EARLY ASSESSMENT OF SERVICE PERFORMANCE USING SIMULATION

TESI DI DOTTORATO DI RICERCA DI:
SAGBO KOUESSI ARAFAT ROMARIC

Matr. R09066

RELATORE
Prof. Ernesto Damiani
Università degli Studi di Milano, Italy

CORRELATORI
Dr. Claudio A. Ardagna
Università degli Studi di Milano, Italy
Prof. Karl Reed
La Trobe University, Australia

DIRETTORE DELLA SCUOLA DI DOTTORATO
Prof. Ernesto Damiani

Anno Accademico 2012 – 2013
To my parents for their support since the beginning of my studies.

A toi, Oriane Graziella, la lueur qui a illuminé ma vie durant cette rédaction et à mon épouse Pélagie pour son infaillible soutien perpétuel et ses nombreux sacrifices.
Abstract

Early Assessment of Service Performance using Simulation

The success of web services is changing the way in which software is designed, developed, and distributed. The increasing diffusion of software in the form services, available as commodities over the Internet, has enabled business scenarios where processes are implemented by composing loosely-coupled services chosen at runtime. Services are in fact continuously re-designed and incrementally developed, released in heterogeneous and distributed environments, and selected and integrated at runtime within external business processes. In this dynamic context, there is the need of solutions supporting the evaluation of service performance at an early stage of the software development process, or even at design time, to support users in an a priori evaluation of the impact, a given service might have when integrated in their business process. A number of performance verification and validation techniques are proposed to test and simulate web services, but they assume the availability of service code or at least of reliable information (e.g., collected by testing) on service behavior. Among these approaches, simulation-based techniques are mostly used to assess the behavior of the service and predict its behavior using historical data. Despite the benefits of such solutions, few proposals have addressed the problem of how service performance can be assessed at design time and how historical data can be replaced by simulation data for performance evaluation at early stage of development cycle.

In this thesis, the notion of simulation is fully integrated within early phases of the software development process in order to predict the behavior of services. We propose model-based approaches that rely on the amount of
information available for the simulation of the performance of service operations. We distinguish full-knowledge, partial-knowledge and zero-knowledge scenarios. In a full-knowledge scenario, the total execution times for each operation and the internal distributions of delays are known and used for performance evaluation. In a partial-knowledge scenario, partial testing results (i.e., the lower and upper bounds to the operation execution times) are used to simulate a service performance. In the zero-knowledge scenario, no testing results are considered; only simulation results are used for performance evaluation.

The main contributions of this thesis can be summarized as follows.

Firstly, we proposed a model-based approach that relies on Symbolic Transition System (STS) to describe the web services as finite state automata and evaluate their performance. This model was extended for testing and simulation. The testing model annotates model transitions with performance idioms, which allow to evaluate the behavior of the service. The simulation model extends the standard STS-based model with transition probabilities and delay distributions. This model is used to generate a simulation script that allows to simulate the service behavior. Our methodology used simulation along the design and pre-deployment phases of the web service lifecycle to preliminarily assess web service performance using coarse-grained information on the total execution time of each service operation derived by testing. We used testing results and provided some practical examples to validate our methodology and the quality of the performance measurements computed by simulation considering the full-knowledge and partial-knowledge scenarios. The results obtained showed that our simulation gives accurate estimation of the execution times.

Secondly, the thesis proposed an approach that permits service developers and software adopters to evaluate service performance in a zero-knowledge scenario, where testing results and service code are not yet available. Our approach is built on expert knowledge to estimate the execution time of the service operation. It evaluates the complexity of the service op-
eration using the input and output Simple Object Access Protocol (SOAP) messages, and the Web Service Description Language (WSDL) interface of the service. Then, the operation interval of execution times is estimated based on profile tables providing the time overhead needed to parse and build SOAP messages, and the performance inferred from the testing of some reference service operations. Our simulation results showed that our zero-knowledge approach gives an accurate approximation of the interval of execution times when compared with the testing results at the end of the development.

Thirdly, the thesis proposed an application of our previous approaches to the definition of a framework that allows to negotiate and monitoring the performance Service Level Agreement (SLA) of the web service based on the simulation data. The solution for SLA monitoring is based on the STS-based model for testing and the solution for SLA negotiation is based on the service model for simulation. This work provides an idea about the SLA of the service in advance and how to handle the violations of the SLA on performance after the service deployment.

**Keywords:** SOA, web service, WSDL, STS model, Performance evaluation, Early performance assessment, Zero-knowledge, Partial-knowledge, Full-knowledge, Complexity
Résumé

Evaluation précoce de service de performance en utilisant la simulation

Le succès des services web est entrain de changer la façon dont le logiciel est conçu, développé et distribué. La diffusion croissante des logiciels sous forme de services, disponibles en tant que produits sur Internet, a permis la définition de scénarios d’entreprise où les processus sont mis en œuvre par la composition de services faiblement couplés, choisis au moment de l’exécution. Les services sont en effet en permanence re-conçus et développés progressivement, publiés dans des environnements hétérogènes et distribués, et sélectionnés et intégrés à l’exécution dans les processus externes d’entreprise. Dans ce contexte dynamique, il est nécessaire d’avoir des solutions permettant l’évaluation de la performance du service à un stade précoce du processus de développement des logiciels, ou encore au moment de la conception, afin de permettre aux utilisateurs de faire une évaluation “a priori” de l’impact qu’un service donné peut avoir quand il est intégré dans leur processus d’entreprise. Un certain nombre de techniques de vérification et de validation des performances utiles sont proposées pour tester et simuler les services web, mais elles requièrent la disponibilité du code source du service ou au moins d’informations fiables (par exemple, recueillies par test) sur le comportement du service. Parmi ces approches, les techniques basées sur la simulation sont principalement utilisées pour évaluer le comportement du service et prédire son comportement en utilisant des données obtenues par test. Malgré les avantages de ces solutions, peu de propositions ont abordé le problème lié à la manière dont la performance du service peut être évaluée au moment de la conception et comment
les données de test peuvent être remplacées par les données de simulation en vue de l’évaluation de la performance à un stade précoce du cycle de développement.


Les principales contributions de cette thèse peuvent être résumées comme suit.

Premièrement, nous avons proposé une approche basée sur l’utilisation de modèle qui s’appuie sur le Système de Transition Symbolique (STS) pour décrire les services web comme des automates à états finis et évaluer leur performance. Ce modèle a été étendu pour les tests et la simulation. Le modèle de test ajoute aux transitions du modèle STS standard des idiomes de performance, qui permettent d’évaluer le comportement du service. Cependant, le modèle de simulation étend le modèle STS standard avec des probabilités de transition et les distributions de délais. Ce modèle est utilisé pour générer un script de simulation permettant de simuler le comportement du service. Notre méthodologie utilise la simulation tout au long des phases de conception et de pré-déploiement du cycle de vie des services web pour une évaluation préliminaire de la performance des services
web en utilisant les informations brutes sur le temps total d’exécution de chaque opération du service web provenant des tests. Nous avons utilisé les résultats des tests et fourni des exemples concrets pour valider notre méthodologie et la qualité des mesures de performance obtenues par simulation en considérant les scénarios full-knowledge et partial-knowledge. Les résultats obtenus ont montré que notre simulation donne une estimation précise des temps d’exécution.


Troisièmement, cette thèse propose une application de nos précédentes approches pour la mise en place d’un framework qui permet de négocier et de surveiller le contrat de niveau de service (SLA) sur la performance du service web en se basant sur les données de simulation. La solution pour le suivi du contrat de niveau de service est basée sur le modèle STS étendu pour le test et la solution de négociation du niveau de service est basée sur le modèle de service étendu pour la simulation. Ce travail fournit à l’avance une idée sur le contrat de performance du service et la façon dont
les violations du contrat sont traitées après le déploiement du service web.

Mots clés : SOA, service web, WSDL, Modèle STS, Evaluation de performance, Evaluation précoce de la performance, Zero-knowledge, Partial-knowledge, Full-knowledge, Complexité
Acknowledgments

I would like to thank all those who have helped, guided, and inspired me throughout this process.
First of all, I would like to sincerely thank my supervisor, Ernesto Damiani and co-supervisors Claudio A. Ardagna and Karl Reed. It has been an honor for me to work with them on my research topic during my Ph.D studies along these last three years in the University of Milan. Their constant guidance helps me to reach the numerous goals initially defined for my Ph.D thesis work.
I would like to thank Gabriele Gianini, for many useful discussion we have during my work about the validation of my experimental results.
I would like to thank Stelvio Cimato and the other members of the SESAR Lab in particular Fulvio Frati and Olga Scotti for their multiple help during my stay.
I would like to thank Kokou Yetongnon for giving to me, the opportunity to stay in the University of Burgundy (Université de Bourgogne) in France for three months in the LE2I Laboratory. Thank for his numerous advices to improve my work.
I would like to thank Patrick C. K. Hung (University of Ontario (Canada)), Antonino Sabetta (SAP Labs (France) and Lionel Brunie (University of Lyon (France)), for having dedicated their precious time to review the thesis, whose valuable feedback helped improving the presentation and the scientific content of this work.
I would like to thank all the participants of the different editions of the Multimedia Distributed Pervasive Systems (MDPS) workshops in Passau (Germany), Lyon (France), Crema (Italy) and Sicily (Italy) for the inspiring ideas and great discussions we have during these brainstorming weeks.
Thanks to Harald Kosch, David Coquil and Nadia Bennani.
I would like to thank my wife for his constant support, guidance, help and encouragement over the years, and the born of my daughter at the end of this thesis which made my Ph.D experience so special and helped me to go ahead.
Last, I would like to express my gratitude to my family for their teaching, their support and their love which help me during this stay abroad. It was really difficult to live far from them.
Merci à vous tous.
Grazie a tutti.
# Contents

Abstract iv  
Résumé vii  
Acknowledgments xi  
List of abbreviations xx  

## Part I Introduction and State of art  

<table>
<thead>
<tr>
<th>Chapter I</th>
<th>Introduction and research questions</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>I.1</td>
<td>Introduction</td>
<td>3</td>
</tr>
<tr>
<td>I.2</td>
<td>Motivations</td>
<td>5</td>
</tr>
<tr>
<td>I.3</td>
<td>Contributions</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>I.3.1 Model-based early assessment of service performance: Full-knowledge and Partial-knowledge scenarios</td>
<td>6</td>
</tr>
<tr>
<td>I.3.2</td>
<td>Model-based early assessment of service performance: Zero-knowledge scenario</td>
<td>6</td>
</tr>
<tr>
<td>I.3.3</td>
<td>Services SLA on performance negotiation and/or monitoring using simulated data</td>
<td>7</td>
</tr>
<tr>
<td>I.4</td>
<td>Structure of the thesis or Overview of the thesis</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter II</th>
<th>State of the art</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>II.1</td>
<td>Web Services and related technologies</td>
<td>9</td>
</tr>
<tr>
<td>II.1.1</td>
<td>Service-Oriented Architecture (SOA)</td>
<td>9</td>
</tr>
<tr>
<td>II.1.2</td>
<td>Web Service</td>
<td>10</td>
</tr>
<tr>
<td>II.1.3</td>
<td>Simple Object Access Protocol (SOAP)</td>
<td>10</td>
</tr>
<tr>
<td>II.1.4</td>
<td>Representational State Transfer (REST)</td>
<td>11</td>
</tr>
<tr>
<td>II.1.5</td>
<td>Web Services Description Language (WSDL)</td>
<td>11</td>
</tr>
<tr>
<td>II.1.6</td>
<td>Universal Description, Discovery and Integration (UDDI)</td>
<td>12</td>
</tr>
<tr>
<td>II.2</td>
<td>Web service QoS guarantees: Service Level Agreement (SLA)</td>
<td>12</td>
</tr>
<tr>
<td>II.2.1</td>
<td>SLAs components</td>
<td>14</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>VI.2.4</td>
<td>Services performance prediction</td>
<td>92</td>
</tr>
<tr>
<td>VI.2.5</td>
<td>Extension to other service models</td>
<td>92</td>
</tr>
<tr>
<td>VI.2.6</td>
<td>Simulation scripts generation according to the load</td>
<td>92</td>
</tr>
<tr>
<td>VI.2.7</td>
<td>Solution for the interference problem in service composition</td>
<td>92</td>
</tr>
<tr>
<td>VI.2.8</td>
<td>Move our solution to Cloud</td>
<td>92</td>
</tr>
</tbody>
</table>

**Publications**

93

**Bibliography**

96

**Random number generator WSDL file**

107

**Medical Meeting Management WSDL file**

109

**Standard STS Model for Medical Meeting Management service**

115

**Complete Java Class for operation CreditAdd performance simulation**

121
## List of Figures

| Figure II.1 | An example of SOAP message | 11 |
| Figure II.2 | An example of WSDL description | 13 |
| Figure II.3 | Basic example of service discovering process | 14 |
| Figure II.4 | Service Level Agreement Life Cycle in six steps | 15 |
| Figure II.5 | Traditional waterfall software development life cycle | 17 |
| Figure II.6 | Basic continuous improvements software development life cycle | 17 |
| Figure III.1 | Performance evaluation framework | 30 |
| Figure III.2 | Traditional waterfall software development life cycle extended with simulation step | 31 |
| Figure III.3 | Basic continuous improvements software development life cycle extended with simulation step | 31 |
| Figure III.4 | An example of STS-based model for the IFX Reverse ATM service | 33 |
| Figure III.5 | An example of STS-based model for testing (STSₜ) | 35 |
| Figure III.6 | An example of STS-based model for simulation (STSₛ) | 36 |
| Figure III.7 | Fragment of STS-based model for simulation with loop | 37 |
| Figure III.8 | Fragment of STS-based model for simulation after loop unroll | 37 |
| Figure III.9 | A fragment of the XML encoding for STSₜ in Figure III.5 | 39 |
| Figure III.10 | A fragment of the XML encoding for STSₛ in Figure III.6 | 40 |
| Figure III.11 | An example of performance interceptor annotation | 41 |
| Figure III.12 | An example of performance interceptor class | 42 |
| Figure III.13 | Algorithm for simulation script generation | 43 |
| Figure III.14 | Java-based simulation script for operation `DebitAdd` | 44 |
| Figure III.15 | Java-based simulation script for operation `CreditAdd` | 45 |
| Figure III.16 | Interface of the framework STS2Java | 46 |
| Figure III.17 | Interface for STS-based simulation model selection | 47 |
| Figure III.18 | Interface for simulation script generation | 47 |
| Figure III.19 | Interface of the Plugin STS2Java for Netbeans | 48 |
| Figure III.20 | Interface for STS-based simulation model selection | 48 |
| Figure III.21 | Interface for simulation script generation | 49 |
| Figure III.22 | Test-based execution times for tc₁ varying rps | 50 |
| Figure III.23 | Test-based execution times for tc₂ varying rps | 51 |
| Figure III.24 | Test-based execution times for tc₃ varying rps | 51 |
List of Tables

Table III.1  Performance Idioms ............................................. 34
Table III.2  Mean and standard deviation of execution times ............. 52
Table IV.1   Classification of XML datatypes .............................. 60
Table IV.2   Scores associated to datatypes complexity .................. 60
Table IV.3   Scores associated to resources complexity .................. 61
Table IV.4   Class of complexity and factor $\gamma$ based on OC ........... 62
Table IV.5   Characteristics of our sample web services .................. 63
Table IV.6   Complexity evaluation for web service 1 operations ........ 64
Table IV.7   Profile tables for DOM APIs .................................. 65
Table IV.8   Values defined for the data-intensive factor ................ 68
Table IV.9   Parameters for the basic and middle classes of complexity ... 70
Table IV.10  Interval of execution times computed for operations in $WS_1$ and $WS_2$ ................................................................. 72
Table IV.11  Intervals of execution times estimated for other service operations 73
Table IV.12  Statistical analysis of the results ............................ 74
Abbreviations and Meaning

ATM  Automatic Teller Machine
BPEL4WS  Business Process Execution Language for Web Services
BPMN  Business Process Modeling Notation
EJB  Enterprise Java Bean
FSA  Finite State Automaton
FSM  Finite State Machine
HMM  Hidden Markov Model
IFX  Interactive Financial Exchange
KLAPER  Kernel LAnguage for PErformance and Reliability
LTS  Labelled Transition Systems
MDA  Model Driven Architecture
MDD  Model-Driven Development
MOF  Meta Object Facility
OMG  Object Management Group
PN  Petri Nets
PUMA  Performance by Unified Model Analysis
PUPPET  Pick UP Performance Evaluation Test-bed
QN  Queuing Network
QoS  Quality of Service
REST  Representational State Transfer
RFC  Request for Comments
SLA  Service Level Agreement
SLC  Software Life Cycle
SOA  Service Oriented Architecture
SOAP Simple Object Access Protocol
STS  Symbolic Transition System
TA  Timed Automata
TPN  Time Petri Nets
UDDI  Universal Description, Discovery and Integration
UML  Unified Modeling Language
URI  Uniform Resource Identifier
W3C  World Wide Web Consortium
WSCL Web Service Conversation Language
WSDL  Web Service Description Language
XML eXtensible Markup Language
Part I

Introduction and State of art

This part of the thesis is composed of two chapters. Chapter I defines the motivations and research questions tackled by the thesis and underlines the original contributions presented in it. Subsequently, it outlines the thesis structure. Chapter II introduces the background of our work and presents an overview of the relevant approaches related to our work and summaries the current state of the research on methods used to assess the performance of the software/service.
Chapter I

Introduction and research questions

Contents

| I.1 | Introduction | 3 |
| I.2 | Motivations   | 5 |
| I.3 | Contributions | 6 |
| I.3.1 | Model-based early assessment of service performance: Full-knowledge and Partial-knowledge scenarios | 6 |
| I.3.2 | Model-based early assessment of service performance: Zero-knowledge scenario | 6 |
| I.3.3 | Services SLA on performance negotiation and/or monitoring using simulated data | 7 |
| I.4 | Structure of the thesis or Overview of the thesis | 7 |

I.1 Introduction

The increasing diffusion of web services is changing the way in which software is designed, developed, and distributed. Despite the success of web services, there are still some open issues that highly affect their widespread adoption especially in critical scenarios where services, supplied by potentially unknown providers, are selected at runtime. Effective and easy to use methodologies should be provided to increase the trustworthiness of services and to guarantee their non-functional properties such as performance, reliability, scalability and security.

In this context, the need of techniques for making accurate the estimation of service performance at design time has become crucial [1], since poor performance discovered after service deployment, can have catastrophic implications. Customers are in fact concerned about the performance of the services they integrate as part of their system, while developers would like to evaluate the performance of their services at an early stage of the service lifecycle. Then, with the emergence of model-driven development, a performance analysis step can be added early to the software development cycle [2, 3] to analyse the non-functional properties and better evaluate the software/services behavior.
Existing approaches for performance evaluation [4, 5, 6, 7] assume the availability of service code or at least of reliable information (e.g., collected by testing) on service behavior. As a consequence, these approaches do not support design time evaluation of service performance without the use of historical data. Based on the amount of information available at design time, we refer to these scenarios as “full-knowledge” and “partial-knowledge” scenarios.

Although model-driven development may support some degree of performance analysis during development, the problem of evaluating service performance is exacerbated by the fact that service code may not be available to or under the control of the party responsible for the evaluation. We refer to this scenario as “zero-knowledge” scenario, where performance evaluation relies on estimation of service behavior and characteristics. Existing estimation models to forecast the cost, size, resource effort, duration or performance of software projects [8, 9, 10, 11, 12] are mainly based on expert analysis, and rely on analogy and statistical methods using historical data. Performance analysis can be carried out by measurement or by modeling techniques, but at early stages, modeling approaches are preferable to develop better abstractions for a model and then permit to study the behavior of the software or the service, using analytical and/or simulation techniques. Most of the existing research activities are based on analytical models [13, 14]. These works only simplify the real service and are not then suitable when we want to assess the behavior of the real service. The performance evaluation approach developed in this thesis allows a more detailed model to be constructed and is less restrictive than existing approaches [9, 10, 11, 12, 14]. The authors of the PUPPET (Pick UP Performance Evaluation Test-bed) approach [15] propose similar model-based solution for functional testing and performance testing. They provide a tool for services integration that support the QoS evaluation of services which are still under development. In order to overcome the drawbacks of existing solutions, we first propose a model-based solution to integrate an early performance analysis steps into the development cycle. This methodology uses simulation during the design and pre-deployment phases of the web service lifecycle to preliminarily assess web service performance. Our technique relies on coarse-grained information on the total execution time of each service operation which is derived by testing, followed by random guesses on the delay introduced by each internal task composing the operation. This solution refers to a full-knowledge and partial-knowledge scenarios.

Secondly, we extend our previous solution to consider the performance evaluation in the zero-knowledge scenario. In particular, our approach is aimed at simulating service performance when no information on real service/operation execution time is available. Our approach estimates the interval of operation execution times for the service from the characteristics of the XML (eXtensible Markup Language) encoding of its input and output parameters and the WSDL interface of the service by using expert knowledge. This information is used to simulate the performance of a given service.

The model-based approaches proposed in this thesis rely on Symbolic Transition Sys-
tems (STSs) [16] to describe web services as finite state automata and evaluate their performance. From the standard STS-based model of the service, we generate two extended models: i) a testing model, which is used to automatically generate monitoring code (i.e., performance interceptors) for the evaluation of the service performance; ii) a simulation model, which is used to generate a simulation script to forecast the service behavior. Our solutions are however not limited to STS-based models and are suitable for any service modeling approach that supports the enrichment of state transitions with annotations.

In this thesis, we provide experimental evaluation of our approaches using both services obtained from the Internet and services developed in-house. We also show the application of our solutions to negotiate and/or evaluate SLAs on service performance, and to select services at runtime on the basis of their claimed performance in the service compositions process.

The following sections present our motivation and research questions, and describe briefly our contributions for early integration of performance analysis into the development cycle of the services via simulation.

I.2 Motivations

Since the emergence of model-driven engineering, there has been a significant effort to include analysis in the software development process. This need arises since the inclusion of tractable performance analysis techniques into existing software development approaches, improves the analysis of non-functional properties such as performance, reliability, scalability and security. With the increasing realisation of software and business process development and distribution as web services, it is important to evaluate their performance using simulation and testing at an early stage of the development process. In the Service-Oriented Architecture (SOA) management, the web service performance is a well-know problem and our aim is to propose model-based solutions that allow to study the behavior of the services.

Our work is motivated by the fact that today there is currently no method which provides a quick and precise approach for web service performance evaluation. Most of the existing model-based approaches do not support performance analysis of services in term of execution times during the design time. Moreover, the testing phases can face two problems which are time-overhead and costs. Furthermore, it is not easy to predict the behavior of the web service before the end of the development process because there is no appropriate framework that helps to study the behavior of the web service. Hence, our research is intended to solve the following research questions:

1. How can the performance of a Web service be estimated at early stage of the development cycle using model-based approaches and simulations?

2. How can the performance of services be predicted without historical or testing data?
I. INTRODUCTION AND RESEARCH QUESTIONS

3. How can we effectively monitor and/or negotiate the performance SLA of service?

The next section summarizes original contributions of this thesis.

I.3 Contributions

As argued in the previous section, this thesis mainly aims to provide solution to integrate the performance analysis step into the development cycle of the software/services using simulation. From our work, the following contributions can be highlighted.

