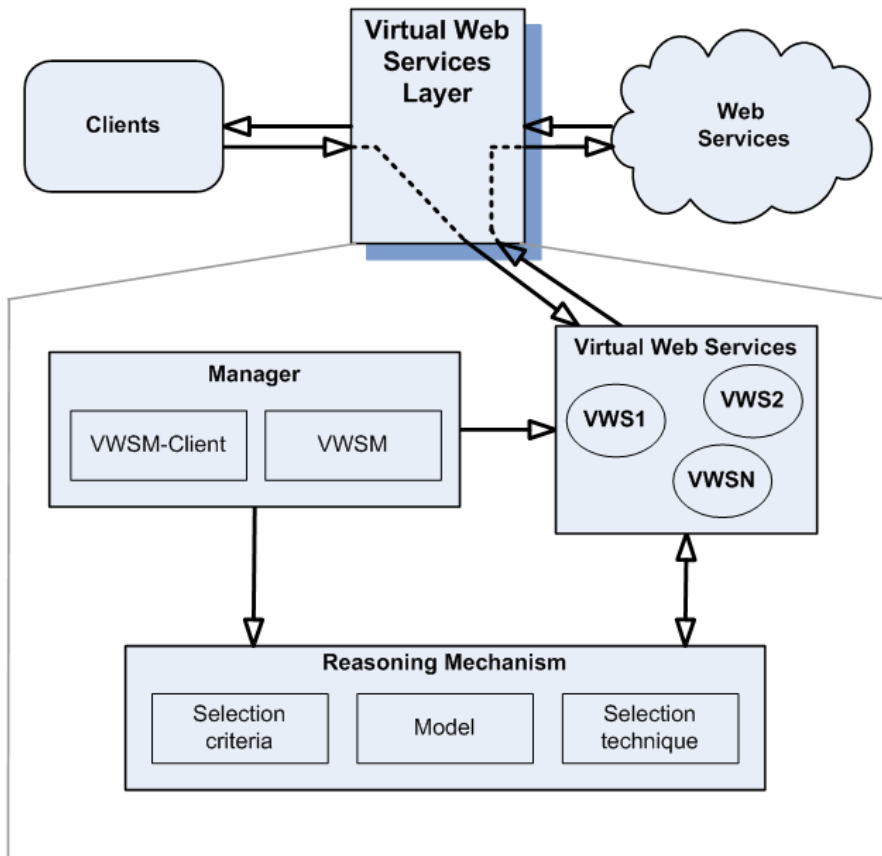
The described architecture is loosely-coupled. It allows the exchange of one element with another



**Figure 3.5:** Reasoning Mechanism

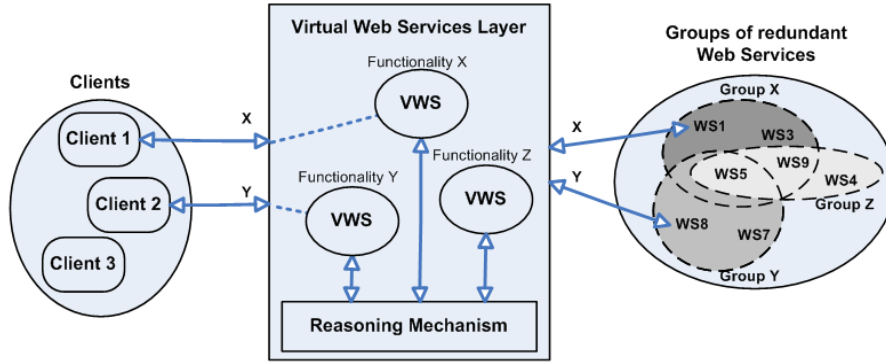


**Figure 3.6:** Reasoning Mechanism - selection criteria, model, and selection technique



**Figure 3.7:** Virtual Web Services Layer - Main Elements





**Figure 3.8:** Example: Virtual Web Services Layer

without modifying the other units of the layer. For example, the reasoning mechanism can be replaced with a different reasoning mechanism that provides the same interface. This change would not affect the Virtual Web Services and the Manager. Depending on the data given to the system about past or future behavior of the Web Services and their clients, various data models and selection techniques can be applied.

### 3.3 Example

Figure 3.8 represents a specific example of the proposed architecture. It consists of three clients and several Web Services. The services are clustered into groups depending on their functionality. For example, Web Services 1, 3, 5, and 9 are in group X, since they offer functionality X. There is a Virtual Web Service that corresponds to each group of redundant services.

When client 1 needs to send a request, it sets the functionality of the request and sends it to the Virtual Web Service that is responsible for the specific functionality. The VWS interacts with the reasoning mechanism in order to find the most appropriate Web Service of the group. Once the service is selected, the request is forwarded to it. Finally, when the result is generated, it is passed to the virtual layer which sends it back to the client.

### 3.4 Design implications

The presented Virtual Web Services Layer contains centralized and decentralized elements. The Virtual Web Services are decentralized and do not influence the proper work of the whole architecture. On the other hand, the reasoning mechanism, a key part of the architecture, is represented as a centralized unit. It means that a failure in the reasoning mechanism will lead to a failure of the whole system.

### 3.4.1 Multiple reasoning mechanisms

The design might be modified in a way that makes the architecture more scalable and without a single point of failure, by specifying a reasoning mechanism to each Virtual Web Service. The trade-offs of this approach are the following:

- Advantages:
  - The Virtual Web Services Layer is decentralized, which assures that there is not a single point of failure;
  - This is a more flexible architecture, since different reasoning mechanisms can be used along with the different Virtual Web Services;
  - The total scalability of the reasoning mechanisms is higher than the scalability of one reasoning mechanism.
  
- Disadvantages:
  - The complexity of the Virtual Web Services Layer is increased;
  - The Virtual Web Services Manager should be able to configure all reasoning mechanisms.

### 3.4.2 Augmented reasoning mechanism

Currently, the Virtual Web Service is responsible for the service invocation. It refers to the reasoning mechanism to obtain the most appropriate service of the group at a given moment of time. This approach separates logically the capability of the Virtual Web Services and the reasoning mechanism, but augments the communication within the Virtual Web Services Layer. In order to reduce that communication, the reasoning mechanism might invoke the selected service. This suggests the following trade-offs:

- Advantages:
  - There is reduced communication between the elements of the Virtual Web Services Layer.
  
- Disadvantages:
  - There is augmented complexity of the reasoning mechanism. The reasoning mechanism should be aware of how to select the most appropriate service, as well as how to invoke it, and how to return the result back to the client.

### 3.4.3 Data consistency

When dealing with redundant components, a mechanism for achieving data consistency must be used. The presented Virtual Web Services Layer architecture allows different techniques for data consistency to be applied. For example:

- *State synchronization.* In order to synchronize the state of the redundant services, the clients' requests should be kept and used for Web Services synchronization, when it is needed (for example, after every request; after a specific period of time; after state transformation). In this case, the developer of the system must implement a state synchronization mechanism;
- *Transactions.* The requests of a client can be considered as a transaction that represents the ACID properties (*atomicity* - the transaction is seen as a single atomic operation; *consistency* - changes the state from one to another; *isolation* - the transactions are performed separately from one another; *durability* - when the transaction is committed, all changes are permanently updated). In addition, recovery mechanism and/or compensation actions should be considered. This case suggests that the developer of the application must implement it as a transactional system;
- *WS Transaction Management.* Web Services Transaction Management [28] is a standard that represents a set of protocols to manage the execution of the Web Services.

# CHAPTER 4

## EVALUATION

This chapter describes the evaluation of the proposed Virtual Web Services Layer (VWSL) approach. Firstly, a feasibility check is done to determine whether it is possible to build such an architecture. Secondly, the behavior of the Web Services is observed and analyzed in different environments. Finally, to observe the behavior of the VWSL with different selection techniques, the simulation method is used, since the study of the optimal strategies is difficult in a real environment where there are many uncontrollable parameters. The evaluation process is divided into the following three phases:

1. *Phase 1: VWSL prototype.* Development of a VWSL prototype system in order to test whether or not the architecture is a feasible technique for managing redundant Web Services on the server side in a dynamic and transparent manner;
2. *Phase 2: Clients' Load Generator and Simulator.* Development of a system that generates clients' load and simulates clients' behavior, in order to observe the response time of the Web Services in different environments. Through building such a system, an overview is obtained regarding the size and the complexity of both the implementation of a real system that deals with Web Services and the analysis of the obtained results;
3. *Phase 3: VWSL simulation.* Development of a simulation model that represents the VWSL architecture, using an existing simulation tool, which provides higher control of the system without dealing with the complexity of a real and resource-consuming environment. In addition, simulation and evaluation of the VWSL architecture is completed in order to observe the behavior of the system with different selection techniques. The question in this phase is: *Which selection technique(s) should be applied, depending on the information available to the system, regarding the QoS of the services?* The selection techniques that are taken into account are with respect to the system's dependability, response time, load-balancing, and overall performance.

## 4.1 Phase 1: VWSL prototype

### 4.1.1 Description

A simplified prototype of the proposed architecture is developed, in order to observe if the VWSL is an applicable approach for dynamic selection of redundant Web Services. It is implemented in the programming language Java [29], version 1.5. The dynamic service selection is realized by reflection that allows us to obtain information at run-time about methods, constructors, and instance fields of classes, as well as to invoke them dynamically [30], [31]. The system runs on the Apache Tomcat server [32], version 5.5 and on the Axis framework for Web Services, based on Java and XML [33], version 1.3.

The reasoning mechanism is based on a random selection technique that does not require any information about the services. Since the decision is done in a random manner, there is no need for selection criteria, nor data about the services must be collected and aggregated by the model. The model of the reasoning mechanism contains only the WSDL descriptions of the redundant Web Services.

The Manager of the layer is implemented as a client-server application in Java using JSP [34] and Servlets [35]. It generates Virtual Web Services and manages the settings of the reasoning mechanism (such as available Web Services).

### 4.1.2 Results

The results of the developed architecture are as follows:

- The Virtual Web Services Layer is a feasible technique for dynamic service selection on the server side. The layer is able to manage the redundant services in a transparent manner. All the necessary information, which should be available to the client, is the service description (WSDL) of the Virtual Web Services. The VWSs hide the reasoning details during the decision-making process. From the consumer's point of view, the VWS is the real Web Service that handles the request.
- The scalability of the system that implements the described layer is expected to be the same as the scalability of a system that does not consist of redundant components. The decentralization of the Virtual Web Services assures that there is no single point of control and respectively of failure. There is a separate VWS component that represents and manages each group of services and does not influence the proper work of the whole system.
- The architecture can be used as a layer that assures a level of security. The Web Services are called by the virtual layer and are never invoked directly by the clients. This technique can

prevent unauthorized users from having access to the real services.

- It is possible for the response time of the proposed architecture to increase due to the reasoning mechanism. This is an expected result since the decision is taken at run-time. Furthermore, it implies a trade-off between the appropriate service selection and the execution time of the system.

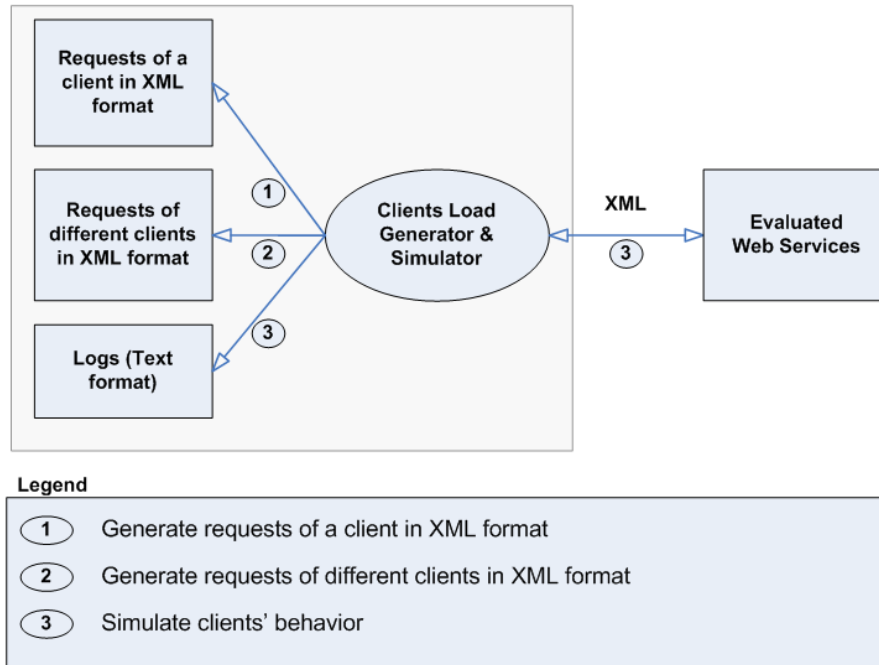
## 4.2 Phase 2: Clients' Load Generator and Simulator

### 4.2.1 Description

In order to examine and observe the behavior of real Web Services, a tool, called the Clients' Load Generator and Simulator, is developed and evaluated, using various scenarios. The implementation is done in Java ([29]). The system creates consumers' load and simulates consumers' behavior. The clients' requests are generated in XML format and are sent to the desired services by the system. The target components are Web Services, accessible via their service descriptions (WSDL). The main goals of the tool, presented in figure 4.1, are the following:

- *To generate requests for one client.* The calls represent the behavior of one client (1 client — multiple Web Services). The user of the application specifies the URL, the method name, and the arguments of the target Web Service for each call and gets as a result the client's requests in XML format;
- *To generate requests for multiple clients.* The application generates behavior of multiple clients (multiple clients — multiple Web Services). The consumers' requests are described in XML format, which is used as an input file for the simulation of the clients;
- *To simulate the clients' behavior.* The application loads the generated XML file, parses it in order to get the necessary information (such as target Web Services, method names, and arguments), and starts the execution of each client in a separate thread. The threads invoke the target services, get the results, and store them into text files (logs). The application keeps logs of all results obtained during the simulation period, such as request start time, request end time, result of the execution, services response times.

The tool is designed in a way that represents real clients' behavior - the calls within a client are synchronized, whereas the requests of different clients are processed in parallel. The behavior of one client consists of one or more requests. When the result of the first request is obtained, the second call is sent to the next service. The behavior of different clients is independent from one to another. The clients are able to send requests simultaneously to the same Web Services.



**Figure 4.1:** Clients' Load Generator and Simulator for Web Services

## 4.2.2 Experiment setups

The application and the target services are tested in a single machine as well as in a distributed environment, using RPC communication style (Apache Axis Framework, version 1.x, [33]) and document communication style (Apache Axis Framework, version 2.0, [36]) [13]. Eight experiment setups have been designed for each communication style:

- **Type 1: Clients and Web Services located in one machine:**

- Scenario 1: 1 client - 1 request per client
- Scenario 2: 1 client - many requests per client
- Scenario 3: many clients - 1 request per client
- Scenario 4: many clients - many requests per client

- **Type 2: Clients and Web Services distributed in different machines with the same characteristics:**

- Scenario 5: 1 client - 1 request per client
- Scenario 6: 1 client - many requests per client
- Scenario 7: many clients - 1 request per client
- Scenario 8: many clients - many requests per client

Experiment Type	Scenario	Min response time, ms	Average response time, ms	Max response time, ms	Number of repetitions
<b>One machine</b>	1 client – 1 request per client	875	1104.7	3250	20
	1 client – 10 requests per client	875	999.6	1203	20
	10 clients – 1 request per client	844	993	1157	20
	5 clients – 5 requests per client	969	1782.3	4094	10
<b>(3) Different machines</b>	1 client – 1 request per client	15	25	125	20
	1 client – 10 requests per client	0	21.1	94	20
	10 clients – 1 request per client	0	19.8	110	20
	5 clients – 5 requests per client	16	658.3	8875	10

**Table 4.1:** Clients’ Load Generator and Simulator: experiment results based on RPC communication style (Axis 1)

These experiment setups represent the full number of situations that could happen when the tool is used. The two communication styles (RPC and document style) are tested with exactly the same Web Services and clients load. The evaluation of the Clients’ Load Generator and Simulator gives an overview of the response times of the Web Services in the different cases. The real-world situation is scenario 8 that represents many clients calling many distributed Web Services. However, it is interesting to observe the other cases as well, since differences in the response times of the services might occur.

### 4.2.3 Results

The obtained results of the experiments (tables 4.1 and 4.2) show that the response times of the Web Services depend on both the communication style as well as the location of the services. For RPC communication style (Axis 1), when the clients and the services are located in the same machine, the response time is higher (average from 990 to 1800 milliseconds) than the response time of the services when they are distributed (average from 20 to 650 milliseconds). The reason for this result is the increased overhead of the machine, since the same machine has to send the clients’ calls, to parse the SOAP requests, to handle the requests, to create the SOAP responses, and to send the results back to the clients. In contrast, when the framework is document communication style (Axis 2), there are no significant differences in the response times (average from 30 to 1500 milliseconds), obtained with distributed and not distributed load, due to the newer version of the framework. For scenario 8: many clients - many requests, the maximum execution times vary significantly for both



Experiment Type	Scenario	Min response time, ms	Average response time, ms	Max response time, ms	Number of repetitions
<b>One machine</b>	1 client – 1 request per client	15	28.9	32	20
	1 client – 10 requests per client	15	48.11	266	20
	10 clients – 1 request per client	15	34.38	188	20
	5 clients – 5 requests per client	15	1348.65	6640	10
<b>(3) Different machines</b>	1 client – 1 request per client	15	24	32	20
	1 client – 10 requests per client	15	61.98	1125	20
	10 clients – 1 request per client	15	26.76	250	20
	5 clients – 5 requests per client	15	1603.36	12968	10

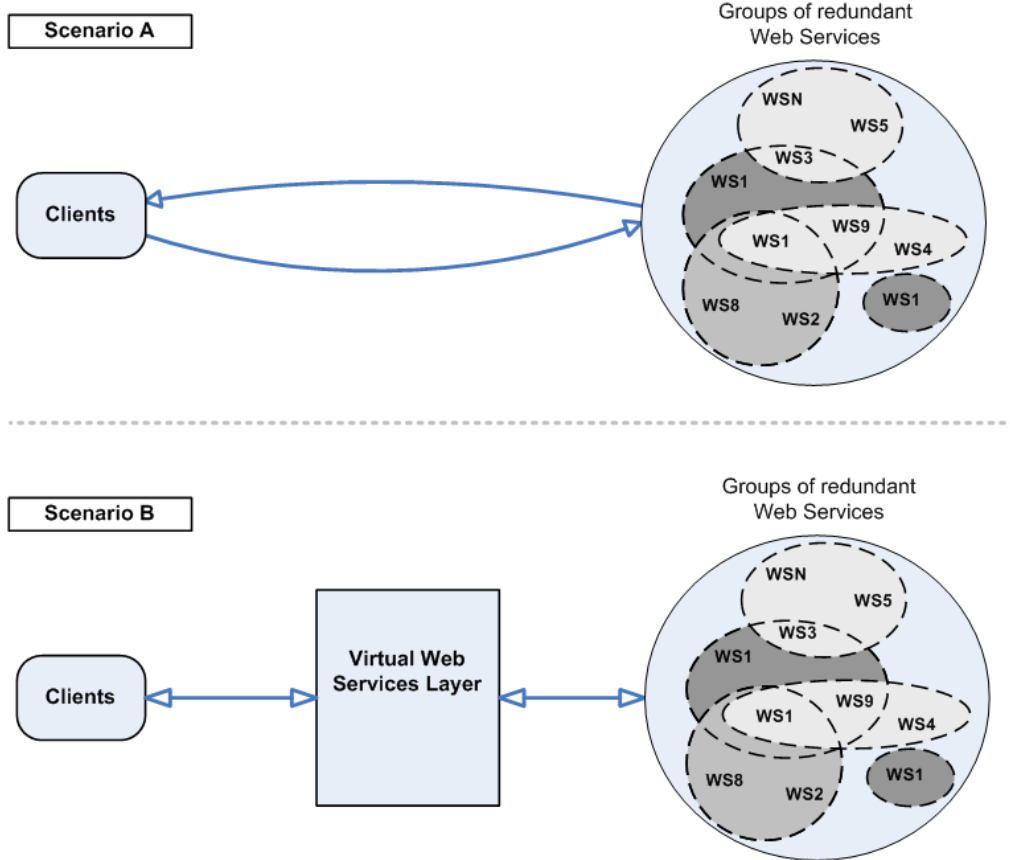
**Table 4.2:** Clients' Load Generator and Simulator: experiment results based on document communication style (Axis 2)

communication styles - from 15 milliseconds to 10-13 seconds. The reason for this result could be the creation of the sockets, garbage collection, and even the configuration of the Tomcat server and the axis settings.

