María Elena Gómez Martínez

Software Perfomance Assessment at Architectural Level: a Methodology and its Application

Departamento

Informática e Ingeniería de Sistemas

Director/es

Merseguer Hernáiz, José Javier

ISSN 2254-7606

SOFTWARE PERFOMANCE ASSESSMENT AT ARCHITECTURAL LEVEL: A METHODOLOGY AND ITS APPLICATION

Autor

María Elena Gómez Martínez

Director/es

Merseguer Hernáiz, José Javier

UNIVERSIDAD DE ZARAGOZA
Informática e Ingeniería de Sistemas

2013

Software Performance Assessment at Architectural Level: a Methodology and its Application

María Elena Gómez Martínez

Ph.D. DISSERTATION

Departamento de Informática e Ingeniería de Sistemas
Universidad de Zaragoza

Advisor: Dr. José Javier Merseguer Hernáiz

December 2013

A los tres hombres de mi vida, con todo mi amor:
José Antonio, Alejandro y Carlos.

Me lo contaron y lo olvidé;
lo vi y lo entendí;
lo hice y lo aprendí.
Confucio

Sólo sé que no sé nada y,
al saber que no sé nada, algo sé;
porque sé que no sé nada.
Socrates

Agradecimientos

En primer lugar, me gustaría agradecer a mi director de tesis, José Merseguer, por su apoyo, estímulo y buenos consejos para realización de esta tesis, pero sobre todo por la infinita paciencia que ha tenido conmigo. Esta tesis ha sido larga, muy larga, pero al fin ha salido. ¡Muchas gracias!

Mi agradecimiento especial es para uno de los mejores profesores que he tenido, Javier Campos, por darme la oportunidad de poder comenzar mi doctorado. Aunque un poco más tarde de lo habitual, me permitió conocer mi verdadera vocación.

No puedo olvidar de mencionar a mis compañeros de laboratorio en la Universidad de Zaragoza durante la primera parte de mi doctorado: Juan Pablo, Simona, María, Anacris, Dario, Diego... me olvido de muchos, pero me es imposible poner a todos.

También quiero agradecer a mis compañeros (y amigos) de I+D de Technosite en el proyecto INREDIS, entre ellos Lourdes, Esteban y Marta, a Sara, por ser mi "oráculo" particular, a Fausto, por su profundo conocimiento de todo lo relacionado con la discapacidad, a José "Guti", por ayudarme con las pruebas y, en especial, a Rafa por su imprescindible aportación en la arquitectura de interoperabilidad. Trabajar con todos ellos fue un auténtico placer. Allí aprendí el valor de la superación personal de muchas personas con discapacidad y de cómo la tecnología puede mejorar sus vidas.

Tampoco me puedo olvidar de mis compañeros de investigación del grupo Babel en la Universidad Politécnica de Madrid, en particular a Guillem, Ricardo, Clara y, en especial, a Álvaro por sus ánimos en esta última etapa. ¡Los necesitaba!

A mi amiga Esther, por estar siempre ahí para escucharme y darme ánimos en los momentos que más me hacían falta. Me siento afortunada de tener una amiga como ella.

Agradecer a mis familias: De la que vengo, a mis padres y hermanos por enseñarme el valor del trabajo y el esfuerzo; a la que me "encontré", mis suegros Antonio y María del Carmen, por tratarme como a una hija y darme todo su apoyo; y la que he formado junto a mi marido, mis hijos Alejandro y Carlos, por dejarme robarles tiempo para esta tesis. Sus sonrisas son el motor de mi vida. Os quiero mucho.

Finalmente, quiero dar las gracias a mi marido José Antonio. No encuentro palabras para escribir en unas líneas todo lo que le tengo que agradecer. Gracias por ser mi compañero, amigo, pareja, esposo, amante, padre de mis hijos... por compartir conmigo este extraño y apasionante viaje que es la vida. Sin ti, sin tu ayuda y sin tus ánimos, hubiera sido imposible hacer esta tesis. Gracias por todo, cielo. Te quiero.

Resumen

El diseño de la arquitectura de un sistema software es una valiosa herramienta para evaluar las propiedades del mismo, tanto cualitativas como cuantitativas. Conseguir el diseño adecuado es crítico para asegurar la bondad de dichas propiedades. Tomar decisiones tempranas equivocadas puede implicar considerables y costosos cambios en un futuro. Dichas decisiones afectarían a muchas propiedades del sistema, tales como su rendimiento, seguridad, fiabilidad o facilidad de mantenimiento.

Desde el punto de vista de la prestaciones del software, la *ingeniería del rendimiento del software* (SPE) es una disciplina de investigación madura y comúnmente aceptada que propone una evaluación basada en modelos en las primeras fases del ciclo de vida de desarrollo software. Un problema en este campo de investigación es que las metodologías hasta ahora propuestas no ofrecen una interpretación de los resultados obtenidos durante el análisis del rendimiento, ni utilizan dichos resultados para proponer alternativas para la mejora de la propia arquitectura software. Hasta la fecha, esta interpretación y mejora requiere de la experiencia y pericia de los ingenieros software, en especial de expertos en ingeniería de prestaciones. Además, a pesar del gran número de propuestas para evaluar el rendimiento de sistemas software, muy pocos de estos estudios teóricos son posteriormente aplicados a sistemas software reales.

El objetivo de esta tesis es presentar una metodología para asesorar decisiones arquitecturales que tienen impacto en el rendimiento del software. La metodología hace uso del Lenguaje Unificado de Modelado (UML) para representar las arquitecturas software y de métodos formales, concretamente redes de Petri, como modelo de prestaciones. El asesoramiento, basado en patrones y antipatrones, intenta detectar los principales problemas que afectan a las prestaciones del sistema y propone cambios para mejorar dichas prestaciones. Como primer paso, estudiamos y analizamos resultados de rendimiento de diferentes estilos arquitecturales. A continuación, sistematizamos los conocimientos previamente obtenidos para proponer una metodología y comprobamos su aplicabilidad asesorando un caso de estudio real, una arquitectura de interoperabilidad para adaptar interfaces a personas con discapacidad conforme a sus capacidades y preferencias. Finalmente, se presenta una herramienta para la evaluación del rendimiento como un producto derivado del propio ciclo de vida software.

Preface

Software architectures have emerged in the last years as the cornerstone for early evaluation of qualitative and quantitative properties of the software. Software architecture design is a critical issue to get the right system. Wrong decisions at early development phases might imply expensive rework and considerable future changes. Architectural decisions may also directly affect other system properties, such as maintainability, reliability, dependability, security or performance, among others.