I.3.1 Model-based early assessment of service performance: Full-knowledge and Partial-knowledge scenarios

In order to assess the behavior of the service in a full-knowledge and partial-knowledge scenarios, we use a Symbolic Transition Systems based models [16] to describe web services as finite state automata and evaluate their performance. The basic STS-based model is extended for simulation and testing. The first extension for testing allows the automatic integration of some performance monitoring code in order to record the performance of the service (e.g., execution time). The second extension for simulation allows to generate a simulation scripts in order to simulate the behavior of the service. This model extends the state transitions of the standard STS-based model with transition probabilities and delay distributions. Transition probabilities model the behavior of the service and the frequency of moving between two states which can be inferred from the executions of similar services. However, delay distributions model the distribution of waiting times that represent the time needed to complete the task associated to the transition. In a full-knowledge scenario, the total execution times for each operation and the internal distributions of delays are known and used for performance evaluation. In a partial-knowledge scenario, coarse-grained partial testing results (i.e., the lower and upper bounds to the operation execution times) are used to simulate a service performance. This contribution includes a methodology that uses simulation during the design and pre-deployment phases of the web service lifecycle to preliminarily assess web service performance using coarse-grained information on the total execution time of each service operation derived by testing. We use testing results and provide some practical examples to validate our methodology and the quality of the performance measurements computed by simulation considering the full-knowledge and partial-knowledge scenarios.

I.3.2 Model-based early assessment of service performance: Zero-knowledge scenario

This contribution allows service developers and software adopters to evaluate service performance in a zero-knowledge scenario, where no testing results on service execution times are considered and only simulation results are used for performance evaluation. Our
I. INTRODUCTION AND RESEARCH QUESTIONS

The approach builds on expert knowledge to estimate the execution time of each service operation and, in turn, the overall service performance. This approach evaluates the complexity of each service operation using the XML encoding of its input and output parameters, and the Web Service Description Language (WSDL) interface of the service. Then, based on profile tables providing the time overhead needed to parse and build SOAP messages with different depths and cardinalities, and the performance inferred by testing some reference service operations, the operation execution times are estimated. We finally experimentally evaluate our approach by using the measured operation execution times to simulate the service performance. These results, compared with the service performance obtained by testing after the development show an accurate approximation.

I.3.3 Services SLA on performance negotiation and/or monitoring using simulated data

This contribution shows how performance information assessed using our approach can be exploited to evaluate and/or negotiate Service Level Agreements (SLAs) on the service. In fact, when the performance analysis step is integrated into the development cycle, the performance measured is used to negotiate the SLA of the service. This part of our thesis presents a complete framework that first allows to negotiate the SLA on the performance from the estimation performed using our approaches and second to monitor the performance of the service after deployment and report the SLA violations.

I.4 Structure of the thesis or Overview of the thesis

The thesis is structured in several parts organized as follows.

The remainder of Part I includes Chapter II which gives the overall background and the related work of the thesis. This chapter presents the background on web service technologies and the most widely used service models in the literature such as STS [16], Petri nets [17], UML, Timed Automata (TA) [18, 19, 20]. Some estimation models and performance models are also presented followed by a description of model-driven engineering followed by the performance measurements tools such as SoapUI, LoadUI, Membrane [21, 22, 23].

Part II includes two chapters and presents our methodologies for early assessment of the service performance. Chapter III presents our model-based approach for evaluating service performance in a full-knowledge and partial-knowledge scenarios in which coarse-grained test-based information on the total execution time interval of each service operation is produced by real testing. Chapter IV outlines our proposed solution for estimating performance of the service in a zero-knowledge scenario, where the testing data are not available and only simulation results are used for performance evaluation. This method relies on the characteristics of the XML encoding of the input and output messages of the service, the WSDL interface of the service and the profiles tables that provide the parsing
and construction times of the input and output messages to approximate the performance of the service.

Part III includes Chapter V that shows the applications of our performance evaluation solutions. It presents our framework that aims \textit{i)} to negotiate the SLA for the service from the performance estimated using the simulation techniques proposed in Part II, \textit{ii)} to evaluate and monitor the SLA of the service in order to report the violations after deployment.

Part IV includes Chapter VI which concludes the thesis and outlines our future work.
Chapter II

State of the art

This chapter provides some background to our research and presents relevant related work.

II.1 Web Services and related technologies

This section presents the general background on web services and related technologies and terms such as SOA, SOAP, WSDL.

II.1.1 Service-Oriented Architecture (SOA)

Service-Oriented Architecture (SOA) represents an architectural model that aims to enhance the agility and cost-effectiveness of an enterprise while reducing the overall
II. STATE OF THE ART

burden of IT on an organization [24]. It is a framework that supports discovery, message exchange, and integration between loosely coupled services using industry standards. Each party complies with agreed-upon protocols and carries out its part in the overall execution of processes involving services from diverse organizations. SOA framework can use a service and integrate it within an application while at the same time it is not aware of the details of the service’s implementation language, platform, location, or status. Its implementation can consist of a combination of technologies, products, APIs, supporting infrastructure extensions, and many other parts.

II.1.2 Web Service

Web service is defined by the World Wide Web consortium (W3C) [25] as follows: “A web service is a software system identified by a URI [RFC 2396], whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols.”

In software architecture and engineering, the web services paradigm is often depicted as the step that allows the development of distributed applications through combinations of services that are located in different places over the Web. It is a piece of business logic, located somewhere on the Internet, that is accessible through standard-based Internet protocols. A web service can be seen as a function, which has an input and an output. The services implemented as web services are commonly described by the following documents: the WSDL definition, XML schema definition, and WS-Policy definition. Service-Oriented Architectures (SOA) refer to software architectures based on web services paradigms [24, 26]. Web services are commonly based on the use of two communication protocols: SOAP and REST.

II.1.3 Simple Object Access Protocol (SOAP)

The interactions with web services which are the requests and responses are based on the Simple Object Access Protocol (SOAP) [27]. SOAP defines a standardized XML-based framework for exchanging structured and typed information between services. SOAP provides the envelope for sending web services messages, which contains an optional header and a body. It is a protocol for messages exchange which defines both a format for messages and a processing model. It defines in addition, how a receiver should process a SOAP message [24, 28]. Figure II.1 shows an example of SOAP message for a web service which implements the sum of two integers. The header of a soap message typically contains information regarding the processing of the message. The body contains the input or output information of the web service to be transferred to the application. In the example, add is the name of the service which is invoked with two values.
II. STATE OF THE ART

II.1.4 Representational State Transfer (REST)

The Representational State Transfer (REST) is a web architectural style presented by Roy Fielding in 2000 [29, 30]. The basic idea of REST is the full exploitation of the HTTP protocol, in particular:

- It focuses on Resources, that is, each service should be designed as an action on a resource.
- It takes full advantage of all HTTP methods: GET, POST, PUT and DELETE.

The GET method is used for requests that are not intended to modify the state of a resource.

The POST method is used to request that the server accepts the entity enclosed in the request as a new subordinate of the resource identified by the URI named in the request.

The PUT method is used to send the modified representation of a resource.

The DELETE method can be used to request the removal of a resource.

The web services implemented with the REST communication protocol are called RESTful web services.

REST components communicate by transferring a representation of a resource in a format matching one of an evolving set of standard data types, selected dynamically based on the capabilities or desires of the recipient and the nature of the resource. REST is really just an abstract architectural style, not a specific architecture, network protocol, or software system. While no existing system exactly adheres to the full set of REST principles, the World Wide Web is probably the most well-known and successful implementation of them.

II.1.5 Web Services Description Language (WSDL)

The Web Services Description Language WSDL [24, 28, 31] is an XML language that is used to describe web services interface in terms of its inputs and outputs. The messages exchanged are described abstractly and associated with a concrete network protocol and
message format. The abstract part of a WSDL contains \(i\)) the types, which are the kinds of messages, the service will send or receive and \(ii\)) the interfaces, which describe the functionalities provided by the web service as a set of operations. An operation is defined as a sequence of input and output messages. Usually, an operation has a name, a message exchange pattern, and the inputs and outputs types. The concrete part contains \(i\)) the bindings describing how the service can be accessed and \(ii\)) the services, which describe where the service can be accessed and consist of a reference to an interface and the endpoints. For every operation in the interface, a binding specifies the message format and the transmission protocol to be used to access the service. WSDL provides specific support for bindings using SOAP and HTTP. Each endpoint contains a reference to a binding and the address of the service, which is typically a URI.

Figure II.2 shows an example of WSDL description for a web service implementing the sum of two numbers.

### II.1.6 Universal Description, Discovery and Integration (UDDI)

In the previous section, we described how the WSDL describes the service by indicating how the service can be invoked, the protocol and the format of the message to be used for the invocation and the location of the service. However, the service needs to be found, before been selected and finally invoked. These steps are known as “service discovery” and are performed using the **Universal Description, Discovery and Integration (UDDI)** standard [24, 32]. UDDI provides standardized descriptions of web services and records them in a catalog called a registry. The registry is queried by the client in order to find the services needed. UDDI defines a set of standard services allowing the description and discovery of the web services providers, the services they publish in the registry and the technical interfaces needed by the service requestors to access those services. UDDI is based on a common set of industry standards, including HTTP, XML, XML Schema, and SOAP. UDDI contains data and metadata about web services, their location and the information needed to invoke them. A UDDI registry supports the discovery-find-use pattern described in Figure II.3.

### II.2 Web service QoS guarantees: Service Level Agreement (SLA)

In order to guarantee a good level of service, they is a need to have an agreement between the service providers and the customers. In SOA environment, the conditions under which services can be delivered are specified in the **Service Level Agreements (SLAs)** [33, 34]. SLAs are the contracts between the service providers and the customers, and contain a set of Quality of Service (QoS) requirements for the services. SLAs need to be defined, negotiated, established, deployed and monitored [35, 36]. The monitoring of the SLAs is important in order to verify whether the service delivered complies with the terms of SLA agreed between the two parties [37].
II. STATE OF THE ART

<?xml version='1.0' encoding='UTF-8'?>
<definitions name="CalculatorWSService">
  <types>
    <xs:schema>
      <xs:element name="add" type="tns:add"/>
      <xs:element name="addResponse" type="tns:addResponse"/>
      <xs:complexType name="add">
        <xs:sequence>
          <xs:element name="i" type="xs:int"/>
          <xs:element name="j" type="xs:int"/>
        </xs:sequence>
      </xs:complexType>
    </xs:schema>
  </types>
  <message name="add">
    <part name="parameters" element="tns:add"/>
  </message>
  <message name="addResponse">
    <part name="parameters" element="tns:addResponse"/>
  </message>
  <portType name="CalculatorWS">
    <operation name="add">
      <input wsam:Action="http://calculator.me.org/CalculatorWS/addRequest"
        message="tns:add"/>
      <output wsam:Action="http://calculator.me.org/CalculatorWS/
        addResponse" message="tns:addResponse"/>
    </operation>
  </portType>
  <binding name="CalculatorWSPortBinding" type="tns:CalculatorWS">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="
      document"/>
    <operation name="add">
      <soap:operation soapAction="/"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="CalculatorWSService">
    <port name="CalculatorWSPort" binding="tns:CalculatorWSPortBinding">
      <soap:address location="http://localhost:8080/CalculatorApp/
        CalculatorWSService"/>
    </port>
  </service>
</definitions>

Figure II.2: An example of WSDL description
II. STATE OF THE ART

II.2.1 SLAs components

Many specifications are defined in order to represent the SLAs. These specifications allow to specify the different conditions on which the services can be delivered and the parties which are concerned [37, 38]. In most of the specifications, the SLAs definition includes three important parts which are:

- The *Parties* part defines the parties involved in the agreement. Here, they are the service providers and the customers.
- The *Service Description* part describes the service concerned by the agreement and specifies its characteristics.
- The *Obligations* part specifies the obligations of the two parties and the actions to be executed when there is a violation.

II.2.2 SLAs Life Cycle

The SLA defined for a service has a life cycle. Sun Microsystems Internet Data Center Group [36] detailed the SLA life cycle into six steps which are defined as follows.

1. Discover Service Providers: where service providers are located according to customers requirements.
2. Define SLA: this includes definition of services, parties involved in the SLA, penalty policies and QoS parameters. This step includes also a negotiation in order to reach a mutual agreement between the parties.
3. Establish Agreement: where an SLA template is established and filled in by specific agreement, and parties involved are starting to commit it.
4. Monitor SLA Violation: in which the performance delivered for the service by the provider is measured in order to handle the violation of the contract.
5. Terminate SLA: in which SLA terminates due to timeout or any party’s violation.
II. STATE OF THE ART

6. Enforce Penalties for SLA Violation: where, if there is any party violating contract terms, the corresponding penalty clauses are invoked and executed.

Figure II.4 illustrates these steps.

II.2.3 SLAs definition languages

In this section, we give a short list and the description of the most popular SLA management languages and frameworks.

- WS-Agreement (Web Service Agreement) [39] is a web service protocol for agreement establishment between the service provider and the consumer. It is an XML-based language that allows to establish and manage dynamically the service SLA.

- WSLA (Web Service Level Agreement) [37, 40] is a framework defined for SLA management. It provides a flexible and extensible language based on XML schema for SLA definition and monitoring. WSLA enables service customers and providers to define different variety of SLAs. The proposed framework configures automatically the SLA, and starts its monitoring to handle the violations.

- SLAng (SLA definition language) [41] is an XML-based language for SLA definition and management. It allows to describe technical and non-technical characteristics of service with the details on QoS requirements and the related set of metrics. SLAng
provides solution for the negotiation and the monitoring of the agreement between the customers and the service providers.

- QML (Quality of service Modeling Language) [42] is a language for quality of service specification for software components. It allows to define the QoS requirements as agreement between the different parties. QML provides solution allowing contract definition in order to monitor the fulfilment of the QoS conditions at runtime for distributed systems.

- SLA* [43] is a rich, comprehensive, extensible and format independent SLA model proposed for SLA definition and monitoring. It defines a solution for SLA management which specifies the parties involved in the agreement, the service description and the terms of the agreement which include the actions to be executed when there is a violation.

II.3 Service Development

In this section, we provide a short overview on software development life cycle and the service models with a particular summary on the Symbolic Transition Systems (STS) which are used in this thesis to model the web services.

II.3.1 Service Development Life Cycle

The software life-cycle (SLC) is the process structure describing the development of a software product [44, 45]. Basically, it starts from the definition of the requirements and finishes with the deployment. The traditional software development life cycle is the waterfall model [46, 47] described in Figure II.5 which begins with requirements and proceeds through the major phases of analysis, design, implementation, and deployment. The drawbacks of this model are that usually the requirement, analysis, and design phases are too long and the performance analysis is performed after the deployment.

Since software and services need continuous improvements during their life cycle, there is another model of development called continuous improvement development life cycle [48, 49, 50] represented in Figure II.6, which allows to continuously improve the software and services from the analyze phase through the design, implement and monitor phases. The current modern methodologies such as Rational Unified Process or Extreme Programming [51, 52, 53] are iterative and use continuous improvements. Depending of the specifics of the methodology, implementations are performed through iteration phases. Whatever the development methodology, during the different phases, the service is modeled using many kinds of model of service and it could be better to analyse also the behavior of the software or service at design time to assess the service performance.
II. STATE OF THE ART

II.3.2 Service Development Models

II.3.2.1 Model-Driven Development (MDD)

Models are mainly used as support in the development process in order to organize and conceptualize the perspectives and the abstractions of the software and service. The Model-Driven Development (MDD) allows to build systems with various levels of modeling abstractions [2, 3, 12]. MDD uses metamodels to capture the concepts and layers of abstraction, and to give service descriptions, service directories, business processes, and information models. The models can be transformed into code or in other models depending on the goal of the modeling. The standard for model-driven engineering defined by the Object Management Group (OMG) is commonly used with the meta-language MOF (Meta Object Facility) [54].

Model-Driven Development supports the different activities performed during the development process of the software or service. It permits the determination of the quality and the behavior of the software or service which include more complex tasks such as per-
formance analysis in term of response time, execution time, throughput and availability. Since MDD allows to analyse the non-functional properties of software/service, the performance analysis is the scope of our work. Our aim is to integrate it at the design phase of the waterfall and continuous improvements model of software development life cycle.

Many modeling languages or notations are used to represent the business logic of a service such as Unified Modeling Language (UML) [55] or Business Process Modeling Notation (BPMN) [56]. Other models such as Queuing Network, Petri-nets, Stochastic Process Algebra and Automaton [12, 57, 58, 59] are also used to add more abstraction in the modeling, to increase the level of abstraction, and to provide a basis for analysis. These models are commonly used to study more complex features of the service, such as performance. They are used to represent the behavior and the performance model of the software/service and can be derived from the extension of the UML and BPMN models. Popular modeling languages like UML and BPMN are routinely used to describe service behavior and extended with some annotations in order to be converted into performance models which can be used to simulate the behavior of the service [3, 5, 45, 60, 61, 62].

II.3.2.2 Modeling techniques

Performance evaluation of software and service can be performed by measurement or by modeling techniques. Measurement technique provides an accurate assessment of the performance but requires that the service be implemented before measurement can take place. In practice, however, we want to assess the behavior of the software or service before the implementation, the development of complex services may be time consuming. The performance of a service can be evaluated early using analytical model, simulation model or a hybrid model, allowing the responsiveness, throughput, reliability, and/or scalability of a service under a given workload to be determined.

In order to model the performance models of the software, services and business processes, many languages and notations have been proposed. Among the most popular, we can mention UML and BPMN for which many extensions are proposed in order to integrate the performance analysis. Additional notations and representations are used to retrieve the behavior of the service and are based on the use of Queuing Network (QN), Petri-nets (PN), Stochastic Process Algebra (SPA) and Finite State Automaton (FSA) [12, 57, 58, 59] such as Symbolic transition systems (STS) techniques.

In our work, we use automaton-based representations to describe the web service. The detail of the Symbolic Transition Systems (STS)-based models used are described below. The specification of the other models can be found in the literature. The review in Section II.4 takes into account most of existing techniques.

- **Finite State Automaton (FSA): Symbolic Transition Systems (STS)**
  Modeling a software/service as a transition system is a traditional approach used to
test functional properties of systems for which the real implementation is not available, and to prove that a software/service respects its specifications [63]. In our work, we choose to use Symbolic Transition Systems (STS) [16, 64] which are derived from the Labelled Transition Systems (LTS) [65] in order to overcome its drawbacks.

An LTS consists of states and transitions labelled with actions between them [65]. The states represent the states of the modelling system and the actions on the transitions model the actions which can be performed by the system. It is formally defined as follows.

**Definition II.3.1 (LTS)**

A Labelled Transition System (LTS) is a tuple \(<S, s_1, I, O, \rightarrow>\) where:

- \(S = \langle s_1, ..., s_n \rangle\) is a set of states;
- \(s_1 \in S\) is the initial state;
- \(I\) is a set of input action labels;
- \(O\) is a set of output action labels with \(I \cap O = \emptyset\);
- \(\rightarrow\) is a transition relation. Each transition \((s, d) \in \rightarrow\) is of the form \(s \xrightarrow{\mu} d\), where:
  - \(s \in S\) is the source state;
  - \(\mu \in (I \cup O)\) is the label on the transition;
  - \(d \in S\) is the destination state.

An LTS allows limited possibility of modelling data values and variables which are mapped in concrete values (actions) to avoid state space explosion problem when the number of state is important. Also, it is not possible to specify additional information on the transition such as the constraints. Then, in order to overcome these disadvantages, Symbolic Transition Systems (STSs) have been introduced by Frantzen et al. [16, 64] and allow to treat symbolically the data values and variables instead of mapping them in concrete values on the model.

A Symbolic Transition System (STS) [16] is a finite state automaton that describes the behavior and evolution of a software/service, and is formally defined as follows.

**Definition II.3.2 (STS)**

A Symbolic Transition System (STS) is a tuple \(<S, s_1, V, I, A, \rightarrow>\) where:

- \(S = \langle s_1, ..., s_n \rangle\) is a set of states;
- \(s_1 \in S\) is the initial state;
- \(V\) is a set of location (internal) variables;
- \(I\) is a set of interaction variables representing operation inputs and outputs;
- \(A\) is a set of actions (operations);
- \(\rightarrow\) is a transition relation. Each transition \((s, d) \in \rightarrow\) is of the form \(s \xrightarrow{\alpha, \gamma, \mu} d\), where:
II. STATE OF THE ART

- \( s \in S \) is the source state;
- \( \alpha \in A \) is an action;
- \( \gamma \) is a guard, that is, a first order formula over variables in \( V \cup I \);
- \( \mu \) is an update mapping on variables in \( V \);
- \( d \in S \) is the destination state.

We note that actions in \( A \) trigger state transitions and can be of two types:

- input actions, denoted as \(?function<parameters>\), where an operation call is received;
- output actions, denoted as \(!function<results>\), where the output of an operation call is returned.

In addition, we note that guards represent conditions on transitions and the update mapping represents new assignments to internal variables.

For performance evaluation, this work extended the STS-based model for testing and simulation as described in Chapter III.

II.4 Previous works

This section presents relevant existing work related to our topic. It covers the performance models, the different approaches proposed to assess the performance of service and other topics discussed along this thesis to provide solution to assess the behavior of the web services at early stage of the development process when historical data is available or not.

The problem of guaranteeing and measuring service performance, and managing QoS from different perspectives has been researched extensively. Some researches consider the challenges of managing QoS to improve service quality and to support QoS-based service selection [66, 67, 68, 69]. Other approaches propose SLA-based solutions to monitor the performance of services [37], to determine the degree of SLA fulfillment by the services [70], and to evaluate the impact of security on service time [4, 71, 72], and to manage cloud service performance [73, 74]. The authors of [37, 39, 40, 42, 43] define languages and frameworks for SLA definition in order to better handle the SLA violations. They provide solutions that allow an easy and extensible definition of SLA that can be used for automatic establishment and management of the agreement between the service providers and the customers. In particular, [43] proposes a solution for SLA definition that contributes in a good monitoring of the terms of the agreement between the parties.

The work of [75] is based on the solution proposed by [43] and analyses the link between SLA negotiation and SLA monitoring. This work presents a novel architecture for establishing and monitoring SLA hierarchies spanning through multiple domains and layers of a service economy. The proposed architecture satisfies the requirements introduced by
II. STATE OF THE ART

SLA establishment which are the availability of historical data for the SLA offers evaluation contrarily to our approach that uses simulation data and the assessment of the capability to monitor the terms of the agreement.

Much in line with our work, other researchers propose simulation techniques that build on fine-grained test data extracted from services to predict behavior for untested inputs or behavior of a composite service integrating these services [2, 76]. The authors of [76] propose a tool that allows the workflow of the service to be described. This description is then used to generate a test code for performance evaluation and a simulation model which is used for intensive performance evaluation. The authors of [77] propose to use simulation to evaluate the performance of the web services in order to have enough information on the behavior of the services and compose efficient web processes.

Other researchers use a model-based approach to analyze different non-functional properties of services/software and to show that performance analysis can be integrated early in the development process [3, 5, 78]. The authors showed that instrumented code can be generated from the extended models for monitoring purposes, but most of the existing works are based on the use of extended UML diagrams, Petri nets, probabilistic time automata, etc. Sometimes, such approaches require the development of expensive and time consuming prototypes, on which representative benchmarks of the system in operation can be run. A detailed survey on model-based performance prediction approaches is given by [79].

The authors of [80] proposes a model-driven approach to integrate the performance prediction into the service composition scenario. The approach is based on performance-enabled WSDL called P-WSDL that allows to add some performance data in the WSDL standard which can be used to predict the behavior of a composite service.

The authors of [81] propose to extend the model-driven engineering with performance engineering in order to perform a performance evaluation process during the different development phases. They extend a UML activity diagram with performance information and transformed it into a simulation model for early performance prediction. They proposed to obtain the performance information used for the prediction from the developers experience and/or the collected performance data on existing systems or similar service. In the same direction, the authors of [82, 83] propose to predict the performance of web service modeled using UML diagrams.

Another relevant aspect of the work in this thesis is the modeling of services as finite state automata for testing and for performance analysis [84, 85].