#### 4.2.4 Phase 1 and phase 2: Conclusions

The proposed architecture - Virtual Web Services Layer - is an appropriate approach for dynamic and transparent Web Service selection on the server side, which should be evaluated further. The model and the selection technique are crucial parts of the reasoning mechanism and are directly related to the response time of the system. Various models and selection techniques should be taken into consideration in order to observe the behavior of the layer and at the same time to evaluate the reasoning mechanisms in different scenarios.

The development of the Virtual Web Services Layer in a real-world application and the evaluation of such a system is a complex procedure. It requires resources in terms of machines that run a set of experiments and time that should be devoted to each experiment setup, run, and analysis. In addition, it is very likely that the results of the experiments depend on the particular machine specifications and environment settings, such as web server, communication style (for example, Tomcat and Axis framework, if the system is implemented in Java). Fluctuations, due to network, memory, CPU, caching, and garbage collection, might appear as well. All this could reflect on the correct analysis of the obtained data and the validity of the conclusions. Furthermore, such an implementation is based on particular standards, protocols, and programming languages.



**Figure 4.2:** Evaluation scenarios A and B

Instead, the method of simulation is used. It provides the possibility to simplify the representation of the world and at the same time, the developer has better control. In order to design a simulation that is close to the real behavior of the Web Services, the obtained response times from the evaluation of the Clients' Load Generator and Simulator are used.

## 4.3 Phase 3: VWSL simulation

### 4.3.1 Description

Simulation is a technique that represents real-world behavior in a simplified manner. It allows observing and analyzing key characteristics of a selected system or a process [37].

In order to evaluate and analyze the Virtual Web Services Layer architecture, a simulation model is built, which runs on one machine. Parallel executions are not necessary since the simulation run is not time-consuming. Time-consuming are the experiment setups, the results gathering, and the analysis of the obtained data.

To observe the main range of possible situations, two scenarios, shown in figure 4.2, are taken

into consideration:

- *Scenario A.* The observed system consists of two participants - clients and groups of redundant Web Services. In this case, the clients know the target services of all calls. Service selection is not needed, since the Web Services are specified in the requests. Although this scenario is the simplest case, it should be evaluated and analyzed as a baseline for comparison to see if the proposed VWSL brings any advantages;
- *Scenario B.* The observed system consists of three participants - clients, Virtual Web Services Layer, and groups of redundant Web Services. This scenario is divided into the following real-world situations:
  - *Scenario B1.* Some of the target services are known by the clients. This case suggests that service selection is not needed all the time. Those requests that contain the target Web Services as part of their parameters, represent scenario A. In contrast, service selection is required for the calls that do not have the target services, but know only the functionalities that the services provide;
  - *Scenario B2.* None of the target services are known by the clients. This case requires service selection to be made for all requests of the clients.

Four Web Services are observed, which are clustered into three groups of redundant services, depending on the functionalities they provide. The response times of the Web Services are preset values, which correspond to some response times, obtained as results during the evaluation of the Clients' Load Generator and Simulator tool, described in the previous section. The capacity of the Web Services, showing the maximum requests that can be processed simultaneously, is the same for all services. In addition, the processing time of a request in each Web Service depends on the current load of the Web Service as well. This allows dealing with services behavior that is close to the real-world.

Two different workload levels are taken into account - low level and high level. When the load level is low, the system is able to handle the incoming requests without getting overloaded. When the load level is high, the system is able to handle the requests, but it is overloaded. The system is considered as *overloaded* when there is at least one request in a service's queue, waiting to be processed. Since the availability of the Web Services changes in the different experiments, a low load level in one case could be a high load level in another cases. On the other hand, in some cases the selection techniques might make the high load level as low load level.

The workload levels, the arrival rate distribution as well as the size distribution of the requests, is presented in table 4.3.

As discussed in chapter 3, both the applied selection technique and the models that collect and aggregate QoS information are important parts of the reasoning mechanism. However, the imple-

No	Workload		
	Workload Level	Arrival Rate Distribution	Request Size Distribution
1	Low	Constant	Constant
2	High	Constant	Constant
3	Low	Uniform-distributed	Uniform-distributed
4	High	Uniform-distributed	Uniform-distributed

**Table 4.3:** Experiments workload



**Figure 4.3:** Model, represented as a black box with a level of accuracy

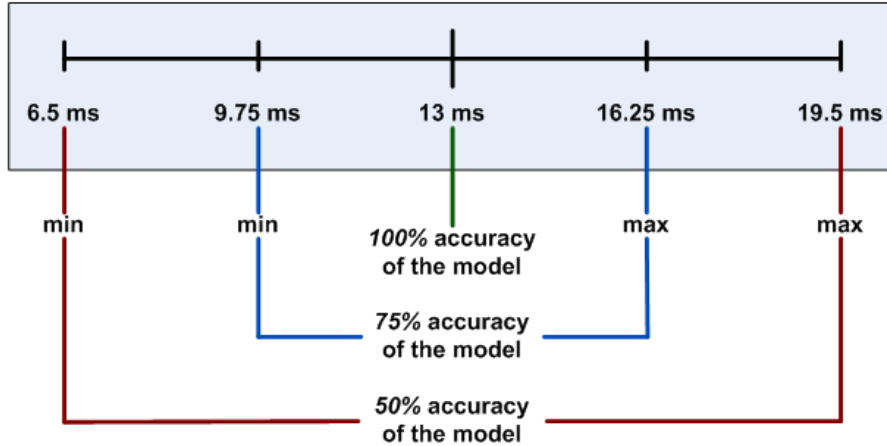
mentation of models that gather information about the services and represent it in an appropriate manner, is a resource- and time-consuming process. In general, every model has a level of accuracy (from 0% to 100% accuracy) that shows how correct is the information about the providers. Therefore, instead of developing a model, the following representation is made - the model is described as a black box with an accuracy about the QoS of the services (figure 4.3). This simplification allows the focus to be on different selection techniques, depending on the correctness of the models.

According to this representation, the predicted value of a QoS criterion belongs to an interval, defined by the value of the criterion in the simulation and the accuracy of the model:

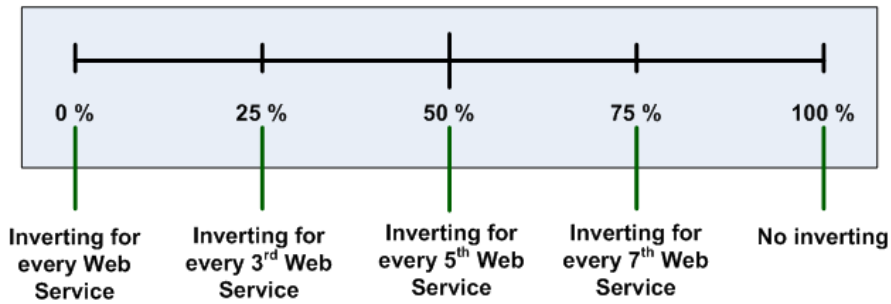
(1)  $X_M = [\frac{X_R * M_A}{100}, 2 * X_R - \frac{X_R * M_A}{100}]$ , where  $X_M$  is the predicted value of the QoS criterion;  $X_R$  is the value of the QoS criterion, specified in the simulation;  $M_A$  is the accuracy of the model.

For example, if the preset response time of a Web Service is 13 ms and the accuracy of the model is 100%, then 13 ms will be used in the system as predicted response time. However, if the accuracy of the model is 75%, the value of the predicted response time is in the interval [9.75 ms, 16.25 ms]. If the accuracy of the model is 50%, then the deviation from the predefined response time increases linearly and the predicted value belongs to the interval [6.5 ms, 19.5 ms]. The higher the accuracy of the model, the lower the interval of possible values; the lower the accuracy, the higher the possibility of deviation from the correct value. This is illustrated in figure 4.4.

In case of a boolean value of the QoS criterion, another approach is considered. For example, the availability of a Web Service can be *true* or *false*. If the model's accuracy is 100%, then the predicted value of the QoS criterion is the same as the real value. If the model's accuracy is 75%, the boolean value of the criterion is inverted for every seventh (7th) Web Service of the input batch of requests. If the model's accuracy is 50%, the value of the criterion is inverted for every fifth



**Figure 4.4:** Example: Execution time intervals, depending on the model accuracy



**Figure 4.5:** Inverting the boolean value of a QoS criterion, depending on the accuracy of the model

(5th) Web Service of the input. This is shown in figure 4.5. Table 4.4 presents an example of the predicted availability of the Web Services depending on the model's accuracy.

The behavior of the simulation depends on the applied selection technique. Four selection techniques are taken into consideration, as follows:

### **Random selection**

The random selection technique chooses a Web Service from a group of redundant services in a random manner. No information about the services is needed. This approach is appropriate when the services' behavior is not known.

### **The fastest service selection**

The fastest service technique selects a Web Service of a group of redundant services, according to its speed. It considers two QoS parameters - availability of the services and response time of the services.

Web Services Predicted Availability							
Model's Accuracy 75% (Inverting for every 7 <sup>th</sup> Web Service)				Model's Accuracy 50% (Inverting for every 5 <sup>th</sup> Web Service)			
Web Service 1	Web Service 2	Web Service 3	Web Service 4	Web Service 1	Web Service 2	Web Service 3	Web Service 4
Real value	Real value	Real value	Real value	Real value	Real value	Real value	Real value
Real value	Real value	Inverted value	Real value	Inverted value	Real value	Real value	Real value
Real value	Real value	Real value	Real value	Real value	Inverted value	Real value	Real value
Real value	Inverted value	Real value	Real value	Real value	Real value	Inverted value	Real value
Real value	Real value	Real value	Real value	Real value	Real value	Real value	Inverted value
Inverted value	Real value	Real value	Real value	Real value	Real value	Real value	Real value
Real value	Real value	Real value	Inverted value	Inverted value	Real value	Real value	Real value
Real value	Real value	Real value	Real value	Real value	Inverted value	Real value	Real value
Real value	Real value	Inverted value	Real value	Real value	Real value	Inverted value	Real value
Real value	Real value	Real value	Real value	Real value	Real value	Real value	Inverted value

**Table 4.4:** Inverting the boolean value of a QoS criterion, depending on the accuracy of the model

### **Load balancing**

The load balancing technique takes into account two features - availability of the services and services workload. The technique selects a Web Service with less load from a group of redundant Web Services, in order to avoid overloading of the services.

### **More accurate selection**

An attempt for a more accurate service selection is made, based on three QoS criteria - response time of the services, workload of the services, as well as the services' availability. It represents a combination of two selection techniques - the fastest service selection and the load balancing technique. The available service with the lowest response time, which is not overloaded, will be selected. However, if all available services are overloaded, the one with minimum load is chosen.

### **4.3.2 Phase 3: Main focus**

The goal of the Virtual Web Services Layer simulation is to answer the following questions:

1. How important is the availability of the Web Services for the system with and without the Virtual Web Services Layer?
2. How important is the models accuracy regarding the QoS of the Web Services in the system with Virtual Web Services Layer?
3. Does the applied selection technique affect the behavior of the system with the Virtual Web Services Layer?
4. Which selection technique(s) should be used in the different scenarios?
5. Does the size, the arrival rate distribution of the requests, as well as the load level (low and high) influence the behavior of the system with and without the Virtual Web Services Layer?

### **4.3.3 Experiment setups**

The scenarios are evaluated with a set of different experiments. Two types of variables are considered - independent variables and dependent variables.

The independent variables are the following:

- *Workload levels.* Two workload levels are considered - low and high. The values of these levels are defined in the case without the Virtual Web Services Layer, when all Web Services are available, and when the arrival rate and the request size distribution are constant as well as uniformly distributed;

- *Arrival rate distribution.* The distribution of the requests entering the system can be two types - constant or uniform-distributed. The former shows that the amount of incoming requests per time unit is the same during the simulation run, whereas the latter specifies that the number of generated requests per time unit changes;
- *Request size distribution.* The size of the request specifies how complex the request is. The size distribution can be constant or uniform-distributed. When the size is constant it means that all requests require the same amount of processing time, whereas in the second case, the size changes;
- *Known target services by the clients.* It specifies how many target services are predefined. Service selection is not needed for those requests that are predefined;
- *Web Services models accuracy.* The Web Service model represents QoS information about the service, which has a level of correctness showing how accurate the data is. The QoS criteria that are represented by the model depend on the applied selection technique. For example, in the case of a random selection, the model is not needed. However, in the case of the fastest service selection, the model represents two QoS criteria - the service's availability and the service's response time, and specifies the level of accuracy - for example 75%;
- *Web Services availability.* It shows how many services are available in the system. There must be at least one service available in each group;
- *Selection techniques.* Four selection techniques (random selection, the fastest service selection, load-balancing, and a more accurate selection) are implemented and applied.

The other type of parameters, the dependent variables, gives a general overview of the behavior of the system in the different scenarios. This allows analyzing the proposed VWSL architecture and studying which selection technique should be applied under certain conditions. The dependent parameters are the following:

- *Execution time of the simulation runs.* This parameter shows how much time it takes to complete the whole simulation run - from sending the first request to receiving the last response;
- *Throughput of the Web Services.* Throughput refers to the number of processed requests per time unit;
- *Dependability of the system.* The dependability is measured in terms of number of repetitions of the requests. If a large number of the requests needs to be re-processed due to inaccurate selection of target services, this will show that the level of dependability of the system is low. Formula 2 presents how the dependability is calculated for every simulation run. In the ideal



case, each request should be processed once and the dependability will be 1. Measuring the number of requests repetitions gives a better overview of the system than simply measuring the dependability as processed requests vs. all requests.

(2)  $D = \sum_{i=1}^N (\frac{1}{R_i} * \frac{1}{N}) \in [0, 1]$ , where  $D$  is the dependability of the system;  $R_i$  is the number of repetitions of request  $i$ ;  $N$  is the number of all generated requests in the system.

If a request is not processed at all,  $(\frac{1}{R_i})$  is considered as 0. As a result, the dependability of the system is lower.

The different scenarios are evaluated from the system's point of view with the same sets of workload, groups of redundant Web Services, and QoS parameters of the services, which assures the same start conditions for each situation. The parameters of the simulation are predefined in input files for every run and contain the following request-specific and system-specific information:

- *Request type.* Every request has a type. There are 10 types of requests. The difference between them is in the values of the predicted response times and workload of the Web Services. For example, if the models accuracy is 75% and the 'real' response of Web Service X is 0.66 seconds, then the predicted response time of the service, calculated using formula 1, is different for the different request types - the value belongs to the interval [0.495, 0.825] s;
- *Arrival rate of the requests.* The arrival rate shows how many requests of a particular type are entering the system per time unit, i.e. how many simultaneous requests of the same type are sent to the system at a particular moment of time. The arrival rate can be constant or uniformly-distributed with mean the same constant value as in the linear case. The interarrival time is constant;
- *Size of the request.* The size of the request shows the complexity of the request. The simplest request has size 1, whereas the haviest (more complex) request has size 6. If the time needed to process a request of size 1 is  $X$  time units, then the service time required to process a request of size  $N$ , where  $N$  is a value in the interval [1, 6], is  $X*N$  time units;
- *Functionality.* The functionality shows the functionality which must be provided by the invoked Web Service;
- *Target Web Service.* The target shows the name of the target Web Service, if it is known;
- *Predicted response times of the Web Services.* These values are calculated using formula 1 for every request type by taking into account the real response time of the services and the models accuracy. The predicted as well as the real response time corresponds to time units necessary for a Web Service to process a request with size 1;

- *Predicted workload of the Web Services.* The workload of the Web Services within a simulation run is dynamic. Therefore, the predicted workload of the Web Services depends on the current number of requests in progress in a given Web Service as well as the models accuracy. In order to specify the predicted values, the following approach is considered: an interval of predicted workload values is calculated using formula 1 for every request type. The low boundary of the interval is 0, when there are no requests in the Web Service, whereas the high boundary is 50, which indicates that at a given moment of time up to 50 requests could be in processing by one service;
- *Predicted availability of the Web Services.* The predicted availability of the Web Services takes into consideration the model's accuracy. If the accuracy is 100%, the real value is taken into account. Otherwise, the value is inverted for every 7th Web Service in case of 75% accuracy, and for every 5th Web Service in case of 50% accuracy;

**Note:** The predicted response times, workload, and availability of the services are or are not used depending on the applied selection technique. For example, if the selection technique is random selection, this QoS information is not taken into account, whereas these values are considered in the case of more accurate selection technique.

- *Real response times of the Web Services.* These values are the same for all scenarios;
- *Real availability of the Web Services.* These values change within a scenario, but are the same between scenarios (100%, 75%, and 50% availability is considered);
- *Others.* Other parameters, used internally in the simulation, such as experiment names, input file names, etc.

Two workload levels are considered - low level, which contains 60 requests during a simulation run and high level, which contains 120 requests during a simulation run. The average number of requests entering the system is 3 for low workload level and 6 for high workload level. The high load level is represented as two low load levels, as shown in figure 4.6. The size of the requests for both workload levels is in the interval  $[1, 6]$  with mean 3 when the distribution is uniform or 3 when the request size distribution is constant; The interarrival time is 2 time units.

The response times of the Web Services as well as the groups of redundant Web Services are shown in table 4.5. The maximum capacity of every Web Service, defined as capacity of the server and the capacity of the queue, is 50 requests - 15 (server capacity) and 35 (queue capacity).

The simulation is designed in a manner that when running different experiments, changes are required only in the input files, not in the simulation. The number of simulation runs is presented in table 4.6. 23 different experiments are conducted, which have 4 different cases and are repeated 3 times (average values are taken into account) for the random selection technique and once for the other cases, which gives 132 simulation runs in total.

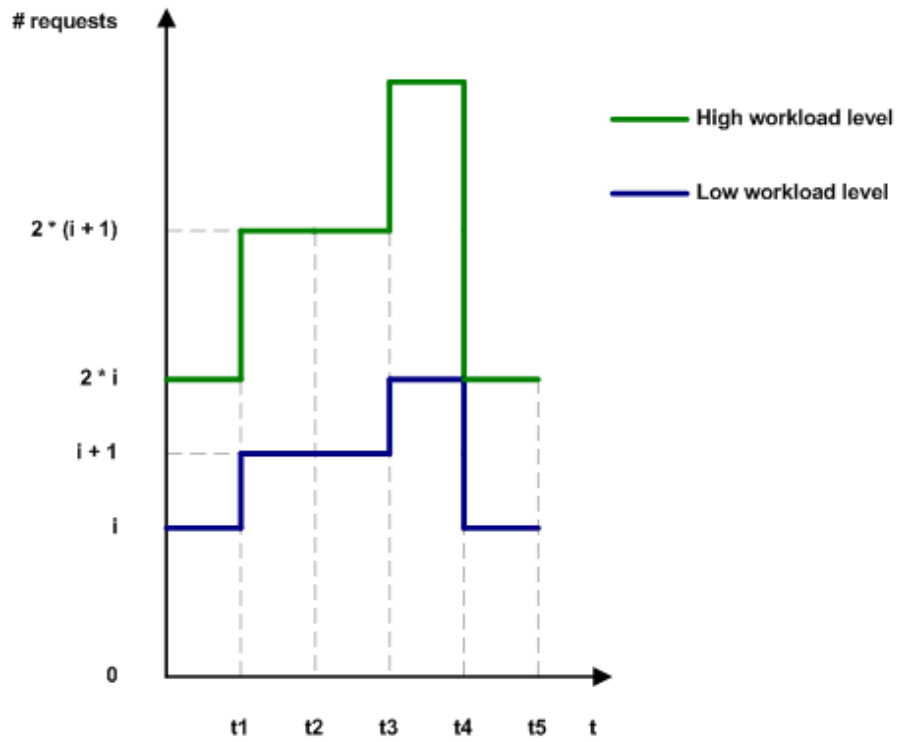


Figure 4.6: Workload levels of the simulation

	Web Service 1	Web Service 2	Web Service 3	Web Service 4
Response time, time units	0.02	1.0	1.6	0.66
Functionality	A, C	B, C	A, B, C	A, B
Groups of redundant Web Services	Group 1, Group 3	Group 2, Group 3	Group 1, Group 2, Group 3	Group 1, Group 2

Table 4.5: Web Services of the simulation

Experiments			
Number of experiments	Cases per experiment (workload level, arrival rate distribution, and request size distribution)	Run repetitions	Total number of runs
23	4	3 – for random selection technique; 1 – for the other experiments, since the same results are observed if the runs are repeated.	132

Table 4.6: Simulation runs

## Scenario A

The goal of scenario A is to evaluate the simulation model without the usage of the VWSL layer. In this case all target Web Services are known by the clients. The experiments for this scenario are shown in tables 4.7, 4.8, 4.9.

The independent variables are the following:

- Known services - all target services are known;
- Web Services availability - 100%, 75%, 50%;
- Request size distribution - constant, uniform-distributed;
- Arrival rate distribution - constant, uniform-distributed;
- Workload levels - low, high.

The dependent variables - execution times of the simulation runs, throughput of the Web Services, and dependability of the system, show how crucial is the high level of availability of the Web Services with respect to the overall performance of the system, when there is not reasoning about the service selection.