From the performance point of view, Software Performance Engineering (SPE) is a mature and well-known research field that proposes a QoS evaluation based on models at early-stages in the life-cycle. A problem in this field is that methodologies generally lack of feedback to interpret performance analysis results and their utilization to propose alternatives to improve the software architecture. Hitherto, it requires skills and experience on the part of software engineers, namely experts in software performance. Furthermore, in spite of the number of proposals to evaluate software performance, very few studies focus on applying these theoretical approaches to the study of real systems.

The aim of this thesis is to present a methodology for assessing software architectures from a performance perspective. The methodology uses the Unified Modeling Language (UML) as a software modeling annotation and the Petri net formalism as performance model. The assessment, based on performance patterns and antipatterns, tries to detect potential performance issues and also tries to enhance software architecture designs for improving system performance, in order to finally propose the optimal configuration of the software architecture. As a first step, we study and analyse performance results of different software architectural styles. Secondly, we build the methodology on the insight previously gained and test its applicability for the performance assessment of a real interoperable architecture for adapting interfaces conforming the preferences and capabilities of people with disabilities. Lastly, we provide a tool for the performance evaluation of software systems as a “by-product” of the software life-cycle.

Contents

List of Figures **v**

List of Tables **vii**

List of Acronyms **xi**

1 Introduction **1**
1.1 Outline 4

2 Preliminary Concepts **5**
2.1 Software Architectures 5
2.1.1 Architectural Style 6
2.1.2 Architecture Assessment 9
2.2 The Unified Modeling Language 10
2.2.1 UML Diagrams 10
2.2.2 SPT/MARTE Profiles 11
2.3 Software Performance Engineering 12
2.3.1 Performance Scenarios and Objectives 13
2.3.2 Performance Models 14
2.3.3 SPE Process 16
2.4 Petri Nets 19
2.4.1 Nets and Net Systems 19
2.4.2 Stochastic Petri Nets 20
2.4.3 Generalized Stochastic Petri Nets 21
2.4.4 Petri Nets Analysis 22
2.5 Conclusions 24

3 Foundations of the Methodology **25**
3.1 CDSS: Web Service Application 26
3.1.1 Performance Issues of Web Services 26
3.1.2 CDSS Web Service 28
3.1.3 Applying SPE to the CDSS 29
3.1.4 Performance Improvements for the CDSS Web Service 33

3.1.5 Related Work . . . 40
3.1.6 Concluding Remarks on CDSS . . . 41
3.2 SPRINGS: Mobile Agents Tracking . . . 42
3.2.1 Mobile Agent Technology . . . 42
3.2.2 System Performance View . . . 46
3.2.3 Performance Models . . . 48
3.2.4 Performance Analysis . . . 54
3.2.5 Related Work . . . 56
3.2.6 Concluding Remarks on SPRINGS . . . 57
3.3 UCH: Universal Control Hub . . . 59
3.3.1 URC-UCH Architecture . . . 60
3.3.2 Applying SPE to UCH . . . 61
3.3.3 Concluding Remarks on UCH . . . 71
3.4 Conclusions . . . 72

4 A Performance Assessment Methodology 73
4.1 Overview of the Methodology . . . 74
4.2 Performance-Oriented Design . . . 76
4.2.1 Software Design: UML diagrams . . . 76
4.2.2 Performance View: Scenarios and Annotations . . . 79
4.3 Performance Model . . . 84
4.4 Performance Analysis . . . 89
4.4.1 Responsiveness . . . 91
4.4.2 Scalability . . . 92
4.4.3 Utilization . . . 93
4.5 Performance Assessment . . . 94
4.5.1 Resource Replication . . . 95
4.5.2 Performance Patterns . . . 96
4.5.3 Performance Antipatterns . . . 100
4.5.4 Optimal Configuration . . . 103
4.6 Related Work . . . 104
4.7 Conclusions . . . 105

5 Industrial Case Study: An Interoperable Architecture 107
5.1 Overview . . . 108
5.2 An Interoperable Architecture . . . 109
5.2.1 First Interaction Scenario . . . 113
5.2.2 Navigation Scenario . . . 118
5.2.3 Device Interaction Scenario . . . 119
5.2.4 Back to Top Scenario . . . 122
5.2.5 Assistive Software Selection Mechanism . . . 122
5.3 Applying the Assessment Methodology . . . 125
5.3.1 Performance-Oriented Design . . . 125
5.3.2 Performance Model . . . 126
5.3.3 Performance Analysis . . . 132

5.3.4 Performance Assessment 139
5.3.5 Optimal Configuration 146
5.3.6 Validation of the Performance Assessment 146
5.3.7 Comparison of Results 148
5.4 Discussion 150
5.4.1 Concerning the Methodology 150
5.4.2 Concerning the Optimal Configuration 152
5.5 Related Work 153
5.6 Conclusions 155

6 The ArgoSPE Tool **157**
6.1 ArgoSPE Features 157
6.1.1 Queries in the Statechart Diagram 158
6.1.2 Queries in the Deployment and Collaboration Diagrams 159
6.1.3 Performance Annotations 159
6.2 Software Architecture 160
6.2.1 Model Editor 160
6.2.2 Model Configurer 161
6.2.3 Model Processor 162
6.3 Related Work 164
6.4 Conclusions 165