The authors of [84] propose to generate a test case for web services using the extended Finite State Machine (FSM) of the service which is built from the WSDL specification. This extended FSM model allows the addition of the dynamic behavior of the service specified in the WSDL which is not available on the FSM standard. This work was restricted to extend the standard automaton for test cases generation and did not take into account the use of the extension for performance evaluation of the service.
Schwarzl et al. [85] proposed to extend the STS-based model by adding other parameters on the state transitions. The extended STS called ESTS proposed to add on transitions, timed behaviors like transition execution times and delay transitions. Each transition has a priority, a traversal probability and an execution duration. They used the timing groups to put together the states that have an outgoing delay transition with the same timeout. This work showed that the STS-based models can be extended by adding some features in order to measure some parameters. The transition probabilities and delay transition are used for model composition and for test cases generation like the previous works. The use of the extension for performance evaluation is not done.

The use of Team Automata, an extended automata-based model used to specify and evaluate software architectural design is proposed by Sharafi [86]. This work highlights the benefits of this model and its extension to evaluate non-functional properties such as security and performance at the software architectures level. Instead of using STS-based model as in our work, the author proposes to use UML diagrams describing the software architecture behavior to generate the extended Team Automata used to study the performance of the web service software architecture. The results obtained show that their framework can be used to estimate the performance aspects of an architecture but it focuses only on the architectural level.

Since the security aspects are important in the business processes definition, and the work of [61] proposed to integrate security requirements into business process modeling. This work extended the BPMN notation and define a business process metamodel in order to define secure business process. Based on Model Driven Architecture (MDA) approach, the business process diagram was augmented with security requirements at early development stages from the business analysts perspective.

The work of [45, 62] extended UML definition to specify temporal restrictions and resource usage and their automatic evaluation. They defined solution that extends UML notation to consider some types of constraints definition on the UML activity diagrams for real-time systems.

Recent work proposed STS-based modeling approaches for the certification of non-functional properties of services and for testing service orchestrations [87, 88]. Finally, some approaches rely on Timed Automata (TA) and Time Petri Nets (TPN) [18, 89] which are also widely-used for workflows modeling and analysis of real-time systems. They are used to model the temporal behavior of workflows sytems to achieve different goals such as testing and simulation of business processes [90]. The authors of [19, 91] use timed automata for test cases generation, while [20, 92] propose to use them for web service verification, fault monitoring and diagnosis of systems. TA are also used by [93, 94] for monitoring the SLAs. The authors of [95] present an approach to verify web services with time restrictions which are defined by BPEL4WS using model checking techniques. They used a formalism based on Timed Automata to translate the description of web service written in BPEL4WS into automata which is used to simulate and verify the correctness
II. STATE OF THE ART

The authors of [17, 96] used TPN for timed modeling and verification of BPEL processes by presenting an approach that verifies the time constraints during service composition processes. Different to the approach we present in this thesis, they did not apply TA and TPN for performance evaluation of web service.

Taking a different line of research to the above-mentioned works, we propose an STS-based approach for an early assessment of service performance, which builds on coarse-grained test data. An STS-based service modeling has been used since it provides a modular and flexible solution that can be extended with new features.

Similar to our method, there are several works that, while not specifically targeted web services, propose service-based or component-based solutions for systems evaluation. The authors of the PUMA (Performance by Unified Model Analysis) method [97], the Palladio framework [98] and the KLAPER (Kernel LAnguage for PErformance and Reliability) approach [99] reduce the system under evaluation to a stochastic state model in order to assess service time distributions and transition probabilities. Their solutions are also based on simulation.

More related to web service domain, the work in [15] proposes the PUPPET (Pick UP Performance Evaluation Test-bed) approach which mock-services concept is close to our notion of simulation scripts. The authors built their solution to assess functional model-based testing and performance testing. PUPPET is used to automatically generate a test-bed environment for a service, in order to check if the specified QoS properties will be respected by the service under development after its deployment in the final environment.

Focusing on design time evaluation, some evaluation models use historical data and expert knowledge [11, 12]. In this thesis, we propose to assess the complexity of the web service in order to estimate their performance without historical data. Some works propose different methods for assessing the complexity of software, web services, processes, workflows, and systems [100, 101, 102, 103]. They define a set of metrics to evaluate the quality of the XML structure of the WSDL in terms of web service maintainability, and to prevent any potential quality issue in the service interfaces. Unlike the solution proposed in this thesis, the above approaches are mostly focused on service complexity analysis, and do not consider performance evaluation. In [104], the authors propose a solution based on queuing theory for an early assessment of the performance of software components. They describe architectural behavior of software systems using UML diagrams, which are then converted into Interface Automata to evaluate software performance at design time based on queuing methods. This solution focuses only on software components and is consequently not applicable to our service-based scenario presented in this thesis.
II.5 Conclusions

This chapter presents the main concepts behind web services, and related technologies and protocols. This chapter also describes the relevant work proposed in the literature about the performance evaluation of software/web services in order to allow a better evaluation of the contributions we will describe in this thesis. In Part II, we present our approaches for early assessment of web service performance using simulation, the implementation of our solutions, and the experimental results.
Part II

Early Assessment of Service Performance via Simulation

Test-based performance evaluation of web services is a type of testing aiming to quantify the responsiveness, throughput, reliability, and/or scalability of a service/software under a given workload. This part of the thesis describes our techniques to achieve comparable results via simulation. First, in Chapter III, we propose a model-based approach that allows to study the behavior of the service from historical data using simulation. Secondly, in Chapter IV, we propose a complete solution that extends the previous work and allows to estimate the behavior of the web services when historical data are not yet available.
Chapter III

Early Assessment of Service Performance: Full-knowledge and Partial-knowledge Scenarios

Contents

III.1 Introduction ......................................................... 28
III.2 Model of service and framework .................................. 29
   III.2.1 Model of service ........................................... 29
   III.2.2 Framework for performance evaluation .................. 29
III.3 Extended service development life cycle with simulation ........ 30
III.4 Reference scenario ................................................. 31
   III.4.1 Overview on the IFX Standard ............................ 31
   III.4.2 Presentation of the service ................................. 32
III.5 Performance modeling ............................................... 33
   III.5.1 STS-based model extended for testing .................... 33
   III.5.2 STS-based model extended for simulation ............... 34
   III.5.3 Loops unroll technique for the STS-based model for simulation ... 36
   III.5.4 XML encoding of the STS-based models .................. 37
III.6 Implementation ..................................................... 38
   III.6.1 Implementation of the performance interceptors .......... 38
   III.6.2 Implementation of the simulation scripts generator ...... 41
   III.6.3 Implementation of our solutions .......................... 44
III.7 Experimental evaluation of our methodology ....................... 49
   III.7.1 Testing and simulation results ............................. 49
   III.7.2 Comparison of testing and simulation results ............ 52
III.8 Conclusions ......................................................... 53

Obtaining an accurate and rapid evaluation of web service performance is a key problem of Service-Oriented Architecture (SOA). It is not possible to evaluate service performance at both design time and runtime. As a result, the performance of the service is accessed later after deployment. This chapter proposes a model-based methodology that
III. EARLY ASSESSMENT OF SERVICE PERFORMANCE: FULL-KNOWLEDGE AND PARTIAL-KNOWLEDGE SCENARIOS

generates a simulation script to be used for an early assessment of service performance, and to negotiate and evaluate SLAs on service performance at runtime. In the following, we present our model-based approach for early assessment of web service performance.

III.1 Introduction

In order to increase the trustworthiness of services and guarantee their non-functional properties (e.g., security, performance, reliability), effective and easy to use methodologies should be provided to facilitate their evaluation. In this scenario, the evaluation of service performance is fundamental for current service-based infrastructures [1]. Customers are in fact concerned about the performance of the services they integrate into their systems, while developers would like to evaluate the performance of their services at the earliest possible stage of the service lifecycle. Most of the current solutions (e.g., [4, 5, 6, 7]), however, are based on interface testing. These do not support early assessment of end to end service performance, including execution of the service itself. Instead, they mainly evaluate the overhead of service protocols and specifications. Also, since such techniques require access to the service code, if the client is to be able to simulate runtime service performance during the selection process.

In this thesis, we propose a model-based approach that relies on Symbolic Transition Systems (STSs) to describe web services as finite state automata and evaluate their performance. From the standard STS-based model of the service, we generate two extended models:

i) a testing model, which is used to automatically generate some monitoring code (i.e., performance interceptors) for the evaluation of the service performance;

ii) a simulation model, which is used to generate a simulation script to forecast the service behavior.

The main contribution of this chapter is a methodology that uses simulation during the design and pre-deployment phases of the web service lifecycle to preliminarily assess web service performance. Our technique relies on coarse-grained information on the total execution time of each service operation derived by testing, followed by statistically generated estimates on the delay introduced by each internal task composing the operation. We consider two scenarios according to the amount of information on the performance of service used in the evaluation process. In a full-knowledge scenario, it is available for the performance evaluation process, the total execution times of each operation and the internal distributions of delays on the transitions. In a partial-knowledge scenario, partial testing results are available for the performance evaluation process. Only, the bounds of the operation execution times interval are know and used to simulate a service performance. We use testing results and provide some practical examples referring to standard IFX [105] financial services’ interfaces to validate our methodology and the quality of the performance
measurements computed by simulation. Our solution to performance evaluation can also be used to negotiate and/or evaluate SLAs on service performance, and select services at runtime on the basis of their claimed performance. To this aim, we assume service provider claims to be reliable and trustworthy.

III.2 Model of service and framework

This section presents an overview on our model of service already defined in Section II.3.2.2 and our framework proposed to evaluate web service performance.

III.2.1 Model of service

Modeling a software/service as a transition system is a traditional approach used to test functional properties of systems for which the real implementation is not available, and to prove that a software/service conforms to its specifications [63]. A Symbolic Transition System (STS)-based [16] model is already formally defined in Definition II.3.2 as a tuple $<S,s_1,V,I,A,\rightarrow>$ where: $S$ is a set of states, $s_1$ the initial state, $V$ is a set of location (internal) variables, $I$ is a set of interaction variables, $A$ is a set of actions and $\rightarrow$ is a transition relation.

In this thesis, we use STSs to also model the service implementation. In this case, $S$ includes both interface and implementation states, and $A$, which usually includes only operations in the WSDL interface of the service, is extended to consider the service internal operations.

The STS-based model of service is chosen in this thesis because of its flexibility to allow an easy extension of the transitions between the states by adding some annotations on them. The extensions of the STS-based models for testing and simulation are detailed in the following sections.

III.2.2 Framework for performance evaluation

We propose a framework that evaluates the performance of a service by measuring some performance indicators like service time and response time.

Figure III.1 shows our framework that is composed by a testing and simulation layers works as follows. The model of the service STS$_o$ defined in Definition II.3.2 is first built from the service interface and code, and given as input to the testing and simulation layers.

The testing layer then produces a testing model STS$_t$ by adding performance idioms to STS$_o$. STS$_t$ is used to automatically generate the code needed to monitor the performance of services.

The simulation layer, instead, produces a simulation model STS$_s$ by adding transition probabilities and delay (waiting time) distributions to transitions representing internal service operation tasks in STS$_o$. Transition probabilities model the normal execution flow.
III. EARLY ASSESSMENT OF SERVICE PERFORMANCE: FULL-KNOWLEDGE AND PARTIAL-KNOWLEDGE SCENARIOS

III.1 Performance evaluation framework

of the service. A delay distribution represents the distribution of the times needed to complete the task represented by a state transition. STS* is used to generate a simulation script that measures performance indicators of the service, when the service code is not available.

The outputs produced by the execution of the test cases (testing layer) are used to verify if the simulation script (simulation layer) provides accurate information to forecast the service performance. If not, training data can be used to refine STS*. They include information on the total execution time of each service operation, and can be produced by our testing approach or by observations on the performance of similar services.

III.3 Extended service development life cycle with simulation

In order to allow an early assessment of service performance by simulation, we propose to modify the traditional development cycle presented in Chapter II and add a simulation step along the development flow. Our solution uses a model-based approach to simulate the behavior of the service. In the traditional development cycle, we proposed to extend the purposes of the design step of the development cycle with simulation. This allows to have more details about the service specification from the analysis step. These details help to build the standard model of the service, which is extended in our work for test and simulation. During the design and simulation step, the performance of the service is simulated using simulation scripts which are generated as discussed in Section III.5.2.

Figure III.2 shows the extended development cycle for the basic waterfall development cycle, whereas Figure III.3 shows the extended continuous development cycle. The simulation step is added at the design step of these cycles and allows to perform an early assessment of the service/software performance. For any other software development life cycle, the simulation phase can be added also in the design phase or after the design phase, where more information is available on the functionalities of the service.
III.4 Reference scenario

In this section, we present an overview of the IFX standard and the reference scenario based on it.

III.4.1 Overview on the IFX Standard

The Interactive Financial Exchange (IFX) standard (http://www.ifxforum.org/) is a content rich, well-designed financial messaging protocol initially defined in the 1997 by financial industry and technology leaders. It is a global, open, and multi-channel messaging protocol developed and maintained by a consortium of financial and information technology companies, which regulates the electronic exchange of financial data between financial institutions, business, and consumers through Internet. IFX provides an XML specification, supporting technical principles of SOA and web services, for electronic financial transactions.

IFX is built with the recognition that no single financial transaction stands on its own, but is an integral part of the relationship among all of the communicating parties.
Currently the standard IFX provides content-rich conversations in the areas of:

- Electronic bill delivery and payment
- Business to Business Payments
- Business to Business Banking (such as balance and transaction reporting, remittance information)
- Automated Teller Machine (ATM) communications
- Branch Banking Services
- Consumer to Business Payments
- Consumer to Business Banking
- Card Management and Services

III.4.2 Presentation of the service

Our reference scenario considers an IFX-based service implementing a deposit and withdrawal service using a Reverse ATM. This service implements the following operations:

- **Signon**, which authenticates users by checking the validity of their credentials;
- **CreditAdd**, which allows authenticated users to deposit funds;
- **DebitAdd**, which allows authenticated users to withdraw funds.

Users rely on a reverse ATM device to connect to the bank via the implemented reverse ATM service.

Figure III.4 shows the STS-based model of the reverse ATM service, where guards are presented within squared brackets, and interface and implementation states are denoted as circles and squares, respectively. The service handles the request of a user by first authenticating her using function $\text{Signon}(\text{login}, \text{pwd})$. After a successful authentication, the user can deposit or withdraw a given amount of money using functions $\text{CreditAdd}(\text{amount}, \text{token})$ or $\text{DebitAdd}(\text{amount}, \text{token})$. We note that $\text{token}$ represents a one-time security token returned as a result of a successful $\text{Signon}$ operation. When the user chooses to deposit funds, the credit is accepted if the amount does not exceed the maximum allowed amount. In case of fund withdraw, the debit is accepted if the amount does not exceed the account balance and the maximum allowed amount. Internal operations $\text{Check\_Money}(\text{amount}, \text{token})$ and $\text{Check\_Balance}(\text{amount}, \text{token})$ are used to perform the above checks, and return result=ok if the amount and token are accepted as valid, failure otherwise. For simplicity, Figure III.4 does not present states modeling the real CreditAdd/DebitAdd implementations, which are traversed after successful Check\_Money/Check\_Balance.
III. EARLY ASSESSMENT OF SERVICE PERFORMANCE: FULL-KNOWLEDGE AND PARTIAL-KNOWLEDGE SCENARIOS

III.5 Performance modeling

This section describes how a standard STS-based model can be extended for the service performance testing (Section III.5.1) and simulation (Section III.5.2).

III.5.1 STS-based model extended for testing

According to Figure III.1, we define an STS-based model for testing (STS_t) by extending the standard STS-based model STS_o with idioms. Idioms express commands to be executed at testing time by a test driver to measure the service performance (i.e., monitoring of execution and service times), as well as for logging and security checks. Our idioms are expressed as annotations to the transitions of the standard model STS_o. Table III.1 shows some performance idioms defined in our work. The Time idioms, startclock(t) and stopclock(t) allow to measure the execution times of the service. The clock is started at the beginning of the execution and stopped at the end of the execution. The Logging idioms, logevent(e) and readevent(e) allow to write and read the notifications sent by the service during its execution. The Security idiom, checkinput(i) allows to check the parameters.
III. EARLY ASSESSMENT OF SERVICE PERFORMANCE: FULL-KNOWLEDGE AND PARTIAL-KNOWLEDGE SCENARIOS

Table III.1: Performance Idioms

<table>
<thead>
<tr>
<th>Idiom</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>startclock(t)</td>
<td>Start time counter t</td>
</tr>
<tr>
<td></td>
<td>stopclock(t)</td>
<td>Stop time counter t</td>
</tr>
<tr>
<td>Logging</td>
<td>logevent(e)</td>
<td>Write an event e in the log file</td>
</tr>
<tr>
<td></td>
<td>readevent(e)</td>
<td>Read an event e from the log file</td>
</tr>
<tr>
<td>Security</td>
<td>checkinput(i)</td>
<td>Check input parameters i</td>
</tr>
</tbody>
</table>

provided to the service.

An STS-based model for testing is defined starting from Definition II.3.2 of the standard STS-based model as follows.

Definition III.5.1 (STS_t)

An STS-based model for testing STS_t is a tuple <S, s_1, V, I, A, ID, id> where:

- ID is the set of performance idioms;
- id, with id ∈ ID, extends the transition relation in Definition II.3.2 with idioms.

We note that id = α·γ·µ·id, with α, γ, and µ defined as in Definition II.3.2.

Figure III.5 shows an example of testing model STS_t for the model in Figure III.4. Idiom startclock(t_1) is added to transition (1,2) to start counter t_1, which measures the execution time of operation Signon. Counter t_1 is ended, using idiom stopclock(t_1), in transition (2,3) or (2,4). After function stopclock is called, counter t_1 contains the execution time of operation Signon. Similarly, idiom startclock(t_2) (startclock(t_3), resp.) is added to transition (4,5) (transition (4,7), resp.) to start the counter measuring the execution time for operation CreditAdd (DebitAdd, resp.). Counter t_2 (t_3, resp.) is ended in transition (5,6) (transition (7,8), resp.), using idiom stopclock(t_2) (stopclock(t_3), resp.) and contains the execution time of operations CreditAdd (DebitAdd, resp).

The testing model STS_t is used to monitor the real performance of the service, which in turn is used to refine the simulation model.

III.5.2 STS-based model extended for simulation

According to Figure III.1, we define an STS-based model for simulation (STS_s) by extending the standard STS-based model STS_o with transition probabilities and delay distributions. Transition probabilities model the behavior of the service and the frequency of moving between two states. Here, we assume probabilities to be derived by the frequencies of the service execution paths under real load conditions, while a priori estimates of such probabilities can be inferred by observations on the executions of similar services in the considered environment. Delay distributions model the distribution of waiting times that represent the time needed to complete a given task, such as, the time necessary to execute an operation, to parse an XML input message, or to build a SOAP output message. Distributions of waiting times are specified using the results of the service testing on the total
III. EARLY ASSESSMENT OF SERVICE PERFORMANCE: FULL-KNOWLEDGE AND PARTIAL-KNOWLEDGE SCENARIOS

Figure III.5: An example of STS-based model for testing (STSₜ)

operation execution time. In general, they can also be estimated by considering top-level execution times of similar services and/or by building a performance library, where each service operation/programming language function (e.g., a Java function) is associated with a performance profile [7, 106].

An STS-based model for simulation is defined starting from Definition II.3.2 as follows.

Definition III.5.2 (STSₛ)
An STS-based model for simulation 𝑮𝑺ₛ is a tuple <𝑺,𝑠₁,𝑽,𝑰,𝑨,.prob⁰,distr⁰> where:

- prob ∈ [0, 1] is a transition probability;
- distr is a probability distribution of waiting times;
- \( \overset{\text{prob},\text{distr}}{\longrightarrow} \) extends the transition relation in Definition II.3.2 using probabilities and delay distributions.

We note that \( \overset{\text{prob},\text{distr}}{\longrightarrow} = \alpha,\gamma,\mu,\overset{\text{prob},\text{distr}}{\longrightarrow} \), with \( \alpha, \gamma, \) and \( \mu \) defined as in Definition II.3.2. We also note that 𝑮𝑺ₛ can contain loops that have no number of iterations set a priori. In Section III.5.3, we present our technique to unroll such loops.
III. EARLY ASSESSMENT OF SERVICE PERFORMANCE: FULL-KNOWLEDGE AND PARTIAL-KNOWLEDGE SCENARIOS

Figure III.6 shows an example of the simulation model STS\(_s\) for the model in Figure III.4. Probabilities take values in [0,1], while delays are uniformly distributed between a lower and an upper bound, corresponding to the min and max times needed to complete a task, and represent a random guess based on service operation testing. For instance, transition (7b,7c) takes delay values in [1ms,1ms] and has probability equal to 0.1 and transition (7b,7d) has probability 0.9 and takes delay values in [4ms,7ms]. We note that, for each state, the sum of the probabilities of outgoing edges is equal to 1.

The simulation model STS\(_s\) permits to generate a simulation script for measuring the performance of a service.

III.5.3 Loops unroll technique for the STS-based model for simulation

Since the STS-based models for simulation can contain loops, we propose a technique that allows designers to unfold these into an equivalent sequential structure according to the number of iterations allowed. In this case, the probability of each iteration will be lower than the probability of the previous one. This means that, assuming entering each iteration is independent, given \( p \) as the probability of a loop iteration and \( (1-p) \) as the probability of exiting the loop, the probability of the \( i \)-th iteration is \( p^i \). We therefore propose to unroll such loops in the simulation model by defining a probability threshold \( p_t \) that limits
III. EARLY ASSESSMENT OF SERVICE PERFORMANCE: FULL-KNOWLEDGE AND PARTIAL-KNOWLEDGE SCENARIOS

Figure III.7: Fragment of STS-based model for simulation with loop

Figure III.8: Fragment of STS-based model for simulation after loop unroll

the number of iterations to \( i \), such that \( p_i \geq p_t \) and \( p_{i+1} < p_t \). After the loops unroll, a new sequential structure is obtained with new transition probabilities.

Figure III.7 shows a fragment of STS-based model for simulation that contains a loop between States 2 and 3. We simplify this model by adding only the probabilities on the transitions. If we set the probability threshold \( p_t \) at 0.01, the loop can be exited at the third iteration when the probability of exiting loop will be lower than \( p_t \). This simulation model is equivalent to the structure in Figure III.8 after the existing loop is unfolded after the second iteration. We can observe that at the second iteration, the probability of exiting the loop is reduced and becomes 0.01 instead of 0.1 at the first iteration. At the third iteration, the loop is exited because the probability of exiting the loop is lower than the probability threshold \( p_t \) (transition between States 3" and 4"). The new sequential structure obtained after the loops unroll is used to generate the simulation script.

III.5.4 XML encoding of the STS-based models

We encode standard STS-based models (Definition II.3.2) as XML files following the approach in [107]. In particular, an STS-based model includes a set of elements location specifying the model states, element initialLocation containing the initial state, element locationVars and element interactionVars defining the location and the interaction variables, respectively, element messages specifying the set of operations and their input/output parameters, and element switches modeling the state transitions. Element locationVars (interactionVars, resp.) includes one or more elements locationVar (interactionVar, resp.) each one with its name (element name) and type (element type).
Element messages includes one or more elements message, each one defining an operation name (element name), the direction of the operation (element kind), and one or more parameters (element param). Element kind takes either value input or value output. Element switches consists of one or more elements switch, each one including a source location (element from), a destination location (element to), a message (element message) and its direction (element kind), a guard (element restriction), and an update of location variables (element update).