Experiment name	Case	Experiment Setup	Repetitions	Approximate time for setup and run
Experiment A-1	Case 1	- Known target Web Services – all; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – low; - Web Services availability – 100%; - Web Services models accuracy – N/A; - Reasoning mechanism – N/A.	1	1.5 days
	Case 2	- Known target Web Services – all; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – high; - Web Services availability – 100%; - Web Services models accuracy – N/A; - Reasoning mechanism – N/A.	1	
	Case 3	- Known target Web Services – all; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – low; - Web Services availability – 100%; - Web Services models accuracy – N/A; - Reasoning mechanism – N/A.	1	
	Case 4	- Known target Web Services – all; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – high; - Web Services availability – 100%; - Web Services models accuracy – N/A; - Reasoning mechanism – N/A.	1	

**Table 4.7:** Experiment setup: Scenario A, Experiment 1

Experiment name	Case	Experiment Setup	Repetitions	Approximate time for setup and run
Experiment A-2	Case 1	<ul style="list-style-type: none"> <li>- Known target Web Services – all;</li> <li>- Request size distribution – constant;</li> <li>- Arrival rate distribution – constant;</li> <li>- Workload level – low;</li> <li>- Web Services availability – 75%;</li> <li>- Web Services models accuracy – N/A;</li> <li>- Reasoning mechanism – N/A.</li> </ul>	1	1.5 days
	Case 2	<ul style="list-style-type: none"> <li>- Known target Web Services – all;</li> <li>- Request size distribution – constant;</li> <li>- Arrival rate distribution – constant;</li> <li>- Workload level – high;</li> <li>- Web Services availability – 75%;</li> <li>- Web Services models accuracy – N/A;</li> <li>- Reasoning mechanism – N/A.</li> </ul>	1	
	Case 3	<ul style="list-style-type: none"> <li>- Known target Web Services – all;</li> <li>- Request size distribution – uniform-distributed;</li> <li>- Arrival rate distribution – uniform-distributed;</li> <li>- Workload level – low;</li> <li>- Web Services availability – 75%;</li> <li>- Web Services models accuracy – N/A;</li> <li>- Reasoning mechanism – N/A.</li> </ul>	1	
	Case 4	<ul style="list-style-type: none"> <li>- Known target Web Services – all;</li> <li>- Request size distribution – uniform-distributed;</li> <li>- Arrival rate distribution – uniform-distributed;</li> <li>- Workload level – high;</li> <li>- Web Services availability – 75%;</li> <li>- Web Services models accuracy – N/A;</li> <li>- Reasoning mechanism – N/A.</li> </ul>	1	

**Table 4.8:** Experiment setup: Scenario A, Experiment 2

Experiment name	Case	Experiment Setup	Repetitions	Approximate time for setup and run
Experiment A-3	Case 1	<ul style="list-style-type: none"> <li>- Known target Web Services – all;</li> <li>- Request size distribution – constant;</li> <li>- Arrival rate distribution – constant;</li> <li>- Workload level – low;</li> <li>- Web Services availability – 50%;</li> <li>- Web Services models accuracy – N/A;</li> <li>- Reasoning mechanism – N/A.</li> </ul>	1	1.5 days
	Case 2	<ul style="list-style-type: none"> <li>- Known target Web Services – all;</li> <li>- Request size distribution – constant;</li> <li>- Arrival rate distribution – constant;</li> <li>- Workload level – high;</li> <li>- Web Services availability – 50%;</li> <li>- Web Services models accuracy – N/A;</li> <li>- Reasoning mechanism – N/A.</li> </ul>	1	
	Case 3	<ul style="list-style-type: none"> <li>- Known target Web Services – all;</li> <li>- Request size distribution – uniform-distributed;</li> <li>- Arrival rate distribution – uniform-distributed;</li> <li>- Workload level – low;</li> <li>- Web Services availability – 50%;</li> <li>- Web Services models accuracy – N/A;</li> <li>- Reasoning mechanism – N/A.</li> </ul>	1	
	Case 4	<ul style="list-style-type: none"> <li>- Known target Web Services – all;</li> <li>- Request size distribution – uniform-distributed;</li> <li>- Arrival rate distribution – uniform-distributed;</li> <li>- Workload level – high;</li> <li>- Web Services availability – 50%;</li> <li>- Web Services models accuracy – N/A;</li> <li>- Reasoning mechanism – N/A.</li> </ul>	1	

**Table 4.9:** Experiment setup: Scenario A, Experiment 3

## Scenario B1

The goal of scenario B1 is to evaluate the simulation model with the usage of the VWSL layer. In this case, half of the target Web Services are known by the clients. For those services, service selection is not needed. The reasoning mechanism is used for the other requests. The experiments for this scenario are shown in tables 4.10, 4.11, 4.12, 4.13, 4.14, 4.15, 4.16, 4.17, 4.18, 4.19.

The independent variables are the following:

- Known services - half of the target services are known;
- Web Services models accuracy - 100%, 75%, 50%;
- Web Services availability - 100%, 75%, 50%;
- Request size distribution - constant, uniform-distributed;
- Arrival rate distribution - constant, uniform-distributed;
- Workload levels - low, high.
- Selection techniques:
  - Random selection;
  - The fastest service selection;
  - Load balancing technique;
  - More accurate selection.

The dependent variables - execution times of the simulation runs, throughput of the Web Services, and dependability of the system, show how the reasoning mechanism influences the behavior of the system. In addition, they indicate which selection techniques are appropriate for this case.

## Scenario B2

The goal of this scenario is to evaluate the simulation model with the usage of the VWSL layer. In this case, none of the target Web Services are known by the clients. Service selection is needed for all requests. The experiments for the scenario are shown in tables 4.20, 4.21, 4.22, 4.23, 4.24, 4.25, 4.26, 4.27, 4.28, 4.29.

The independent variables are the following:

- Known services - none of the target services is known;
- Web Services models accuracy - 100%, 75%, 50%;
- Web Services availability - 100%, 75%, 50%;

Experiment name	Case	Experiment Setup	Repetitions	Approximate time for setup and run
Experiment B1-1	Case 1	- Known target Web Services – half; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – low; - Web Services availability – 100%; - Web Services models accuracy – N/A; - Selection technique – random selection.	3	1.5 days
	Case 2	- Known target Web Services – half; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – high; - Web Services availability – 100%; - Web Services models accuracy – N/A; - Selection technique – random selection.	3	
	Case 3	- Known target Web Services – half; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – low; - Web Services availability – 100%; - Web Services models accuracy – N/A; - Selection technique – random selection.	3	
	Case 4	- Known target Web Services – half; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – high; - Web Services availability – 100%; - Web Services models accuracy – N/A; - Selection technique – random selection.	3	

**Table 4.10:** Experiment setup: Scenario B1, Experiment 1

Experiment name	Case	Experiment Setup	Repetitions	Approximate time for setup and run
Experiment B1-2	Case 1	- Known target Web Services – half; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – low; - Web Services availability – 75%; - Web Services models accuracy – N/A; - Selection technique – random selection.	3	1.5 days
	Case 2	- Known target Web Services – half; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – high; - Web Services availability – 75%; - Web Services models accuracy – N/A; - Selection technique – random selection.	3	
	Case 3	- Known target Web Services – half; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – low; - Web Services availability – 75%; - Web Services models accuracy – N/A; - Selection technique – random selection.	3	
	Case 4	- Known target Web Services – half; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – high; - Web Services availability – 75%; - Web Services models accuracy – N/A; - Selection technique – random selection.	3	

**Table 4.11:** Experiment setup: Scenario B1, Experiment 2



Experiment name	Case	Experiment Setup	Repetitions	Approximate time for setup and run
Experiment B1-3	Case 1	- Known target Web Services – half; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – low; - Web Services availability – 50%; - Web Services models accuracy – N/A; - Selection technique – random selection.	3	1.5 days
	Case 2	- Known target Web Services – half; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – high; - Web Services availability – 50%; - Web Services models accuracy – N/A; - Selection technique – random selection.	3	
	Case 3	- Known target Web Services – half; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – low; - Web Services availability – 50%; - Web Services models accuracy – N/A; - Selection technique – random selection.	3	
	Case 4	- Known target Web Services – half; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – high; - Web Services availability – 50%; - Web Services models accuracy – N/A; - Selection technique – random selection.	3	

**Table 4.12:** Experiment setup: Scenario B1, Experiment 3

Experiment name	Case	Experiment Setup	Repetitions	Approximate time for setup and run
Experiment B1-4	Case 1	- Known target Web Services – half; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – low; - Web Services availability – 75%; - Web Services models accuracy – 100%; - Selection technique – the fastest service selection.	1	1.5 days
	Case 2	- Known target Web Services – half; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – high; - Web Services availability – 75%; - Web Services models accuracy – 100%; - Selection technique – the fastest service selection.	1	
	Case 3	- Known target Web Services – half; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – low; - Web Services availability – 75%; - Web Services models accuracy – 100%; - Selection technique – the fastest service selection.	1	
	Case 4	- Known target Web Services – half; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – high; - Web Services availability – 75%; - Web Services models accuracy – 100%; - Selection technique – the fastest service selection.	1	

**Table 4.13:** Experiment setup: Scenario B1, Experiment 4

Experiment name	Case	Experiment Setup	Repetitions	Approximate time for setup and run
Experiment B1-5	Case 1	- Known target Web Services – half; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – low; - Web Services availability – 50%; - Web Services models accuracy – 75%; - Selection technique – the fastest service selection.	1	1.5 days
	Case 2	- Known target Web Services – half; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – high; - Web Services availability – 50%; - Web Services models accuracy – 75%; - Selection technique – the fastest service selection.	1	
	Case 3	- Known target Web Services – half; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – low; - Web Services availability – 50%; - Web Services models accuracy – 75%; - Selection technique – the fastest service selection.	1	
	Case 4	- Known target Web Services – half; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – high; - Web Services availability – 50%; - Web Services models accuracy – 75%; - Selection technique – the fastest service selection.	1	

**Table 4.14:** Experiment setup: Scenario B1, Experiment 5

Experiment name	Case	Experiment Setup	Repetitions	Approximate time for setup and run
Experiment B1-6	Case 1	- Known target Web Services – half; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – low; - Web Services availability – 75%; - Web Services models accuracy – 100%; - Selection technique – load balancing technique.	1	1.5 days
	Case 2	- Known target Web Services – half; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – high; - Web Services availability – 75%; - Web Services models accuracy – 100%; - Selection technique – load balancing technique.	1	
	Case 3	- Known target Web Services – half; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – low; - Web Services availability – 75%; - Web Services models accuracy – 100%; - Selection technique – load balancing technique.	1	
	Case 4	- Known target Web Services – half; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – high; - Web Services availability – 75%; - Web Services models accuracy – 100%; - Selection technique – load balancing technique.	1	

**Table 4.15:** Experiment setup: Scenario B1, Experiment 6

Experiment name	Case	Experiment Setup	Repetitions	Approximate time for setup and run
Experiment B1-7	Case 1	- Known target Web Services – half; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – low; - Web Services availability – 50%; - Web Services models accuracy – 75%; - Selection technique – load balancing technique.	1	1.5 days
	Case 2	- Known target Web Services – half; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – high; - Web Services availability – 50%; - Web Services models accuracy – 75%; - Selection technique – load balancing technique.	1	
	Case 3	- Known target Web Services – half; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – low; - Web Services availability – 50%; - Web Services models accuracy – 75%; - Selection technique – load balancing technique.	1	
	Case 4	- Known target Web Services – half; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – high; - Web Services availability – 50%; - Web Services models accuracy – 75%; - Selection technique – load balancing technique.	1	

**Table 4.16:** Experiment setup: Scenario B1, Experiment 7

Experiment name	Case	Experiment Setup	Repetitions	Approximate time for setup and run
Experiment B1-8	Case 1	- Known target Web Services – half; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – low; - Web Services availability – 75%; - Web Services models accuracy – 100%; - Selection technique – more accurate selection.	1	1.5 days
	Case 2	- Known target Web Services – half; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – high; - Web Services availability – 75%; - Web Services models accuracy – 100%; - Selection technique – more accurate selection.	1	
	Case 3	- Known target Web Services – half; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – low; - Web Services availability – 75%; - Web Services models accuracy – 100%; - Selection technique – more accurate selection.	1	
	Case 4	- Known target Web Services – half; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – high; - Web Services availability – 75%; - Web Services models accuracy – 100%; - Selection technique – more accurate selection.	1	

**Table 4.17:** Experiment setup: Scenario B1, Experiment 8

Experiment name	Case	Experiment Setup	Repetitions	Approximate time for setup and run
Experiment B1-9	Case 1	- Known target Web Services – half; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – low; - Web Services availability – 50%; - Web Services models accuracy – 75%; - Selection technique – more accurate selection.	1	1.5 days
	Case 2	- Known target Web Services – half; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – high; - Web Services availability – 50%; - Web Services models accuracy – 75%; - Selection technique – more accurate selection.	1	
	Case 3	- Known target Web Services – half; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – low; - Web Services availability – 50%; - Web Services models accuracy – 75%; - Selection technique – more accurate selection.	1	
	Case 4	- Known target Web Services – half; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – high; - Web Services availability – 50%; - Web Services models accuracy – 75%; - Selection technique – more accurate selection.	1	

**Table 4.18:** Experiment setup: Scenario B1, Experiment 9

Experiment name	Case	Experiment Setup	Repetitions	Approximate time for setup and run
Experiment B1-10	Case 1	- Known target Web Services – half; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – low; - Web Services availability – 50%; - Web Services models accuracy – 50%; - Selection technique – more accurate selection.	1	1.5 days
	Case 2	- Known target Web Services – half; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – high; - Web Services availability – 50%; - Web Services models accuracy – 50%; - Selection technique – more accurate selection.	1	
	Case 3	- Known target Web Services – half; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – low; - Web Services availability – 50%; - Web Services models accuracy – 50%; - Selection technique – more accurate selection.	1	
	Case 4	- Known target Web Services – half; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – high; - Web Services availability – 50%; - Web Services models accuracy – 50%; - Selection technique – more accurate selection.	1	

**Table 4.19:** Experiment setup: Scenario B1, Experiment 10

- Request size distribution - constant, uniform-distributed;
- Arrival rate distribution - constant, uniform-distributed;
- Workload levels - low, high.
- Selection techniques:
  - Random selection;
  - The fastest service selection;
  - Load balancing technique;
  - More accurate selection.

The dependent variables - execution times of the simulation runs, throughput of the Web Services, and dependability of the system, show how the reasoning mechanism influences the behavior of the system, when a decision-making process needs to be done for all requests.

Experiment name	Case	Experiment Setup	Repetitions	Approximate time for setup and run
Experiment B2-1	Case 1	- Known target Web Services – none; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – low; - Web Services availability – 100%; - Web Services models accuracy – N/A; - Selection technique – random selection.	3	1.5 days
	Case 2	- Known target Web Services – none; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – high; - Web Services availability – 100%; - Web Services models accuracy – N/A; - Selection technique – random selection.	3	
	Case 3	- Known target Web Services – none; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – low; - Web Services availability – 100%; - Web Services models accuracy – N/A; - Selection technique – random selection.	3	
	Case 4	- Known target Web Services – none; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – high; - Web Services availability – 100%; - Web Services models accuracy – N/A; - Selection technique – random selection.	3	

**Table 4.20:** Experiment setup: Scenario B2, Experiment 1

Experiment name	Case	Experiment Setup	Repetitions	Approximate time for setup and run
Experiment B2-2	Case 1	- Known target Web Services – none; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – low; - Web Services availability – 75%; - Web Services models accuracy – N/A; - Selection technique – random selection.	3	1.5 days
	Case 2	- Known target Web Services – none; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – high; - Web Services availability – 75%; - Web Services models accuracy – N/A; - Selection technique – random selection.	3	
	Case 3	- Known target Web Services – none; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – low; - Web Services availability – 75%; - Web Services models accuracy – N/A; - Selection technique – random selection.	3	
	Case 4	- Known target Web Services – none; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – high; - Web Services availability – 75%; - Web Services models accuracy – N/A; - Selection technique – random selection.	3	

**Table 4.21:** Experiment setup: Scenario B2, Experiment 2

Experiment name	Case	Experiment Setup	Repetitions	Approximate time for setup and run
Experiment B2-3	Case 1	- Known target Web Services – none; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – low; - Web Services availability – 50%; - Web Services models accuracy – N/A; - Selection technique – random selection.	3	1.5 days
	Case 2	- Known target Web Services – none; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – high; - Web Services availability – 50%; - Web Services models accuracy – N/A; - Selection technique – random selection.	3	
	Case 3	- Known target Web Services – none; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – low; - Web Services availability – 50%; - Web Services models accuracy – N/A; - Selection technique – random selection.	3	
	Case 4	- Known target Web Services – none; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – high; - Web Services availability – 50%; - Web Services models accuracy – N/A; - Selection technique – random selection.	3	

**Table 4.22:** Experiment setup: Scenario B2, Experiment 3

Experiment name	Case	Experiment Setup	Repetitions	Approximate time for setup and run
Experiment B2-4	Case 1	- Known target Web Services – none; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – low; - Web Services availability – 75%; - Web Services models accuracy – 100%; - Selection technique – the fastest service selection.	1	1.5 days
	Case 2	- Known target Web Services – none; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – high; - Web Services availability – 75%; - Web Services models accuracy – 100%; - Selection technique – the fastest service selection.	1	
	Case 3	- Known target Web Services – none; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – low; - Web Services availability – 75%; - Web Services models accuracy – 100%; - Selection technique – the fastest service selection.	1	
	Case 4	- Known target Web Services – none; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – high; - Web Services availability – 75%; - Web Services models accuracy – 100%; - Selection technique – the fastest service selection.	1	

**Table 4.23:** Experiment setup: Scenario B2, Experiment 4

Experiment name	Case	Experiment Setup	Repetitions	Approximate time for setup and run
Experiment B2-5	Case 1	- Known target Web Services – none; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – low; - Web Services availability – 50%; - Web Services models accuracy – 75%; - Selection technique – the fastest service selection.	1	1.5 days
	Case 2	- Known target Web Services – none; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – high; - Web Services availability – 50%; - Web Services models accuracy – 75%; - Selection technique – the fastest service selection.	1	
	Case 3	- Known target Web Services – none; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – low; - Web Services availability – 50%; - Web Services models accuracy – 75%; - Selection technique – the fastest service selection.	1	
	Case 4	- Known target Web Services – none; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – high; - Web Services availability – 50%; - Web Services models accuracy – 75%; - Selection technique – the fastest service selection.	1	

**Table 4.24:** Experiment setup: Scenario B2, Experiment 5

Experiment name	Case	Experiment Setup	Repetitions	Approximate time for setup and run
Experiment B2-6	Case 1	- Known target Web Services – none; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – low; - Web Services availability – 75%; - Web Services models accuracy – 100%; - Selection technique – load balancing technique.	1	1.5 days
	Case 2	- Known target Web Services – none; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – high; - Web Services availability – 75%; - Web Services models accuracy – 100%; - Selection technique – load balancing technique.	1	
	Case 3	- Known target Web Services – none; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – low; - Web Services availability – 75%; - Web Services models accuracy – 100%; - Selection technique – load balancing technique.	1	
	Case 4	- Known target Web Services – none; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – high; - Web Services availability – 75%; - Web Services models accuracy – 100%; - Selection technique – load balancing technique.	1	

**Table 4.25:** Experiment setup: Scenario B2, Experiment 6



Experiment name	Case	Experiment Setup	Repetitions	Approximate time for setup and run
Experiment B2-7	Case 1	- Known target Web Services – none; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – low; - Web Services availability – 50%; - Web Services models accuracy – 75%; - Selection technique – load balancing technique.	1	1.5 days
	Case 2	- Known target Web Services – none; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – high; - Web Services availability – 50%; - Web Services models accuracy – 75%; - Selection technique – load balancing technique.	1	
	Case 3	- Known target Web Services – none; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – low; - Web Services availability – 50%; - Web Services models accuracy – 75%; - Selection technique – load balancing technique.	1	
	Case 4	- Known target Web Services – none; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – high; - Web Services availability – 50%; - Web Services models accuracy – 75%; - Selection technique – load balancing technique.	1	

**Table 4.26:** Experiment setup: Scenario B2, Experiment 7

Experiment name	Case	Experiment Setup	Repetitions	Approximate time for setup and run
Experiment B2-8	Case 1	- Known target Web Services – none; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – low; - Web Services availability – 75%; - Web Services models accuracy – 100%; - Selection technique – more accurate selection.	1	1.5 days
	Case 2	- Known target Web Services – none; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – high; - Web Services availability – 75%; - Web Services models accuracy – 100%; - Selection technique – more accurate selection.	1	
	Case 3	- Known target Web Services – none; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – low; - Web Services availability – 75%; - Web Services models accuracy – 100%; - Selection technique – more accurate selection.	1	
	Case 4	- Known target Web Services – none; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – high; - Web Services availability – 75%; - Web Services models accuracy – 100%; - Selection technique – more accurate selection.	1	

**Table 4.27:** Experiment setup: Scenario B2, Experiment 8

Experiment name	Case	Experiment Setup	Repetitions	Approximate time for setup and run
Experiment B2-9	Case 1	- Known target Web Services – none; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – low; - Web Services availability – 50%; - Web Services models accuracy – 75%; - Selection technique – more accurate selection.	1	1.5 days
	Case 2	- Known target Web Services – none; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – high; - Web Services availability – 50%; - Web Services models accuracy – 75%; - Selection technique – more accurate selection.	1	
	Case 3	- Known target Web Services – none; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – low; - Web Services availability – 50%; - Web Services models accuracy – 75%; - Selection technique – more accurate selection.	1	
	Case 4	- Known target Web Services – none; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – high; - Web Services availability – 50%; - Web Services models accuracy – 75%; - Selection technique – more accurate selection.	1	

**Table 4.28:** Experiment setup: Scenario B2, Experiment 9

Experiment name	Case	Experiment Setup	Repetitions	Approximate time for setup and run
Experiment B2-10	Case 1	- Known target Web Services – none; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – low; - Web Services availability – 50%; - Web Services models accuracy – 50%; - Selection technique – more accurate selection.	1	1.5 days
	Case 2	- Known target Web Services – none; - Request size distribution – constant; - Arrival rate distribution – constant; - Workload level – high; - Web Services availability – 50%; - Web Services models accuracy – 50%; - Selection technique – more accurate selection.	1	
	Case 3	- Known target Web Services – none; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – low; - Web Services availability – 50%; - Web Services models accuracy – 50%; - Selection technique – more accurate selection.	1	
	Case 4	- Known target Web Services – none; - Request size distribution – uniform-distributed; - Arrival rate distribution – uniform-distributed; - Workload level – high; - Web Services availability – 50%; - Web Services models accuracy – 50%; - Selection technique – more accurate selection.	1	

**Table 4.29:** Experiment setup: Scenario B2, Experiment 10

# CHAPTER 5

## SIMULATION

Simulation is a method that represents real-world behavior in a simplified manner. It allows observing key characteristics of a selected system or a process [37]. AnyLogic is a flexible and powerful tool that is used to model, simulate, visualize, and analyze diverse real-world problems. It is a simulation software for continuous, discrete, and hybrid systems, which can be applied in many areas, such as control systems, traffic, system dynamics, manufacturing, networks, computer systems, and others [38], [7]. The real objects are represented by classes of active objects. These objects can be atomic or compound. The compound ones encapsulate other active objects to any depth. This allows building a hierarchy of objects and constructing simulations in a modular way (figure 5.1).