7 Final Conclusions and Future Work **167**
7.1 Achievements 167
7.2 Scientific Results 169
7.3 Future Research 172

Bibliography **173**

Index **190**

List of Figures

Figure 2.1: SPE process 17
Figure 2.2: PUMA approach 18

Figure 3.1: CDSS: UML Sequence Diagram for the invocation process . . . 28
Figure 3.2: CDSS: Performance results obtained by Catley et al. (2004) . . 29
Figure 3.3: CDSS: CSM for invocation 30
Figure 3.4: CDSS: CSM for process 31
Figure 3.5: CDSS: GSPN representing the whole scenario 32
Figure 3.6: CDSS: Response times obtained from the GSPN model 33
Figure 3.7: CDSS: UML Sequence Diagram with tunning 35
Figure 3.8: CDSS: Response times for different XML parsers 35
Figure 3.9: CDSS: UML Sequence Diagram with SOAP toolkit 37
Figure 3.10: CDSS: Response times for different SOAP implementations . . 37
Figure 3.11: CDSS: Response times when EPR file increases in size 38
Figure 3.12: CDSS: Response times when CDSS processing time increases . 39
Figure 3.13: SPRINGS: Architecture for mobile agent platforms 43
Figure 3.14: SPRINGS: UML Sequence Diagram for agent movement 45
Figure 3.15: SPRINGS: UML Sequence Diagram for agents communication 46
Figure 3.16: SPRINGS: UML Deployment Diagram of the architecture . . . 47
Figure 3.17: SPRINGS: UML IOD of the performance scenario 48
Figure 3.18: SPRINGS: CSM for agent movement 50
Figure 3.19: SPRINGS: CSM for agents communication 51
Figure 3.20: SPRINGS: GSPN for agent movement 52
Figure 3.21: SPRINGS: GSPN for agents communication 53
Figure 3.22: SPRINGS: Response times 54
Figure 3.23: SPRINGS: Response times when multithreading contexts . . . 55
Figure 3.24: SPRINGS: Response times when multithreading (II) 56
Figure 3.25: SPRINGS: Sensitivity analysis 57
Figure 3.26: UCH: Components of the architecture 61
Figure 3.27: UCH: UML Sequence Diagram for device control process . . . 63
Figure 3.28: UCH: UML Sequence Diagram for a value set request 64
Figure 3.29: UCH: UML Deployment Diagram 65
Figure 3.30: UCH: CSM 67

Figure 3.31: UCH: GSPN for a value set request 68
Figure 3.32: UCH: Response times for 40 users 69
Figure 3.33: UCH: Response times 70
Figure 3.34: UCH: Response times when multithreading 71
Figure 3.35: UCH: Optimal configuration 71

Figure 4.1: Performance assessment methodology loop 74
Figure 4.2: Producer-Consumer example: UML Deployment Diagram. . . . 77
Figure 4.3: Producer-Consumer example: UML State Machine Diagram. . 78
Figure 4.4: Producer-Consumer example: UML Sequence Diagram. 79
Figure 4.5: MARTE: subset of GQAM and GRM 81
Figure 4.6: MARTE: pre-declared NFP types 82
Figure 4.7: Producer-Consumer example: GSPN. 88
Figure 4.8: Producer-Consumer example: Closed-cycle GSPN. 90
Figure 4.9: Producer-Consumer: Scalability 93
Figure 4.10: Producer-Consumer: Utilization 94
Figure 4.11: Producer-Consumer: Resource replication 95
Figure 4.12: Producer-Consumer: Utilization when multithreading 96
Figure 4.13: Pattern detection algorithm 99
Figure 4.14: Antipattern detection algorithm 102

Figure 5.1: INREDIS: UML Deployment Diagram 111
Figure 5.2: INREDIS: UML IOD of the main processes 112
Figure 5.3: INREDIS: UML Sequence Diagram for first interaction 114
Figure 5.4: INREDIS: UML Sequence Diagram for perimeter calculation . 115
Figure 5.5: INREDIS: UML Sequence Diagram for initial interface generation 117
Figure 5.6: INREDIS: UML Sequence Diagram for a navigation 118
Figure 5.7: INREDIS: UML Sequence Diagram for an interface generation 120
Figure 5.8: INREDIS: UML Sequence Diagram for a user's device interaction 121
Figure 5.9: INREDIS: UML Sequence Diagram for a back-to-top 122
Figure 5.10: INREDIS: UML Activity Diagram for AS selection 124
Figure 5.11: INREDIS: GSPN for first interaction scenario 127
Figure 5.12: INREDIS: GSPN for navigation scenario 128
Figure 5.13: INREDIS: GSPN for device interaction scenario 129
Figure 5.14: INREDIS: GSPN for back-to-top scenario 130
Figure 5.15: INREDIS: GSPN for ASSM scenario 131
Figure 5.16: INREDIS: Empirical response times 135
Figure 5.17: INREDIS: Response times of performance models 136
Figure 5.18: INREDIS: Empirical response times of ASSM 137
Figure 5.19: INREDIS: Response times of ASSM using models 138
Figure 5.20: INREDIS: Resource utilization 140
Figure 5.21: INREDIS: Response times when multithreding 141
Figure 5.22: INREDIS: Resource utilization (II) 141
Figure 5.23: INREDIS: Response times when multithreding (II) 142
Figure 5.24: INREDIS: Response times when multithreding (III) 142

Figure 5.25: INREDIS: Response times with Fast Path pattern 143
Figure 5.26: INREDIS: The Ramp antipattern 145
Figure 5.27: INREDIS: Response times for specific recommender of ASSM . 146
Figure 5.28: INREDIS: Response times for the optimal configuration 147
Figure 5.29: INREDIS: Comparison of results (I) 148
Figure 5.30: INREDIS: Comparison of results (II) 149

Figure 6.1: ArgoSPE: An example 158
Figure 6.2: ArgoSPE: Architecture proposed by OMG 161
Figure 6.3: ArgoSPE: Snapshot 162
Figure 6.4: ArgoSPE: Example of a configuration file 163

List of Tables

Table 3.1: CDSS: Mean execution times of XML parsers 34
Table 3.2: CDSS: Mean execution times of SOAP toolkits 36
Table 3.3: CDSS: Mean execution times of EPR size 38
Table 3.4: SPRINGS: Mean execution times of operations 49
Table 3.5: UCH: Mean execution times of operations 65

Table 4.1: Subset of performance annotations in MARTE. 85
Table 4.2: Translation rules of the performance annotations in MARTE into GSPN models. 86
Table 4.3: Performance patterns 98

Table 5.1: INREDIS: Results in seconds for each performance scenario . . 136
Table 5.2: INREDIS: Probabilities rates of ASSM 137

Table 6.1: ArgoSPE: Supported performance annotations 160

List of Acronyms

AD Activity Diagram

AS Assistive Software

ASSM Assistive Software Selection Mechanism

CDSS Clinical Decision Support Systems

CSM Core Scenario Model

CTMC Continuous-Time Markov Chain

DD Deployment Diagram

DOM Document Object Model

EQN Extended Queueing Network

FTP File Transfer Protocol

GRM Generic Resource Modeling

GSPN Generalized Stochastic Petri Nets

GQAM Generic Quantitative Analysis Modeling

HCI Human Computer Interaction

HTTP HyperText Transfer Protocol

ICT Information and Communications Technology

INREDIS INterfaces for RElations between Environment and people with DISabilities