To define the STS-based models for testing and simulation, we extend the XML encoding in [107] with the following additional elements included within element switch:

- \(<\text{idiom}>\text{idiom1}; \text{idiom2};</\text{idiom}>\) to annotate the model with the performance idioms;
- \(<\text{probability}>\text{value}</\text{probability}>\) to define the probability associated with state transitions;
- \(<\text{distribution}>\text{value}</\text{distribution}>\) to define the delay distribution associated with state transitions.

Figure III.9 shows a fragment of the XML encoding for the STS-based model for testing in Figure III.5 and Figure III.10 shows a fragment of the XML encoding for the STS-based model for simulation in Figure III.6.

## III.6 Implementation

This section presents the implementation of performance interceptors and simulation scripts generator based on the STS-based models for testing STS_t and for simulation STS_s discussed in Section III.5. We note that the interceptors are used to provide reliable values for the total execution time of each service operation. As we shall discuss in Section III.7, in fact, when based on “good” educated guesses, the simulation script provides results whose quality is comparable to the testing ones.

### III.6.1 Implementation of the performance interceptors

Performance interceptors consist of performance monitoring code that is automatically integrated within the service code on the basis of the performance idioms added on the transitions of testing model STS_t as annotations. They are implemented using the Enterprise Java Bean (EJB) interceptors, since EJB interceptors allow the use of separate common code for logging, auditing, performance monitoring, and security checks from business methods [108, 109]. The approach in this chapter considers interceptors for performance monitoring, and is based on the simple idea of using them to monitor the execution time of a single operation from its call to the return of its results. As discussed in [3, 109, 110], a solution based on interceptors is a suitable approach to monitor the service performance.
III. EARLY ASSESSMENT OF SERVICE PERFORMANCE: FULL-KNOWLEDGE AND PARTIAL-KNOWLEDGE SCENARIOS

Figure III.9: A fragment of the XML encoding for STSₗ in Figure III.5
The use of interceptors in fact guarantees a level of synchronization with the real code execution, providing a good approach for a close evaluation of execution and service times.
III. EARLY ASSESSMENT OF SERVICE PERFORMANCE: FULL-KNOWLEDGE AND PARTIAL-KNOWLEDGE SCENARIOS

```java
@Interceptors (ExecutionTimeMeasure.class)
public String DebitAdd (Double amount, String token) {
    // Your code here
}
```

Figure III.11: An example of performance interceptor annotation

In this chapter, we assume service-specific XML serialization/de-serialization and parsing times to be approximated with high accuracy [7, 106] and to not influence the quality of our methodology. Therefore, we do not take them into account neither in testing nor in simulation.

Our approach can be summarized as follows. After the STS-based model for testing STS<sub>t</sub> has been released, our solution iterates through the XML file encoding it, searching for idioms \textit{startclock}. For each idiom, we automatically annotate the relevant service operation with annotation \texttt{@Interceptors} that specifies the interceptor to be used for performance monitoring. Then, each call to the annotated operation triggers the execution of the interceptor (i.e., the method/Java class within the interceptor and annotated with \texttt{@AroundInvoke}). As an example, consider STS<sub>t</sub> in Figure III.5. The model specifies idioms \textit{startclock} and \textit{stopclock}, for operations \texttt{Signon}, \texttt{DebitAdd}, and \texttt{CreditAdd}. Using the idioms, we extend all operations with annotation \texttt{@Interceptors}, as showed in Figure III.11 for operation \texttt{DebitAdd}. We note that the annotation forces the execution of class \texttt{ExecutionTimeMeasure} implementing the interceptor shown in Figure III.12 at each operation call. We also note that the same annotation and interceptor are used for operations \texttt{Signon} and \texttt{CreditAdd}. Interceptor \texttt{ExecutionTimeMeasure} includes method \texttt{ServiceTime}, which is annotated with \texttt{@AroundInvoke} and handles the operation time monitoring. As an example, upon a call to \texttt{DebitAdd} in Figure III.12, the interceptor is executed and first starts the clock (line 4). Then, it calls method \texttt{proceed} (line 7) that monitors the service operation until the result of the operation is returned. After the return of method \texttt{proceed}, the instruction in clause \texttt{finally} is executed to stop the clock and compute the operation (\texttt{DebitAdd} in our example) time (line 12).

The proposed approach is generic, that is, it can be used for programming languages other than Java, and is extensible, meaning that additional interceptors can be integrated by simply extending the set of idioms.

### III.6.2 Implementation of the simulation scripts generator

Simulation scripts provide an estimate of service performance. They are automatically produced by a script generator that takes as input an STS<sub>s</sub> and generates as output a Java-based simulation script. Figure III.13 describes the algorithm of our script generator that works as follows. First, as discussed in Section III.5.2, it unrolls all loops in STS<sub>s</sub> using threshold \( p_t \) (function \texttt{loop_unroll} in \texttt{main}). All loops are converted into a finite
III. EARLY ASSESSMENT OF SERVICE PERFORMANCE: FULL-KNOWLEDGE AND PARTIAL-KNOWLEDGE SCENARIOS

```java
public class ExecutionTimeMeasure {
    @AroundInvoke
    public Object ServiceTime(InvocationContext ctx) throws Exception {
        long startclock = System.currentTimeMillis();
        Object[] parameters = ctx.getParameters();
        try {
            return ctx.proceed();
        } catch (Exception e) {
            logger.warning("Error calling ctx.proceed method");
            return null;
        } finally {
            long stopclock = System.currentTimeMillis() - startclock;
        }
    }
}
```

Figure III.12: An example of performance interceptor class

sequence of switches, all having an alternative corresponding to the backward “stay-in-loop” selection, but with decreasing probability. Then, the generation process visits the XML tree encoding of the “unrolled” STS, using a `foreach` cycle, to set flag `Not visited` for all transitions of the STS-based model. Subsequently, it calls procedure `process_state` that receives as input initial state `s_1`, and recursively visits each state of the model. For each state `s`, it checks if `s` has children (`|children(s)|\geq1`). If that is the case, procedure `add_delay` is called with `s` as input. According to the delay distribution and probability annotations in STS, procedure `add_delay` performs the script code generation for all transitions between `s` and its children by i) producing the delay distribution and ii) generating the code that simulates the service flow using probabilities. To this aim, the procedure searches in the XML file of STS all tags `switch` such that the source state is the state given as input to `add_delay`. If current state `s` has a single child, procedure `add_delay` generates the script code according to the delay distribution of the STS transition between `s` and its child, using method `generate_delay`. As an example, let us consider transition `(7,7a)` in Figure III.6 labeled with `distr="delay in [0ms,4ms]"`. As presented in Figure III.14, method `generate_delay` adds the instruction `Delay(Uniform(0,4))` to the simulation script (line 6), where function `Uniform` denotes a uniform distribution in the specified interval. We note that, at simulation time, the delay is chosen as a random value in interval `[0,4]`.

When state `s` has more than one child, procedure `generate_prob_delay` is called with the set of transitions originating at `s` as input. `generate_prob_delay` first adds an instruction that generates a random number to select the next execution step (i.e., one of the multiple transitions); then, it generates a `switch case` conditional statement modeling all transitions, using the probabilities in STS. For each of such transitions an instruction `Delay` is added using `generate_delay` and the corresponding delay distribution in STS. As an example, let us consider transitions `(7b,7c)` and `(7b,7d)` in Figure III.6 labeled with `p=0.1` and `p=0.9`, respectively. As presented in Figure III.14, method `gener-
III. EARLY ASSESSMENT OF SERVICE PERFORMANCE: FULL-KNOWLEDGE AND PARTIAL-KNOWLEDGE SCENARIOS

INPUT: STS,

OUTPUT: Simulation script

MAIN

Let $e=(s_i,s_j)$ be a transition between states $s_i$ and $s_j$ and $p_t$ the probability threshold

$STS_s = \langle S,s_1,\mathcal{V},\mathcal{A},prob,distr\rangle \Rightarrow loop\_unroll(STS_s,p_t)$

foreach $e_i \in prob,distr$ do

flag($e_i$) := "Not visited"

process_state($s_1$)

PROCESS_STATE($s$)

if $|children(s)| \geq 1$

add_delay($s$)

foreach $s_i \in children(s)$ do

process_state($s_i$)

ADD_DELAY($s$)

if $|children(s)|=1$

if $e.distr \neq null$ with $e=(s,children(s))$

generate_delay($\{e\}$)

else

generate_prob_delay($\{(s,s_i)|s_i \in children(s)\}$)

Figure III.13: Algorithm for simulation script generation

ate_prob_delay first adds a random number generator (line 9) and then a switch case statement following the probabilities and delay distributions in the model (lines 10–19). The script generation ends when all transitions have been visited using generate_delay and generate_prob_delay (i.e., flag set to Visited for all transitions).

Figure III.14 shows the Java-based simulation script that computes the execution time for operation DebitAdd in Figure III.6, generated by applying the algorithm in Figure III.13 to the implementation states of DebitAdd. Method EvaluateServiceTime first starts a counter (line 2) and calls for each state with a single transition instruction Delay. This operation i) randomly selects a waiting time (ms) using the probability distribution given as input, and ii) implements a Java thread that sleeps for the generated waiting time. A switch case statement is then generated for each state with more than one outgoing transitions (lines 14–31). Instructions in lines 3 and 13 permit to generate a random number that is used to select a given transition (conditional statement) and, in turn, the delay associated with this transition. At the end of the simulation the counter contains the sum of the delays generated for each transition. The value of the variable SimulationT is the execution time computed by simulation (line 33).

Similarly, Figure III.14 shows the simulation script generated for operation DebitAdd in Figure III.6.

We note that our solutions for performance interceptor generation and integration and for simulation script generation are developed as part of our framework and also proposed
III. EARLY ASSESSMENT OF SERVICE PERFORMANCE: FULL-KNOWLEDGE AND PARTIAL-KNOWLEDGE SCENARIOS

```java
public long EvaluateServiceTime() {
    long SimulationT = 0;
    Distribution event = new GenerateRandomEvent();

    // transition (7,7a)
    delayvalue = Uniform(0.4);
    Delay(delayvalue);
    SimulationT += delayvalue

    // transition (7a,7b)
    delayvalue = Uniform(1.4);
    Delay(delayvalue);
    SimulationT += delayvalue
    Double pevent = event.nextRandom();

    switch (pevent) {
        // transition (7b,7c) and (7c,7)
        case pevent <= 0.1:
            delayvalue = Uniform(1.1);
            Delay(delayvalue);
            SimulationT += delayvalue
            delayvalue = Uniform(1.1);
            Delay(delayvalue);
            SimulationT += delayvalue
            break
        // transition (7b,7d) and (7d,7)
        case pevent > 0.1:
            delayvalue = Uniform(4.7);
            Delay(delayvalue);
            SimulationT += delayvalue
            delayvalue = Uniform(2.9);
            Delay(delayvalue);
            SimulationT += delayvalue
            break
    }
    return SimulationT;
}
```

Figure III.14: Java-based simulation script for operation DebitAdd

as plugin to be integrated within existing web service development tools Netbeans\(^1\) and Eclipse\(^2\). The framework which integrates the functionalities presented in this section and the derived plugins are shown in Section III.6.3.

III.6.3 Implementation of our solutions

III.6.3.1 Framework STS2Java

The solutions presented in this chapter are proposed as part of our framework called “STS2Java” \cite{111}. This framework provides the functionalities implemented in Section III.6.1 and Section III.6.2 \cite{112} such as:

- Automatic integration of the performance interceptors within the service code based on the STS-based models for testing;

\(^1\)www.netbeans.org
\(^2\)www.eclipse.org

44
III. EARLY ASSESSMENT OF SERVICE PERFORMANCE: FULL-KNOWLEDGE AND PARTIAL-KNOWLEDGE SCENARIOS

```
public long EvaluateServiceTime() {
    long SimulationT = 0;
    Distribution event = new GenerateRandomEvent();

    // transition (5,5a)
    delayvalue = Uniform(0,4);
    Delay(delayvalue);
    SimulationT += delayvalue

    // transition (5a,5b)
    delayvalue = Uniform(1,4);
    Delay(delayvalue);
    SimulationT += delayvalue

    Default pevent = event.nextRandom();
    switch (pevent) {
        // transition (5b,5c) and (5c,5)
        case pevent <= 0.1:
            delayvalue = Uniform(1,1);
            Delay(delayvalue);
            SimulationT += delayvalue
            delayvalue = Uniform(1,1);
            Delay(delayvalue);
            SimulationT += delayvalue
            break
        // transition (5b,5d) and (5d,5)
        case pevent > 0.1:
            delayvalue = Uniform(4,7);
            Delay(delayvalue);
            SimulationT += delayvalue
            delayvalue = Uniform(2,9);
            Delay(delayvalue);
            SimulationT += delayvalue
            break
    }

    return SimulationT;
}
```

Figure III.15: Java-based simulation script for operation CreditAdd

- Automatic generation of the performance interceptor code;
- Automatic simulation script generation based on the STS-based models for simulation.

The framework is developed in Java and allows to generate a Java-based code for the performance idioms and the simulation scripts. Figure III.16 shows the interface of our framework STS2Java. Button “Open STS Model” allows to choose the STS-based models and Button “Generate Code” allows to generate the performance interceptors code from the testing model of service encoded in XML. Button “Generate Script” allows to generate the simulation script from the model of service extended for simulation. The functionalities associated to the remaining buttons visible on this interface are under development and will allow to add more features to our framework.

The framework is provided as plugin for Eclipse and Netbeans to support developers and expert users in the generation of ready-to-use Java-based simulation scripts, starting from an XML-based encoding of the STS-based model of the services.
III. EARLY ASSESSMENT OF SERVICE PERFORMANCE: FULL-KNOWLEDGE AND PARTIAL-KNOWLEDGE SCENARIOS

III.6.3.2 Simulation Plugin STS2Java for Eclipse

This section presents the plugin STS2Java for simulation script generation developed for Eclipse IDE.

Eclipse plugin STS2Java (available at http://sesar.dti.unimi.it/sts2java/) helps developers and expert users in assessing the behavior of a service at design time, via simulation. It is compatible with Eclipse 3.4 and JavaSE-1.6, and implements two main components:

- a parser that checks the validity of the XML encoding of the STS-based simulation model before the generation of the script;

- a generator that generates a Java-based simulation script from a valid STS-based model for simulation.

After the plugin STS2Java has been installed following a traditional Eclipse installation procedure, entry “STS2Java” is added to the Eclipse main menu. Upon starting STS2Java by clicking on the new entry in the menu, the interface in Figure III.17 is shown to the user. The user can then choose the STS-based model for simulation representing the service to be evaluated, and click on button “Next” to reach the interface for simulation script generation shown in Figure III.18. At this point, the user clicks on button “Generate script” to generate the simulation script and on button “Save” to save the script for

Figure III.16: Interface of the framework STS2Java
III. EARLY ASSESSMENT OF SERVICE PERFORMANCE: FULL-KNOWLEDGE AND PARTIAL-KNOWLEDGE SCENARIOS

III.6.3.3 Simulation plugin STS2Java for Netbeans

Similarly to the previous plugin, we provide the same plugin STS2Java for Netbeans IDE. The plugin provides the same functionalities and allows to select the XML encoding of the service models extended for simulation to generate the simulation scripts used to estimate the execution time of the service. It is compatible with Netbeans 7.2.1 and JavaSE-1.6, and is composed of an XML parser and a generator.
III. EARLY ASSESSMENT OF SERVICE PERFORMANCE: FULL-KNOWLEDGE AND
PARTIAL-KNOWLEDGE SCENARIOS

After the plugin STS2Java installation in Netbeans, a new entry “STS2Java” is added as submenu of the Netbeans menu ”Tools”. After clicking on this entry, the main interface of our plugin appears as shown in Figure III.19. After a click on button “Next”, the interface in Figure III.20 allows to select the service model for simulation, which is used at the next step to generate the simulation script (Figure III.21), and save it for execution.
III.7 Experimental evaluation of our methodology

In this section, we experimentally evaluate our simulation approach using the IFX Reverse ATM service (IFX service below) presented in Section III.4. We set up an experimental environment that consists of:

- a workstation (server) containing Apache Tomcat 7 integrated with Axis 2 and a relational database Apache Derby DB 10.9, and equipped with two Intel Core 3 GHz, 4GB RAM, and 200GB of disc storage running Linux Ubuntu 12.04 server;

- a workstation (client) containing SOAP testing tool soapUI (http://www.soapui.org/), and equipped with two Intel Core 2.66 GHz and 4GB of RAM running Mac OS 10.6.8.

Our IFX service was deployed on the server and its performance is tested and simulated using our methodology and the client software in Section III.7.1. The testing and simulation results are compared in Section III.7.2.

III.7.1 Testing and simulation results

We present in the following the performance results obtained for operations SignOn, DebitAdd, and CreditAdd using the following three test case scenarios:

- Test case 1 (tc1): The operation SignOn is invoked with valid credentials, or invalid credentials;
III. EARLY ASSESSMENT OF SERVICE PERFORMANCE: FULL-KNOWLEDGE AND PARTIAL-KNOWLEDGE SCENARIOS

![Graphs showing execution times for different loads of requests]

Figure III.22: Test-based execution times for tc1 varying rps

- Test case 2 (tc2): The operation CreditAdd is invoked with a positive amount that does not exceed the maximum allowed amount, or a positive amount that exceeds it;
- Test case 3 (tc3): DebitAdd is invoked with a positive amount that does not exceed the account balance and the maximum allowed amount, or a positive amount that exceeds at least one of them.

Each test case and simulation consists of 1500 requests where we assume that we have a probability equal to 0.1 to have an amount that exceeds the limits and equal to 0.9 for the case where the amount does not exceed the maximum allowed. For clarity, the Figures in this section present the execution time of the first 150 requests, while the discussion refers to the whole set of 1500 requests.

We first used soapUI to send different loads of requests and test the service operation performance at server side using our interceptors. In particular, based on real data provided by Rototype (http://www.rototype.com/) [113], an important player in the area of self-service kiosks, we selected a baseline load of 5 requests per second (rps), and increased it to 25rps and 100rps to monitor the behavior of the service in case of stressful conditions. Figures III.22, III.23 and III.24 show the results of the execution of tc1 (operation SignOn), tc2 (operation CreditAdd), and tc3 (operation DebitAdd), respectively. The execution times computed for tc1 are between 3.19ms and 22.61ms for 5rps, 3.25ms and 23.06ms for 25rps, and increase up to 37.88ms for 100rps. The execution times computed for tc2 are between 3.02ms and 24.2ms for 5rps, 3.03ms and 25.11ms for 25rps, and increase up to 43.31ms for 100rps. The execution times computed for tc3 are between 3ms and 23.7ms for 5rps, 3.07ms and 24.78ms for 25rps, and increase up to 41.67ms for 100rps. We note that the peaks retrieved for 100rps are due to exceeding thread pools on the server side. We also
III. EARLY ASSESSMENT OF SERVICE PERFORMANCE: FULL-KNOWLEDGE AND PARTIAL-KNOWLEDGE SCENARIOS

Figure III.23: Test-based execution times for $tc_2$ varying rps

Figure III.24: Test-based execution times for $tc_3$ varying rps

note that execution times less than 10ms usually correspond to execution failures, that is, an amount exceeding the limits is given as input to the operations.

We then used the simulation scripts generated for the different operations. For operations $DebitAdd$ and $CreditAdd$, Figures III.14 and III.15 show respectively the simulation scripts used to simulate the IFX service performance. The scripts have been generated following the baseline load (5rps) and their results compared with the test results obtained for the test cases $tc_1$, $tc_2$ and $tc_3$ for the same load. We note that probabilities associated
III. EARLY ASSESSMENT OF SERVICE PERFORMANCE: FULL-KNOWLEDGE AND PARTIAL-KNOWLEDGE SCENARIOS

Table III.2: Mean and standard deviation of execution times

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Testing results</th>
<th>Simulation results (full-knowledge)</th>
<th>Simulation results (partial-knowledge)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>STD</td>
<td>Mean</td>
</tr>
<tr>
<td>tc1</td>
<td>10.09 ms</td>
<td>1.373</td>
<td>10.72 ms</td>
</tr>
<tr>
<td>tc2</td>
<td>10.97 ms</td>
<td>1.985</td>
<td>11.93 ms</td>
</tr>
<tr>
<td>tc3</td>
<td>11.25 ms</td>
<td>1.923</td>
<td>11.97 ms</td>
</tr>
</tbody>
</table>

with transitions in the STS-based model for simulation $STS_s$ are derived by the frequencies of the service execution paths used in the testing phase, while delays of internal tasks are taken from uniform distributions and consider the total operation execution time obtained by testing. Uniform distributions have been used just to evaluate our methodology, while they can be substituted by more complex, even empirical, distributions. To make our simulation independent by a specific set of uniform delay distributions, we produced 100 simulation scripts for $SignOn$, 100 for $CreditAdd$ and 100 for $DebitAdd$, where for each script the delay distributions in STSs are produced as follows. Given the states of a single operation implementation in STSs (e.g., states 7, 7a, 7b, 7c in Figure III.6) and test-based lower $lb$ and upper $ub$ bounds of the execution times for this operation, we randomly associated a uniform delay distribution with each of the corresponding transitions, such that, for each linearly independent path (e.g., path 7-7a-7b-7c-7 in Figure III.6), the sum of the lower (upper, resp.) bounds of all distributions in the path is equal to $lb$ ($ub$, resp.) obtained by testing.

III.7.2 Comparison of testing and simulation results

We evaluated the quality of our simulation approach by comparing testing and simulation results on the basis of the amount of knowledge available on the operation execution times. First, a full-knowledge scenario has been assumed, where the total operation execution times and the internal distributions of delays are known and used in the generation of a single simulation script for $SignOn$, $CreditAdd$ and $DebitAdd$; then, a partial-knowledge scenario has been considered, where only $lb$ and $ub$ are known and 100 simulation scripts randomly generated for $SignOn$, $CreditAdd$ and $DebitAdd$. The full knowledge scenario provides a baseline for the partial knowledge one, which is closer to the “educated guess” of an expert user.

Table III.2 summarizes the mean and standard deviation of the execution times generated by testing and by simulation in the full-knowledge scenario, and the mean of the means and standard deviations generated by the 100 simulation scripts in the partial-knowledge scenario. To better evaluate the quality of our simulation results, we statistically compared the similarity between testing and simulation distributions using a Chi-square test [114]. Chi-square estimates the degree of confidence with which we can claim that two data samples derive from the same distribution. The $\chi^2$ distance is computed for our testing $d_{test}$.
III. EARLY ASSESSMENT OF SERVICE PERFORMANCE: FULL-KNOWLEDGE AND PARTIAL-KNOWLEDGE SCENARIOS

and simulation $d_{sim}$ data using the standard formula:

$$D_{\chi^2}(d_{test}, d_{sim}) = \frac{1}{n} \left( \sum_{i=1}^{n} \left( \frac{(d_{test}(i) - d_{sim}(i))^2}{d_{sim}(i)} \right) \right).$$

Using this formula and table Chi-square, our experiments evaluated the probability that the distributions behind testing and simulation are the same, with 8 degrees of freedom. Degrees of freedom have been estimated using cardinality value of the test data. In the full-knowledge scenario, this probability is higher than 0.96 for the scripts generated from STSs in Figure III.6. In the partial-knowledge scenario, the probability is higher than 0.91 for all simulation scripts.

Our experiments show that simulation scripts can represent a suitable solution for an early assessment of service performance, when only coarse-grained information on the total execution time of each service operation is available. Our approach can then be used at design and development time to evaluate the potential impact of a service on system performance, and at deployment and selection time to support negotiation and evaluation of performance SLAs between the service and its customers as discussed in Chapter V.