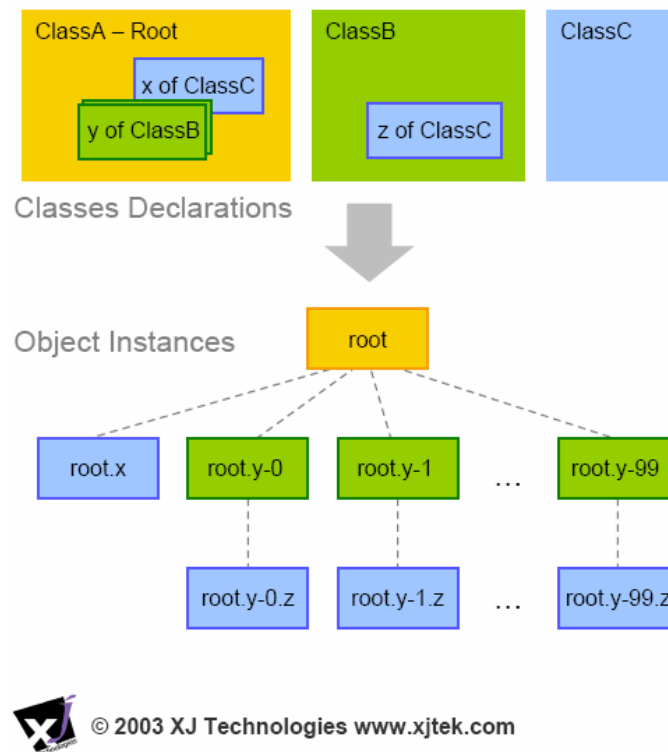


Figure 5.1: AnyLogic: Hierarchy of active objects presented in [7]

## 5.1 AnyLogic Enterprise Library

AnyLogic provides libraries for simulating systems in various domains. The Enterprise Library can be applied in discrete systems, such as manufacturing, services, business processes, etc. [39]. It is able to “create flexible models, collect basic and advanced statistics, and effectively visualize the process”, in order to represent the system. At the same time, it “provides a higher-level interface for fast creation of discrete event models in the style of flowcharts”, using objects like source, sink, queue, delay, server, and others, in a drag-and-drop manner [1].

Entity is a basic concept in the Enterprise Library. The entities represent individual units that are evaluated in the simulation. They can enter and leave objects through one directional ports. The connections between the ports are established by connectors.

In order to observe the behavior of the Virtual Web Services Layer architecture, described in chapter 3, the AnyLogic Enterprise Library is used. The components used in the simulation of the proposed architecture are described in tables 5.1 and 5.2.

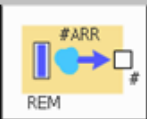
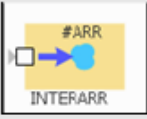
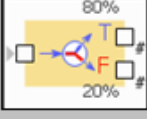
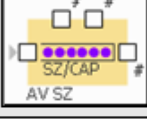
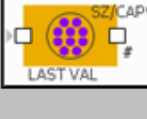
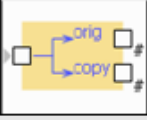
## 5.2 Simulation design of the proposed architecture

This section provides an overview of the design of the simulation model that is used to evaluate the VWSL approach.

The architecture, proposed in chapter 3, consists of three main components - Clients, Virtual Web Services Layer, and Web Services, forming the key compound objects of the high-level view of the simulation design, presented in figure 5.2. Object *LG* (load generator) generates clients' requests (entities), object *VWSL* represents the behavior of the virtual layer, and each object *WS* corresponds to a Web Service. Object *LS* (load sink) is used as the end point of the entity flow. It accepts the incoming from the VWSL entities and disposes of them.

### 5.2.1 Load Generator

Object *Load Generator*, presented in figure 5.3, is responsible for entities' generation and specifies the way that requests enter the system, which allows evaluation of the proposed architecture from a system point of view. The object consists of one source element which creates requests based on uniform distribution, where the interarrival time of the requests is constant, but the number of incoming entities varies. Every request has a set of parameters - request type, request size, functionality which must be provided by the invoked Web Service, target Web Service (if known), and other parameters used in the business logic of the simulation (whether the request is processed or not, number of attempts for processing the request, etc.). The input parameters of the simulation are read from external files for every experiment in order to assure that the results can be repeated

Enterprise Library Objects			
No	Icon	Name	Description
1		<b>Source</b>	It is used as a starting point of the entity flow since it generates entities.
2		<b>Sink</b>	It is used as an end point of the entity flow. It accepts incoming entities and disposes them.
3		<b>SelectOutput</b>	It accepts an entity and then forwards it to one of the output ports depending on the user-specified condition. The condition may depend on the entity itself as well as any external data.
4		<b>Queue</b>	It stores incoming entities in a specified order (FIFO, LIFO, Random, etc.).
5		<b>Server</b>	Delays entities until they receive the specified amount of service time. Multiple entities (up to the specified capacity) share the object. Entities served simultaneously affect each other. Once the entity has received the full amount of service, it leaves the object.
6		<b>Split</b>	It creates a specified amount of copies of the incoming entities which leave the object through the outputCopy port. The class and the initialization of the new entities are specified by the developer.

**Table 5.1:** Enterprise Library objects [1] used in the simulation



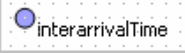
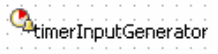
No	Enterprise Library Objects		
	Icon	Name	Description
1		<b>Port</b>	Each object has a port. The connections between objects are established through their ports.
2		<b>Connector</b>	It is used to connect objects. Once a connection between the objects is created, they can communicate with one another.
3		<b>Variable</b>	Variable, called interarrivalTime. Variables are used to store parameters of objects. The values may or may not change over time.
4		<b>Static timer</b>	Timer, called timerInputGenerator. When the time of the timer expires, it performs a user-defined action.

Table 5.2: Enterprise Library - other components used for the VWSL simulation

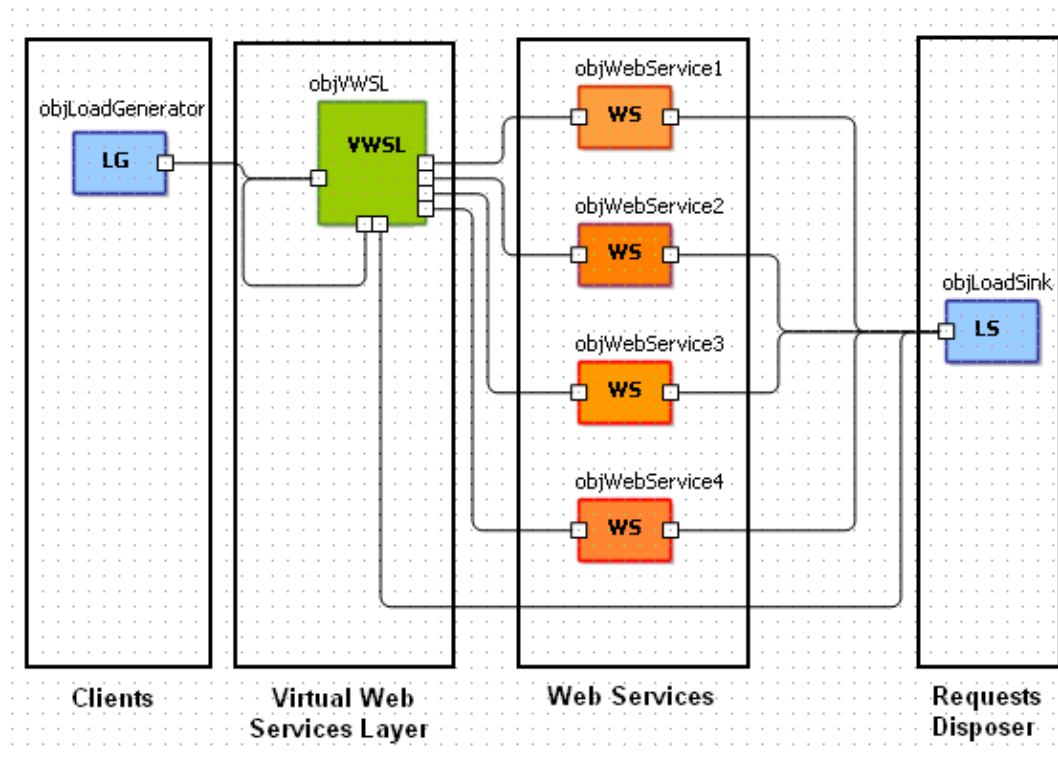


Figure 5.2: Simulation: Clients — VWSL — Web Services

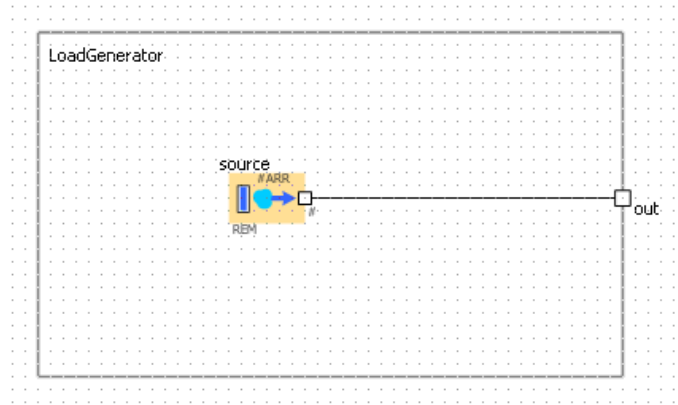


Figure 5.3: Simulation: Load Generator

if needed.

### 5.2.2 Virtual Web Services Layer

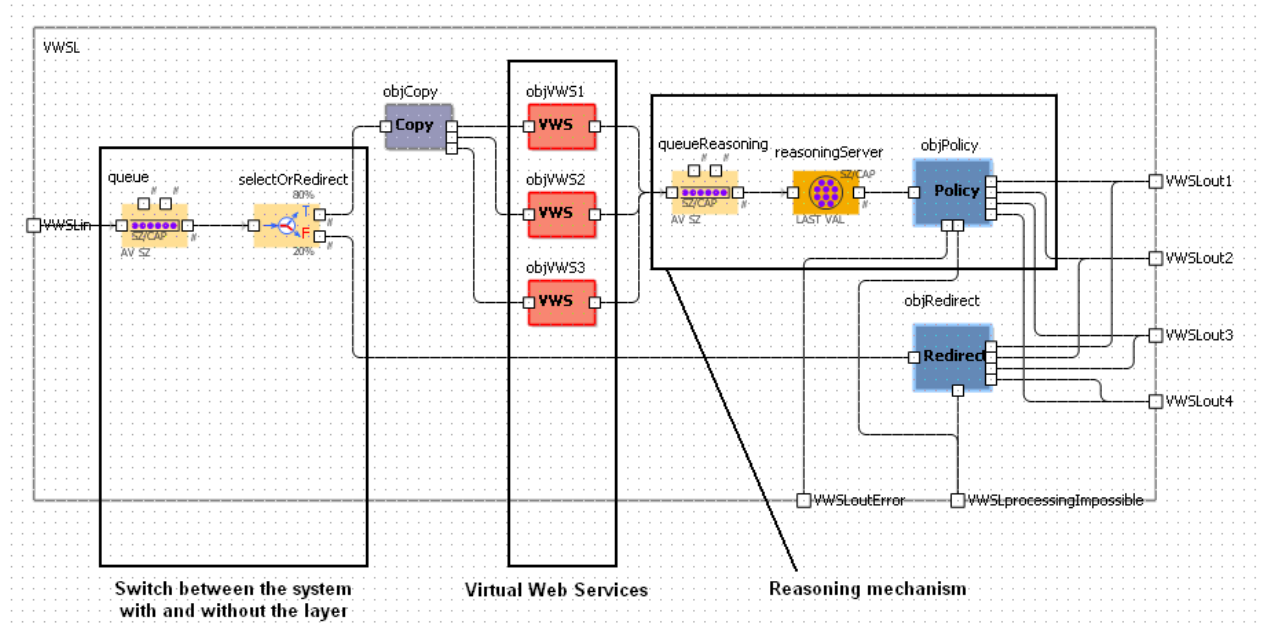
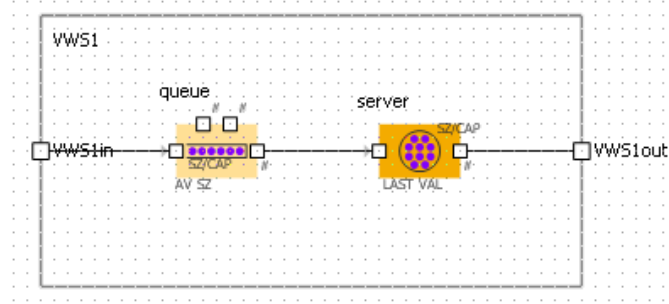


Figure 5.4: Simulation: Virtual Web Services Layer

As discussed in chapter 3, the Virtual Web Services Layer consists of a reasoning mechanism, Virtual Web Services, and a Manager. The reasoning mechanism is an important component of the VWSL architecture, which takes into account selection criteria, models, and selection techniques:

- *Selection criteria.* Selection criteria are used by the selection technique when a service selection is needed;
- *Models.* The models collect data about the Web Services and the interactions between the



**Figure 5.5:** Simulation: Virtual Web Service

clients and the services. They interpret the gathered data sets and represent them in an aggregated way. However, instead of building such models, collecting and evaluating data, the aggregated measures (such as service response time, service workload, and service availability) can be considered as given. Each model has a level of accuracy that shows how precise its information about the services is. This information is used by the applied selection technique.

- *Selection techniques.* Four selection techniques are considered in the simulation - random service selection, the fastest service selection, load balancing technique, and an attempt to achieve a more accurate service selection by combining the fastest service selection and load-balancing technique.

The simulation design of the VWSL is presented in figure 5.4. Component Manager is not included, since the settings of the reasoning mechanism are defined before the simulation runs. There is a Virtual Web Service corresponding to each group of redundant services with particular functionality. The reasoning mechanism is presented by a queue, a server, and a policy object. The *Policy* is a compound object which represents the applied selection technique. In addition, the Virtual Web Services Layer of the simulation is designed in a way that allows testing the system with and without using the Virtual Web Services and the reasoning mechanism, i.e. the simulation is able to switch between a system without the operation of the Virtual Web Services Layer and a system with the operation of the Virtual Web Services Layer. The system-switching mechanism consists of a queue and a select object. The *selectOrRedirect* object checks if the target Web Service is specified. If yes, then there is no need of service selection, and the request is sent to the *Redirect* object. However, if the endpoint is not known, then the entity is sent to the VWSs.

There are three Virtual Web Services in the simulation design. Each of them corresponds to a group of redundant Web Services with a particular functionality. The design of a VWS is shown in figure 5.5. It consists of a queue that keeps the incoming requests and a server that simulates the behavior of a real server.

In order to simplify the simulation design, there is a slightly difference between the proposed architecture and the object interaction, presented here. Instead of returning the name of the selected



Web Service, the reasoning mechanism calls the service itself. When the request is processed, it is disposed of by the object load sink. This approach simplifies the simulation and at the same time does not reflect on the behavior of the proposed VWSL architecture.

Object *Copy* copies the incoming request and sends it to all Virtual Web Services. However, only one Virtual Web Service processes the entity - the VWS that is responsible for the functionality, specified as part of the call. The other virtual services simply ignore and remove the entity.

Object *Redirect* is used when the simulation is observed without the operation of the VWSL. In this case, the target Web Services are specified in the generated requests and service selection is not needed.

### **Object *Policy***

The object *Policy*, presented in figure 5.6, is used during the service selection process. The incoming request enters the queue of the element via port *in*. Object *selectWSOrRejectRequest* chooses a Web Service according to the applied selection technique by taking into account the selection criteria and the predicted information about the services. As a result, the block forwards the request to the selected target Web Service, if the service is available, or rejects the request in case the service is not available. If the target Web Service is available, object *selectWSOrRejectRequest* returns true and the request is sent to it via port *policyOut1*, *policyOut2*, *policyOut3*, or *policyOut4*, depending on which port is connected to the target Web Service. Otherwise, the entity is sent to object *tryToSelectWSAgain*, which checks whether the maximum attempts for processing the request has been reached or not. If the maximum is exceeded, the request leaves object *Policy* via port *exceededMaximumRepetitions* and leaves object VWSL via port *VWSLprocessingImpossible* and is disposed by object *Load Sink* without being processed. However, if the maximum request repetitions is not reached, the entity leaves the policy block via port *policyOutError* and leaves the VWSL via port *VWSLoutError* and enters block VWSL again.

### **Object *Redirect***

Object *Redirect*, presented in figure 5.7, is used when there is no need of service selection, since the target Web Service is specified in the client's request. The incoming request enters the queue of the element via port *redirectIn*. Object *redirectOrRejectRequest* forwards the entity to the target Web Service if it is available via port *redirectOut1*, *redirectOut2*, *redirectOut3*, or *redirectOut4*, depending on which port is connected to the specified Web Service. However, if the service is not available, object *redirectOrRejectRequest* returns false and the entity leaves the element via port *redirectOutError* as well as leaves block VWSL via port *VWSLprocessingImpossible* and is disposed by object *Load Sink*, without being processed (since the target Web Service is not available).