IOD Interaction Overview Diagram

ISO International Standard Organization

KB Knowledge Base

LQN Layered Queueing Network

MARTE Modeling and Analysis of Real-Time and Embedded systems

MDA Model Driven Architecture

NFP Non Functional Properties

OMG Object Management Group

PASA Performance Assessment of Software Architectures

PDF Probability Distribution Function

PEPA Performance Evaluation Process Algebra

pmf probability mass function

PN Petri nets

PSM Platform Specific Models

PUMA Performance by Unified Model Analysis

QN Queueing Network

QoS Quality of Service

SA Software Architecture

SAX Simple API for XML

SD Sequence Diagram

SM State Machine Diagram

SMG Stochastic Marked Graph

SMTP Simple Mail Transfer Protocol

SOA Service-Oriented Architecture

SOAP Simple Object Access Protocol

SPA Stochastic Process Algebra

SPE Software Performance Engineering

SPN Stochastic Petri net

SPPN Stochastic Process Petri Net

SPT UML profile for Schedulability, Performance and Time

TCP/IP Transmission Control Protocol/Internet Protocol

TPN Timed Petri net

TVL Tag Value Language

UCH Universal Control Hub

UC Use Case Diagram

UDDI Universal Description Discovery and Integration

UIML User Interface Markup Language

UML Unified Modeling Language

URC Universal Remote Console

VSL Value Specification Language

W3C World Wide Web Consortium

WS Web Service

WSDL Web Service Description Language

XHTML eXtensible HyperText Markup Language

XMI XML Metadata Interchange

XML eXtensible Markup Language

XPP XML Pull Parser

Chapter 1

Introduction

Software architectures have emerged in the last years as the cornerstone for early evaluation of qualitative and quantitative properties of the software. Software architecture design represents one of the earliest decisions made in the software development life-cycle. This decision is critical to get the right system, because of the difficulties to introduce changes once the system has been deployed. Wrong decisions at early development phases may imply an expensive rework to fix involving considerable changes at any stage of the software development life-cycle.

ISO 42010:2011 standard defines the concept of software architecture as *the fundamental organization of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its designs and evolution.* This standard, as well as most of the definitions regarding software architecture, chiefly focuses on modeling the structure and behaviour of a system. Obviously, the software architecture must deliver functional requirements, i.e, how it should behave. Nevertheless their compliance is not enough for assuring its quality, which is also determined by how well non-functional requirements (or non-functional properties, NFP) are met, i.e., how it should perform. Thus, architectural decisions may directly affect non-functional properties, such as maintainability, reliability, dependability, security or performance, among others.

Software performance is a pervasive quality difficult to understand, because it is affected by every aspect of the design, code, and execution environment (Woodside et al., 2007). Performance influences all the components in a software system. Hence, it must be analysed by considering the software architecture as a whole and how its components interact. Software performance is then a non-functional property related to software quality that it is not always taken into account at the design stage notwithstanding.

Software performance can be defined by means of performance objectives, which specify quantitative criteria for evaluating the performance characteristics of the software system (Smith and Williams, 2002b). According to Smith (1990), performance objectives can be expressed in several different ways, including response time, throughput or constrains on resource usage (or utilization).

Therefore, software performance evaluation focuses on the dynamic behaviour analysis and the prediction of these performance indices and measures. Unfortunately, performance evaluation of software systems has been traditionally accomplished after deployment. This is the well-known *"fix-it later"* approach and it has well-known problems (Smith, 1990). For example, the cost of re-architecting, re-implementing and re-deploying the system when performance goals are not fulfilled. Also the over-budget for being out of schedule as Woodside et al. (2007) described.

Software Performance Engineering (SPE) was defined by Smith in 1981 as *a systematic and quantitative approach to build software systems that meet performance objectives*. SPE represents the entire collection of software engineering activities and related analyses used throughout the software development cycle, which are directed to meeting performance requirements (Woodside et al., 2007; Cortellessa et al., 2011).

The usual way in SPE for introducing a performance specification is by annotating the design diagrams. Annotations account for properties such as workload, host demands or routing rates. Following SPE principles, the aforementioned design diagrams annotated with performance information are translated into performance models. Performance models are formal models that help to obtain performance measures of interest (e.g. system response time) by analysis or simulation, and thence to evaluate software systems.

Nowadays, the Unified Modeling Language (UML), defined by the Object Management Group (OMG, 2011b), is the industry-standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems (Rumbaugh et al., 1999; Fowler, 2000). UML is enriched with profiles, i.e., tailored subsets of the UML metamodel for specific purposes, such as transactional, dependability or fault-tolerant systems. The UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) described by OMG (2011a) is a standard that customizes UML for the modeling and analysis of performance properties.

Regarding performance models, there are different kinds of performance formalisms widely accepted in SPE: queuing networks (Lazowska et al., 1984), stochastic process algebras (Hermanns et al., 2002) and stochastic Petri nets (Ajmone Marsan et al., 1995). There exist SPE methodologies that translate performance-annotated UML models into the formalisms above mentioned. For example, the work in (Petriu and Woodside, 2002; Petriu and Shen, 2002) to obtain queuing networks, the work in (Tribastone and Gilmore, 2008) to obtain process algebras or (Bernardi and Merseguer, 2007; Distefano et al., 2011) to obtain Petri nets. Using the latter formalisms, PUMA (Performance by Unified Model Analysis) (Woodside et al., 2005, 2013) is a transformation model chain that proposes to carry out the performance analysis using intermediate performance models, concretely Core Scenario Models (CSM). Some of these approaches have associated tools that automate the translation process.

Nevertheless, these approaches generally lack of feedback to interpret performance analysis results and their utilization to propose alternatives to improve the software architecture is not immediate. Hitherto, it requires skills and experience on the part of software engineers, namely experts in software performance. Furthermore, in spite of the large number of proposals to evaluate software performance, very few studies focus on applying these theoretical approaches in the study of real systems.

A solution to this problem could be the development of a systematic and structured approach to evaluate architectural decisions in order to close the "assessment loop", that is, Design $\rightarrow$ Performance Model $\rightarrow$ Analysis $\rightarrow$ Results $\rightarrow$ new Design. This performance assessment methodology based on SPE principles and techniques allows us to systematically evaluate performance of software architectures in the design stage, the detection of design errors and subsequent assessment for their correction. It tries thus to enhance the initial software architecture design for improving system performance.

To the best of our knowledge, there are very few initiatives to assess architectures based on SPE principles. An exception is the PASA (Performance Assessment of Software Architectures) method, proposed by Williams and Smith (2002). PASA is a performance scenario-based software architecture analysis method that provides a framework for the whole assessment process. Nevertheless, some steps of PASA entrust in the software engineer expertise to be applied and to identify alternatives for improvements.