III.8 Conclusions

This chapter presented our methodology based on simulation to preliminary assess web service performance. Differently from the previous works presented in the literature such as [4, 66, 67, 37, 70, 73, 74, 15], the proposed solution models services as STSs, much in the same line with some model-based testing works [63, 87], and extends them to gain an early understanding of the service performance. Our methodology builds on coarse-grained measurements of operation execution times to generate a simulation script that well approximate the results obtained by real testing. We experimentally evaluated our approach by comparing testing and simulation results on an IFX Reverse ATM service and we obtained good estimation of the execution times using the simulation scripts generated from the STS-based model extended for simulation. The proposed solution represents a significant step towards the definition of a more general approach that permits to generate a simulation script when the service code and the results of real service executions are not yet available (zero-knowledge scenario). In the next chapter (Chapter IV), we present a solution for the early estimation of service performance assuming a zero-knowledge scenario.
III. EARLY ASSESSMENT OF SERVICE PERFORMANCE: FULL-KNOWLEDGE AND PARTIAL-KNOWLEDGE SCENARIOS
In this chapter, we present our approach that allows service developers and software adopters to evaluate service performance in a zero-knowledge scenario, where neither the service code nor (test-based) information on service execution times are available. Our approach is built on using expert knowledge to estimate the execution time of each service operation and from this, deriving the overall service performance. To achieve this, we first evaluate the complexity of each operation based upon the XML encoding of its input and output parameters, and the Web Service Description Language (WSDL) interface of the service. We then use profile tables providing the time overhead needed to parse and build SOAP messages with different depths and cardinalities, and the performance (retrieved by
testing) of some reference service operations to estimate the operation execution times. We finally experimentally evaluate our approach by using the measured operation execution times to simulate the service performance.

IV.1 Introduction

In the context of early assessment of service performance, it has become crucial to develop techniques capable of using educated guesses of service performance at design time [1], since poor performance, discovered after service deployment, can have catastrophic implications. These techniques will allow designers to make a priori evaluations of the impact a given service might have when integrated in their business process. Although model-driven approaches may support some degree of performance analysis during development [2, 3], the problem is exacerbated by the fact that service code may be not available to or under the control of the party responsible for the evaluation.

Existing approaches to performance evaluation (e.g., [4, 5, 6]) assume the availability of service code or at least of reliable information (e.g., collected by testing) on service behavior. As a consequence, these approaches do not support design-time evaluation of service performance. Our work was inspired by existing estimation models for forecasting the cost, size, resource effort, duration or performance of software projects [8, 9, 10, 11, 12, 115, 116]. These approaches are mainly based on expert analysis, and rely on analogy and statistical methods using historical data. We refer to the scenario in which a priori execution data cannot be either extracted or derived from the testing data as zero-knowledge scenario, where performance evaluation relies on guessing of service behavior and characteristics. In Chapter III, we proposed a model-based approach to evaluate service performance when coarse-grained information on the total execution time interval of each service operation is produced by real testing [112].

In this chapter, we extend the previous solution proposed in order to consider the performance evaluation in the zero-knowledge scenario. In particular, our approach is aimed at simulating service performance when no information on real service/operation execution time is available. As we have said, our approach estimates a range of operation execution times by using expert knowledge, and uses this information to simulate the performance of a given service. In particular, our method first evaluates the complexity of each operation using the XML encoding of its input and output parameters and the WSDL interface of the service. The estimated service complexity together with (i) profile tables providing the time overhead needed to parse and build SOAP messages with different depths and cardinalities, and (ii) the performance of some reference service operations, are used to generate execution time interval estimations for each operation. The produced estimations are then fed into the solution in Chapter III to evaluate service performance. An experimental evaluation is provided along two lines. First, we validate our methodology computing operation complexity using some services crawled on the Internet. Second, we
validate our entire approach to performance simulation using many services developed in-house.

IV.2 Working Assumptions and our Framework

This section presents our working assumptions and an overview of our performance evaluation framework.

IV.2.1 Working Assumptions

Our work is based on the following three working assumptions about the information available for performance evaluation as follows.

**Framework knowledge.** We assume that no real (e.g., measured by testing) information on service performance is available. The framework has access to the WSDL interface of the service, which contains descriptions of service operations, and to the Web Service Conversation Language (WSCL) document specifying the communications between the client and the service in the form of the service operation workflow.

**Expert knowledge.** Along with many existing estimation models (e.g., COCOMO [9]), our framework relies on expert knowledge to tune the simulation results and improve their accuracy. Our framework in fact is targeted at expert users (e.g., service developers, software adopters), who can provide some information on the service under evaluation, as for instance, an estimation of the amount of accesses to internal/external resources (e.g., database, files) required by service operations, and the volume of requested input/output tasks. We note that the more precise the expert knowledge, the more accurate the simulation results.

**Service model.** As described in Chapter III, we assume a model of the service under evaluation as a state automaton that specifies, for each operation, the execution flows with the quickest and longest execution time. We note that the execution time measured in our work is the time needed to serve a request at the server side.

IV.2.2 Performance Evaluation Framework

In addition to our zero-knowledge scenario, evaluation of service performance may involve other scenarios used in Chapter III, called full-knowledge and partial-knowledge, depending on the amount of performance information about the real service used in the evaluation process [112]. In the zero-knowledge scenario, no testing results are considered; only simulation results are used for performance evaluation.

Our framework for the zero-knowledge scenario aims to provide an estimate of service performance by simulation at design time. Our framework defines the relationship between execution time of service operations and parsing and construction times of SOAP messages.
IV. EARLY ASSESSMENT OF SERVICE PERFORMANCE: ZERO-KNOWLEDGE SCENARIO

It relies on the WSDL interface, the operation input/output, and the model of the service. It classifies service operations by classes of complexity on the basis of some reference operations.

Figure IV.1 shows our performance evaluation framework that consists of two main parts:

- **Execution Time Generator (ETG)** estimates the interval of execution times for each service operation;

- **Performance Simulator (PS)** receives as input the interval of execution times and returns as output the simulation results.

We note that the part PS implements the framework proposed in the previous chapter (Chapter III), which generates simulation scripts for performance simulation in a partial-knowledge scenario.

The part ETG which is the main contribution of this chapter extends the solution in Chapter III to the zero-knowledge scenario, that is, by providing PS with reliable simulation information in place of test results. In particular, **Complexity Evaluator** receives as input the WSDL interface and the model of the service to determine the class of complexity of each service operation to be simulated (Section IV.3). Based on the class of complexity, **Execution Time Interval Estimator** uses the structure of the input/output SOAP messages of the operation, a reference service operation (i.e., an operation at the same class of complexity over which an extensive testing has been conducted), and profile tables measuring SOAP message parsing and construction times, to estimate the execution time interval of the operation (Section IV.4.2). A **Monte Carlo procedure** [117, 118] is then applied to the produced intervals to generate different random extractions of the delay distributions. The delay distributions model the distribution of the execution times among the tasks of an operation and are used, together with task execution probabilities, to annotate the transitions.
in the service model representing such tasks. The service model annotated with probabilities and delay distributions in PS represents the simulation model used to evaluate service performance, and is given as input to Simulation Script Generator. Simulation Script Generator generates simulation scripts, which are executed (Execution in Figure IV.1) to predict the behavior of the operation (Result in Figure IV.1). The simulation results are finally compared with real testing results to evaluate their accuracy.

IV.3 Operation Complexity Assessment

This section presents our approach to the evaluation of service operation complexity, which is then used to estimate the interval of execution times and in turn simulate operation performance in the zero-knowledge scenario.

IV.3.1 Building Blocks

Complexity assessment is performed for each operation according to the following building blocks: i) the type of operation input/output parameters in the WSDL interface and ii) the access to internal and/or external resources required by the operation (expert guess).

IV.3.1.1 Operation Types Processing Complexity (OTPC)

It considers the types of the input/output parameters used by the operation under evaluation. To compute the operation types processing complexity OTPC, the primitive and complex datatypes are classified into three classes based on their processing time:

i) easy for the logic, binary, and number datatypes,

ii) medium for date and time datatypes,

iii) difficult for text and XML complex types.

Table IV.1 shows our classification of the datatypes.

A weight is associated with each class according to the datatype processing times we retrieved by testing: 1 for the easy class, 2 for the medium class, and 4 for the difficult class. OTPC is defined as follows.

**Definition IV.3.1 (OTPC)** Given the WSDL of the service, OTPC of operation $o_i$, denoted $OTPC(o_i)$, is the weighted sum of the number of input/output datatypes used by $o_i$ and is computed as

$$OTPC(o_i) = N_c + 2 \times N_m + 4 \times N_d$$

---

1 In our previous solution, we used Symbolic Transition System (STS) to describe the behavior and evolution of a service [16], however our solution is not limited to STS-based models and is suitable for any service modeling approach that allows to enrich state transitions with annotations.
Table IV.1: Classification of XML datatypes

<table>
<thead>
<tr>
<th>Class</th>
<th>Easy</th>
<th>Medium</th>
<th>Difficult</th>
</tr>
</thead>
<tbody>
<tr>
<td>weight=1</td>
<td>weight=2</td>
<td>weight=4</td>
<td></td>
</tr>
</tbody>
</table>

- boolean
- base64Binary-hexBinary
- decimal-double-float
- integer-negativeInteger
- nonNegativeInteger
- positiveInteger
- nonPositiveInteger
- byte-unsignedByte
- int-unsignedInt
- long-unsignedLong
- short-unsignedShort

- date-datetime
- duration-gDay
- gMonth
- gMonthDay
- gYear
- gYearMonth
time

- anyURI-language
- normalizedString
- string-token
- Name-NCName
- NOTATION-QName
- ENTITIES-ENTITY
- ID-IDREF-IDREFS
- NMENTITY
- NMTOKEN
- NMTOKENS

Table IV.2: Scores associated to datatypes complexity

<table>
<thead>
<tr>
<th>Score</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTPC</td>
<td>0</td>
<td>1-12</td>
<td>13-32</td>
<td>33-64</td>
<td>65-100</td>
<td>101+</td>
</tr>
</tbody>
</table>

where $N_e$, $N_m$, and $N_d$ are the number of input/output datatypes in the easy, medium, and difficult classes, respectively.

We note that there is a non-linear relationship between OTPC and its impact on the overall complexity of a service operation. To reduce errors due to our estimations, Table IV.2 maps OTPC onto the ordinal scale 0 to 5 of scores, ranking the operations from the least to the most complex. We denote with $S_{OTP C}(o_i)$, the score of operation $o_i$.

IV.3.1.2 Resource Complexity (RC)

Operation complexity is affected by the amount of resources (i.e., computational resources, information/components) accessed during operation execution, such as databases, files/documents, and applications/services. We distinguish between internal and external resources, where internal resources are inside the perimeter of the service under evaluation, while external resources are outside that perimeter. Resource Complexity (RC) is computed by a weighted sum of the number of internal/external accesses to resources. We assign 1 unit as weight for the internal resources (e.g., access to a local database) and 5 units for each access to the external resources accessible through the network (e.g., call to another web service). The choice of the weights is based on experimental results showing that accesses to local resources are around 5 times faster than accesses to external resources of the same type. The number of internal/external accesses is an estimation given by our expert users. In practice, expert users might give more detailed estimates of the amount of internal/external accesses distinguishing them by type (e.g., number of accesses to databases, files, services); in this case, different weights should be defined for different types of accesses. We assume an internal/external coarse-grained classification to reduce the impact of expert knowledge on our approach. RC is defined as follows.
IV. EARLY ASSESSMENT OF SERVICE PERFORMANCE: ZERO-KNOWLEDGE SCENARIO

Table IV.3: Scores associated to resources complexity

<table>
<thead>
<tr>
<th>Score</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC</td>
<td>0</td>
<td>1-6</td>
<td>7-15</td>
<td>16-25</td>
<td>26-40</td>
<td>41+</td>
</tr>
</tbody>
</table>

**Definition IV.3.2 (RC)** Given the WSDL of the service, RC of operation $o_i$, denoted $RC(o_i)$, is the weighted sum of the number of accesses to internal and external resources required by $o_i$ and is computed as

$$RC(o_i) = N_{int} + 5 * N_{ext} \quad (IV.2)$$

where $N_{int}$ and $N_{ext}$ are the number of accesses to internal and external resources, respectively.

Similarly to OTPC, Table IV.3 maps each RC on scores between 0 and 5. We denote with $S_{RC(o_i)}$, the score of operation $o_i$.

**IV.3.2 Operation Complexity (OC)**

Operation Complexity (OC) is the sum of scores $S_{OTPC}$ and $S_{RC}$ associated with OTPC (Definition IV.3.1) and RC (Definition IV.3.2), and takes values in $[0,10]$. Based on OC, we calculate a class of complexity for the operation and produce a complexity factor, denoted $\gamma$, used to predict the interval of execution times for such operation. OC is defined as follows.

**Definition IV.3.3 (OC)** OC of an operation $o_i$, denoted $OC(o_i)$, estimates the overall complexity of $o_i$, and is calculated as

$$OC(o_i) = S_{OTPC(o_i)} + S_{RC(o_i)} \quad (IV.3)$$

where $S_{OTPC(o_i)}$ and $S_{RC(o_i)}$ are the scores assigned to $OTCP(o_i)$ and $RC(o_i)$ in Equations (IV.1) and (IV.2), respectively.

OC is used to assign an operation to a class of complexity in $\{Basic, Middle, High, Extra High\}$. Table IV.4 shows an example of this assignment. Complexity factor $\gamma \ (\gamma \geq 1)$ is then defined, according to the computed complexity OC and the number $n$ of classes of complexity, to predict the behavior of the operation. $\gamma$ is calculated as

$$\gamma = 1 + \left( \frac{OC}{1+OC} \right)^n \quad (IV.4)$$

We note that in our case, given the above equation and $n=4$, $\gamma$ assumes values between 1 (less complex operation) and 1.68 (most complex operation). We also note that different $\gamma$ can be defined within the same class of complexity.
IV. EARLY ASSESSMENT OF SERVICE PERFORMANCE: ZERO-KNOWLEDGE SCENARIO

Table IV.4: Class of complexity and factor $\gamma$ based on OC

<table>
<thead>
<tr>
<th>Class</th>
<th>Operation Complexity (OC)</th>
<th>Factor $\gamma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1.063</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1.2</td>
</tr>
<tr>
<td>Middle</td>
<td>3</td>
<td>1.32</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1.41</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1.48</td>
</tr>
<tr>
<td>High</td>
<td>6</td>
<td>1.54</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>1.59</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>1.62</td>
</tr>
<tr>
<td>Extra High</td>
<td>9</td>
<td>1.66</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1.68</td>
</tr>
</tbody>
</table>

IV.3.3 Example of complexity evaluation

What follows is an example on the use of our methodology for the evaluation of operation complexity. We obtained from the Internet a sample of 8 web services supplied by the same provider. These services, which provides a total of 67 operations defined in their WSDL files are presented in Table IV.5. A short definition of each of them can be given as follows.

- Service StadiumTransaction contains 12 operations which are used for selling stadium and theater tickets.
- Service CinemaTransaction contains 24 operations which are used for selling cinema tickets.
- Service StadiumData has 7 operations providing users with data about Events and Performances currently available at particular Stadiums and Theaters.
- Service CinemaData has 9 operations providing users with details of current Films and Performances for all available cinemas.
- Service BuyerData provides 5 operations from which you can obtain data about buyers that have been registered.
- Service CinemaSynchronization provides 3 operations which provide synchronization of external applications with the data providers.
- Service StadiumSynchronization provides 3 operations which provide synchronization of external applications with the data providers.
- Service CinemaReservation exposes 4 operations which are used for cinema ticket reservation.

We performed our analysis on the first five services. Then, Table IV.6 shows the results obtained for services StadiumTransaction, CinemaTransaction, StadiumData, CinemaData and BuyerData. Since the selected services are not under our control, we cannot
evaluate the exact operation execution time at the server side. As a consequence, we used the response times (last column in Table IV.6) measured for each operation as the means to evaluate the correctness of an operation complexity estimation. We note that, by choosing a set of services/operations provided by a unique provider, the additional time introduced by client-server communications can be assumed constant and therefore to not substantially influence the comparative complexity evaluation. For these services, we estimate the resources complexity RC by counting the number of tables that involves in the execution of the service operation using the specification in the WSDL.

Our results show that classes of complexity are consistent with response times for the majority of the evaluated operations, meaning that operation response times increase with an increase in complexity. Some inconsistencies can be observed when OC is near to the border between two classes of complexity, where small errors in the expert knowledge can have a huge impact on the complexity estimation. As an example, while operations GetAvailablePerformances and GetRetainSet shows response times that are typical of basic operations, a middle complexity class is associated with both of them. The technique used to estimate the resources complexity may also explain the inconsistencies observed in the overall complexity computed for each service operation. We note that these inconsistencies can also be influenced by client-side network conditions, which can partially affect the calculated response times.

Figure IV.2 shows an example of the evolution of the response times measured with the complexities for the operations proposed by the web service StadiumTransaction.
**IV. EARLY ASSESSMENT OF SERVICE PERFORMANCE: ZERO-KNOWLEDGE SCENARIO**

Table IV.6: Complexity evaluation for web service 1 operations

<table>
<thead>
<tr>
<th>WS</th>
<th>Operation Name</th>
<th>Complexity</th>
<th>Class of Compl.</th>
<th>Factor γ</th>
<th>Resp. Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>OTEC</td>
<td>OTEC</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value Score</td>
<td>Value Score</td>
<td></td>
<td></td>
</tr>
<tr>
<td>StadiumTransaction</td>
<td>CheckPerformanceSeats</td>
<td>3 1 2 1 2</td>
<td>Basic</td>
<td>1.2</td>
<td>304</td>
</tr>
<tr>
<td></td>
<td>CheckPerformanceSectionAreaSeats</td>
<td>8 1 4 1 2</td>
<td>Basic</td>
<td>1.2</td>
<td>306</td>
</tr>
<tr>
<td></td>
<td>CancelTransaction</td>
<td>10 1 2 1 2</td>
<td>Basic</td>
<td>1.2</td>
<td>305</td>
</tr>
<tr>
<td></td>
<td>GetAvailablePerformances</td>
<td>14 2 3 1 3</td>
<td>Middle</td>
<td>1.32</td>
<td>305</td>
</tr>
<tr>
<td></td>
<td>GetRetainSeat</td>
<td>15 2 4 1 3</td>
<td>Middle</td>
<td>1.32</td>
<td>303</td>
</tr>
<tr>
<td></td>
<td>CalculateServiceCharge</td>
<td>22 2 6 1 3</td>
<td>Middle</td>
<td>1.32</td>
<td>272</td>
</tr>
<tr>
<td></td>
<td>CloseTransaction</td>
<td>20 2 6 1 3</td>
<td>Middle</td>
<td>1.32</td>
<td>274</td>
</tr>
<tr>
<td></td>
<td>GetPerformanceSectionAreaPrices</td>
<td>28 2 5 1 3</td>
<td>Middle</td>
<td>1.32</td>
<td>266</td>
</tr>
<tr>
<td></td>
<td>GetPerformanceSectionAreaMap</td>
<td>27 2 6 1 3</td>
<td>Middle</td>
<td>1.32</td>
<td>276</td>
</tr>
<tr>
<td></td>
<td>RenewSeat</td>
<td>14 2 3 2 4</td>
<td>Middle</td>
<td>1.41</td>
<td>379</td>
</tr>
<tr>
<td></td>
<td>GetDeliveryOptions</td>
<td>14 2 8 2 4</td>
<td>Middle</td>
<td>1.41</td>
<td>379</td>
</tr>
<tr>
<td></td>
<td>CloseTransactionAndGetData</td>
<td>116 5 5 1 6</td>
<td>High</td>
<td>1.54</td>
<td>944</td>
</tr>
<tr>
<td>CinemaTransaction</td>
<td>GetSeasonTicketPerformances</td>
<td>19 2 3 1 3</td>
<td>Middle</td>
<td>1.32</td>
<td>301</td>
</tr>
<tr>
<td></td>
<td>GetAvailablePerformances</td>
<td>22 2 3 1 3</td>
<td>Middle</td>
<td>1.32</td>
<td>304</td>
</tr>
<tr>
<td></td>
<td>CheckPerformanceSeats</td>
<td>3 1 2 1 2</td>
<td>Basic</td>
<td>1.2</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td>CheckPerformanceAreaSeats</td>
<td>1 1 3 1 2</td>
<td>Basic</td>
<td>1.2</td>
<td>295</td>
</tr>
<tr>
<td></td>
<td>GetPerformanceAreaDetail</td>
<td>16 2 4 1 3</td>
<td>Middle</td>
<td>1.32</td>
<td>301</td>
</tr>
<tr>
<td></td>
<td>GetPerformancePrices</td>
<td>15 2 4 1 3</td>
<td>Middle</td>
<td>1.32</td>
<td>302</td>
</tr>
<tr>
<td></td>
<td>GetPerformancePromotions</td>
<td>18 2 5 1 3</td>
<td>Middle</td>
<td>1.32</td>
<td>321</td>
</tr>
<tr>
<td></td>
<td>GetPerformanceSupersavers</td>
<td>17 2 5 1 3</td>
<td>Middle</td>
<td>1.32</td>
<td>317</td>
</tr>
<tr>
<td></td>
<td>GetSeasonTicketPrices</td>
<td>15 2 4 1 3</td>
<td>Middle</td>
<td>1.32</td>
<td>303</td>
</tr>
<tr>
<td></td>
<td>GetPerformanceSections</td>
<td>8 1 4 1 2</td>
<td>Basic</td>
<td>1.2</td>
<td>296</td>
</tr>
<tr>
<td></td>
<td>GetPerformanceSectionMap</td>
<td>27 2 6 1 3</td>
<td>Middle</td>
<td>1.32</td>
<td>175</td>
</tr>
<tr>
<td></td>
<td>GetSeasonTicketMap</td>
<td>25 2 5 1 3</td>
<td>Middle</td>
<td>1.32</td>
<td>777</td>
</tr>
<tr>
<td></td>
<td>CalculateServiceCharge</td>
<td>21 2 5 1 3</td>
<td>Middle</td>
<td>1.32</td>
<td>316</td>
</tr>
<tr>
<td></td>
<td>CalculateServiceChargeForAnythingRetention</td>
<td>10 4 10 2 6</td>
<td>High</td>
<td>1.54</td>
<td>871</td>
</tr>
<tr>
<td></td>
<td>RetainSeats</td>
<td>28 2 3 2 4</td>
<td>Middle</td>
<td>1.41</td>
<td>783</td>
</tr>
<tr>
<td></td>
<td>RetainProducts</td>
<td>21 2 4 1 3</td>
<td>Middle</td>
<td>1.32</td>
<td>780</td>
</tr>
<tr>
<td></td>
<td>RetainSeatsAndProducts</td>
<td>34 3 8 2 5</td>
<td>Middle</td>
<td>1.48</td>
<td>614</td>
</tr>
<tr>
<td></td>
<td>RetainAnything</td>
<td>75 4 10 2 6</td>
<td>High</td>
<td>1.54</td>
<td>816</td>
</tr>
<tr>
<td></td>
<td>GetRetainSeat</td>
<td>15 2 5 1 3</td>
<td>Middle</td>
<td>1.32</td>
<td>692</td>
</tr>
<tr>
<td></td>
<td>CloseTransaction</td>
<td>20 2 4 1 3</td>
<td>Middle</td>
<td>1.32</td>
<td>781</td>
</tr>
<tr>
<td></td>
<td>CloseTransactionAndGetData</td>
<td>116 5 7 2 7</td>
<td>High</td>
<td>1.59</td>
<td>916</td>
</tr>
<tr>
<td></td>
<td>CloseTransactionAndMarkPrintedOk</td>
<td>116 5 8 2 7</td>
<td>High</td>
<td>1.59</td>
<td>931</td>
</tr>
<tr>
<td></td>
<td>CancelTransaction</td>
<td>10 1 3 1 2</td>
<td>Basic</td>
<td>1.2</td>
<td>305</td>
</tr>
<tr>
<td></td>
<td>RefundTransaction</td>
<td>15 2 4 1 3</td>
<td>Middle</td>
<td>1.32</td>
<td>806</td>
</tr>
<tr>
<td>StadiumData</td>
<td>GetVersion</td>
<td>4 1 0 0 0</td>
<td>Basic</td>
<td>1.063</td>
<td>308</td>
</tr>
<tr>
<td></td>
<td>GetStadiumActive</td>
<td>2 1 0 0 0</td>
<td>Basic</td>
<td>1.063</td>
<td>308</td>
</tr>
<tr>
<td></td>
<td>GetAllStadiums</td>
<td>12 1 2 1 2</td>
<td>Basic</td>
<td>1.2</td>
<td>308</td>
</tr>
<tr>
<td></td>
<td>GetAllEvents</td>
<td>14 2 2 1 3</td>
<td>Middle</td>
<td>1.32</td>
<td>309</td>
</tr>
<tr>
<td></td>
<td>GetStadiumInfo</td>
<td>33 3 2 1 4</td>
<td>Middle</td>
<td>1.41</td>
<td>310</td>
</tr>
<tr>
<td></td>
<td>GetEventsByStadium</td>
<td>15 2 2 1 3</td>
<td>Middle</td>
<td>1.32</td>
<td>309</td>
</tr>
<tr>
<td></td>
<td>GetPerformances</td>
<td>11 1</td>
<td>Middle</td>
<td>1.32</td>
<td>308</td>
</tr>
<tr>
<td></td>
<td>GetVersion</td>
<td>4 1 0 0 0</td>
<td>Basic</td>
<td>1.063</td>
<td>306</td>
</tr>
<tr>
<td></td>
<td>GetStadiumActive</td>
<td>2 1 0 0 0</td>
<td>Basic</td>
<td>1.063</td>
<td>301</td>
</tr>
<tr>
<td></td>
<td>GetAllStadiums</td>
<td>5 1 1 1 2</td>
<td>Basic</td>
<td>1.2</td>
<td>310</td>
</tr>
<tr>
<td></td>
<td>GetAllShows</td>
<td>8 1 1 1 2</td>
<td>Basic</td>
<td>1.2</td>
<td>308</td>
</tr>
<tr>
<td></td>
<td>GetCinemaInfo</td>
<td>33 3 4 1 4</td>
<td>Middle</td>
<td>1.41</td>
<td>314</td>
</tr>
<tr>
<td></td>
<td>GetStadiumByCinema</td>
<td>11 1 2 1 2</td>
<td>Basic</td>
<td>1.2</td>
<td>309</td>
</tr>
<tr>
<td></td>
<td>GetCinemaByShow</td>
<td>11 1 2 1 2</td>
<td>Basic</td>
<td>1.2</td>
<td>309</td>
</tr>
<tr>
<td></td>
<td>GetShowInfo</td>
<td>29 2 2 1 3</td>
<td>Middle</td>
<td>1.32</td>
<td>310</td>
</tr>
<tr>
<td></td>
<td>GetPerformances</td>
<td>22 2 3 1 3</td>
<td>Middle</td>
<td>1.32</td>
<td>310</td>
</tr>
<tr>
<td>BuyerData</td>
<td>GetVersion</td>
<td>4 1 0 0 0</td>
<td>Basic</td>
<td>1.063</td>
<td>305</td>
</tr>
<tr>
<td></td>
<td>GetByEmail</td>
<td>5 1 4 1 4</td>
<td>Middle</td>
<td>1.41</td>
<td>708</td>
</tr>
<tr>
<td></td>
<td>GetInternalMail</td>
<td>20 3 4 1 4</td>
<td>Middle</td>
<td>1.41</td>
<td>788</td>
</tr>
<tr>
<td></td>
<td>Update</td>
<td>53 3 5 1 4</td>
<td>Middle</td>
<td>1.41</td>
<td>788</td>
</tr>
<tr>
<td></td>
<td>AddNew</td>
<td>54 3 4 1 4</td>
<td>Middle</td>
<td>1.41</td>
<td>798</td>
</tr>
</tbody>
</table>