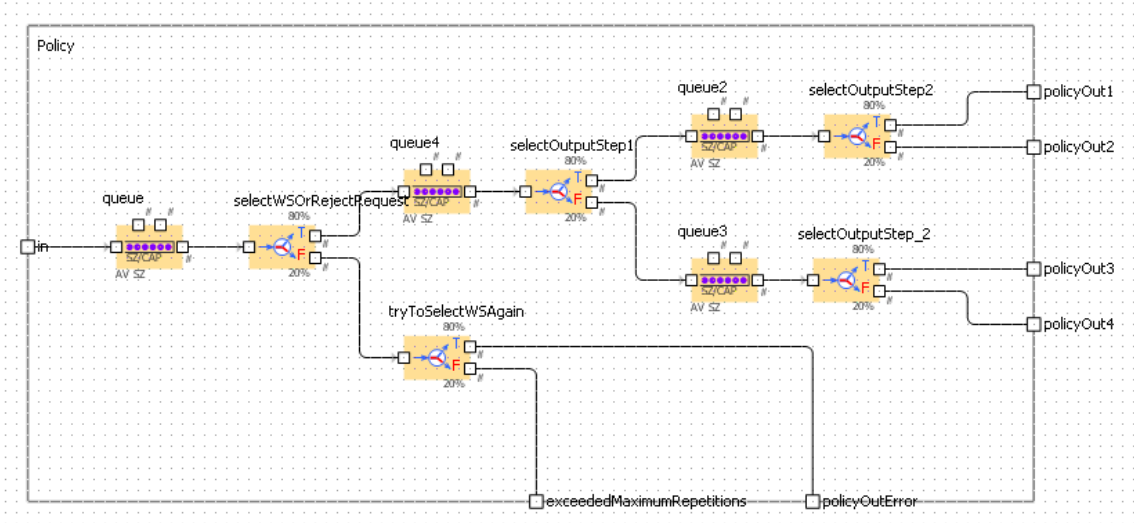


Figure 5.6: Simulation: Object *Policy* of the Virtual Web Services Layer

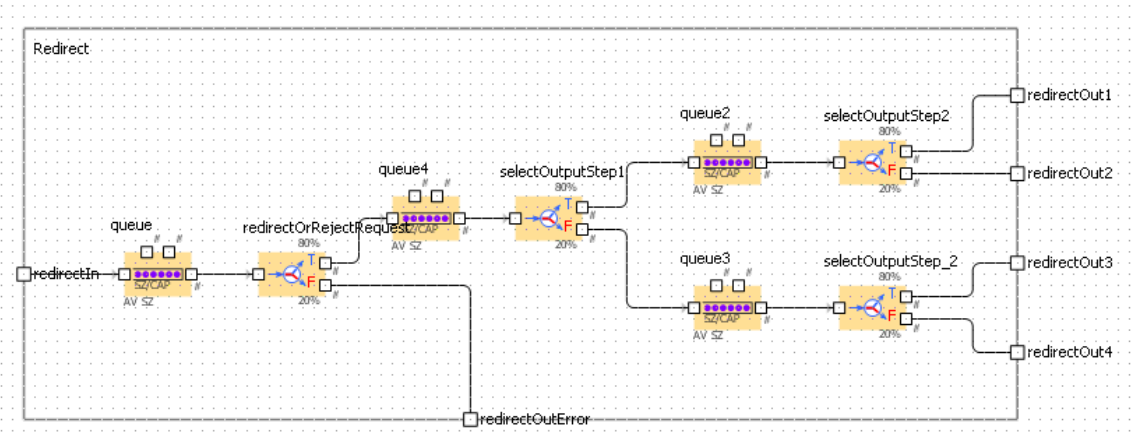


Figure 5.7: Simulation: Object *Redirect* of the Virtual Web Services Layer

### Entity flow of the Virtual Web Services Layer

When a request comes to the *VWSLin* port of the VWSL (figure 5.4), it enters the queue block of the switch mechanism. Then, the entity is analyzed by object *selectOrRedirect*, as follows:

- If the target Web Service is specified, object *selectOrRedirect* returns false. This indicates that the request goes to object *Redirect*. If the specified service is available in the system, the entity is sent to it through one of the outgoing ports - *VWSLout1*, *VWSLout2*, *VWSLout3*, or *VWSLout4*, depending on which port is connected to the particular Web Service. However, if the service is not available, the request leaves the VWSL through port *VWSLprocessingImpossible*. The request goes directly to the load disposer object, since it cannot be handled due to unavailability of the target Web Service.
- If the target Web Service is not specified in the request, the switch mechanism returns true. The entity is accepted by object *Copy* which makes two copies of the entity. The replicas and the original request enter the Virtual Web Services objects. The VWS whose functionality is specified in the request parameters, sends the entity to the reasoning mechanism. The other VWSs ignore and remove the received entity. The reasoning mechanism selects a Web Service according to the specified selection criteria, model, and selection technique. If the selected service is available, the entity is sent to it for processing. Otherwise, if the maximum number of possible attempts for processing is not reached, the request leaves the VWSL block through port *VWSLoutError* and enters again the queue of the switch mechanism via port *VWSLin*. This repeats until the selected Web Service is available and can process the request or until reaching the maximum number of possible attempts for processing. If the maximum number is reached, then the request leaves the VWSL block via port *VWSLprocessingImpossible*, which indicates that it is not processed due to unavailability of the selected target Web Service. This prevents infinite loops in the entity flow.

### 5.2.3 Web Services

Each Web Service consists of a queue and a server that simulates requests processing (figure 5.8). The Web Services provide one or more functionalities. Each service has a unique name, a predefined response time, and a capacity level.

### 5.2.4 Requests disposer - object *Load Sink*

Object *Load Sink*, presented in figure 5.9, accepts incoming requests via port *in* and disposes them via object *sinkProcessedRequests* or *sinkNotProcessedRequests*. It represents the end of the entity flow of the simulation. Object *selectOutput* checks whether the request is processed by a Web

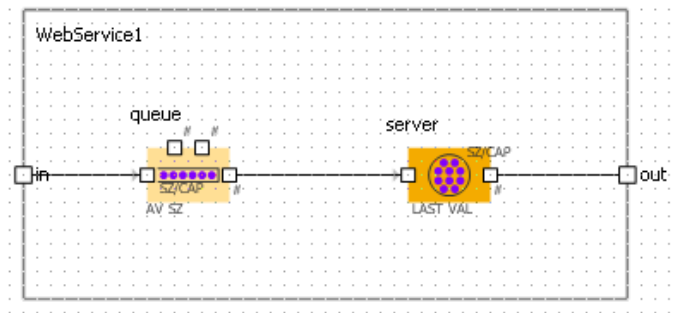


Figure 5.8: Simulation: Web Service

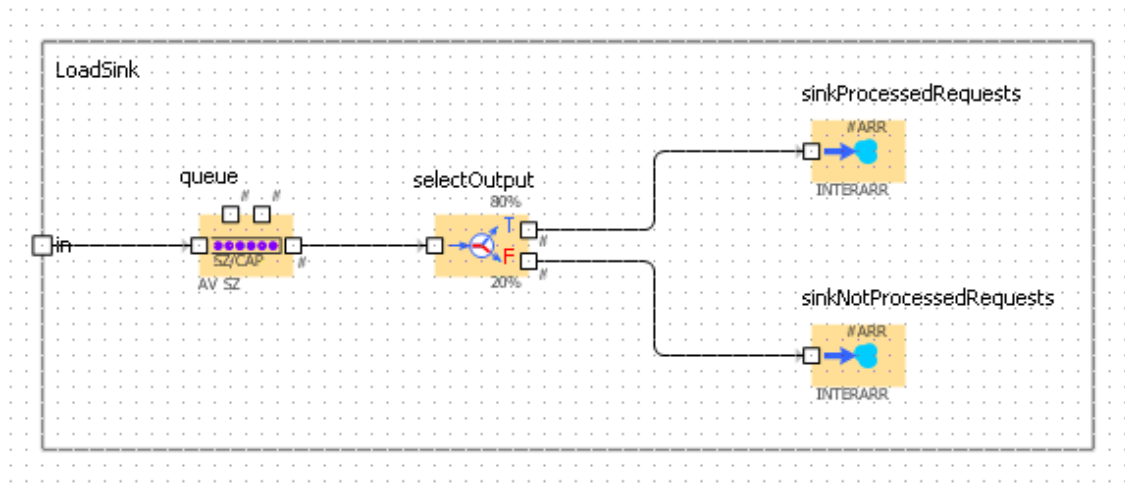


Figure 5.9: Simulation: Requests disposer

Service or not. If yes, then it is disposed by object *sinkProcessedRequests*, otherwise - by object *sinkNotProcessedRequests*. Two different sink elements are used in order to obtain the results of the simulation run in an easy manner.

### 5.3 Summary

This chapter describes the simulation environment AnyLogic and presents the design of the simulation that corresponds to the proposed Virtual Web Services Layer architecture. The clients are represented by object *Load Generator*; the behavior of the virtual layer is simulated by object *VWSL*; and the Web Services are represented by objects *WS*.

## CHAPTER 6

### VIRTUAL WEB SERVICES LAYER SIMULATION RESULTS

This chapter presents the results of the conducted experiments based on the described simulation design.

The ideal setting (scenario A, experiment 1; without Virtual Web Services Layer) when all target Web Services are known and available is presented in table 6.1 taking into account different load levels as well as request size and arrival rate distributions (figures 6.1 and 6.2). The values of the workload as well as the arrival rate and request size distribution are predefined and applied to all experiments.

In the ideal case, the dependability of the system reaches its maximum value - 1, since all requests are processed successfully due to the availability of all Web Services.

#### 6.1 Experiments results

As is expected, when the availability of the Web Services drops down to 75% (Web Service 2 is not available) and 50% (Web Services 1 and 2 are not available), the dependability of the system decreases linearly to 0.75 and 0.5 respectively, as shown in tables 6.2 and 6.3. In order to deal with this situation in a graceful manner, the Virtual Web Services Layer is used in the next experiments.

When random selection is applied for half of the Web Services (Scenario B1: Experiments 1, 2, and 3), the behaviour of the system varies between the simulation runs with the same configuration

	Total # processed requests	Workload level	Arrival rate distribution	Request size distribution	Execution time, time units	System dependability	Processed requests by a Web Service (Maximum requests in the queue of the Web Service)			
							Web Service 1 (Queue)	Web Service 2 (Queue)	Web Service 3 (Queue)	Web Service 4 (Queue)
Case 1	60	Low	Constant	Constant	72.5	1	15 (0)	15 (0)	15 (0)	15 (0)
Case 2	120	High	Constant	Constant	144.5	1	30 (0)	30 (9)	30 (15)	30 (3)
Case 3	60	Low	Uniform	Uniform	100.5	1	10 (0)	18 (0)	15 (0)	17 (0)
Case 4	120	High	Uniform	Uniform	188.5	1	20 (0)	36 (15)	30 (13)	34 (5)

**Table 6.1:** Simulation results: Ideal setting (Scenario A, Experiment 1)

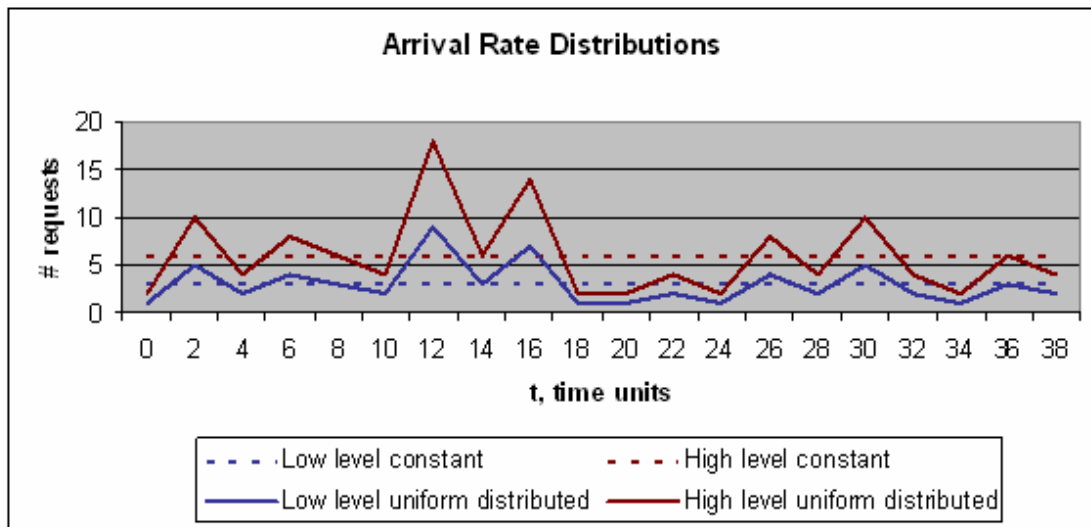


Figure 6.1: Simulation results: Arrival rate distributions

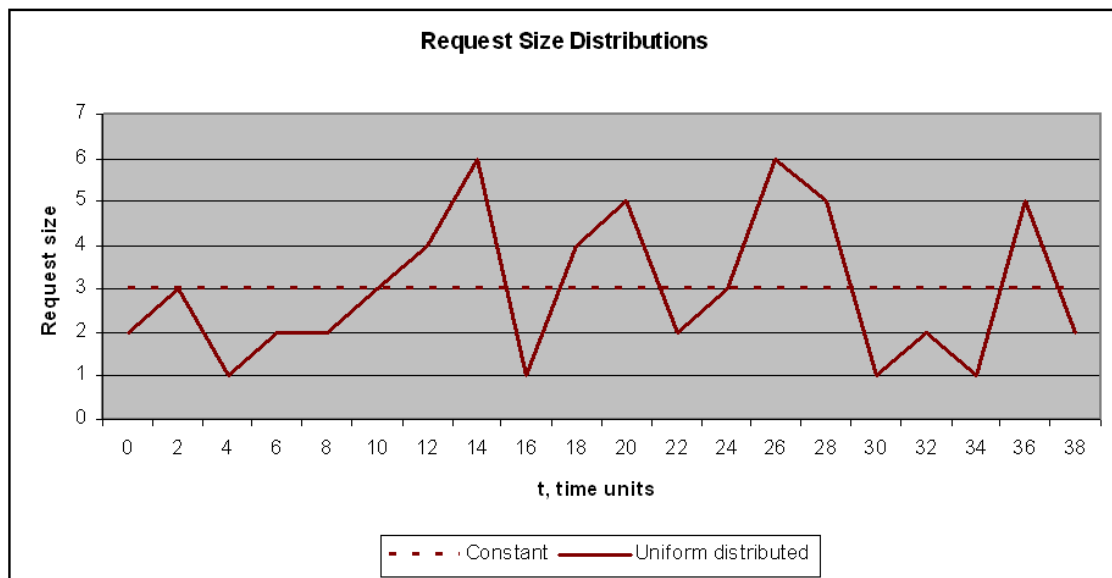


Figure 6.2: Simulation results: Request size distributions

	Total # processed requests	Workload level	Arrival rate distribution	Request size distribution	Execution time, time units	System dependability	Processed requests by a Web Service (Maximum requests in the queue of the Web Service)			
							Web Service 1 (Queue)	Web Service 2 (Queue)	Web Service 3 (Queue)	Web Service 4 (Queue)
Case 1	45 out of 60	Low	Constant	Constant	72.5	0.75	15 (0)	N/A	15 (0)	15 (0)
Case 2	90 out of 120	High	Constant	Constant	144.5	0.75	30 (0)	N/A	30 (15)	30 (3)
Case 3	42 out of 60	Low	Uniform	Uniform	100.5	0.70	10 (0)	N/A	15 (0)	17 (0)
Case 4	84 out of 120	High	Uniform	Uniform	188.5	0.70	20 (0)	N/A	30 (13)	34 (5)

**Table 6.2:** Simulation results: Availability 75% (Scenario A, Experiment 2)

	Total # processed requests	Workload level	Arrival rate distribution	Request size distribution	Execution time, time units	System dependability	Processed requests by a Web Service (Maximum requests in the queue of the Web Service)			
							Web Service 1 (Queue)	Web Service 2 (Queue)	Web Service 3 (Queue)	Web Service 4 (Queue)
Case 1	30 out of 60	Low	Constant	Constant	72.5	0.50	N/A	N/A	15 (0)	15 (0)
Case 2	60 out of 120	High	Constant	Constant	144.5	0.50	N/A	N/A	30 (15)	30 (3)
Case 3	32 out of 60	Low	Uniform	Uniform	100.5	0.53	N/A	N/A	15 (0)	17 (0)
Case 4	64 out of 120	High	Uniform	Uniform	188.5	0.53	N/A	N/A	30 (13)	34 (5)

**Table 6.3:** Simulation results: Availability 50% (Scenario A, Experiment 3)

	Total # processed requests	Workload level	Arrival rate distribution	Request size distribution	Execution time, time units	System dependability	Overloaded system	Processed requests by a Web Service (Maximum requests in the queue of the Web Service)			
								Web Service 1 (Queue)	Web Service 2 (Queue)	Web Service 3 (Queue)	Web Service 4 (Queue)
Case 1	60 out of 60	Low	Constant	Constant	83.67	1	Yes	12 (0)	21 (2)	16 (1)	11 (0)
Case 2	120 out of 120	High	Constant	Constant	161.5	1	Yes	16.33 (0.33)	48.33 (31.33)	29.33 (14.33)	26 (5)
Case 3	60 out of 60	Low	Uniform	Uniform	68.83	1	Sometimes (1 out of 3 times)	14 (0)	21 (0)	7.67 (0)	17.33 (1.33)
Case 4	120 out of 120	High	Uniform	Uniform	185.33	1	Yes	19.67 (0)	42 (17)	27.67 (13.67)	30.67 (10)

**Table 6.4:** Simulation results: Availability 100%, random selection for half of the Web Services (Scenario B1, Experiment 1)

	Total # processed requests	Workload level	Arrival rate distribution	Request size distribution	Execution time, time units	System dependability	Overloaded system	Processed requests by a Web Service (Maximum requests in the queue of the Web Service)			
								Web Service 1 (Queue)	Web Service 2 (Queue)	Web Service 3 (Queue)	Web Service 4 (Queue)
Case 1	45 out of 60	Low	Constant	Constant	120.33	0.63	Yes	10 (0)	N/A	25 (0)	10 (0)
Case 2	90 out of 120	High	Constant	Constant	160.17	0.67	Yes	30.33 (0)	N/A	33.33 (18.33)	26.33 (6.67)
Case 3	42 out of 60	Low	Uniform	Uniform	85.67	0.64	Yes	9.33 (0)	N/A	17 (4.33)	15.67 (1.33)
Case 4	84 out of 120	High	Uniform	Uniform	144.33	0.65	Yes	29 (0)	N/A	25.33 (6)	29.67 (15.67)

**Table 6.5:** Simulation results: Availability 75%, random selection for half of the Web Services (Scenario B1, Experiment 2)

due to the nature of the selection technique. In most of the cases the execution time is increased as well as the system is overloaded even when low workload levels are observed (tables 6.4, 6.5, and 6.6). When the availability of the Web Services drops down to 75% and 50%, the random selection technique cannot assign immediately an available Web Service from a group of redundant services to process the request since the technique does not take into consideration any QoS criteria of the Web Services. This increases the number of attempts for request processing and in some cases the maximum attempts (5) is exceeded. As a result, the entity is considered as not processed, which decreases the system's dependability. Therefore, the role of the reasoning mechanism of the Virtual Web Services Layer is crucial and it is important to apply selection techniques which take into account QoS of the Web Services.

When the fastest service selection technique is used for half of the Web Services and the availability of the services is 75% and 50% (Scenario B1: Experiments 4 and 5), the behavior of the system is improved compared to the same settings but with random technique. The total number of processed requests is increased, the system execution time and the Web Services overloading is lower as well as the system's dependability is higher (tables 6.7, 6.8). These results show that



	Total # processed requests	Workload level	Arrival rate distribution	Request size distribution	Execution time, time units	System dependability	Overloaded system	Processed requests by a Web Service (Maximum requests in the queue of the Web Service)			
								Web Service 1 (Queue)	Web Service 2 (Queue)	Web Service 3 (Queue)	Web Service 4 (Queue)
Case 1	36 out of 60	Low	Constant	Constant	99.5	0.48	Sometimes (2 out of 3 times)	N/A	N/A	20.67 (4.67)	15.33 (0)
Case 2	78 out of 120	High	Constant	Constant	214.5	0.51	Yes	N/A	N/A	44.67 (29.67)	33.33 (12.33)
Case 3	38 out of 60	Low	Uniform	Uniform	170	0.51	Yes	N/A	N/A	30 (14)	8 (0)
Case 4	76.67 out of 120	High	Uniform	Uniform	269.33	0.53	Yes	N/A	N/A	47.33 (30.33)	29.33 (5.33)

**Table 6.6:** Simulation results: Availability 50%, random selection for half of the Web Services (Scenario B1, Experiment 3)

	Total # processed requests	Workload level	Arrival rate distribution	Request size distribution	Execution time, time units	System dependability	Overloaded system	Processed requests by a Web Service (Maximum requests in the queue of the Web Service)			
								Web Service 1 (Queue)	Web Service 2 (Queue)	Web Service 3 (Queue)	Web Service 4 (Queue)
Case 1	45 out of 60	Low	Constant	Constant	46.5	0.75	No	24 (0)	N/A	6 (0)	15 (0)
Case 2	90 out of 120	High	Constant	Constant	69.5	0.75	Yes	48 (0)	N/A	12 (0)	30 (9)
Case 3	42 out of 60	Low	Uniform	Uniform	46	0.7	No	26 (0)	N/A	3 (0)	13 (0)
Case 4	84 out of 120	High	Uniform	Uniform	58.5	0.7	Yes	53 (3)	N/A	6 (0)	26 (3)

**Table 6.7:** Simulation results: Availability 75%, Accuracy 100%, the fastest service selection for half of the Web Services (Scenario B1, Experiment 4)

the fastest selection technique, which takes into account Web Services availability and response time as QoS criteria, improves the performance of the target system in terms of execution time, dependability, and overloading.