Therefore, the main objective of this dissertation thesis is to devise a methodology for the assessment of software performance in the early phases of the software development life-cycle, and apply it to a complex architecture, as well as its automation. So, it is our objective that the resulting performance assessment should be obtained as a "by-product" of the software life-cycle, avoiding the software engineer to perform tasks for which a strong mathematical background is required.

From the viewpoint of the performance modeling and analysis, we base our methodology on the use of UML (OMG, 2011b), MARTE profile (OMG, 2011a) and the Petri nets formalism, concretely Generalized Stochastic Petri nets (GSPN) developed by Ajmone Marsan et al. (1995). From the performance assessment perspective, we use the concepts of response time and resource utilization studied by Jain (1991), performance patterns proposed by Smith and Williams (2002b) and derived from design patterns of Gamma et al. (1995) and performance antipatterns defined by Smith and Williams (2000). We inspire our work in the PASA method and PUMA approach. Chapter 2 introduces these concepts.

In order to accomplish this objective, we analyse diverse software architectures using SPE techniques and interpreted the obtained performance results, as well we have studied the impact of some technological decisions. We put the focus on three different kind of software architecture: a web service-based application, a mobile agents platform, and a remote console framework. We analyse these samples applying PUMA approach with GSPNs as formal method. As outcome of this research, we gain insight in performance issues, how to interpret the results of performance analysis and the first steps for improving initial architecture designs. Chapter 3 describes this work, which paves the development of the methodology.

Once we achieved a deep insight in the performance analysis of software architectures and a certain knowledge on closing the "assessment loop" by interpreting the performance results, we apply our methodology to a real-complex case study. This industrial case study is an interoperable architecture to automatically adapt user interfaces (device controllers or web services) according to the capabilities or preferences of people with special needs. We assess this software architecture in order to obtain its optimal configuration. Moreover, the theoretical performance results are validated against the experimental ones obtained in the user testing phase. In addition, other hypothetical situations are studied. Chapter 5 describes the main components of the aforementioned software architecture and presents the application of the performance assessment methodology proposed in this dissertation thesis.

Moreover, we develop a tool for the automation of the performance evaluation of software systems in the first stages of the development process. This tool implements the algorithms given in (Bernardi and Merseguer, 2007; Distefano et al., 2011) and follows the architecture proposed in the UML-SPT (OMG, 2005). It permits to expedite the performance analysis of software systems. Chapter 6 details the most of the features of this tool.

1.1 Outline

This thesis comprises seven chapters including this one. The balance is as follows.

Chapter 2 reviews the main research fields considered in this work and introduces preliminary concepts needed to follow the rest of the dissertation, such as Software Architecture, UML models, basic principles for SPE and Petri net formalism.

In Chapter 3, we address some sample applications to explore performance analysis. We analyse a Web-based application, a mobile agent tracking approach and a gateway oriented architecture for devices controllers.

Chapter 4 describes a scenario-based methodology for the performance assessment of software architectures using Software Performance Engineering principles. This methodology aims to assess performance as a “by-product” of the software development life-cycle.

In Chapter 5, we apply our methodology to an industrial case study. An interoperable architecture to automatically adapt interfaces for people with disabilities.

Chapter 6 introduces ArgoSPE, we have developed to automatize some aspects of the methodology. For this purpose, ArgoSPE translates UML diagrams annotated with performance properties into Generalized Stochastic Petri Nets. ArgoSPE is implemented according to the architecture proposed in the UML-SPT (OMG, 2005) profile and it is plugged into the open source ArgoUML CASE tool.

Finally, Chapter 7 concludes this dissertation thesis. It gives a summary of the achieved results and the main contributions. In addition, it establishes the open issues and the proposed future work.

Chapter 2

Preliminary Concepts

This chapter relates the work of this thesis to relevant research fields. It is subdivided into a number of sections. First, software architectures are described including architectural styles in Section 2.1. This is followed by the explanation of UML in Section 2.2 that allows to model software architecture. Then, software performance engineering in Section 2.3 is contextualized to be aware with its principles and objectives and to know the techniques and models used in this area. Finally, a brief introduction to Petri nets in Section 2.4 is presented. In each section, there is also an explanation of how the thesis is related to each area. Finally, some conclusions concerning this chapter are given in Section 2.5.

2.1 Software Architectures

The Software Architecture (SA) of a program or computer system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them (Bass et al., 2005).

A software architecture is an abstraction of the run-time elements of a software system during some phase of its operation. A system may be composed of many levels of abstraction and many phases of operation, each with its own software architecture (Shaw, 1990; Shaw and Garlan, 1996).

Hence, at the heart of software architecture is the principle of abstraction: hiding some of the details of a system through encapsulation in order to better identify and sustain its properties (Shaw, 1990). A complex system will contain many levels of abstraction, each with its own architecture. An architecture represents an abstraction of system behaviour at that level, such that architectural elements are delineated by the abstract interfaces they provide to other elements at that level (Bass et al., 2005).

Other definition can be found in the 42010:2011 standard, which defines a architecture as *"the fundamental organization of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its designs and evolution"* (ISO/IEC/IEEE, 2011). This standard describes an architecture using Unified Modeling Language (UML) class diagrams to represent these elements.

Therefore, a software architecture models the structure and behaviour of a system. Focussing on these aspects, quality attributes such as usability, performance, reliability, and security indicates the success of the design and the overall quality of the software application. Moreover, some of these properties are determined by the structure or architectural style.

In this work, we analyse and assess software architectures in the design phase with respect to software performance using Software Performance Engineering (SPE) techniques. Thus, we describe software architectures expressed in terms of UML diagrams.

2.1.1 Architectural Style

Software architecture of a large system can be guided by using architectural styles and design patterns (Bass et al., 2005; Monroe et al., 1997). It describes the organization of a family of systems that share common features.

Architectural styles are closely related to design patterns in two ways. First, architectural styles can be viewed as kinds of patterns (Shaw and Garlan, 1996) or perhaps more accurately as pattern languages (Kerth, 1995). There are many recognized architectural patterns and styles (Garlan and Shaw, 1993):

- **Data-centered architectures.** They aim to achieve the quality of integrability data. Basically, a centralized data store communicates with a number of clients, which use data manipulation protocols to work with the data. Some of the most well-known example are a database architecture or web architecture.

- **Data-flow architectures.** They aim to achieve the quality of reuse and modifiability. This style is characterized by viewing the system as a series of transformations on successive pieces of input data. Some examples are networking architectures.