**IV.4 Execution Time Estimation**

The use of complexity assessment in Section IV.3 and the profile tables providing SOAP message parsing and construction times to predict the interval of execution times of a single operation can now be discussed.
IV. EARLY ASSESSMENT OF SERVICE PERFORMANCE: ZERO-KNOWLEDGE SCENARIO

Table IV.7: Profile tables for DOM APIs

<table>
<thead>
<tr>
<th>Depth</th>
<th>Card</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.097</td>
<td>0.096</td>
<td>0.098</td>
<td>0.102</td>
<td>0.104</td>
<td>0.104</td>
<td>0.105</td>
<td>0.106</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.1</td>
<td>0.099</td>
<td>0.099</td>
<td>0.102</td>
<td>0.106</td>
<td>0.107</td>
<td>0.112</td>
<td>0.112</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.097</td>
<td>0.098</td>
<td>0.103</td>
<td>0.105</td>
<td>0.105</td>
<td>0.105</td>
<td>0.106</td>
<td>0.111</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>–</td>
<td>0.101</td>
<td>0.105</td>
<td>0.106</td>
<td>0.108</td>
<td>0.107</td>
<td>0.112</td>
<td>0.112</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>–</td>
<td>–</td>
<td>0.103</td>
<td>0.106</td>
<td>0.106</td>
<td>0.107</td>
<td>0.109</td>
<td>0.113</td>
<td></td>
</tr>
</tbody>
</table>

(a) Parsing time (ms)

<table>
<thead>
<tr>
<th>Depth</th>
<th>Card</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.455</td>
<td>1.44</td>
<td>1.47</td>
<td>1.53</td>
<td>1.56</td>
<td>1.56</td>
<td>1.575</td>
<td>1.59</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1.5</td>
<td>1.485</td>
<td>1.485</td>
<td>1.53</td>
<td>1.59</td>
<td>1.605</td>
<td>1.68</td>
<td>1.68</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1.455</td>
<td>1.47</td>
<td>1.545</td>
<td>1.575</td>
<td>1.575</td>
<td>1.575</td>
<td>1.59</td>
<td>1.65</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>–</td>
<td>1.515</td>
<td>1.575</td>
<td>1.59</td>
<td>1.62</td>
<td>1.605</td>
<td>1.68</td>
<td>1.68</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>–</td>
<td>–</td>
<td>1.545</td>
<td>1.59</td>
<td>1.59</td>
<td>1.605</td>
<td>1.635</td>
<td>1.695</td>
<td></td>
</tr>
</tbody>
</table>

(b) Construction time (ms)

IV.4.1 Parsing and Construction Profile Tables

Profile tables are used to provide an estimation of the time overhead introduced by parsing and building of SOAP messages with different depths and cardinalities. The depth of a SOAP message is the depth of its XML tree encoding, while the cardinality represents its number of nodes. The overhead depends on and varies according to the type of APIs (i.e., DOM, SAX) used to parse and construct the XML-based SOAP messages. In the following, we consider DOM APIs and build two profile tables, one for the parsing times and one for the construction times. We note that the same approach is used to generate the profile tables for SAX APIs.

To build the parsing profile table for DOM APIs, we write a Java code that receives as input SOAP messages with different depths and cardinalities. For each message, we perform 1500 trials and take the mean parse time for each message. The value obtained for each SOAP message is used to populate the parsing profile table presented as example in Table IV.7(a)), which associates a parsing time with a depth and cardinality. Cells denoted with – represent invalid combinations of depths and cardinalities, that is, scenarios in which the cardinality of an XML message is lower than/equal to its depth.

Starting from the parsing table, we build the construction table for DOM-APIs using approximation. This approach reduces the impact of biased results due to particular structures of SOAP messages, and the generation of a profile table with an estimation of construction times for each combination of depths and cardinalities.\(^2\) Assuming a linear relation between parsing and construction times, our goal is to find the ratio between construction and parsing times; this ratio is then applied to the parsing profile table to generate the construction profile table. Then, we take two examples of SOAP messages

\(^2\)For small scenarios, SOAP constructions times can be measured rather than approximated.
and write a Java program to build them, and measure the individual construction times. The first SOAP message has depth equal to 2 and cardinality equal to 7, while the second 5 and 11. The measured construction times are 1.57ms for the first message and 1.715ms for the second one. If we consider the parsing times for the same depths and cardinalities (0.102ms and 0.113ms in Table IV.7(a)), we note that, as expected, the ratio between construction and parsing times is approximatively the same, that is, the construction times are 15 times slower than the parsing times. We then generate the construction profile table by multiplying each cell in the parsing table by our ratio. Table IV.7(b) shows the construction profile table for DOM APIs. We note that given the ratio between construction and parsing times, we can generate the construction time for a SOAP message with any depth and cardinality, given the corresponding parsing time.

IV.4.2 Execution Time Interval Estimation

We use the profile tables (Section IV.4.1) and factor $\gamma$ (Section IV.3) to estimate the execution time interval of service operations. In particular, we start from the assumption that the execution times of a particular operation can be approximated by applying a degradation factor $\alpha \in [0,1]$ to parsing and construction times. Let us then denote $ET$ the execution time of an operation, $PT$ the parsing time of input SOAP messages, and $CT$ the construction time of output SOAP messages. Our goal is to find factor $\alpha$ that well approximate the operation execution time $ET$ according to the following equation:

$$ET = \frac{PT + CT}{\alpha} \quad \text{(IV.5)}$$

where $PT + CT$ represents the XML management execution time. Since $ET$ is equal to the sum of the XML management execution time and the Business Logic execution time $BL$. Equation (IV.5) can be rewritten as follows:

$$\alpha = \frac{PT + CT}{PT + CT + BL} \quad \text{(IV.6)}$$

We rewrite Equation (IV.6) as:

$$\alpha = \beta * (PT + CT) \quad \text{(IV.7)}$$

where $\beta = \frac{1}{PT + CT + BL}$ is called Business Logic Complexity (BLC). While $PT$ and $CT$ are independent from the considered domain and class of complexity, $BL$ changes with them.

To calculate $\alpha$ for a given service operation, we use reference service operations under our control that can be fully tested and validated. One such operation is implemented for each class of complexity specified in Table IV.4. As an example, for the basic class, we use operation $add$ of a calculator service that implements the sum of two numbers; as another example, for the middle class, we use operation $CreditAdd$ of a reverse ATM service based on the Interactive Financial Exchange (IFX) standard, which allows authenticated
users to deposit funds presented in Chapter III. Given the reference operation having the same class of complexity as that estimated for the operation under evaluation using the approach presented in Section IV.3, we calculate $\alpha_{\text{REF}}$ for the reference operation according to Equation (IV.5) as follows:

$$\alpha_{\text{REF}} = \frac{[PT + CT]_{\text{REF}}}{[ET]_{\text{REF}}} \quad (IV.8)$$

Then, using an approach similar to that used in the COCOMO model [8, 9] and the value obtained for the complexity assessment for the operation under evaluation, we calculate the value of $\alpha$ by applying operation complexity factor $\gamma$ to the factor $\alpha_{\text{REF}}$ of the reference operation as follows:

$$\alpha = [\alpha_{\text{REF}}]^{\gamma} \quad (IV.9)$$

Now, in order to produce the execution time interval of the operation, we need to compute two values for $\alpha$, denoted $\alpha_{\text{min}}$ and $\alpha_{\text{max}}$, which represents the factor for the quickest and longest operation execution times. Then, following our approach we first calculate $\alpha_{\text{REF, min}}$ and $\alpha_{\text{REF, max}}$, which consider the minimum ($[ET]_{\text{REF, min}}$) and maximum ($[ET]_{\text{REF, max}}$) execution times for the reference operation. Clearly, $\alpha_{\text{REF, min}}$ and $\alpha_{\text{REF, max}}$ are computed from Equation (IV.8) as follows:

$$\alpha_{\text{REF, min}} = \frac{[PT + CT]_{\text{REF}}}{[ET]_{\text{REF, min}}} \quad (IV.10)$$

$$\alpha_{\text{REF, max}} = \frac{[PT + CT]_{\text{REF}}}{[ET]_{\text{REF, max}}} \quad (IV.11)$$

and $\alpha_{\text{min}}$ and $\alpha_{\text{max}}$ are finally given by:

$$\alpha_{\text{min}} = [\alpha_{\text{REF, min}}]^{\gamma} \quad (IV.12)$$

$$\alpha_{\text{max}} = [\alpha_{\text{REF, max}}]^{\gamma} \quad (IV.13)$$

The execution times associated to the quickest and longest execution flow for the operation under evaluation are finally obtained from Equation (IV.5) as follows:

$$ET_{\text{min}} = \frac{PT + CT}{\alpha_{\text{min}}} = \frac{PT + CT}{[\alpha_{\text{REF, min}}]^{\gamma}} \quad (IV.14)$$

$$ET_{\text{max}} = \frac{PT + CT}{\alpha_{\text{max}}} = \frac{PT + CT}{[\alpha_{\text{REF, max}}]^{\gamma}} \quad (IV.15)$$

$ET_{\text{min}}$ and $ET_{\text{max}}$ are the outputs of component $ETG$ of our framework in Figure IV.1 and used by component $PS$ to generate the simulation script and estimate the operation performance.
IV. EARLY ASSESSMENT OF SERVICE PERFORMANCE: ZERO-KNOWLEDGE SCENARIO

Table IV.8: Values defined for the data-intensive factor

<table>
<thead>
<tr>
<th>Level (l)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data-intensive Weight Factor $F_{DI}$</td>
<td>$F_{DI} = 1 + \left( \frac{l}{1 + l} \right)^5$</td>
<td>1.03</td>
<td>1.13</td>
<td>1.24</td>
<td>1.33</td>
</tr>
</tbody>
</table>

IV.4.3 Execution Time Adjustment: Data-Intensive Factor

To further refine the execution time interval $[ET_{min}, ET_{max}]$ computed in Section IV.4.2, we evaluate the impact of input/output tasks performed by the operation during its execution flow. We therefore define the data-intensive factor, $F_{DI}$, that takes into account how the use of resources impacts the operation performance. $F_{DI}$ is calculated according to the expert knowledge used to estimate the amount of input/output tasks (data-intensive weight) performed by the operation. Data intensive weights are classified in five different levels $l$ (from 1 to 5), which are then used to calculate the adjustment factor to be applied to the execution times $ET_{min}$ and $ET_{max}$. In particular, given level $l$ calculated according to the data-intensive weight in the expert knowledge and the number of levels $n$, data-intensive factor $F_{DI}$ is calculated as follows:

$$F_{DI} = 1 + \left( \frac{l}{1 + l} \right)^n$$ (IV.16)

Table IV.8 presents the different values computed for $F_{DI}$ according to levels $l$ of data-intensive weights. We note that $F_{DI} > 1$. We also note that in our case the number $n$ of levels is equal to 5, while they can be extended to increase the impact of $F_{DI}$ on the estimation of the execution time interval. We finally note that in case no adjustment is needed (i.e., the data-intensive weight is not specified), $l=0$ and $F_{DI}=1$.

Adjustment factor $F_{DI}$ is applied to Equations (IV.14) and (IV.15), to compute the adjusted bounds for the interval of execution times, as follows.

$$ET_{min} = \frac{PT + CT}{\alpha_{min}} \cdot F_{DI} = \frac{PT + CT}{[\alpha_{REF_{min}}]^\gamma} \cdot F_{DI}$$ (IV.17)

$$ET_{max} = \frac{PT + CT}{\alpha_{max}} \cdot F_{DI} = \frac{PT + CT}{[\alpha_{REF_{max}}]^\gamma} \cdot F_{DI}$$ (IV.18)

The new bounds are then used in the simulation script generation process.

IV.4.4 Generic algorithm for simulation script generation

The simulation scripts allow us to estimate the behavior of a given web service. As we have described earlier, in our zero-knowledge scenario, the simulation scripts are generated by an algorithm that takes as inputs, the SOAP message, the STS-based model describing the simulation, the different values of $\alpha_{REF}$ associated to the reference web service, $\alpha_{REF_{min}}$ and $\alpha_{REF_{max}}$, the degree $\gamma$ associated to the class of complexity and the data-intensive
IV. EARLY ASSESSMENT OF SERVICE PERFORMANCE: ZERO-KNOWLEDGE SCENARIO

INPUT: $S_{\text{Msg}}$: SOAP Message
STS$_s$: STS-based model for simulation
$\alpha_{\text{REF}}_{\text{min}}, \alpha_{\text{REF}}_{\text{max}}$: the values of $\alpha_{\text{REF}}$
$\gamma$: degree associated to the class of the service
$F_{\text{DI}}$: Data-intensive factor
OUTPUT: Simulation script

MAIN

// Retrieve the depth and the cardinality from the XML SOAP message
d = getdepth($S_{\text{Msg}}$)
c = getcard($S_{\text{Msg}}$)

// Read the parsing and construction times in the profile tables
$pct = \text{read\_ProfileTable}(d, c)$

// Compute the bounds of the execution time interval
$ST_{\text{min}} = \frac{pct}{\alpha_{\text{REF}}_{\text{min}}} * F_{\text{DI}}$
$ST_{\text{max}} = \frac{pct}{\alpha_{\text{REF}}_{\text{max}}} * F_{\text{DI}}$

// Use the bounds to run a Monte Carlo process and annotate the model with one random extraction
annotateSTS(STS$_s$, ST$_{\text{min}}$, ST$_{\text{max}}$)

// Generate the simulation script using the algorithm presented in [112]
generatescript(STS$_s$)

Figure IV.3: Algorithm for simulation script generation

factor $F_{\text{DI}}$, and gives as output a Java-based simulation script. The factor $\gamma$ is computed as explained in Section IV.3 and the factor $F_{\text{DI}}$ is chosen by the developer. The algorithm presented in Figure IV.3, is the extension of our algorithm for simulation script generation presented in Chapter III (Section III.6.2). The proposed algorithm works as follows. First, it retrieves the depth and the cardinality of the SOAP message using functions getdepth and getcard (lines 10–11), respectively. The depth and the cardinality are used by function read_ProfileTable to get in the profile table the value of parsing and construction times ($PT+CT$) associated to the given SOAP message (line 13). This value denoted $pct$ is used to compute the lower and upper bounds of service times interval (lines 15–17). The bounds of the interval are used by function annotateSTS which uses a Monte Carlo process to generate randomly many extractions of delay distributions and choose one to annotate the STS-based model for simulation (line 19). The updated STS-based model is given to function generatescript, which is implemented in the simulation script generator presented in Section III.6.2, to generate automatically the Java-based simulation script (line 21). Then, the simulation script is executed and the behavior of the web service obtained can be compared subsequently with the real performance of the service at the end of the development cycle.

IV.4.5 Example of evaluation of the complexity classes parameters

What follows is an example on the use of our methodology to the computation of parameters $\alpha$ and $F_{\text{DI}}$ for the different classes of complexity. These parameters are then used in
IV. EARLY ASSESSMENT OF SERVICE PERFORMANCE: ZERO-KNOWLEDGE SCENARIO

Table IV.9: Parameters for the basic and middle classes of complexity

<table>
<thead>
<tr>
<th>Class of Service</th>
<th>Reference Service</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASIC</td>
<td>Operation Add of Calculator WS</td>
<td>$\gamma \in {1, 1.063, 1.2}$</td>
</tr>
<tr>
<td></td>
<td>$[ST]<em>{REF</em>{\text{min}}} = 3$ ms</td>
<td>$\alpha_{\text{min}} = 0.523$</td>
</tr>
<tr>
<td></td>
<td>$[ST]<em>{REF</em>{\text{max}}} = 15$ ms</td>
<td>$\alpha_{\text{max}} = 0.105$</td>
</tr>
<tr>
<td></td>
<td>$[PT + CT]_{REF} = 1.568$ ms</td>
<td>$\alpha_{\text{min}} \in {0.523, 0.502, 0.46}$</td>
</tr>
<tr>
<td></td>
<td>$\alpha_{\text{max}} = 0.105$</td>
<td>$\alpha_{\text{min}} \in {0.105, 0.091, 0.067}$</td>
</tr>
<tr>
<td>MIDDLE</td>
<td>Operation CreditAdd of IFX-based WS</td>
<td>$\gamma \in {1.32, 1.41, 1.48}$</td>
</tr>
<tr>
<td></td>
<td>$[ET]<em>{REF</em>{\text{min}}} = 3$ ms</td>
<td>$\alpha_{\text{min}} = 0.976$</td>
</tr>
<tr>
<td></td>
<td>$[ET]<em>{REF</em>{\text{max}}} = 24$ ms</td>
<td>$\alpha_{\text{min}} \in {0.969, 0.966, 0.965}$</td>
</tr>
<tr>
<td></td>
<td>$[PT + CT]_{REF} = 2.928$ ms</td>
<td>$\alpha_{\text{max}} = 0.122$</td>
</tr>
<tr>
<td></td>
<td>$\alpha_{\text{max}} = 0.063, 0.052, 0.044$</td>
<td></td>
</tr>
</tbody>
</table>

Section IV.5 to estimate the interval of execution times of an operation that belongs to the same class of complexity and to validate our approach.

As explained before in this section, our methodology relies on profile tables and reference service operations to compute parameters $\alpha$ and $F_{DI}$. In the following, these parameters are evaluated for the class Basic and Middle.

In particular, the reference web service for the class Basic is a simple calculator web service that makes the sum of two numbers and has only one operation, operation add which sends the request of computation and returns the response. For this reference web service, the service times vary between 3 and 15 ms, then $[ST]_{REF_{\text{min}}} = 3$ ms and $[ST]_{REF_{\text{max}}} = 15$ ms. The SOAP message used to query this web service has depth equal to 3 and cardinality equal to 5. Using our profile tables, we obtain 0.098ms for the parsing time and estimate the construction time at 1.47ms, yielding that $[PT + CT]_{REF} = 1.568$ ms. An example of a SOAP message for this service was shown in Figure II.1.

The reference service operation for class Middle is the IFX-based web service operation CreditAdd with execution times varying between 3ms and 24ms, that is, $[ET]_{REF_{\text{min}}} = 3$ms and $[ET]_{REF_{\text{max}}} = 24$ms. Its input SOAP message has depth equal to 9 and cardinality equal to 32 and its output message depth 10 and cardinality 48. Using our profile tables, we obtain 0.183ms for the parsing time and estimate 2.745ms for the construction time, with the result that $[PT + CT]_{REF} = 2.928$ms. According to Equations (IV.10) and (IV.11), $\alpha_{REF_{\text{min}}}$ and $\alpha_{REF_{\text{max}}}$ are calculated. Based on these values, and on $\gamma$, $\alpha_{\text{min}}$ and $\alpha_{\text{max}}$ are computed for the operation under evaluation. Table IV.9 summarizes the parameters of the basic and middle classes of complexity, according to the different values of $\gamma$ presented in Table IV.4.

To conclude, when $\alpha_{\text{min}}$ and $\alpha_{\text{max}}$ have been calculated according to $\gamma$ of the operation under evaluation (see Table IV.9 for class of complexity middle), the lower bound $ET_{\text{min}}$ (the upper bound $ET_{\text{max}}$, resp.) of the execution time interval is computed using Equation (IV.14) (Equation (IV.15), resp.). $ET_{\text{min}}$ and $ET_{\text{max}}$ can then be further refined with adjustment factor $F_{DI}$ using Equations (IV.17) and (IV.18), respectively.
IV.5 Experimental Results and Validation of our approach

We evaluated our approach by setting an experimental environment composed by:

i) a server, which is a workstation containing Apache Tomcat 7 integrated with Axis 2 equipped with a two Intel Core 3 GHz and 4GB RAM running Linux Ubuntu 12.04 server;

ii) a client, which is a workstation containing SOAP testing tool soapUI [21], and equipped with a two Intel Core 2.66 GHz and 4GB RAM running Mac OS 10.6.8.