The obtained data of scenario B1, experiments 6 and 7 (tables 6.9 and 6.10) show that the execution times of the simulation runs are higher for the load balancing technique compared to the fastest service selection but lower compared to the random selection technique. In terms of Web Services overloading, both the fastest and the load balancing techniques present similar results.

When the more accurate selection technique is applied (Scenario B1, Experiments 8, 9, and 10), the results show that the accuracy of the predicted QoS information about the Web Services impacts the system behavior (tables 6.11, 6.12, and 6.13). When an unavailable Web Service is selected due to incorrect information about a particular service, the request cannot be handled and another selection is made. However, the same service is chosen due to the predicted information about the QoS criteria, but actually this service is not available, and another attempt is made again. This continues until the maximum number of attempts for request processing is reached, then the request

	Total # processed requests	Workload level	Arrival rate distribution	Request size distribution	Execution time, time units	System dependability	Overloaded system	Processed requests by a Web Service (Maximum requests in the queue of the Web Service)			
								Web Service 1 (Queue)	Web Service 2 (Queue)	Web Service 3 (Queue)	Web Service 4 (Queue)
Case 1	39 out of 60	Low	Constant	Constant	101	0.65	Yes	N/A	N/A	21 (3)	18 (0)
Case 2	78 out of 120	High	Constant	Constant	202	0.65	Yes	N/A	N/A	42 (27)	36 (9)
Case 3	39 out of 60	Low	Uniform	Uniform	133	0.65	Yes	N/A	N/A	21 (5)	18 (0)
Case 4	78 out of 120	High	Uniform	Uniform	258	0.65	Yes	N/A	N/A	42 (25)	36 (11)

**Table 6.8:** Simulation results: Availability 50%, Accuracy 75%, the fastest service selection for half of the Web Services (Scenario B1, Experiment 5)

	Total # processed requests	Workload level	Arrival rate distribution	Request size distribution	Execution time, time units	System dependability	Overloaded system	Processed requests by a Web Service (Maximum requests in the queue of the Web Service)			
								Web Service 1 (Queue)	Web Service 2 (Queue)	Web Service 3 (Queue)	Web Service 4 (Queue)
Case 1	45 out of 60	Low	Constant	Constant	62	0.75	No	21 (0)	N/A	12 (0)	12 (0)
Case 2	90 out of 120	High	Constant	Constant	110.5	0.75	Yes	44 (0)	N/A	23 (2)	23 (0)
Case 3	42 out of 60	Low	Uniform	Uniform	51	0.7	No	17 (0)	N/A	9 (0)	16 (0)
Case 4	84 out of 120	High	Uniform	Uniform	106	0.7	Yes	35 (0)	N/A	20 (0)	29 (1)

**Table 6.9:** Simulation results: Availability 75%, Accuracy 100%, load balancing technique for half of the Web Services (Scenario B1, Experiment 6)

	Total # processed requests	Workload level	Arrival rate distribution	Request size distribution	Execution time, time units	System dependability	Overloaded system	Processed requests by a Web Service (Maximum requests in the queue of the Web Service)			
								Web Service 1 (Queue)	Web Service 2 (Queue)	Web Service 3 (Queue)	Web Service 4 (Queue)
Case 1	39 out of 60	Low	Constant	Constant	130	0.65	Yes	N/A	N/A	27 (9)	12 (0)
Case 2	78 out of 120	High	Constant	Constant	230.5	0.65	Yes	N/A	N/A	48 (33)	30 (3)
Case 3	39 out of 60	Low	Uniform	Uniform	162	0.65	Yes	N/A	N/A	25 (9)	14 (0)
Case 4	78 out of 120	High	Uniform	Uniform	267.5	0.65	Yes	N/A	N/A	44 (27)	34 (5)

**Table 6.10:** Simulation results: Availability 50%, Accuracy 75%, load balancing technique for half of the Web Services (Scenario B1, Experiment 7)

	Total # processed requests	Workload level	Arrival rate distribution	Request size distribution	Execution time, time units	System dependability	Overloaded system	Processed requests by a Web Service (Maximum requests in the queue of the Web Service)			
								Web Service 1 (Queue)	Web Service 2 (Queue)	Web Service 3 (Queue)	Web Service 4 (Queue)
Case 1	45 out of 60	Low	Constant	Constant	46.5	0.75	No	24 (0)	N/A	6 (0)	15 (0)
Case 2	90 out of 120	High	Constant	Constant	97.5	0.75	Yes	48 (0)	N/A	20 (0)	22 (3)
Case 3	42 out of 60	Low	Uniform	Uniform	46	0.7	No	26 (0)	N/A	3 (0)	13 (0)
Case 4	84 out of 120	High	Uniform	Uniform	61	0.7	Yes	52 (3)	N/A	8 (0)	24 (1)

**Table 6.11:** Simulation results: Availability 75%, Accuracy 100%, more accurate selection for half of the Web Services (Scenario B1, Experiment 8)

	Total # processed requests	Workload level	Arrival rate distribution	Request size distribution	Execution time, time units	System dependability	Overloaded system	Processed requests by a Web Service (Maximum requests in the queue of the Web Service)			
								Web Service 1 (Queue)	Web Service 2 (Queue)	Web Service 3 (Queue)	Web Service 4 (Queue)
Case 1	39 out of 60	Low	Constant	Constant	101	0.65	Yes	N/A	N/A	21 (3)	18 (0)
Case 2	78 out of 120	High	Constant	Constant	202	0.65	Yes	N/A	N/A	42 (27)	36 (9)
Case 3	39 out of 60	Low	Uniform	Uniform	133	0.65	Yes	N/A	N/A	21 (5)	18 (0)
Case 4	78 out of 120	High	Uniform	Uniform	258	0.65	Yes	N/A	N/A	42 (25)	36 (11)

**Table 6.12:** Simulation results: Availability 50%, Accuracy 75%, more accurate selection for half of the Web Services (Scenario B1, Experiment 9)

is considered as not handled, which decreases the dependability even below the level obtained when random selection is applied (6.13). Therefore, the accuracy of the QoS information plays a key role in the reasoning mechanism.

The level of dependability is not high in scenario B1 since it is based on using a reasoning mechanism only for half of the target Web Services. For the other half, if the target Web Service is not available, the request cannot be processed which decreases the system’s dependability. This indicates that conclusions about which selection technique should be applied under different conditions cannot be made only from the obtained results of scenario B1. Therefore, in order to be able to analyse the advantages and the disadvantages of the selection techniques of the Virtual Web Services Layer, service selection for all requests must be observed as well (scenario B2).

In scenario B2, when the selection technique is random selection (tables 6.14, 6.15, and 6.16) the Web Services are overloaded even more compared to the same settings in scenario B1 due to the nature of the scenario – selection for all requests. In addition, the execution times of the simulation runs are higher than those in scenarios A and B1. On the other hand, the dependability is increased

	Total # processed requests	Workload level	Arrival rate distribution	Request size distribution	Execution time, time units	System dependability	Overloaded system	Processed requests by a Web Service (Maximum requests in the queue of the Web Service)			
								Web Service 1 (Queue)	Web Service 2 (Queue)	Web Service 3 (Queue)	Web Service 4 (Queue)
Case 1	33 out of 60	Low	Constant	Constant	86.5	0.55	No	N/A	N/A	18 (0)	15 (0)
Case 2	66 out of 120	High	Constant	Constant	173	0.55	Yes	N/A	N/A	36 (21)	30 (9)
Case 3	25 out of 60	Low	Uniform	Uniform	75.5	0.42	No	N/A	N/A	12 (0)	13 (0)
Case 4	50 out of 120	High	Uniform	Uniform	143	0.42	Yes	N/A	N/A	24 (7)	26 (3)

**Table 6.13:** Simulation results: Availability 50%, Accuracy 50%, more accurate selection for half of the Web Services (Scenario B1, Experiment 10)

	Total # processed requests	Workload level	Arrival rate distribution	Request size distribution	Execution time, time units	System dependability	Overloaded system	Processed requests by a Web Service (Maximum requests in the queue of the Web Service)			
								Web Service 1 (Queue)	Web Service 2 (Queue)	Web Service 3 (Queue)	Web Service 4 (Queue)
Case 1	60 out of 60	Low	Constant	Constant	97.17	1	Sometimes (1 time out of 3)	14.33 (0)	13.33 (0)	10 (5)	12.33 (0)
Case 2	120 out of 120	High	Constant	Constant	178.33	1	Yes	22 (0)	31 (12)	36.67 (21.67)	30.33 (6.67)
Case 3	60 out of 60	Low	Uniform	Uniform	151	1	Yes	12 (0)	12 (0)	30 (13.33)	6 (0)
Case 4	120 out of 120	High	Uniform	Uniform	168	1	Yes	20 (0)	40 (17)	40 (24.33)	20 (4.33)

**Table 6.14:** Simulation results: Availability 100%, random selection for all Web Services (Scenario B2, Experiment 1)

since selection is made for all Web Services. However, when the availability drops down to 50%, there are still unprocessed requests, since random selection assigns unavailable Web Services and the maximum attempts for request processing can be reached. Furthermore, when high workload level is observed (Scenario B2, Experiment 3), in order to run the simulation, extension of the Web Services queues is required due to Web Services overloading.

The random selection technique should be applied when there is incomplete information about the QoS of the Web Services. However, using this selection technique, it is not possible to predict the behavior of the system and it varies between different runs of the same situation. In order to avoid uncertainties in the system's performance, selection techniques which take into consideration QoS of the Web Services should be considered and applied.

When the availability and the response times of the Web Services are taken into account by the fastest service selection (tables 6.17 and 6.18), the system's overall performance is much better compared to the random selection technique. However, some available services are not used at all (i.e. Web Service 3), since their response time is high. This leads to overloading of the fast Web

	Total # processed requests	Workload level	Arrival rate distribution	Request size distribution	Execution time, time units	System dependability	Overloaded system	Processed requests by a Web Service (Maximum requests in the queue of the Web Service)			
								Web Service 1 (Queue)	Web Service 2 (Queue)	Web Service 3 (Queue)	Web Service 4 (Queue)
Case 1	60 out of 60	Low	Constant	Constant	122.5	0.88	Yes	19.33 (0)	N/A	24.33 (9.33)	16.33 (0)
Case 2	120 out of 120	High	Constant	Constant	210.5	0.91	Yes	38.67 (0)	N/A	43.33 (27.67)	38 (14)
Case 3	60 out of 60	Low	Uniform	Uniform	146	0.84	Yes	7 (0)	N/A	30 (12)	23 (0.67)
Case 4	120 out of 120	High	Uniform	Uniform	158.33	0.84	Yes	29.33 (0.33)	N/A	32.67 (17)	58 (28.67)

**Table 6.15:** Simulation results: Availability 75%, random selection for all Web Services (Scenario B2, Experiment 2)

	Total # processed requests	Workload level	Arrival rate distribution	Request size distribution	Execution time, time units	System dependability	Overloaded system	Processed requests by a Web Service (Maximum requests in the queue of the Web Service)			
								Web Service 1 (Queue)	Web Service 2 (Queue)	Web Service 3 (Queue)	Web Service 4 (Queue)
Case 1	52 out of 60	Low	Constant	Constant	157.5	0.65	Yes	N/A	N/A	32.67 (17.33)	19.33 (0)
Case 2	103 out of 120	High	Constant	Constant	312.33	0.65	Yes	N/A	N/A	64 (49)*	39 (11.67)
Case 3	57.67 out of 60	Low	Uniform	Uniform	182.33	0.82	Yes	N/A	N/A	33.67 (17.33)	24 (4.33)
Case 4	116.67 out of 120	High	Uniform	Uniform	341.33	0.72	Yes	N/A	N/A	66.67 (47.67)*	50 (24.33)

**Table 6.16:** Simulation results: Availability 50%, random selection for all Web Services (Scenario B2, Experiment 3); \* - extended queue

	Total # processed requests	Workload level	Arrival rate distribution	Request size distribution	Execution time, time units	System dependability	Overloaded system	Processed requests by a Web Service (Maximum requests in the queue of the Web Service)			
								Web Service 1 (Queue)	Web Service 2 (Queue)	Web Service 3 (Queue)	Web Service 4 (Queue)
Case 1	60 out of 60	Low	Constant	Constant	48	1	No	36 (0)	N/A	0 (0)	24 (0)
Case 2	120 out of 120	High	Constant	Constant	95.5	1	Yes	72 (0)	N/A	0 (0)	48 (21)
Case 3	60 out of 60	Low	Uniform	Uniform	55.5	1	No	33 (0)	N/A	0 (0)	27 (0)
Case 4	120 out of 120	High	Uniform	Uniform	99	1	Yes	66 (3)	N/A	0 (0)	54 (20)

**Table 6.17:** Simulation results: Availability 75%, Accuracy 100%, the fastest service selection for all Web Services (Scenario B2, Experiment 4)

	Total # processed requests	Workload level	Arrival rate distribution	Request size distribution	Execution time, time units	System dependability	Overloaded system	Processed requests by a Web Service (Maximum requests in the queue of the Web Service)			
								Web Service 1 (Queue)	Web Service 2 (Queue)	Web Service 3 (Queue)	Web Service 4 (Queue)
Case 1	51 out of 60	Low	Constant	Constant	105	0.85	Yes	N/A	N/A	21 (6)	30 (3)
Case 2	102 out of 120	High	Constant	Constant	206	0.85	Yes	N/A	N/A	42 (27)	60 (30)
Case 3	56 out of 60	Low	Uniform	Uniform	165	0.93	Yes	N/A	N/A	24 (7)	32 (3)
Case 4	112 out of 120	High	Uniform	Uniform	324.5	0.93	Yes	N/A	N/A	48 (29)	64 (32)

**Table 6.18:** Simulation results: Availability 50%, Accuracy 75%, the fastest service selection for all Web Services (Scenario B2, Experiment 5)

Services and not usage of the other services. However, when the accuracy of the QoS information decreases, the reasoning mechanism selects services which are not available and/or with high response time.

When the load balancing technique is applied and the accuracy of the QoS information is 100% (table 6.19) the execution times of the simulation runs are lower compared to the fastest selection technique. This result is observed since the load balancing technique tries to prevent the system from overloading as much as possible which leads to fewer requests waiting in the queues of the Web Services. However, when the models' accuracy drops down to 75% (table 6.20), the system experiences a high level of overloading which requires the size of the queues of the Web Services to be extended in order to run the experiments. This shows the importance of the accuracy of the QoS information.

When a more accurate selection technique is applied and the accuracy of the QoS criteria is 100% (table 6.21), the system has good overall performance. It deals better with Web Services overloading which suggests slightly increased execution time since not only the fastest Web Services

	Total # processed requests	Workload level	Arrival rate distribution	Request size distribution	Execution time, time units	System dependability	Overloaded system	Processed requests by a Web Service (Maximum requests in the queue of the Web Service)			
								Web Service 1 (Queue)	Web Service 2 (Queue)	Web Service 3 (Queue)	Web Service 4 (Queue)
Case 1	60 out of 60	Low	Constant	Constant	41	1	No	33 (0)	N/A	8 (0)	19 (0)
Case 2	120 out of 120	High	Constant	Constant	82	1	Yes	72 (0)	N/A	17 (2)	31 (6)
Case 3	60 out of 60	Low	Uniform	Uniform	54.5	1	No	27 (0)	N/A	17 (0)	16 (0)
Case 4	120 out of 120	High	Uniform	Uniform	90	1	Yes	55 (0)	N/A	23 (7)	42 (9)

**Table 6.19:** Simulation results: Availability 75%, Accuracy 100%, load balancing technique for all Web Services (Scenario B2, Experiment 6)

	Total # processed requests	Workload level	Arrival rate distribution	Request size distribution	Execution time, time units	System dependability	Overloaded system	Processed requests by a Web Service (Maximum requests in the queue of the Web Service)			
								Web Service 1 (Queue)	Web Service 2 (Queue)	Web Service 3 (Queue)	Web Service 4 (Queue)
Case 1	57 out of 60	Low	Constant	Constant	163.5	0.95	Yes	N/A	N/A	34 (19)	23 (0)
Case 2	114 out of 120	High	Constant	Constant	379.5	0.95	Yes	N/A	N/A	79 (64)*	35 (6)
Case 3	55 out of 60	Low	Uniform	Uniform	211.5	0.92	Yes	N/A	N/A	34 (16)	21 (0)
Case 4	120 out of 120	High	Uniform	Uniform	467.5	1	Yes	N/A	N/A	82 (61)*	38 (13)

**Table 6.20:** Simulation results: Availability 50%, Accuracy 75%, load balancing technique for all Web Services (Scenario B2, Experiment 7); \* - extended queue

	Total # processed requests	Workload level	Arrival rate distribution	Request size distribution	Execution time, time units	System dependability	Overloaded system	Processed requests by a Web Service (Maximum requests in the queue of the Web Service)			
								Web Service 1 (Queue)	Web Service 2 (Queue)	Web Service 3 (Queue)	Web Service 4 (Queue)
Case 1	60 out of 60	Low	Constant	Constant	48	1	No	36 (0)	N/A	0 (0)	24 (0)
Case 2	120 out of 120	High	Constant	Constant	106.5	1	Yes	72 (0)	N/A	20 (5)	28 (1)
Case 3	60 out of 60	Low	Uniform	Uniform	55	1	No	33 (0)	N/A	0 (0)	27 (0)
Case 4	120 out of 120	High	Uniform	Uniform	84	1	Yes	66 (3)	N/A	18 (0)	36 (3)

**Table 6.21:** Simulation results: Availability 75%, Accuracy 100%, more accurate selection technique for all Web Services (Scenario B2, Experiment 8)

are in use. This effect is observed when dealing with constant arrival rate or uniform-distributed low level arrival rate. However, when the arrival rate is uniform high level, the execution time of the simulation run is minimized – the load balancing leads to effective utilization of the resources and lower execution time of the simulation run. This means that the more accurate selection technique is appropriate when both Web Services response time and their load levels are important.

However, when the accuracy of the QoS information decreases to 75% and 50% (tables 6.22 and 6.23), the total number of processed requests and the system’s dependability decrease as well as load balancing cannot be achieved. This result is due to the nature of the selection technique – it is based on three QoS criteria of the Web Services: availability, response time, and load. Therefore, when the correctness of the information regarding these three criteria is not guaranteed, the system’s behavior is not predictable and the overall performance is low. Even more, the size of the queues of some of the Web Services has to be increased in order to deal with the high level of overloading. This result shows that the accuracy of the QoS information is crucial.

More accurate selection technique should be applied in the cases when the accuracy of the QoS criteria which are taken into consideration is guaranteed and the goal is to have a high level of overall performance in terms of execution time, overloading, and dependability.

## 6.2 Conclusions

This section provides conclusions regarding the conducted experiments which cover the extreme cases (scenarios A and B2), as well as the situation which is a combination of the boundary cases (scenario B1). The results of the conducted experiments provide an overview of the behavior of the system under the different conditions and known information about the services.

A summary of the applied selection techniques is presented in table 6.24, as follows:

- When nothing is known about the system, the Virtual Web Services Layer should not be



	Total # processed requests	Workload level	Arrival rate distribution	Request size distribution	Execution time, time units	System dependability	Overloaded system	Processed requests by a Web Service (Maximum requests in the queue of the Web Service)			
								Web Service 1 (Queue)	Web Service 2 (Queue)	Web Service 3 (Queue)	Web Service 4 (Queue)
Case 1	51 out of 60	Low	Constant	Constant	105	0.85	Yes	N/A	N/A	21 (6)	30 (3)
Case 2	96 out of 120	High	Constant	Constant	316.5	0.8	Yes	N/A	N/A	65 (50)*	31 (10)
Case 3	56 out of 60	Low	Uniform	Uniform	165	0.93	Yes	N/A	N/A	24 (7)	32 (3)
Case 4	108 out of 120	High	Uniform	Uniform	369	0.9	Yes	N/A	N/A	60 (41)*	48 (20)*

**Table 6.22:** Simulation results: Availability 50%, Accuracy 75%, more accurate selection technique for all Web Services (Scenario B2, Experiment 9); \* - extended queue

	Total # processed requests	Workload level	Arrival rate distribution	Request size distribution	Execution time, time units	System dependability	Overloaded system	Processed requests by a Web Service (Maximum requests in the queue of the Web Service)			
								Web Service 1 (Queue)	Web Service 2 (Queue)	Web Service 3 (Queue)	Web Service 4 (Queue)
Case 1	42 out of 60	Low	Constant	Constant	90.5	0.7	No	N/A	N/A	18 (3)	24 (0)
Case 2	82 out of 120	High	Constant	Constant	195.5	0.68	Yes	N/A	N/A	40 (25)	42 (15)
Case 3	40 out of 60	Low	Uniform	Uniform	75.5	0.67	No	N/A	N/A	13 (0)	27 (0)
Case 4	80 out of 120	High	Uniform	Uniform	161	0.67	Yes	N/A	N/A	28 (9)	52 (18)

**Table 6.23:** Simulation results: Availability 50%, Accuracy 50%, more accurate selection technique for all Web Services (Scenario B2, Experiment 10)

Information about the system		High level of dependability	Minimum overloading	Minimum execution time
None	-	Virtual Web Services Layer is not needed		
Web Services availability & Web Services models accuracy	Low	1 (X)	1 (~)	1 (~)
		2 (~)	2 (✓)	2 (✓)
		3 (~)	3 (~)	3 (~)
		4 (~)	4 (~)	4 (~)
	High	1 (X)	1 (X)	1 (X)
		2 (~)	2 (~)	2 (~)
		3 (~)	3 (~, ✓)	3 (✓)
		4 (~)	4 (✓)	4 (~)
<b>1 – Random selection; 2 – The fastest selection; 3 – Load balancing; 4 – More accurate selection</b> <b>✓ – The best choice; X – The worst choice; ~ – Similar;</b> <b>Low Web Services availability – 50% or less      Low Web Services models accuracy – 75% or less</b> <b>High Web Services availability – 75% or high      High Web Services models accuracy – 100%</b>				

**Table 6.24:** Simulation results: Selection Techniques Analysis

applied. Although, selection techniques which do not take any QoS parameters into consideration can be used (such as random selection or round-robin) they do not provide a specific behavioral pattern and the performance of the system depends on undefined characteristics.