- **Abstraction layer architectures.** The structure of the system is organized into set of layers. These architectures focus on a hierarchical distribution of roles and responsibilities. The role denotes the type and the mode of interaction with other layers and the responsibility indicates the functionality. Operating Systems are typically organized into layers, as well as network protocol stack, such as TCP/IP.

- **N-tier architectures.** The functionality is divided into different segments, but each segment is physically located on separated platforms. Each layer interacts with only the layer directly below, and has specific function that it is responsible for. Client-Server systems are defined with two-tier architectures. Most of the web applications are based on three-tier architectures, which typically comprise a presentation tier, a business or data access tier, and a data tier.

- **Notification architectures.** The information and activity is propagated by a notification mechanism. Thus, when an event occurs the interested component is notified.

- **Remote invocation and service architectures.** These architectures involve distributed processing components. Typically, a client component invokes a method (function) on a remote component. Service-oriented architectures introduce a special component where services are registered. Any component interested in a services asks that component for the address of that service.

- **Heterogeneous architectures.** Since no real system follows strictly only a single style, architectures might be conceptually, structurally, executional heterogeneous. For instance, a Web-based search engine might be conceptually data-centric, layered and three tier. From the structure viewpoint might follow layered and notification style. And its execution might be distributed, service-oriented and notification.

Next, we briefly describe Service-Oriented Architectures and Mobile Agent-Based System, since our work is mainly focused on them.

Service-Oriented Architecture

Service-Oriented Architecture (SOA) is a technology architectural model for service oriented solutions with distinct characteristics in support of realizing service orientation and the strategic goals associated with service oriented computing (Erl, 2007). Hence, SOA is a paradigm for developing and deploying business applications as a set of reusable services. SOA is not based on a specific technology or programming languages.

Therefore, SOA is an architectural style whose goal is to achieve loose coupling among interacting software service, some of them acting as providers and others, as consumers (Erl, 2007). A service is a unit of work done by a service provider to achieve desired end results for a service consumer in SOA, services are the mechanism by which needs and capabilities are brought together. When a service is deployed on the net, it is named Web Service (WS).

Web Services are universally accessible software components deployed on the Web, thus in a Web service architecture clients and services are loosely coupled and geographically distributed. One important characteristic is the heterogeneous and architecture-neutral of the computing platform. Therefore, one of the key ideas is that a Web service's implementation and deployment platform are not relevant to the application that is invoking the service (Alonso et al., 2004).

A refined definition is supplied by the World Wide Web Consortium (W3C) in (W3C, 2004): *A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typ-*

ically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

A Web service is a collection of protocols and standards used for exchanging data between applications. Software applications written in various programming languages and running on various platforms can use Web Services to exchange data over computer networks like the Internet, in a manner similar to inter-process communication on a single computer. This interoperability is due to the use of open standards such as the following and others:

- **Simple Object Access Protocol (SOAP)** (W3C, 2007) is a W3C specification describing a way to use XML and HTTP to create information delivery and remote procedure mechanisms. It is a lightweight protocol for the exchange of information decentralized, distributed environment; therefore, it is a specification that defines a uniform way of passing XML-encoded data. Examples of transport protocols that may be used are HTTP, SMTP, FTP or POP3. Currently, the most used is HTTP over port 80, originally reserved by Web browsers.
- **Web Service Description Language (WSDL)** (W3C, 2001) is an XML format for describing network services as a set of endpoint operating on messages containing either document-oriented or procedure-oriented information. WSDL is used to describe what services does, where they resides and how to invoke them.
- **Universal Description Discovery and Integration (UDDI)** (OASIS, 2005) specifications define a way to publish and discover information about Web Services. It consists of several related documents and an XML schema that defines a SOAP-based programming protocol for registering and discovering Web Services. UDDI is a framework for Web Services integration. From a performance perspective, the discovery process involves the time to look up the service in the Web Services directory using UDDI. It is possible to assume a more simplistic approach, where the client has prior knowledge of the service addressed.

In our work, we focus on the performance issues of industrial cases based on UCH and SOA with web services, cfr. Chapters 3 and 5.

Mobile Agent Architecture

In the traditional *client/server* architecture, a server at a certain computer offers a set of services to interested parties. Then, three steps take place:

1. a client located at another computer requests the execution of a service by interacting with the server,
2. the server performs the requested service,
3. the server returns the result to the client.

As opposed to this classical approach, a *mobile agent* (Milojičić et al., 1999) is a software component that can move autonomously among computers, and so it can decide itself when and where to move in order to perform its tasks.

Thanks to their mobility, mobile agents offer a range of unique advantages, such as autonomy, flexibility, and effective usage of network bandwidth (Lange and Oshima, 1999). For example, in a distributed information system a mobile agent can travel where the data are stored and process them locally, avoiding the need to communicate all the data over the network. Furthermore, in certain contexts they also exhibit a *good performance* compared with the traditional client/server approach (Spyrou et al., 2004; Mena et al., 2002).

We analyse SPRINGS, a mobile agent-based architecture, from performance perspective in Chapter 3.

2.1.2 Architecture Assessment

In the 24765:2010 Standard for Vocabulary of Systems and software engineering (ISO/IEC/IEEE, 2010), *software quality* is defined as the degree to which a system, a component, or a process meets specified requirements.

Architecture assessment is an activity of the architecting process that is targeted to evaluate the degree of fulfillment of quality, or non-functional requirements (Del Rosso, 2006).

Although there is a lack of consensus about the meaning and of non functional requirements or properties (NFP), as pointed out some authors (Glinz, 2007; Chung and Prado Leite, 2009), along this work, we follow the simple, but in our opinion very concise, definition proposed in (Espinoza et al., 2005): Functional properties describe what a system model does, and non-functional properties how it does it. Hence, non-functional requirements mostly define the overall attributes of the "resulting" system. The 29148:2011 Standard for Requirements engineering of Systems and software engineering (ISO/IEC/IEEE, 2011) enumerates a lists of 13 non-functional properties, among them: safety, security, usability, reliability and performance requirements.

The 25010:2011 Standard for System and software quality models of Systems and software engineering (ISO/IEC, 2011b) denotes *performance* as a quality attribute that meets the functionality of the system with timeless and correctness. It defines also performance as efficiency that requires two main factors: time behaviour and resource efficiency. These factors usually address common performance metrics such as response time, throughput and utilization.

Software performance assessment aims to evaluate and refine software architectures with respect of performance objectives as early as possible in the system development life-cycle. Thus, performance objectives include performance metrics with specific threshold.