The subject services were deployed on the server, while the client was used to test and simulate their performance.

To validate our simulation approach, we developed many different services. This allowed us to fully test them, and compare the measured execution times with the ones obtained by simulation by using the approach in this chapter. In the following, we present the detailed results retrieved by considering two of our most relevant services. The results obtained for the remaining services are presented at the end of this section in Table IV.11.

The first web service, denoted $WS_1$, implements a single operation $Generate$ that produces a random number; its input/output SOAP messages have both depth equal to 3 and cardinality equal to 4. The second web service, denoted $WS_2$, implements a single operation $AskDoc$ that allows the users of a medical meeting management system to ask for an appointment with the doctor; its input/output SOAP messages have both depth equal to 3 and cardinality equal to 6. Its standard STS-based model is shown in Figure IV.4. The WSDL files of the two services are provided in Appendix with the standard STS-based model encoded in XML of the medical web service. All the information about the rest of services are available at [http://sesar.dti.unimi.it/hase2014.html](http://sesar.dti.unimi.it/hase2014.html).

To compute the execution time intervals for the two operations, we first evaluated operation complexity $OC$. We obtained $OC=1$ for operation $Generate$ of $WS_1$. This corresponds to the basic class of complexity and implies $\gamma=1.063$. We then considered $F_{DI}=1.03$, 

![Figure IV.4: Standard STS-based model of the service AskDoc](image-url)
because WS_1 does not make intensive access to data. For operation AskDoc of WS_2, OC=4, which corresponds to the middle class of complexity and \( \gamma = 1.41 \). This operation needs up to 6 data accesses to perform its tasks, then we considered \( F_{DI} = 1.24 \). We then computed execution time intervals taking into account information on shortest and longest execution flows. As an example, the values associated with \( \alpha_{min} \) and \( \alpha_{max} \) of operation AskDoc were taken from Table IV.9 for the middle class of complexity. Parsing \( PT \) and construction \( CT \) times were read from the profile tables. Table IV.10 presents our results.

In particular, the interval of execution times [3.18ms, 17.57ms] has been estimated for operation Generate and [2.12ms, 39.3ms] for operation AskDoc. To evaluate the accuracy of our estimation, we tested WS_1 and WS_2, and obtained an execution time interval [3ms, 18ms] for operation the Generate and [2.77ms, 39.76ms] for the operation AskDoc. These results, which are close to our estimation, show that our zero-knowledge approach can contribute to assessing the overall behavior of an operation without any knowledge on its performance. We note that, as discussed in Section IV.3.3, the quality of the complexity estimation provided by our framework can be affected by the accuracy of the expert knowledge for those cases in which the complexity of an operation lies in between two classes of complexity. However, in any case, our solution can give a preliminary insight into the time interval of a given operation, and in turn, as discussed in the remainder of this section, on its performance.

Table IV.11 summarizes the results obtained for the two previous service operation and additional results obtained for other service operations we used to validate our approach. These results show also a good estimation of the interval of execution times for each service operation compared with the real service operation.

The interval of execution times computed with our approach were then given as input to a Monte Carlo procedure that generated a large number of delay distributions, used by our tool introduced in Chapter III. In particular, our tool selected one of the delay distributions and generated the script used to simulate the execution times of 1500 requests for the operation under evaluation. At the same time, operation execution times were
collected by testing the real operation with 1500 requests. We note that the probabilities of executing the quickest and longest execution flows are known in advance, and used in testing and simulation activities. We can extend the experiments using different distributions for the delays which, in our work are assumed uniformly distributed in a range. Figures IV.5(a) and IV.5(b) present the first 150 results obtained for operations \textit{Generate} and \textit{AskDoc} by testing and simulation.

To compute the similarity between the distributions of testing and simulation results and in turn the accuracy of our simulation approach, as in Chapter III, we used the Chi-Square test [114]. The results presented in Table IV.12 show that execution times obtained by simulation are similar to the testing results with a probability up to 0.94 for operation \textit{Generate} and up to 0.79 for operation \textit{AskDoc}. These probabilities confirm for both operations that the two distributions of execution times (testing and simulation) are likely to come from the same population. Table IV.12 also shows the mean and standard deviation of the Chi-Square probability value (P-value).

Figure IV.6 shows the similarity variations for the two examples of web service using Chi-Square test for different random tests conducted on these services. The results are
Table IV.12: Statistical analysis of the results

<table>
<thead>
<tr>
<th>Service operation</th>
<th>Evaluation type</th>
<th>Intervals (ms)</th>
<th>Mean (ms)</th>
<th>STD</th>
<th>Chi-Square (P-value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate WS1</td>
<td>Simulation</td>
<td>[3.18 - 17.57]</td>
<td>11.94</td>
<td>3.51</td>
<td>0.94 0.76 0.077</td>
</tr>
<tr>
<td></td>
<td>Testing</td>
<td>[3 - 18]</td>
<td>12.3</td>
<td>3.07</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Testing</td>
<td>[2.77 - 39.76]</td>
<td>17.33</td>
<td>4.65</td>
<td>0.79 0.59 0.075</td>
</tr>
</tbody>
</table>

Figure IV.6: Chi-Square variation for different tests on WS1 and WS2

higher than 0.5 in all cases.

In summary, as expected, the results in this chapter are less accurate than the one in Chapter III, which considered full-knowledge and partial-knowledge scenarios and where the similarity probabilities were greater than 0.91 in all cases. However, we can observe that our solution to zero-knowledge performance evaluation provides a first estimation on the performance of service operations that can be used for evaluating operation performance at design time. Also, we note that given the WSCL conversation, we can use operation performance obtained by simulation to give an estimation of the performance of the entire service executions. This can be simply achieved by summing up the performance of each operation in the considered service execution flow. Finally, the accuracy of our results can be further refined by:

i) defining different reference services depending on the domain of the service under evaluation (e.g., financial, shopping);

ii) tuning the classes of complexity and related factor $\gamma$;

iii) refining the definition of types of resource access.

Regarding point iii), currently we are only assuming that expert users can estimate the number of required internal and external accesses to resources; in the future, we can assume that expert users are able to estimate also the type (e.g., database, service) of internal and external resources, which are accessed during operation execution.
IV.6 Conclusions

In this chapter, we presented a framework that supports early assessment of service performance. This zero-knowledge scenario requires the definition of new solutions able to guess service execution times at early steps of the service development process. Existing solutions use historical data and expert knowledge [11, 12] to assess the service behavior and the methods methods for assessing the complexity of software, web services, processes, workflows, and systems can not be used in our case [100, 101, 102, 103, 119]. The work in [104] proposes a solution based on queuing theory for an early assessment of the performance of software components using UML diagrams, they focus only on software components and therefore the solution proposed is not applicable to our service-based scenario. Our approach integrates service performance analysis in the early phases of the development process, by estimating the performance of each service operation, using information in the WSDL interface, the input/output SOAP message structure, a model of the service, and some expert knowledge on the complexity of service operations. Our approach first defined a process to evaluate the complexity of a given service operation. Then, it provided a process to estimate the execution time interval of the operation. Finally, it has been integrated within the tool in [112] to produce operation performance estimation by simulation. We experimentally evaluated our approach using different scenarios of complexity. Our simulation results showed a good level of approximation of operation performance; in turn, operation performance may be used to calculate the overall service execution performance (e.g., using a WSCL description of the service conversation).
In the previous chapters, we have proposed our approaches for early assessment of web service performance. In this part of the thesis, we propose the application of our approaches to real world scenarios. Chapter V presents our solutions for Service Level Agreement (SLA) negotiation and monitoring based on performance estimated using our zero-knowledge approach. A use case based on SLA* is proposed to show how this SLA management model can be used to negotiate and monitor service SLAs.
Chapter V

Applications of our approaches for SLA negotiation and monitoring

Contents

V.1 Introduction ................................................. 77
V.2 Framework for service SLA negotiation .......................... 78
V.3 Framework for service SLA monitoring .......................... 79
V.4 SLA negotiation and monitoring: a real use case based on SLA∗ .................. 81
  V.4.1 Overview on SLA∗ ........................................ 81
  V.4.2 SLA generation using SLA∗ abstract syntax ............... 81
  V.4.3 SLA negotiation solution with SLA∗ ....................... 83
  V.4.4 SLA monitoring solution with SLA∗ ....................... 85
V.5 Conclusions ............................................... 85

This chapter presents how the solutions proposed in this thesis that aimed to assess the service performance can be used to negotiate the performance SLAs with service providers, and monitor the performance of the services.

V.1 Introduction

The methodology presented in this thesis aims to evaluate the performance of the service at early stages of the development process using simulation. The simulation data can be used \textit{i}) to negotiate the preliminary performance SLA of the service between the service providers and the customers searching for a given service with a given level of performance, and \textit{ii}) to monitor it. After the development of the service, the SLA can be refined or updated to further match the real performance measured for the service by testing.

For the performance SLA negotiation, the service model for simulation (\textit{STS}$_s$) described in Chapter III, is used to estimate the interval of execution times for the service, generate the simulation script, and study the service behavior before the negotiation.
V. APPLICATIONS OF OUR APPROACHES FOR SLA NEGOTIATION AND MONITORING

For the monitoring of the negotiated performance SLA for the service, we propose to use its STS-based model for testing ($STS_t$), which allows to measure the performance of the service with performance interceptors. The real service implements the performance idioms defined in the service model and uses them to record its performance of the service. Our performance interceptors are the real monitoring probes and provide an accurate evaluation of service performance.

Among existing solutions for SLA management [37, 41, 120, 70], which define the language and describe how the SLA can be negotiated and established between the different parties, we define in this chapter a solution based on SLA$^\ast$ [43]. SLA$^\ast$ allows a machine-readable SLA definition for the service.

In the following of this chapter, we present our frameworks for SLA negotiation and monitoring based on simulation results.

V.2 Framework for service SLA negotiation

Using our approach for service performance simulation, we estimate the performance of the service under evaluation using the model of service extended for simulation. The estimation results are used to generate the first template of the service performance SLA. The SLA template can be used as the basis for advanced negotiation protocols because it provides a starting point of comparison with the service providers proposals [37, 70, 43].

Figure V.1 shows our framework for SLA negotiation. It works as follows. The STS-based models for simulation provided by the service providers are given to our framework $STS2Java$ (See Section III.6.3), which generates the simulation scripts. The $Simulation$ module of this framework is used for the generation. The simulation scripts are executed to have the behavior of the services. This information is saved for each service evaluated in the
V. APPLICATIONS OF OUR APPROACHES FOR SLA NEGOTIATION AND MONITORING

database Simulation Data. The performance stored is used by the SLA Generation module to generate a preliminary template for the service performance SLA. In the template, the level of the execution time to be guaranteed is set using the bounds of the execution times estimated by simulation. The generated SLA drives the SLA evaluation during the SLA negotiation process with the customers in order to choose the most appropriate offer using the SLA Negotiation module. For this, the SLA Negotiation Module works following three possible scenarios:

- the customer receives a service with the generated SLA provided by the service provider. The customer simulates the performance of the service and evaluates how much the proposed SLA for the service is realistic and matches its performance requirements. In the case of positive evaluation, the service is selected, otherwise another service is evaluated.

- The customer receives from the service provider a service with the SLA template generated for the service with the related simulation script. The customer is then able to evaluate the performance SLA of the service using the simulation script. This kind of disclosure by the provider permits to increase the trust of the customer in the service performance SLA.

- The service provider can give an a priori estimation of SLAs to customers using our approach to simulate and discover the behavior of the service, when the service is not still developed. Then, the customers can select the service that matches its requirements, and can verify its behavior with the real performance retrieved after development. The SLA generated for a given service can then be used as the basis in the selection of the service.

After the selection of the service, the SLA is established between the two parties and saved in a database.

The framework for SLA negotiation allows the customers of service to predict the performance impact of the service before the selection according to the performance obtained by simulation. The SLAs database is used as input for the SLA monitoring framework (See Section V.3), after the service deployment and the SLA establishment to detect the violations.

V.3 Framework for service SLA monitoring

To monitor the SLA of the service, the service model for testing is used to add to the implementation of the service our performance interceptors that allow to measure the performance and record all results in a database. The monitoring of the SLA allows to check the non-functional properties of the service and to execute appropriate countermeasures when a problem occurs. The monitoring framework uses the real time information
measured for the service performance with the interceptors. Figure V.2 presents our framework for monitoring existing services, which implements our performance interceptors. This framework is derived from our previous work presented in [121], where we defined a quality architecture for resources allocation in Cloud. The monitoring framework works as follows.

- The Service Unit presents the services offered by the provider to the users and customers. This unit manages also the resources available for the service. The services, which implement the performance interceptors, send their performance to the monitoring information database. The Service Unit communicates with the Monitoring Unit and the Providers Unit by providing the information about the services and their owners.

- The Monitoring Unit monitors the service in order to handle the SLA violations. It collects information about the services and has two modules: Qualitative Measurement Module and SLA Module.

The Qualitative Measurement Module manages all metrics which allow to measure the performance of the service from the monitoring information. This information is saved in the database and is compared with the information intercepted by our performance interceptors in order to evaluate the SLA fulfilment. The metrics are defined based on the information available in the SLAs negotiated for the different
service operations. Our framework allows to measure the trustworthiness of the service provider by checking the SLA violations using the performance intercepted by the interceptors.

The SLA Module has access to the information about the SLAs negotiated for each service contained in the SLAs database and provides these inputs to the Qualitative Measurement Module to evaluate the metrics and handle the possible SLA violations. This module is linked with the Providers Unit to keep information about each service provider and its agreement.

- The Providers Unit manages the service providers. Each service provider is in relation with its SLA and services in order to associate the SLA violations to the appropriate provider and to know who refunds the customers in case of violations.

V.4 SLA negotiation and monitoring: a real use case based on SLA*

In this section, we propose to use SLA* [43], a solution defined for SLA definition and management.

V.4.1 Overview on SLA*

SLA* [43] is a rich, comprehensive, extensible and format independent SLA model defined by the European project SLA@SOI\(^1\) for SLA management. It offers a language for SLA definition using domain specific vocabularies. The syntax defined for SLA* in [43] shows its expressivity for the definition in few statements of the SLAs for the service operation.

An SLA*-based template describes:

- the parties involved in the SLA agreement,
- the definition of the interface of the service concerned by the agreement,
- the specification of the terms of the agreement, which specifies the QoS requirements guaranteed by the agreement.

An example of SLA*-based template is given in the following sections, showing the expressiveness of the syntax defined to describe the service SLA.

V.4.2 SLA generation using SLA* abstract syntax

To allow our framework to have a machine-readable SLA definition that allows the service providers to negotiate the SLA with the customers, we use SLA* abstract syntax in order to generate our SLA templates based on BNF (Backus Naur Form) Grammar.

\(^{1}\)http://sla-at-soi.eu/
The SLA generated has different parts that specify the content of the agreement between the parties and is used as the basis for the negotiation process. It is composed by the description of the parties involved in the agreement, the information about the service concerned by the agreement, and the terms of the agreement that define the performance guarantees from the simulation results. The upper bound of the execution time interval is used to specify that the execution time to be guaranteed for the service operation, needs to be equal to or lower than that bound.

Using SLA$^*$, Figure V.3 shows an example of template generated for the IFX-based web service operation \textit{CreditAdd}, which execution times are between 3 and 24 ms. In this
V. APPLICATIONS OF OUR APPROACHES FOR SLA NEGOTIATION AND MONITORING

template, the completion time allowed for the service operation requests is set to 24 ms. This template is defined by the service providers and provides information that allows the customers to verify the performance requirements during the negotiation process. The information available in the template is compared with the performance required by the customers in order to find the service which can satisfy its expectations.

The SLA generated in Figure V.3 can be described as follows. First, the version of the SLA template is specified (line 2), then the vocabulary used by the template is given (lines 3–4). Here, the core SLA∗ vocabulary is declared to be used in the template. Lines 5–11 define the parties concerned by the established agreement. Here, Alice is the provider of the service and Bob is the customer of the service. Following these declarations, the interface is specified (lines 12–21) and contained the information about the endpoint of the service operation specifying the location of the operation and the protocol used. The url, where the interface of the service is available, is also specified in the template. Lines 22–29 define the variables used in the SLA. The first variable REQ, specifies the request to be sent to the interface IF1 previously defined (lines 23–25). The expression associated to this variable allows to send the request to the service interface and to get the response. The second variable OPT specifies the option of the SLA. It defines two options, basic and high (line 28), and declares the option basic as the default option (line 27). The basic option is the case where the service is executed without considering a strict execution time. The option high is chosen for executions where we need more responsiveness and measures execution times closer to the results obtained from simulation. The last component of the template (lines 30–40) specifies the terms of the agreement decomposed into actions. Depending on the option chosen in the execution of the SLA, the appropriate action is executed. The first action A1 specifies for the option basic that the completion time guaranteed for the service operation is less than 30 ms. The second action A2 sets the completion time equal to or less than 24 ms. For monitoring purposes, another action will be added in Section V.3 to handle the SLA violations and give penalties.

V.4.3 SLA negotiation solution with SLA∗

The selection of the service is based on the SLA definition generated for the service operation which matches better the performance requirements specified by the customers. We note that after the best SLA has been selected, its content can be refined through a negotiation between the service provider and the customer. Initially, the customers send their performance requirements to the negotiation module in Figure V.1. The performance requirements are compared with the SLA template generated for the service operation. In our case, the execution time guaranteed by each provider for the service and specified in its SLA, is compared with the one specified by the customer. The SLA selected is the one whose execution time is equal or near the one required by the customer. In the case where no SLAs do not match the requirements, all of them are dropped. After selection, the SLA of the service operation is updated as shown in Figure V.4. The completion time
is changed in the SLA and the selected provider guarantees less than 25 ms as execution
time when we need high responsiveness of the operation instead of 24 ms required by the
initial template (line 38). It guarantees also an execution time less than 35 ms instead of
30 ms when we select a basic responsiveness for the operation (line 34). The request of the
customers is not satisfied, but among the SLAs proposed by the service providers, it is the
one which values are closer to the customers requirements.

The updated SLA is used in the next section to monitor the SLA violations. Then,
in order to handle the SLA violations, a new action, that specifies the conditions and the
penalty associated to the violations is added to the SLA following SLA* standard definition

![Figure V.4: Example of the final SLA template updated for operation CreditAdd](image)

---

84
V. APPLICATIONS OF OUR APPROACHES FOR SLA NEGOTIATION AND MONITORING

(See Section V.4.4 and Figure V.5). The penalty associated to each violation is also taken into account in the SLA selection criteria.

V.4.4 SLA monitoring solution with SLA∗

We propose in this section a solution based on SLA∗ to monitor SLAs. Our solution extends the set of actions defined in the terms of the SLA in order to specify the action to be performed when the SLA is violated. In particular, the violation condition is specified and the penalty associated to it is given to the customer by the service provider. Initially, the service operation SLA template generated for the negotiation process contains the monitoring action. This is also used by the customers to classify the SLAs proposed by the providers. But, we note that the most important criteria in the selection is the execution time, since the goal of the service provider is to have less violations. Some providers can propose a good execution time and an attractive penalty, when they are sure about the quality of service they propose to the customers.

Figure V.5 shows the final SLA template used to monitor the performance delivered to the users of the IFX-based web service operation CreditAdd. In our case, we add an action A3 (lines 40–49), which monitors the SLA, handles the violations, and gives the penalties to the service provider. Action A3 is mandatory (line 42) and has to be executed each month by service provider Alice (line 44), in order to pay the penalties to the customer which is the recipient in this case. This action triggers the violations of the conditions specified for the two previous actions A1 or A2 according to the responsiveness option chosen by the customer. A penalty of 0.1 euro is earned by the customer Bob for each violation of the agreement by the service provider (line 47).

V.5 Conclusions

We presented our solutions for SLA negotiation and monitoring. The model of the service extended for simulation allows to have an idea about the performance of a service and can be used to negotiate the performance SLA for the service. Using our performance interceptors inside the web service code, the performance of the service can be measured and help to handle the SLA violations. To show the applicability of our approach, we defined an SLA∗-based solution to negotiate and monitor the service SLA. The SLA template is generated first after the simulation results are obtained and is updated after the selection of the service by the customer following different scenarios. The SLA specified the conditions on which it can be monitored and how the violations of the terms of the agreement are handled and the penalties enforced.
V. APPLICATIONS OF OUR APPROACHES FOR SLA NEGOTIATION AND MONITORING

```plaintext
slatemplate {
  version: sla-creditadd-v3
  vocabularies:
    http://sla-at-soi.eu/core
  parties:
    Alice: party{
      role: provider
    }
    Bob: party{
      role: customer
    }
  interfaceDeclrs:
    IF1: interfaceDeclr{
      provider: Alice
      endpoints:
        E1: endpoint{
          location: http://ifxservice.com/IFXService/CreditAdd
          protocol: soap
        }
      interface: http://ifxservice.com/IFXService/CreditAdd
    }
  variables:
    REQ: var{
      expr: IF1/request
    },
    OPT: var{
      expr: basic
      domain: one-of { high, basic }
    }
  terms:
    AT1: agreementTerm{
      A1: state{
        pre: OPT == basic
        post: completion-time(REQ) < 35 ms
      },
      A2: state{
        pre: OPT == high
        post: completion-time(REQ) < 25 ms
      },
      A3: action{
        actor: Alice
        policy: mandatory
        pre: violated[A1.post and A2.post]
        limit: 4 weeks
        post: payment{
          recipient: customer
          value: 0.1 euro
        }
      }
    }
}
```

Figure V.5: Example of final SLA template used to monitor operation CreditAdd
Part IV

Conclusions and future work

This part presents the summary of our main contributions and the future work.
Chapter VI

Conclusions and future work

Contents

| VI.1 | Summary of the contributions | 89 |
| VI.2 | Future work | 90 |
| VI.2.1 | Service composition framework using simulation data | 90 |
| VI.2.2 | Using Simulation as Part of Service Development Cycle | 91 |
| VI.2.3 | Application of our approach for services certification | 92 |
| VI.2.4 | Services performance prediction | 92 |
| VI.2.5 | Extension to other service models | 92 |
| VI.2.6 | Simulation scripts generation according to the load | 92 |
| VI.2.7 | Solution for the interference problem in service composition | 92 |
| VI.2.8 | Move our solution to Cloud | 92 |

The goal of this thesis was to propose a methodology for early assessment of service performance. To reach this aim, our methodology integrates the simulation step in the development cycle of services. In this chapter, we provide the summary of our contributions and discuss some possible research directions our work can consider in the future.

VI.1 Summary of the contributions

In this thesis, we proposed solutions for the early assessment of service performance. We defined a set of techniques that allow to estimate the performance of the service when historical data and source code are not yet available using model-based approach and simulation. Our main contributions can be summarized as follows.

• Early assessment of service performance: Full-knowledge and Partial-knowledge scenarios

We proposed a model-based approach that relies on STS to describe the web services as finite state automata and evaluate their performance. This model was extended for testing and simulation. The testing model annotates model transitions with performance idioms, which allow to evaluate the behavior of the service. The simulation model extends the standard STS-based model with transition probabilities and delay
VI. CONCLUSIONS AND FUTURE WORK

distributions. This model is used to generate a simulation script that allows to simulate the service behavior. Our methodology used simulation along the design and pre-deployment phases of the web service lifecycle to preliminarily assess web service performance using coarse-grained information on the total execution time of each service operation derived by testing. We used testing results and provided some practical examples to validate our methodology and the quality of the performance measurements computed by simulation considering the full-knowledge and partial-knowledge scenarios. The results obtained showed that our simulation gives accurate estimation of the execution times.