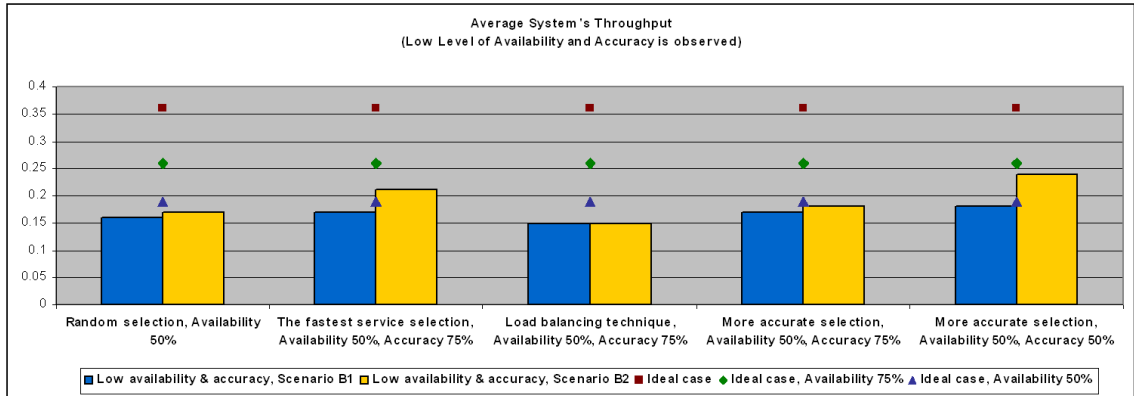
- The system's dependability is not directly related to any selection technique which takes into account QoS criteria. Therefore, conclusions regarding the best selection technique which should be applied in order to assure a high level of dependability when the Web Services availability and models accuracy are low cannot be made. The conducted experiments show that the fastest selection, the load-balancing, and the more accurate selection technique provide similar results. However, the random selection technique presents an unstable level of dependability due to the fact that it does not take into consideration any QoS criteria, and is considered as the worst technique in this situation.
- In terms of the system's overloading, the fastest service selection is the best choice when the availability and the models accuracy are low. This is due to the fact that both the load balancing technique and the more accurate selection technique are based on the current load of the Web Services. Since the accuracy of that information is decreased, the selection techniques are not able to take an appropriate decision and both selection techniques give similar results in terms of Web Services overloading. On the other hand, the fastest service selection takes into account the availability of the services and their response time and does not depend on the services' load. Therefore, this selection technique provides better load balancing than the load balancing technique. However, a conclusion regarding the worst selection technique in this situation cannot be made since the random selection, the load balancing, and the more accurate selection techniques present similar results in terms of Web Services overloading, i.e. they require extension of some of the queues of the services.
- When the goal is to minimize the execution time of the system, the fastest service selection shows the best results in the case when low level of availability and accuracy are observed. When the workload level is low, the more accurate selection technique provides the same results as the fastest service selection but cannot deal with high workload level in a graceful manner due to the overloading of the Web Services. On the other hand, a conclusion regarding the worst selection technique cannot be made since all other techniques provide similar bad results in terms of execution time when the services' availability and models accuracy are low.
- When the availability and the models accuracy are high, the dependability of the system is high and even reaches its maximum value - 1, for all selection techniques which take into account QoS criteria. This is an expected result, since a correct service selection can be done. However, when the availability drops down to 75%, the random selection technique provides lower dependability compared to the other techniques. Therefore, random selection

is considered as the worst choice in this situation. The other techniques - the fastest selection technique, the load balancing technique, and the more accurate technique show similar results in terms of system dependability, and therefore a conclusion regarding the best selection technique cannot be made.

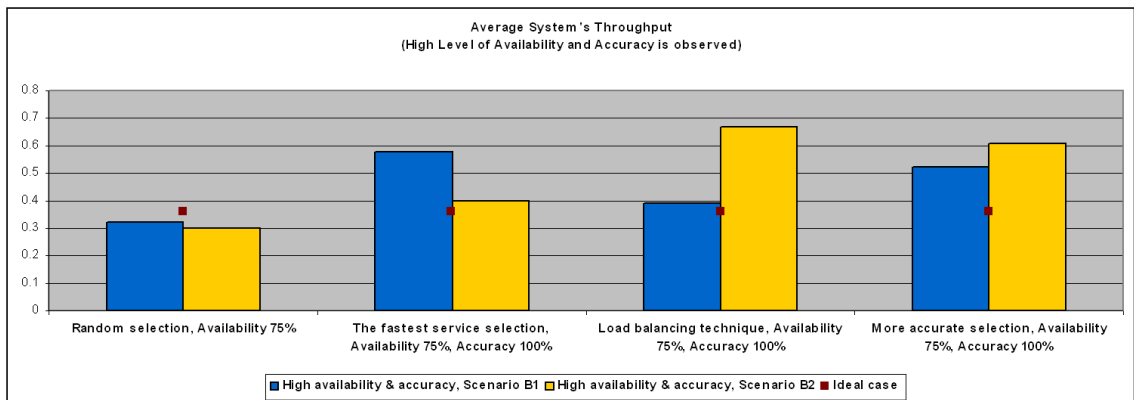
- In order to assure minimum overloading of the Web Services when the availability and the models accuracy have high values, the more accurate selection technique should be applied. Since this technique takes into account availability, response time, and load of the services, it is able to select the fastest available service which is not overloaded. If all available services from a group are overloaded, the one with minimum overloading is chosen. Therefore, the more accurate selection technique shows better results than the load balancing technique in terms of Web Services overloading and the load balancing technique is the second best technique in this situation. The worst technique is random selection due to its nature, i.e. it is not based on any QoS criteria.
- When the Web Services availability and the models accuracy have a high level and the goal is to minimize the execution time of the system, the load balancing technique shows the best results. The worst choice is the random selection technique.
- In terms of overall performance (dependability, overloading, and execution time), when the availability of the Web Services as well as the accuracy of the models are low, the fastest service selection should be applied. In contrast, when the availability and the correctness of predicted information about the Web Services are high, the load balancing technique as well as the more accurate selection technique should be used.

A comparison of the average system's throughput in the different experiments is presented in figure 6.3 for low level Web Services availability and models accuracy as well as in figure 6.4 for high level Web Services availability and models accuracy. Scenario B1 provides service selection for half of the Web Services, and therefore conclusions regarding the observed throughput cannot be made from the results of this scenario, since the number of processed requests is reduced due to unavailability of the given target Web Services. However, the results of both scenarios are presented for completeness.

When low level Web Services availability and accuracy is observed (figure 6.3), the overall system throughput in scenario B1 and B2 has lower values compared to the ideal case (without the Virtual Web Services Layer, scenario A). The reason for this result is the increased overall execution time due to the incorrect information about the Web Services. However, although the throughput of scenario A is higher, the performance is low since when the Web Services' availability decreases, the system's dependability decreases linearly, and therefore, the number of processed requests is reduced. On the other hand, in scenario B2 when the accuracy is 50%, the throughput is higher



**Figure 6.3:** System's Throughput, Low level of Web Services' availability and models accuracy



**Figure 6.4:** System's Throughput, High level of Web Services' availability and models accuracy

than the throughput when the accuracy is 75%. This result shows the importance of the accuracy of the QoS information: When the accuracy of the models' of the Web Services drops down, the selection technique is not able to select an appropriate service and as a result, the total number of processed requests decreases. In addition, the Web Services are less overloaded which helps reduce the execution time. Therefore, the observed throughput has a higher value, but the system's performance is lower.

When the Virtual Web Services Layer is used with high level Web Services availability and accuracy (figure 6.4), the overall system throughput and performance (in terms of dependability, overloading, and execution time) are better than scenario A. The figure shows that the random selection technique provides the worst throughput since it does not take into account any QoS criteria, and therefore the execution time to process the requests is increased. The throughput of scenario B1 with the fastest selection technique and the more accurate selection technique is higher than the other techniques of the same scenario, because the execution time of the simulation runs are lower and the same number of requests is processed, which increases the system's throughput. This result is observed since both selection techniques take into account the response time of the Web Services. However, in scenario B2, the load balancing technique and the more accurate selection technique give the highest throughput due to the fact that service selection is done for all requests and these techniques assure better execution time and/or less overloading of the system.

The conclusions below are broken into sub-sections, according to the independent variables of the conducted experiments, as follows:

- Known target Web Services;
- Web Services models accuracy;
- Web Services availability;
- Arrival rate and request size distribution;
- Workload levels;
- Selection techniques.

### **Known target Web Services**

The conducted experiments show that it is possible to combine two different systems into one – a system which does not require a service selection and a system which is based on dynamic selection of Web Services.

When half of the target Web Services are known (scenario B1), the Virtual Web Services Layer is not used for the requests which have predefined their target service. As a result, it is hard to

analyse the behavior of the system since both selection and redirecting are mixed together and conclusions regarding the most appropriate selection technique cannot be made.

However, when service selection is applied for all requests (scenario B2), behavioral patterns of the system are found and analyzed.

### **Web Services models accuracy**

The accuracy of the Web Services models has a significant impact on the behavior of the Virtual Web Services Layer. In particular, the predicted availability of the Web Services is crucial during the decision-making process. In case of the fastest service selection technique, if the fastest service from a particular group of redundant Web Services is not available, but the predicted information defines the Web Service as available, every time the service is selected as target Web Service, since it provides the lowest response time, but actually the request is never handled. As a result, the dependability of the system drops down. In contrast, if the fastest service from a group of redundant Web Services is available, but the predicted information considers it as not available, then the fastest service is never selected. As a result, the dependability of the system is not affected, but the execution time is increased.

The same conclusions can be applied to the load-balancing technique in terms of Web Services workload as well as to the more accurate selection technique in terms of Web Services response times and workload.

Therefore, assuring the correctness of the QoS criteria of the Web Services should have the highest priority when QoS techniques are taken into account during the decision-making process. If the accuracy of the predicted information cannot be guaranteed, selection techniques which take into consideration fewer criteria should be applied.

In addition, the results show that the accuracy is more important than the availability. That is, even when the availability is 50%, the system can continue working properly and assure a high level of dependability if the models accuracy of the Web Services is high.

### **Web Services availability**

The availability of the Web Services is crucial when the target service is predefined. As seen, when the availability drops down, the system's dependability decreases linearly. However, when the Virtual Web Services Layer is applied, the availability of the services does not play such an important role. As long as there is at least one available Web Service from a particular group of redundant Web Services, the system can continue working properly, only if selection techniques which consider availability are applied as well as the accuracy of this information is guaranteed.

### **Arrival rate and request size distribution**

The distribution of the size of the requests influences the behavior of the system in terms of execution time, services' overloading, and dependability. However, a more significant role is played by the accuracy of the QoS criteria and the applied selection technique. If the correctness of the Web Services' models is guaranteed, the dependability of the system is not changed when the arrival rate and the request size vary. In this situation, the execution time and the services' overloading is influenced, which is an expected result, since a different number of requests enters the system at a given moment of time requiring different processing time.

### **Workload levels**

The results show that it is important to observe the behavior of the system with low and high workload levels. They give an insight regarding the importance of the accuracy of the Web Services' models as well as the applied selection technique in terms of overloading and in which cases the overloading can be handled gracefully.

### **Selection techniques**

When the Virtual Web Services layer is applied, the choice of selection technique is important. Depending on the desired system's characteristics as well as the information which is available to the system and its correctness, different selection techniques should be used during the decision-making process. The results show that the random selection technique gives the worst overall system performance. The other techniques should be applied as follows:

- *To achieve a high level of dependability.* All selection techniques which take into consideration QoS criteria can be applied.
- *To achieve minimum overloading of the system.* If the Web Services' availability and accuracy is low, the fastest service selection gives minimum overloading of the system since this technique does not take into account the current load of the services. If the Web Services' availability and accuracy is high, the more accurate selection and the load balancing techniques should be applied since they take into account the load of the services and are able to select the service according to its overloading.
- *To achieve minimum execution time.* If the Web Services' availability and accuracy are low, the fastest service selection should be used since it assures minimum overloading of the system, which leads to better execution time. In contrast, when the Web Services' availability and accuracy are high, the load balancing technique gives the best system execution time since it avoids overloading as much as possible.

## CHAPTER 7

### CONCLUSIONS AND FUTURE WORK

#### 7.1 Conclusions

The domain of Web Services consists of redundant services provided by different parties. The services with the same functionality can be clustered into a group of redundant services. If a service offers more than one functionality, it belongs to more than one group. The variety of Web Services suggests the necessity of a mechanism that selects the most appropriate Web Service at a given moment of time. However, in most of the cases the consumers do not have information about the QoS of the services in order to do the service evaluation and selection. Even if such information is available to the clients, this increases their complexity and consumes additional resources. On the other hand, it is possible that some clients have different knowledge about the service than others, which can reflect on the correctness of the service selection.

In order to deal with redundant Web Services on the server side, virtualization of services is applied. A Virtual Web Services Layer is presented between the clients and the services, which is based on dynamic and transparent service selection and invocation. The layer hides the system's complexity from the clients and assures a level of security, since the consumers do not have direct access to the Web Services. Furthermore, the layer allows adding services with new functionality to the system at run-time, as well as various reasoning mechanisms can be applied in order to achieve load-balancing, a high level of dependability, minimum response time, and high overall performance, without modifying the other components of the system.

The evaluation of the Virtual Web Services Layer shows that such an architecture can be built using a set of standard programming languages and protocols. However, in order to avoid fluctuations of the experimental results due to particular machine specifications, programming languages, and protocols, as well as to observe a large range of system parameters, the layer can be evaluated using an existing simulation tool. The results of the experiments show that:

- The accuracy of the predicted QoS criteria has a big impact on the selected Web Service;
- The Virtual Web Services Layer helps improve the dependability of the system when a low level of Web Services availability is observed;



- The arrival rate and request size distribution impact the behavior of the system, but a more crucial role is played by the accuracy of the predicted QoS criteria and the applied selection technique;
- The low and high workload levels give insight regarding the importance of the accuracy of the predicted QoS criteria as well as the applied selection technique in terms of overloading;
- The choice of selection technique is important and depends on the desired system's characteristics as well as the information available to the system and its correctness:
  - To achieve a high level of system dependability, all selection techniques which take into account availability of the services can be applied;
  - To achieve minimum overloading of the system, the fastest service selection should be used when the Web Services' availability and accuracy is low, whereas the more accurate selection and the load balancing technique should be applied in case of high level of Web Services' availability and accuracy;
  - To achieve minimum execution time, the fastest service selection should be used when low level of Web Services' availability and accuracy is observed, whereas the load balancing technique should be applied in case of high level of Web Services' availability and accuracy of the predicted QoS criteria.

The suggested Virtual Web Services Layer can be applied according to the standard design pattern - Enterprise Service Bus (ESB). ESB is a complex and flexible conceptual solution for large-scale heterogeneous applications which require characteristics such as transactions, security, robustness, scalability, extensibility, messaging, routing, synchronization, integration, and others [6], [10]. This design pattern can be viewed as a set of several simpler design patterns embedded together, which can be implemented in various ways.

The Virtual Web Services Layer focuses on routing and can be seen as a subcomponent of ESB, which deals with the dynamic selection of the redundant services and can be embedded in a particular ESB implementation. In addition, the flexibility of the layer allows different selection strategies to be deployed at run-time according to the requirements of the system.

On the other hand, the Virtual Web Services Layer can be applied in the area of Multi-Agent systems. As discussed in [19], a MAS is a composition of autonomous agents which do not have a single point of control and agent replication is needed in order to assure a fault-tolerant application. The VWSL architecture can help managing redundant agents in a transparent manner taking into account different QoS criteria of the agents.

Finally, the proposed architecture can be used in Grid computing as well, whose focus is on virtualization for dynamic allocation of distributed resources. The problem with dynamic and

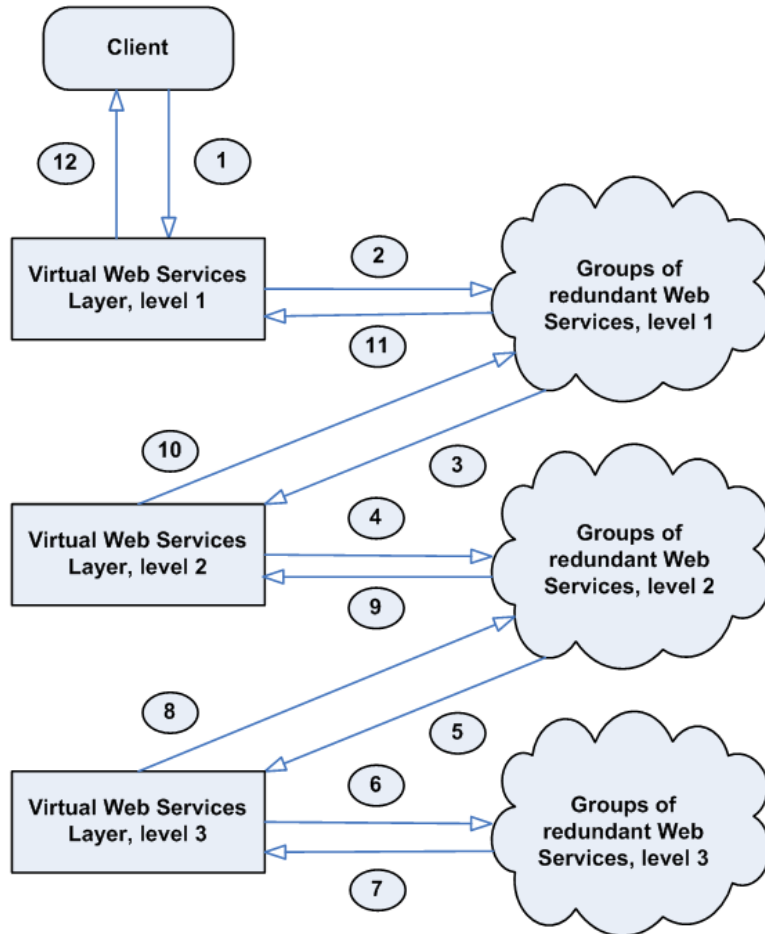
transparent selection of services exists in Grid computing and the Virtual Web Services Layer is an applicable solution which takes into account QoS criteria of the services. The layer can help improve dependability, response time, load balancing, and system overall performance, which is crucial for the Grid applications.

## 7.2 Future work

At this point, the Virtual Web Services Layer is evaluated for atomic Web Services, without taking into consideration composition of Web Services. However, the proposed architecture can be applied in composite services which consist of two or more atomic ones. Such a scenario should include transactions in order to assure consistency of the states of the system. Figure 7.1 shows the data flow in case of composite services which incorporate a Virtual Web Services Layer at every level of service composition. The client sends a request to the Virtual Web Services Layer at level 1 (step 1). The layer selects a composite service from one of the groups of redundant services, depending on the functionality of the request (step 2). The chosen service is invoked and when a request must be sent to another service, the request is sent to the Virtual Web Services Layer at level 2 (step 3), which selects the target Web Service (step 4). Again, when another service must be invoked during the execution of the selected service at level 2, the layer at the next level receives the request (step 5), which selects a Web Service at level 3 (step 6). In this example, the Web Services at level 3 are atomic, whereas the ones at levels 1 and 2 are composite. When the last request is processed by the atomic Web Service, the obtained result is returned back to the VWSL at level 3 (step 7), which forwards the result back to the selected service at the previous level (step 8). Once the service at level 2 completes its execution, it returns the calculated result to the layer at level 2 (step 9). Similarly, the layer at level 2 returns the result back to the selected service at level 1 (step 10). When the execution of the Web Service at level 1 is completed, it sends the final result to the layer at level 1 (step 11). Finally, the VWSL at level 1 returns the result back to the client (step 12).