In this dissertation, we propose a scenario-based software architecture assessment that aims to meet performance objectives and is carried out using principles and techniques of the Software Performance Engineering (SPE) field, which are described in the following sections.

2.2 The Unified Modeling Language

The UML, defined by the Object Management Group (OMG, 2011b), is a semi formal general-purpose visual modeling language that is designed to specify, visualize, construct and document the artifacts of software systems (Rumbaugh et al., 1999; Fowler, 2000). UML offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions, as well as concrete things such as programming language statements, database schemas, and reusable software components (Rumbaugh et al., 1999). UML has been formally published as the standard 19505-1:2012 by the International Standard Organization (ISO/IEC, 2012).

2.2.1 UML Diagrams

UML defines thirteen types of diagrams, divided into three categories: Six diagram types represent static application structure; three represent general types of behaviour; and four represent different aspects of interactions.

- **Structural Diagrams.** Structure (or static) diagrams are intended to model the static structure (logical and architectural) of the objects in a system. That is, they depict those elements in a specification that are irrespective of time. They include the Class Diagram, Object Diagram, Component Diagram, Composite Structure Diagram, Package Diagram, and Deployment Diagram.
- **Behaviour Diagrams** Behaviour diagrams show the dynamic behaviour of the objects in a system, including their methods, activities, and state histories. The dynamic behaviour of a system can be described as a series of changes to the system over time. They include the Use Case Diagram (used by some methodologies during requirements gathering), Activity Diagram, and State Machine Diagram.
- **Interaction Diagrams** Interaction diagrams are all derived from the more general Behaviour Diagram. They include the Sequence Diagram, Communication Diagram, Timing Diagram, and Interaction Overview Diagram.

In the following paragraphs, we outline the diagrams used for our work. For a more extended description, see the specification (OMG, 2011b).

UML Deployment Diagram

A UML Deployment Diagram (DD) depicts a static view of the execution architecture of a system that represents the allocation (deployment) of software artifacts to deployment nodes. It identifies the system software components as well as the hardware nodes in which the former are deployed and their relationships. Nodes (drawn as cubes) represent either hardware devices or software execution environments. Software components (represented as a rectangle and associated to a node through an arrow labelled `deploy`) represent the software that is installed. The relationships (drawn as lines) represent the middleware or the network used to connect the machines to one another.

Use Case Diagrams

A UML Use Case Diagram (UC) is used to model user (called *actor*) or system interactions in a horizontal way. They defines behaviour, requirements and constraints in the form of scripts or scenarios. UCs do not only represent details of individual features of a system, but they show all of its available functionality. A UC can extend another UC when the former is a special case behaviour of the latter. An actor does not necessarily represent a specific physical entity but merely a particular facet or role. An actor is represented by a “stick man” with its name.

UML Activity Diagrams

A UML Activity Diagram (AD) emphasizes the sequence of activities and conditions for coordinating lower-level behaviours of an object. Therefore, it shows the work flow from a start point to the finish point detailing the many decision paths that exist in the progression of events contained in the activity.

State Machine Diagrams

A UML State Machine Diagram (SM) depicts the various states that an object may be in and the transitions between those states. A state represents a stage in the behaviour pattern of an object. An initial state is the one that an object is in when it is first created, whereas a final state is one in which no transitions lead out of.

UML Sequence Diagrams

A UML Sequence Diagram (SD) is used primarily to show the interactions between objects in the sequential order that those interactions occur. It is used to model usage scenarios with respect to a timeline.

Interaction Overview Diagrams

A UML Interaction Overview Diagram (IOD) constitutes a high-level structuring mechanism that is used to compose scenarios through sequence, iteration, concurrency or choice. IODs are a special and restricted kind of ADs where the activity nodes are interactions or interaction uses and the activity edges denote control flow only.

2.2.2 SPT/MARTE Profiles

UML is enriched with *Profiles*, i.e., tailored subsets of the UML metamodel for specific purposes, such as safety (de Miguel et al., 2008), dependability (Bernardi et al., 2012) or fault-tolerant systems (OMG, 2008). The field of real-time embedded software systems is one such domain for which extensions to UML are required to provide more precise expression of domain specific phenomena (e.g., mutual exclusion mechanisms, concurrency, deadlines, and the like). The OMG had already issued a UML profile for this purpose, called the *UML profile for Schedulability, Performance and Time* (SPT), (OMG, 2005), Since SPT is based on UML 1.4, when UML 2.0 was adopted, it became necessary to upgrade the SPT profile to new one, named MARTE (*Modeling and Analysis of Real-Time and Embedded systems*) (OMG, 2011a). MARTE is also compliant with the *UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms* (QoS & FT) (OMG, 2008), developed by de Miguel (2003), in order to define Quality of Service (QoS) properties and requirements. The content of this section is taken from (OMG, 2005, 2011a).

MARTE builds on previous UML profiles and consists of a set of sub-profiles dedicated for different aspects, globally divided into foundations, design, analysis and annexes. The foundation part defines general concepts such as non-functional properties and time. The modeling part contains concepts that are useful for modeling, e.g. the component model and high level application modeling concepts dealing for instance with time properties of service invocations. The modeling part also contains for instance the possibility to characterize the properties of hardware resource like for example bandwidth/jitter of busses. In the context of components, it is a useful to annotate target platforms (deployment) with these properties in order to enable timing analysis. The annex of MARTE standardizes common non-functional characteristics, for instance durations, frequencies or arrival patterns.

Since, MARTE introduces the notion of time as annotations in the UML diagrams, it is the most comprehensive proposal for performance assessment using UML.

Key features of MARTE are the Non Functional Properties (NFP) framework and the Value Specification Language (VSL). NFP framework is used to define data-types characterized by several properties: measurements by means of magnitude and unit (e.g. energy, data size and duration); and qualifiers, such as value source, statistical measure and value precision. VSL is an expression language which VSL aims at the specification of NFPs, say performance, dependability and security. VSL specifies mathematical expressions (such as arithmetic and logical), time expressions (delays, periods, triggers conditions and so on) and variables, i.e., placeholders for unknown parameters.

Since the present work spanned a long period of time, we use in an interchangeably way a subset of SPT and MARTE profiles to address performance aspects in software designs. In addition, we partly automate the translation process from UML diagrams to GSPNs using ArgoSPE, an ArgoUML plugin, see a detailed description in Chapter 6. Performance annotations supported by ArgoSPE are not in MARTE, but in SPT profile format.