- **Early assessment of service performance:** *Zero-knowledge scenario*

  We proposed an approach that permits service developers and software adopters to evaluate service performance in a zero-knowledge scenario, where testing results and service code are not yet available. Our approach is built on expert knowledge to estimate the execution time of the service operation. It evaluates the complexity of the service operation using the input and output SOAP messages, and the Web Service Description Language (WSDL) interface of the service. Then, the operation interval of execution times is estimated based on profile tables providing the time overhead needed to parse and build SOAP messages, and the performance inferred from the testing of some reference service operations. Our simulation results showed that our zero-knowledge approach gives an accurate approximation of the interval of execution times when compared with the testing results at the end of the development.

- **Application of simulation methodology to real world scenarios:** Negotiation and monitoring of service SLA on performance using simulated data

  We proposed an application of our previous approaches to define frameworks that allow to negotiate and monitor the performance SLA of the web service based on the simulation data. The solution for SLA monitoring is based on the STS-based model for testing and the solution for SLA negotiation is based on the service model for simulation. This work allows to have an idea about the SLA of the service in advance and after deployment to handle the SLA violations. An SLA*-based solution is proposed for SLA negotiation and monitoring.

VI.2 Future work

The work presented in this thesis leaves space for further improvements and extensions, which are described in the remainder of this section.

VI.2.1 Service composition framework using simulation data

We propose to extend the application of our solutions to provide a service composition based on the performance information provided by the STS-based model extended
VI. CONCLUSIONS AND FUTURE WORK

for simulation $ST_{S_s}$. Simulation-based approach is chosen because the performance of a composite service cannot be evaluated a priori. The performance of a composite service needs to be evaluated before integration and the code of the service is not available. The simulation script generated from this model helps to guess the performance of the service before its integration in the composed process.

Indeed, the composition scenario recruits the component of the composite service based on the performance claimed by the STS-based model for simulation. Before the recruitment of the component, we need to trust in the claimed performance of the component estimated from the extended STS-based model. For that, a trusted third party will sign the service model of the component used to evaluate its performance. The composition process takes into account two different scenarios.

- First, the scenario where a component of the composite service falls down and there is a need to recruit a new one to replace it following the initial conditions specified in the SLA of the process.

- The second scenario allows to compose all of the process by recruiting the different components compatible with the SLA of the composite service. These components will be recruited in such a way that they satisfy the performance conditions specified in the SLA.

In both scenarios, our work will show that if each of the different component provides a trustworthy STS-based model extended for simulation of the performance, the resulting composite service is compatible with the SLA.

VI.2.2 Using Simulation as Part of Service Development Cycle

In Service-Oriented Architectures (SOA), the quick and accurate evaluation of web service performance is a fundamental problem. Despite the fact that the integration of the simulation step into the development cycle of software/services can allow to evaluate in advance the performance, the integration of simulation as part of service development cycle is still a challenge [2, 3, 81, 112]. To show the application of our methodologies presented along this thesis, we propose to use simulation as part of the development process to assess the performance of a family of web services. The goal is to apply our technique to a family of services in the same domain, which have some correlation and show how the performance of the entire family can be guessed. For that, one of the services is chosen as reference, developed and its performance results are used to estimate the performance of the other services, members of the family. The estimation process uses our zero-knowledge scenario presented in this thesis.
VI.2.3 Application of our approach for services certification

We plan to extend the application of our approach to services certification. The goal is to propose an approach that virtually certifies the model of services used in the evaluation of the service performance in our approach for a given set of performance properties. This can be also used in the case of service compositions.

VI.2.4 Services performance prediction

We plan to provide a solution that allows to predict the performance of services from monitoring data in order to setup an auto-scaling technique. Our solution will use Hidden Markov Models (HMM) and/or Time Series Models to predict the execution times of the services based on the simulated data obtained. This prediction is used in order to reduce the SLA violation by allocating more resources to the service when needed.

VI.2.5 Extension to other service models

The aim is to provide solutions for other service models that exist like UML, petri nets and timed automata. This will allow to extend the usage of our solution to more developers according to their familiarity with one model or another. This extension will allow to generate also the simulation from these models.

VI.2.6 Simulation scripts generation according to the load

This extension will allow to generate simulation scripts according to the load of requests on the service. We plan to use regression technique to estimate the load evolution and to generate the interval of execution times as a function of that and then to generate the load-based simulation scripts.

VI.2.7 Solution for the interference problem in service composition

This extension will propose solution for the interference problem during the composition. It will study how different orders of the components of the composite service can impact the overall performance.

VI.2.8 Move our solution to Cloud

Our goal is to apply our solution to Cloud, to allow an early evaluation of the service performance used to monitor and negotiate the SLA with the cloud provider and also to manage the resources allocation. The simulation model used to estimate the performance of the service will help to predict the behavior of the service or composite service in order to scale automatically the need of resources by the cloud infrastructure provider.
Publications

The works presented in this thesis is presented/submitted to many international conferences. We give here the title and the abstract of the relevant publications.

1. **Zero-Knowledge Evaluation of Service Performance Based on Simulation**
   
   **Acceptance rate:** 40%
   
   **Abstract:** The success of web services is changing the way in which software is designed, developed, and distributed. Services are in fact continuously re-designed and incrementally developed, released in heterogeneous and distributed environments, and selected and integrated at runtime within external business processes. In this dynamic context, there is the need of solutions supporting the evaluation of service performance at an early stage of the software development process, or even at design time, to support users in an a priori evaluation of the impact a given service might have when integrated in their business process.

   In this paper, we provide an approach that permits service developers and software adopters to evaluate service performance in a zero-knowledge scenario, where neither the service code nor (test-based) information on service execution times are available. Our approach builds on expert knowledge to estimate the execution time of each service operation and, in turn, the overall service performance. To this aim, we first evaluate the complexity of each operation using the XML encoding of its input and output parameters, and the Web Service Description Language (WSDL) interface of the service. We then use profile tables providing the time overhead needed to parse and build SOAP messages with different depths and cardinalities, and the performance (retrieved by testing) of some reference service operations to estimate the operation execution times. We finally experimentally evaluate our approach by using the measured operation execution times to simulate the service performance.

2. **STS2Java: An Eclipse Plugin for Early Assessment of Service Performance Based on Simulation**
   (co-authors: Claudio A. Ardagna, Ernesto Damiani) in Proc. of 8th Workshop of the Italian Eclipse Community (Eclipse-IT 2013), Crema, Italy, September 19–20, 2013.

   **Abstract:** Since the emergence of the model-driven development paradigm, there has been a significant effort towards the integration of solutions for the assessment of software performance in the early phases of the software development process. Along this line, we have proposed a framework based on simulation that estimates the performance of web services modeled as Symbolic Transition Systems (STSs). Our
framework uses the STS-based model of the service under evaluation to automatically produce a simulation script for performance estimation. In this paper, we present STS2Java, an implementation of the framework as a plugin for Eclipse IDE, which produces Java-based simulation scripts.

3. Early Assessment of Service Performance Based on Simulation
(co-authors: Claudio A. Ardagna, Ernesto Damiani) in Proc. of 10th International Conference on Services Computing (SCC 2013), Santa Clara, CA, USA, June 27–July 2, 2013.

Acceptance rate: 18%

Abstract: Accurate and rapid evaluation of web service performance is a key problem of Service-Oriented Architecture (SOA), where services are continuously being (re-)designed and released, and integrated within heterogeneous environments. Unfortunately, pre-deployment testing of services is not suitable to evaluate service performance at both design time and runtime. As a result, often process designers get a reliable assessment of service performance only very late in the lifecycle, once services have been deployed, while customers cannot evaluate service behavior at selection time. In this paper we tackle these problems by proposing a methodology that generates a simulation script that can be used for an early assessment of service performance, and to negotiate and evaluate SLAs on service performance at runtime.

4. Quality architecture for resource allocation in cloud computing

Acceptance rate: 32%

Abstract: Quality features are important to be taken into account while allocating resource in Cloud Computing, since it allows to provide to the users or customers, high Quality of Service (QoS) with best response time as example and respects the Service Level Agreement (SLA) established.

Indeed, it is not easy to handle efficiently resource allocation processes in Cloud, since, the applications deployed on Cloud present non-uniform usage patterns, and the cloud allocation architecture needs to provide different scenarios of resource allocation to satisfy the demands and provide quality. In order to provide the measurement of quality indexes, the Cloud resource allocation architecture needs to be proactive and reactive.

The goal of this paper is to propose a resource allocation architecture for Cloud Computing that provides the measurement of quality indicators identified between the Key Performance Indicators (KPI) defined by the Cloud Services Measurement Initiative.
VI. CONCLUSIONS AND FUTURE WORK

Consortium (CSMIC). Our architecture proposes different resource allocation policies: predictive and reactive. The allocation decisions are taken in this architecture, according to the SLA. Finally, the preliminary experimental results show that our proposed architecture can improve quality in Cloud.

5. Resources Provisioning in a Cloud Environment (Research plan)

Abstract: Efficient resources allocation is important in Cloud Computing to satisfy the agreement the provider has with the users and customers. The virtual resources need then to be available when need it. Our research activities will help to handle the allocation of resources in Cloud Computing efficiently by ensuring high availability and scalability. Our work aims to have a high Quality of Service in the Cloud. We will use the most used allocation approaches to manage the resources by proposing some algorithms following reactive and proactive models. The migration of resources for server consolidation will also be used to provide the resources in the Cloud.

6. Study, Design, and Development of Architectural Patterns for Multitenant Cloud (Research plan)

Abstract: Recent advent of cloud computing offers some tangible prospects of reducing the costs of hardware and software monitoring, management, and maintenance. Multi-tenancy, which allows a single application to emulate multiple application instances, has been proposed as a solution. Our research activities will focus on SOA-as-a-service use case and will study and define architectural patterns based on virtualization and models for dynamic improvements of resources that support reliability, scalability and multi-tenancy at PaaS (Platform as a Service) level.
Bibliography


[113] Claudio Santacesaria. *Personal communication on the use of ATMs: load statistics*. Email received on 14th December 2012.


APPENDIX

Random number generator: WSDL file
This web service implements a single operation that generates a random number.

```xml
xmlns:tns="http://rsagbo.org/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://rsagbo.org/" name="GenerateNumber">
<types>
<xsd:schema>
<xsd:import namespace="http://rsagbo.org/"/>
<x:s:element name="generatenumber" type="tns:generatenumber"/>
<x:s:element name="generatenumberResponse" type="tns:generatenumberResponse"/>
<x:s:complexType name="generatenumber">
<x:s:sequence>
<x:s:element name="start" type="xs:int"/>
<x:s:element name="end" type="xs:int"/>
</x:s:sequence>
</x:s:complexType>
<x:s:complexType name="generatenumberResponse">
<x:s:sequence>
<x:s:element name="return" type="xs:int"/>
</x:s:sequence>
</x:s:complexType>
</xsd:schema>
</types>
<message name="generatenumber">
<part name="parameters" element="tns:generatenumber"/>
</message>
```
<message name="generatenumberResponse">
  <part name="parameters" element="tns:generatenumberResponse"/>
</message>

<portType name="GenerateNumber">
  <operation name="generatenumber">
    <input wsam:Action="http://rsagbo.org/GenerateNumber/generatenumberRequest"
      message="tns:generatenumber"/>
    <output wsam:Action="http://rsagbo.org/GenerateNumber/generatenumberResponse"
      message="tns:generatenumberResponse"/>
  </operation>
</portType>

<binding name="GenerateNumberPortBinding" type="tns:GenerateNumber">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
  <operation name="generatenumber">
    <soap:operation soapAction=""/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
</binding>

<service name="GenerateNumber">
  <port name="GenerateNumberPort" binding="tns:GenerateNumberPortBinding">
    <soap:address location="http://localhost:8080/Generate/GenerateNumber"/>
  </port>
</service>
</definitions>
Medical Meeting Management: WSDL file

This web service allows the users of a medical meeting management system to ask for an appointment with the doctor.

```
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401
-wss-wssecurity-utility-1.0.xsd" xmlns:wsp="http://www.w3.org/ns/ws-policy"
xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/
xmlns:tns="http://rsagbo.org/
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://rsagbo.org/" name="AskDocService">
  <types>
    <xsd:schema>
      <xs:import namespace="http://rsagbo.org/"
      <xs:element name="AskDoc" type="tns:AskDoc"/>
      <xs:element name="AskDocResponse" type="tns:AskDocResponse"/>
      <xs:element name="AssignMeeting" type="tns:AssignMeeting"/>
      <xs:element name="AssignMeetingResponse" type="tns:AssignMeetingResponse"/>
      <xs:element name="CheckID" type="tns:CheckID"/>
      <xs:element name="CheckIDResponse" type="tns:CheckIDResponse"/>
      <xs:element name="CheckMeeting" type="tns:CheckMeeting"/>
      <xs:element name="CheckMeetingResponse" type="tns:CheckMeetingResponse"/>
      <xs:element name="generatenumber" type="tns:generatenumber"/>
      <xs:element name="generatenumberResponse" type="tns:generatenumberResponse"/>
      <xs:element name="searchId" type="tns:searchId"/>
      <xs:element name="searchIdResponse" type="tns:searchIdResponse"/>
      <xs:element name="searchMeeting" type="tns:searchMeeting"/>
      <xs:element name="searchMeetingResponse" type="tns:searchMeetingResponse"/>
    </xsd:schema>
    <xs:complexType name="CheckMeeting">
      <xs:sequence>
        <xs:element name="id" type="xs:string" minOccurs="0"/>
        <xs:element name="token" type="xs:string" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="CheckMeetingResponse">
      <xs:sequence>
        <xs:element name="return" type="xs:boolean"/>
      </xs:sequence>
    </xs:complexType>
  </types>
</definitions>
```
<xs:complexType name="generatenumber">
  <xs:sequence>
    <xs:element name="start" type="xs:int"/>
    <xs:element name="end" type="xs:int"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="generatenumberResponse">
  <xs:sequence>
    <xs:element name="return" type="xs:int"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="AskDoc">
  <xs:sequence>
    <xs:element name="id" type="xs:string" minOccurs="0"/>
    <xs:element name="token" type="xs:string" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="AskDocResponse">
  <xs:sequence>
    <xs:element name="return" type="xs:string" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="AssignMeeting">
  <xs:sequence>
    <xs:element name="code" type="xs:string" minOccurs="0"/>
    <xs:element name="isNew" type="xs:boolean"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="AssignMeetingResponse">
  <xs:sequence>
    <xs:element name="return" type="xs:string" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="searchId">
  <xs:sequence>
    <xs:element name="id" type="xs:string" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="searchIdResponse">
  <xs:sequence>
  </xs:sequence>
</xs:complexType>
<xs:element name="return" type="xs:boolean"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="CheckID">
<xs:sequence>
<xs:element name="id" type="xs:string" minOccurs="0"/>
<xs:element name="token" type="xs:string" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="CheckIDResponse">
<xs:sequence>
<xs:element name="return" type="xs:boolean"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="searchMeeting">
<xs:sequence>
<xs:element name="id" type="xs:string" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="searchMeetingResponse">
<xs:sequence>
<xs:element name="return" type="xs:boolean"/>
</xs:sequence>
</xs:complexType>
</xsd:schema>
<message name="generatenumber">
<part name="parameters" element="tns:generatenumber"/>
</message>
<message name="generatenumberResponse">
<part name="parameters" element="tns:generatenumberResponse"/>
</message>
<message name="searchId">
<part name="parameters" element="tns:searchId"/>
</message>
<message name="searchIdResponse">
<part name="parameters" element="tns:searchIdResponse"/>
</message>
<message name="searchMeeting">
<part name="parameters" element="tns:searchMeeting"/>
</message>
<message name="searchMeetingResponse">
<part name="parameters" element="tns:searchMeetingResponse"/>
</message>

<message name="CheckID">
<part name="parameters" element="tns:CheckID"/>
</message>

<message name="CheckIDResponse">
<part name="parameters" element="tns:CheckIDResponse"/>
</message>

<message name="CheckMeeting">
<part name="parameters" element="tns:CheckMeeting"/>
</message>

<message name="CheckMeetingResponse">
<part name="parameters" element="tns:CheckMeetingResponse"/>
</message>

<message name="AssignMeeting">
<part name="parameters" element="tns:AssignMeeting"/>
</message>

<message name="AssignMeetingResponse">
<part name="parameters" element="tns:AssignMeetingResponse"/>
</message>

<message name="AskDoc">
<part name="parameters" element="tns:AskDoc"/>
</message>

<message name="AskDocResponse">
<part name="parameters" element="tns:AskDocResponse"/>
</message>

<portType name="AskDocService">
<operation name="AskDoc">
<input wsam:Action="http://rsagbo.org/AskDocService/AskDocRequest"
message="tns:AskDoc"/>
<output wsam:Action="http://rsagbo.org/AskDocService/AskDocResponse"
message="tns:AskDocResponse"/>
</operation>
</portType>

</binding name="AskDocServicePortBinding" type="tns:AskDocService">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document">
<operation name="generatenumber">
<soap:operation soapAction=""/>
</operation>
</soap:binding>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
<operation name="searchId">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
<operation name="searchMeeting">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
<operation name="CheckID">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
<operation name="CheckMeeting">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
<soap:body use="literal"/>
</output>
</operation>
<operation name="AssignMeeting">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
<operation name="AskDoc">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
</binding>
<service name="AskDocService">
<port name="AskDocServicePort" binding="tns:AskDocServicePortBinding">
<soap:address location="http://localhost:8080/MedicalMeeting/AskDocService"/>
</port>
</service>
</definitions>
Standard STS Model for Medical Meeting Management service
This is the XML encoding of the STS model that represents the Medical Meeting Management web service as an automaton.

<STS>
  <location>1</location>
  <location>2</location>
  <location>3</location>
  <location>4</location>
  <location>5</location>
  <location>5a</location>
  <location>5b</location>
  <location>5c</location>
  <location>5d</location>
  <location>5e</location>
  <location>5f</location>
  <location>6</location>
  <initialLocation>1</initialLocation>
  <locationVars/>
  <interactionVars>
    <interactionVar>
      <name>login</name>
      <type>String</type>
    </interactionVar>
    <interactionVar>
      <name>pwd</name>
      <type>String</type>
    </interactionVar>
    <interactionVar>
      <name>result</name>
      <type>String</type>
    </interactionVar>
    <interactionVar>
      <name>token</name>
      <type>String</type>
    </interactionVar>
    <interactionVar>
      <name>id</name>
      <type>String</type>
    </interactionVar>
  </interactionVars>
<messages>
  <message>
    <name>Signon</name>
    <kind>input</kind>
    <param>login</param>
    <param>pwd</param>
  </message>
  <message>
    <name>Signon</name>
    <kind>output</kind>
    <param>result</param>
    <param>token</param>
  </message>
  <message>
    <name>AskDoc</name>
    <kind>input</kind>
    <param>id</param>
    <param>token</param>
  </message>
  <message>
    <name>AskDoc</name>
    <kind>output</kind>
    <param>result</param>
  </message>
  <message>
    <name>CheckMeeting</name>
    <kind>input</kind>
    <param>id</param>
    <param>token</param>
  </message>
  <message>
    <name>CheckMeeting</name>
    <kind>output</kind>
    <param>result</param>
  </message>
  <message>
    <name>AssignMeeting</name>
    <kind>input</kind>
    <param>result</param>
  </message>
</messages>
<message>
   <name>AssignMeeting</name>
   <kind>output</kind>
   <param>result</param>
</message>

<switches>
   <switch>
      <from>1</from>
      <to>2</to>
      <message>Signon</message>
      <kind>input</kind>
      <restriction>
         login!=null &amp;&amp; pwd!=null
      </restriction>
   </switch>

   <switch>
      <from>2</from>
      <to>3</to>
      <message>Signon</message>
      <kind>output</kind>
      <restriction>
         result==failure
      </restriction>
   </switch>

   <switch>
      <from>2</from>
      <to>4</to>
      <message>Signon</message>
      <kind>output</kind>
      <restriction>
         result==ok
      </restriction>
   </switch>

   <switch>
      <from>4</from>
      <to>5</to>
      <message>117</message>
   </switch>
</switches>
<message>AskDoc</message><kind>input</kind><restriction>id!=null &amp;&amp; token!=null</restriction><update/> <switch><from>5</from><to>5a</to><message>AskDoc</message><kind>input</kind><restriction>id!=null &amp;&amp; token!=null</restriction><update/></switch><switch><from>5a</from><to>5b</to><message>CheckMeeting</message><kind>input</kind><restriction/></switch><switch><from>5b</from><to>5c</to><message>CheckMeeting</message><kind>output</kind><restriction>result==failure</restriction><update/></switch><switch><from>5b</from><to>5d</to><message>CheckMeeting</message><kind>output</kind>
<restriction>
result=new
</restriction>
<update/>
</switch>
<switch>
  <from>5c</from>
  <to>5</to>
  <message>AskDoc</message>
  <kind>output</kind>
  <restriction/>
<update/>
</switch>
<switch>
  <from>5d</from>
  <to>5e</to>
  <message>AssignMeeting</message>
  <kind>input</kind>
  <restriction/>
<update/>
</switch>
<switch>
  <from>5e</from>
  <to>5f</to>
  <message>AssignMeeting</message>
  <kind>output</kind>
  <restriction>
result=ok
</restriction>
<update/>
</switch>
<switch>
  <from>5f</from>
  <to>5</to>
  <message>AskDoc</message>
  <kind>output</kind>
  <restriction/>
<update/>
</switch>
<switch>
<from>5</from>
<to>6</to>
<message>AskDoc</message>
<kind>output</kind>
<restriction/>
<update/>
</switch>
</switches>
</STS>
Complete Java Class for operation *CreditAdd* performance simulation

This is a Java code that allows to simulate the service execution time for the operation CreditAdd of the IFX web service.

```java
/**
 * Template generated by STS2JAVA from your STS model 'STSModelIFXProba.sax'.
 * @author romaric
 */

// Your package name
//package org.rsagbo;

// Add the import here
import java.util.Random;
import statistics.*;

public class STS2Java1 {

  public long EvaluateServiceTime () {
    long beginT = System.currentTimeMillis () ;
    Distribution event = new GenerateRandomEvent () ;
    // transition (5,5a)
    Delay(Uniform(0,4));
    // transition (5a,5b)
    Delay(Uniform(1,4));
    Double pevent = event.nextRandom() ;
    switch ( pevent ) {
      // transition (5b,5c) and (5c,5)
      case pevent <= 0.1:
        Delay(Uniform(1,1));
        Delay(Uniform(1,1));
      // transition (5b,5d) and (5d,5)
      case pevent > 0.1:
        Delay(Uniform(4,7));
        Delay(Uniform(2,9));
    }
```
return System.currentTimeMillis() - beginT;
}

// Delay method that performs the waiting feature
public void Delay(int start, int end) {
    int time = generatedelay(start, end);
    try {
        Thread.sleep(time);
    } catch (InterruptedException ex) {
        Thread.currentThread().interrupt();
    }
}

public int generatedelay(int start, int end) {
    Random randomGenerator = new Random();
    int randomdelay;
    int Start = start;
    int End = end;
    long range = (long) End - (long) Start + 1;
    // compute a fraction of the range, 0 <= frac < range
    long fraction = (long) (range * randomGenerator.nextDouble());
    randomdelay = (int) (fraction + Start);
    return randomdelay;
}

public static void main(String[] arg) {
    // Your code here
    STS2Java sts = new STS2Java();
    long st = 0;
    st = sts.EvaluateServiceTime();
    System.out.println(" The simulated service time is: "+ st);
}
}