Currently, the Virtual Web Services Layer has a potential single point of failure - the reasoning mechanism. In order to design the layer as a completely distributed architecture, a reasoning mechanism can be associated with every Virtual Web Service. As a result, a failure in one of the reasoning mechanisms would lead to a failure in one specific part of the system, but it would not lead to a failure in the entire application.

In future, other technologies that can be applied in the Virtual Web Services Layer are Semantic Web technologies, in particular ontologies. By describing the data in a machine-understandable manner and creating ontologies of QoS criteria, the decision-making process would be based on more features as well as their relationships would be represented in a better and more flexible way. This would allow applying more complex selection techniques in order to choose the appropriate



**Figure 7.1:** Future work: Composite Web Services and Virtual Web Services Layers

Web Service at a given moment of time. They can be based on a large range of QoS criteria such as availability, reliability, response time, trust and reputation, load, execution price as well as domain-specific criteria. Furthermore, the reasoning mechanism can be augmented in a way that it considers not only the QoS of the providers, but also the client side in terms of its importance, needs, and preferences.

## REFERENCES

- [1] XJ Technologies Company Ltd. Anylogic Enterprise Library Reference Guide. <http://www.xjtek.com/files/docs/en/EnterpriseLibraryReferenceGuide.pdf> Last access: 2007-06-18, (121 pages).
- [2] Y. Liu, S. Ngu, and L. Zeng. QoS Computation and Policing in Dynamic Web Service Selection. *WWW2004*, (8 pages), May 2004.
- [3] J. Day and R. Deters. Selecting the Best Web Service. *CASCON*, pages 293–307, October 2004.
- [4] A. Padovitz, S. Krishnaswamy, and S. Loke. Towards Efficient Selection of Web Services. *2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems*, (9 pages), July 2003.
- [5] E. Maximilien and M. Singh. Toward Autonomic Web Services Trust and Selection. *ICSOC'04*, pages 212–221, November 2004.
- [6] R. Robinson. Understand Enterprise Service Bus scenarios and solutions in Service-Oriented Architecture, Part 1. <http://www-128.ibm.com/developerworks/webservices/library/ws-esbscen/> Last access: 2007-06-18, June 2004.
- [7] Alexei Filippov. AnyLogic Technical Overview. <http://www.anylogic.jp/download/any5presentation.pdf> Last access: 2006-11-14, November 2003.
- [8] IBM Web Services Architecture team. Web Services architecture overview. <http://www-128.ibm.com/developerworks/web/library/w-ovr/?dwzone=web> Last access: 2007-06-18, (7 pages), 2000.
- [9] M. Papazoglou and D. Georgakopoulos. Service-Oriented Computing. *Communications of the ACM*, 46(10):25–28, October 2003.
- [10] M. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, and B. Kramer. Service-Oriented Computing Research Roadmap. <http://drops.dagstuhl.de/opus/volltexte/2006/524/pdf/05462.SWM.Paper.524.pdf> Last access: 2007-06-18, (29 pages), April 2006.
- [11] H. Kreger. Web Services Conceptual Architecture, WSCA 1.0. <http://www.cs.uoi.gr/zarras/mdw-ws/WebServicesConceptualArchitectu2.pdf> Last access: 2007-06-18, (41 pages), May 2001.
- [12] W3C. SOAP Version 1.2 Part 0: Primer. <http://www.w3.org/TR/2003/REC-soap12-part0-20030624> Last access: 2007-06-18.
- [13] Chris Peltz. Applying Design Issues and Patterns in Web Services. <http://www.devx.com/enterprise/Article/10397/1954?pf=true> Last access: 2007-06-18, (7 pages), 2005.
- [14] A. Tanenbaum and M. Steen. *Distributed Systems Principles and Paradigms*. Prentice Hall, 2002.
- [15] A. Avizienis, J. Laprie, and B. Randell. Fundamental Concepts of Dependability. *3rd Information Survivability Workshop, ISW2000*, pages 7–12, October 2000.

- [16] S. Ran. A Model for Web Services Discovery With QoS. *ACM 2003*, Volume 4(Issue 1):1–10, March 2003.
- [17] E. Maximilien and M. Singh. A Framework and Ontology for Dynamic Web Services Selection. *IEEE Internet Computing*, pages 84–93, September-October 2004.
- [18] S. Ludwig and S. Reyhani. Selection Algorithm for Grid Services based on a Quality of Service Metric. *21st International Symposium on High Performance Computing Systems and Applications (HPCS'07)*, (7 pages), 2007.
- [19] A. Fedoruk and R. Deters. Improving Fault-Tolerance by Replicating Agents. *AAMAS'02*, pages 737–744, July 2002.
- [20] A. Vidmar. BalanceNG: A simple approach to load balancing. <http://www.linux.com/articles/60871> Last access: 2007-08-09, March 2007.
- [21] H. Bryhni, E. Klovning, and O. Kure. A Comparison of Load Balancing Techniques for Scalable Web Servers. *IEEE Network*, pages 58–64, July-August 2000.
- [22] J. Silva and N. Mendonca. Dynamic Invocation of Replicated Web Services. *10th Brazilian Symposium on Multimedia and the Web 2nd Latin American Web Congress*, (8 pages), 2004.
- [23] S. Palmer. The Semantic Web: An Introduction. <http://infomesh.net/2001/swintro/> Last access: 2007-06-18, 2001.
- [24] E. Maximilien and M. Singh. A Framework and Ontology for Dynamic Web Services Selection. *IEEE Internet Computing*, pages 84–93, September-October 2004.
- [25] J. Hanson. Implementing an Enterprise Service Bus in Java. <http://www.devx.com/Java/Article/28127/1954?pf=true> Last access: 2007-06-18, 2005.
- [26] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. Grid Services for Distributed System Integration. *Computer*, pages 37–46, June 2002.
- [27] H. Vhecn, D. Guo, X. Qun-Wei, X. Luo, and W. Zhang. Service Virtualization in Large Scale, Heterogeneous and Distributed Environment. (7 pages).
- [28] D. Bunting, M. Chapman, O. Hurley, M. Little, J. Mischkinsky, E. Newcomer, J. Webber, and K. Swenson. Web Services Transaction Management (WS-TXM). <http://developers.sun.com/techtopics/webservices/wscaf/wstxm.pdf> Last access: 2006-11-14, (111 pages), July 2003.
- [29] Sun Microsystems. Java Sun. <http://java.sun.com/> Last access: 2007-06-18.
- [30] Sun Microsystems. Trail: The Reflection API. <http://java.sun.com/docs/books/tutorial/reflect/> Last access: 2007-08-09.
- [31] Dale Green. Trail: The Reflection API. <http://java.sun.com/docs/books/tutorial/reflect/index.html> Last access: 2007-06-18, 1995-2005.
- [32] The Apache Software Foundation. Apache Tomcat. <http://tomcat.apache.org/> Last access: 2007-06-18.
- [33] The Apache Software Foundation. Web Services - Axis. <http://ws.apache.org/axis/> Last access: 2007-06-18.
- [34] Sun Microsystems. JavaServer Pages Technology. <http://java.sun.com/products/jsp/> Last access: 2007-06-18.
- [35] Sun Microsystems. Java Servlet Technology. <http://java.sun.com/products/servlet/> Last access: 2007-06-18.

- [36] The Apache Software Foundation. Welcome to Apache Axis2. <http://ws.apache.org/axis2/>  
*Last access: 2007-06-18.*
- [37] R. Smith. Simulation Article. <http://www.modelbenders.com/encyclopedia/encyclopedia.html>  
*Last access: 2007-08-09, 1998.*
- [38] XJ Technologies Company. Anylogic. <http://www.xjtek.com/> *Last access: 2007-06-18.*
- [39] XJ Technologies Company Ltd. Anylogic Enterprise Library Tutorial.  
<http://www.xjtek.com/files/docs/en/EnterpriseLibraryTutorial.pdf> *Last access: 2007-06-18, (121 pages).*

# APPENDIX A

## VWSL PROTOTYPE

### A.1 Manager of the VWSL prototype

Figure A.1 shows the user interface of the Manager component of the Virtual Web Services Layer prototype. It is a client-server application that consists of two parts - VWSM-Client and VWSM. The figure represents a page that creates a Virtual Web Service. The meaning of the elements of the page is as follows:

- Field "Create a jws" means create a Virtual Web Service;
- Field "Policy" refers to a selection technique;
- Field "ID" refers to the unique name of the VWS;
- Field "Destination folder" specifies the folder where the created VWSL is stored;
- Link "Add a wsdl" adds a Web Service to a specified group of redundant Web Services;
- Link "Set VWS policy" specifies the selection technique that will be used in a particular Virtual Web Service;
- Link "Delete a wsdl" removes a Web Service from a particular group of redundant services;
- Link "List all wsdl" lists all Web Services in the system.

### A.2 Automatically generated Virtual Web Service in Java

Figure A.2 shows the source code of a generated Virtual Web Service by the Manager of the VWSL prototype system.



Virtual Web Service Manager						User: svetlana <a href="#">Logout</a>
<a href="#">[Home]</a>	<a href="#">[Create a jws]</a>	<a href="#">[Add a wsdl]</a>	<a href="#">[Set VWS policy]</a>	<a href="#">[Delete a wsdl]</a>	<a href="#">[List all wsdl]</a>	
<b>Create a jws</b>						
WSDL*	<input type="text" value="http://vama.usask.ca:8080/axis/Tests/HelloServer1.jws"/>					
Policy*	<input type="text" value="PolicyRandomSelection"/>					
ID	<input type="text" value="generatedJWS"/>					
Destination folder	<input type="text" value="C:/Tomcat/webapps/axis/Tests/Research/Applications/wsdl2jws/jws/"/>					
<input type="button" value="Create"/>						
2005 All rights reserved, Svetlana Slavova, MADMUC LAB						

Figure A.1: Virtual Web Services Manager GUI

```

import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import javax.xml.namespace.QName;
import java.util.Hashtable;
import java.util.Enumeration;
import java.util.Vector;
import java.io.*;

public class generatedJWS
{
    public java.lang.String Hello()
    {
        try
        {
            String endpoint = "";
            Hashtable sharedHashTable = new Hashtable();
            //Read the hash table from a file
            try
            {
                String file = "C:/Tomcat/webapps/axis/Tests/Research/Applications/wsdl2jws/src/sharedVariables/HashTable";

                //Create a stream for reading
                FileInputStream fis = new FileInputStream(file);

                //Create an object that can read from that file
                ObjectInputStream inStream = new ObjectInputStream(fis);

                //Retrieve the Serializable object
                sharedHashTable = (Hashtable) inStream.readObject();

                //Close the stream
                inStream.close();
            }
            catch (Exception ex)
            {
                System.out.println("generatedJWS: Error: in reading the hash table from a file");
                System.out.println(ex);
            }

            String policyPackageAndClass = "";
            //Get the name of the policy from the hash table
            for (Enumeration e = sharedHashTable.keys(); e.hasMoreElements();)
            {
                String key = e.nextElement().toString();
                if (key.equals("generatedJWS"))
                {
                    //The id is found in the hash table
                    Vector temp[] = (Vector[]) sharedHashTable.get(key);
                    policyPackageAndClass = (String) temp[1].elementAt(0);
                }
            }

            Policies.PolicyInterface policyInterface = (Policies.PolicyInterface) Class.forName(policyPackageAndClass).newInstance();
            endpoint = policyInterface.selectWebService("generatedJWS");

            Service service = new Service();
            Call call = (Call) service.createCall();
            call.setTargetEndpointAddress(new java.net.URL(endpoint));
            call.setOperationName(new QName(endpoint, "Hello"));
            return (java.lang.String) call.invoke( new Object[] { } );
        }
        catch(Exception ex)
        {
            System.out.println("VWS generatedJWS: Error");
            System.out.println(ex);
            return null;
        }
    }
}

```

Figure A.2: Example of a generated Virtual Web Service in Java

# APPENDIX B

## CLIENTS LOAD GENERATOR AND SIMULATOR

Figures B.1, B.2, and B.3 show some GUI components of the Clients Load Generator and Simulator. Figure B.4 represents a log file that is generated during the clients' simulation.

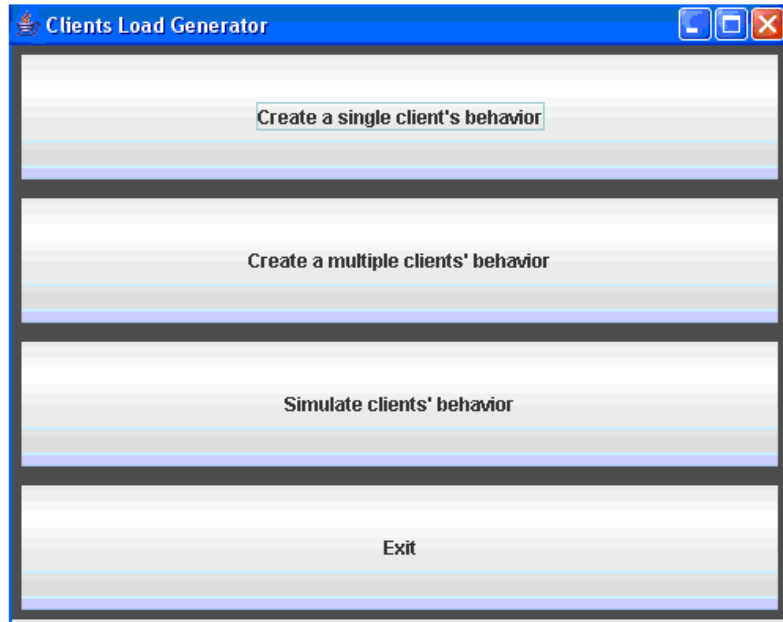


Figure B.1: Clients Load Generator and Simulator: Main window

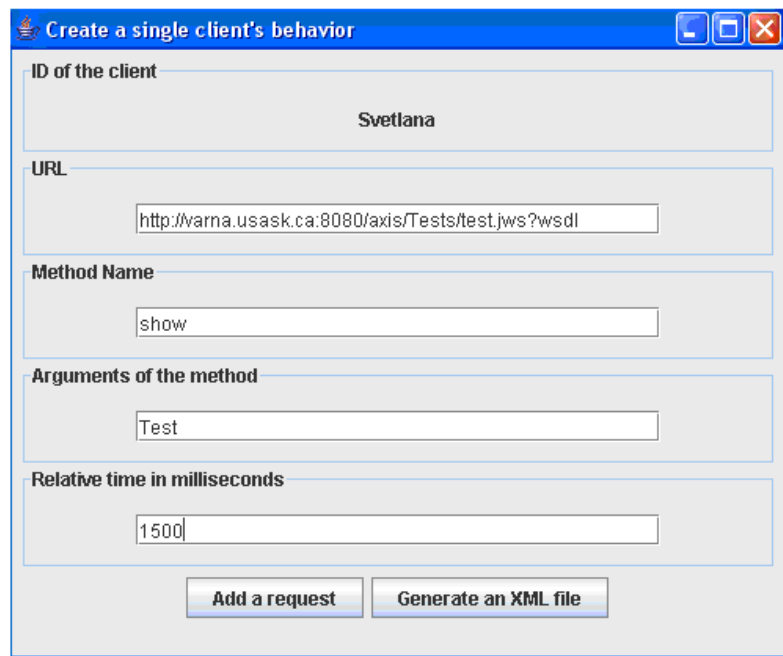


Figure B.2: Clients Load Generator and Simulator: Create single client's behavior

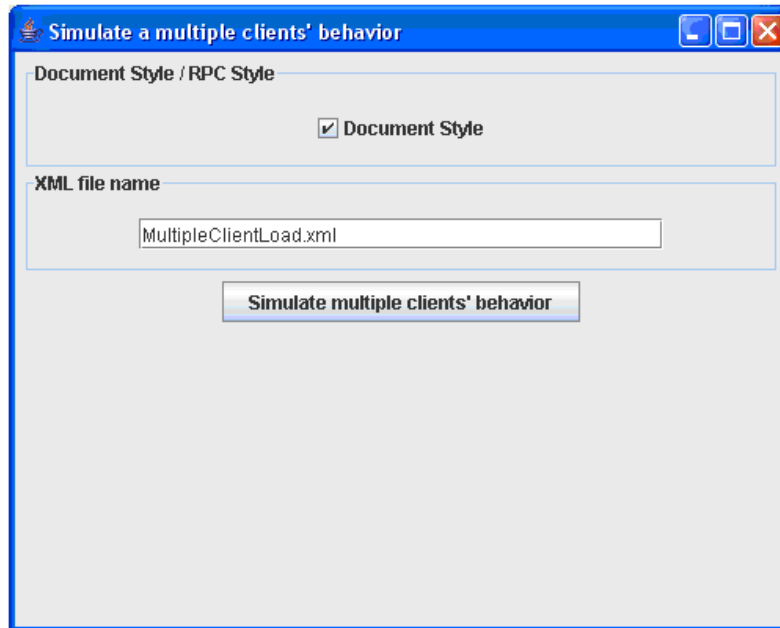


Figure B.3: Clients Load Generator and Simulator: Simulate multiple clients' behavior

No	ClientID	RequestID	Exec(ms)	Start time	End time	Result	URL	Method
1	Svetlana1	number_1	1157	Tue May 09 12:56:56 CST 2006	Tue May 09 12:56:57 CST 2006	Hello test	http://vacna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
2	Svetlana1	number_1	844	Tue May 09 12:57:01 CST 2006	Tue May 09 12:57:01 CST 2006	Hello test	http://vacna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
3	Svetlana1	number_1	1015	Tue May 09 12:57:05 CST 2006	Tue May 09 12:57:06 CST 2006	Hello test	http://vacna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
4	Svetlana1	number_1	985	Tue May 09 12:57:12 CST 2006	Tue May 09 12:57:13 CST 2006	Hello test	http://vacna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
5	Svetlana1	number_1	954	Tue May 09 12:57:17 CST 2006	Tue May 09 12:57:18 CST 2006	Hello test	http://vacna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
6	Svetlana1	number_1	1016	Tue May 09 12:57:21 CST 2006	Tue May 09 12:57:22 CST 2006	Hello test	http://vacna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
7	Svetlana1	number_1	1016	Tue May 09 12:57:25 CST 2006	Tue May 09 12:57:26 CST 2006	Hello test	http://vacna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
8	Svetlana1	number_1	1031	Tue May 09 12:57:29 CST 2006	Tue May 09 12:57:30 CST 2006	Hello test	http://vacna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
9	Svetlana1	number_1	1000	Tue May 09 12:57:32 CST 2006	Tue May 09 12:57:33 CST 2006	Hello test	http://vacna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
10	Svetlana1	number_1	953	Tue May 09 12:57:37 CST 2006	Tue May 09 12:57:38 CST 2006	Hello test	http://vacna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
11	Svetlana1	number_1	1047	Tue May 09 12:57:41 CST 2006	Tue May 09 12:57:42 CST 2006	Hello test	http://vacna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
12	Svetlana1	number_1	969	Tue May 09 12:57:46 CST 2006	Tue May 09 12:57:47 CST 2006	Hello test	http://vacna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
13	Svetlana1	number_1	1000	Tue May 09 12:57:50 CST 2006	Tue May 09 12:57:51 CST 2006	Hello test	http://vacna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
14	Svetlana1	number_1	1000	Tue May 09 12:57:54 CST 2006	Tue May 09 12:57:55 CST 2006	Hello test	http://vacna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
15	Svetlana1	number_1	1140	Tue May 09 12:57:59 CST 2006	Tue May 09 12:58:00 CST 2006	Hello test	http://vacna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
16	Svetlana1	number_1	984	Tue May 09 12:58:03 CST 2006	Tue May 09 12:58:04 CST 2006	Hello test	http://vacna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
17	Svetlana1	number_1	1015	Tue May 09 12:58:07 CST 2006	Tue May 09 12:58:08 CST 2006	Hello test	http://vacna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
18	Svetlana1	number_1	1031	Tue May 09 12:58:11 CST 2006	Tue May 09 12:58:12 CST 2006	Hello test	http://vacna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
19	Svetlana1	number_1	937	Tue May 09 12:58:18 CST 2006	Tue May 09 12:58:19 CST 2006	Hello test	http://vacna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
20	Svetlana1	number_1	953	Tue May 09 12:58:23 CST 2006	Tue May 09 12:58:24 CST 2006	Hello test	http://vacna.usask.ca:8080/axis/Tests/test.jws?wsdl	show

Figure B.4: Clients Load Generator and Simulator: Log file, created during the clients' simulation