2.3 Software Performance Engineering

Architectural decisions are among the earliest made in a software project, as Williams and Smith (2002) point out. Early performance assessment for system architecture is highly desirable to prevent underperformance during system deployment. Nevertheless, performance evaluation of software systems has been traditionally accomplished after deployment. This is the well-known *fix-it later approach* and it has well-known problems (Smith, 1990). For example, the cost of re-architecting, re-implementing and re-deploying the system when performance goals are not fulfilled. Also the over-budget for being out of schedule as Woodside et al. (2007) described.

Software Performance Engineering (SPE) was introduced by Smith in 1981 as *a systematic, quantitative approach to constructing software systems that meet performance objectives.*

Smith (1990) refined the latter SPE definition as a research field that tries to use quantitative methods and performance models in order to assess the performance effects of different design and implementation alternatives during the development of a system. SPE promotes the integration of performance analysis into the software development process from its earliest life-cycle stages, in order to assure that the system will meet its performance objectives.

Therefore, SPE represents the entire collection of software engineering activities and related analyses used throughout the software development cycle, which are directed to meeting performance requirements (Woodside et al., 2007). As Smith (1990) pointed out, SPE augments others software engineering methodologies; it does not replace them.

2.3.1 Performance Scenarios and Objectives

Performance scenarios are those Use Cases of a software system that are executed frequently or are critical to the user's perception of performance (Williams and Smith, 2002). Each performance scenario corresponds to a workload.

A *workload* is the collection of request made by the users of the system. The workload intensity is a measure of the number of request made by a workload in a given time interval Smith and Williams (2000), i.e., it specifies the level of usage of a scenario. Workloads can be classified in closed, open or partly open.

Performance objectives specify quantitative criteria for evaluating the performance characteristics of performance scenarios in the system (Smith and Williams, 2002b). Thus, performance objectives not only include performance metrics with specific threshold; it is typically a set of numbers describing a combination of the input-processing-output for a particular situation in a performance scenario.

Performance objectives can be expressed in several different ways, including response time, throughput or constrains on resource usage (Smith, 1990). Jain (1991) defines these performance indices as follows:

- *Response time* is the time interval between a user request of a service and the response time of the system

- *Throughput* of a system is the number of requests that can be processed in some specified time interval. It is defined as a rate and is measured in request per time.

- *Utilization* is the ratio of busy time of a resource and the total elapsed time of measurement period.

Taking these quantitative measures, two important dimensions to software performance timeless are introduced by Smith (1990):

- *Responsiveness* is the ability of a system to meet its objectives for response time or throughput.

- *Scalability* is the ability of a system to continue to meet its response time or throughput objectives as the demand for the software functions increases.

This dissertation proposes a scenario-based performance assessment methodology for software architectures to meet performance objectives.

2.3.2 Performance Models

Performance models are formal models that help to obtain measures of interest (e.g., system response time) by analysis or simulation. They describe the system behaviour, expressed as scenarios which are realizations of Use Cases, and have the special capability of predicting some of its properties before it is built.

Performance models can be created from *scenarios*, which denotes a realizations of Use Cases that are frequently executed; or from *objects and components*, which interact between them and modify the complete system behaviour.

There are different kinds of performance formalisms widely accepted in SPE. In the following paragraphs they are briefly introduced.

Queueing Network Queueing Network (QN) was introduced by Lazowska et al. (1984) and it models a system as a network of queues. A network of queues consists of a set of queues, named stations. Each station consist of a queue and one or more resources, called servers (either single-server, multiple-server, or infinitive-server). Entities called customers or jobs generated by an external arrival process join the queue to receive service at one of the servers.

Extended Queueing Network Extended Queueing Network (EQN) (Cortellessa and Mirandola, 2002) inherits all components of queuing networks, i.e., sources where jobs enter the network, sinks where jobs leave the network, active queues consisting of a number of servers in which jobs are served and waiting areas where jobs wait for service, and finally interconnections between those network elements. Additionally, passive queues are introduced. A passive queue consists of one token pool and a number of network nodes assigned to this token pool. Passive queues allow for simulation of simultaneous resource possession and resource consumption.

Layered Queueing Network Layered Queueing Network (LQN) is an extension of QN model (Petriu and Woodside, 2002; Petriu and Shen, 2002). A LQN model is an acyclic graph, with nodes representing software entities and hardware devices (both known as tasks), and arcs denoting service requests. The software entities are drawn as rectangles with thick lines, and the hardware devices as ellipses. The nodes with outgoing but no incoming arcs play the role of clients, the intermediate nodes with both incoming and outgoing arcs are usually software servers and the leaf nodes are hardware servers (such as processors, I/O devices, communication network, etc.) A software or hardware server node can be either a single-server or a multi-server.

Stochastic Process Algebra In the process algebra approach systems are modelled as collections of entities, called agents, which execute atomic actions. These actions are the building blocks of the language and they are used to describe sequential behaviours which may run concurrently, and synchronisations or communications between them. Stochastic Process Algebra (SPA) in (Clark et al., 2007) extends process algebra by associating a random variable, representing duration, with every action (Hermanns et al., 2002). Performance Evaluation Process Algebra (PEPA) extends classical process algebras by introducing probabilistic branching and timing of transitions (Gilmore and Hillston, 1994).

Simulation Models Simulation models establish a correspondence between the system behaviours in a simulation model in order to obtain performance measures (Banks et al., 2009), i.e., the simulation model is used to derive a simulation program.

Stochastic Petri Net Stochastic Petri net (SPN), or similar token-based state model, is outlined in Section 2.4.3. We concretely use generalized SPN (GSPN) developed by Ajmone Marsan et al. (1995). Some of the reasons for using GSPNs were: their capacity to represent routing rates, competition for shared resources, stochastic duration of the host demands, parallel executions and forks and joins.

Performance Models from UML Diagrams

There exist SPE methodologies that translate performance-annotated UML models into the formalisms above mentioned. For example, the work in (Petriu and Woodside, 2002) to obtain queuing networks, the work in (Tribastone and Gilmore, 2008) to obtain process algebras, the work in (Balsamo and Marzolla, 2003a) to obtain simulation models or (Bernardi and Merseguer, 2007; Distefano et al., 2011) to obtain Petri nets.

Some of these methodologies have associated tools that automate the translation process. Concretely, we use ArgoSPE, detailed in Chapter 6, which converts UML diagrams into GSPNs. We translate each critical performance scenario and obtain the corresponding structure of a GSPN. Nevertheless, the translation, although automatic, require some additional effort as shown in Chapter 5.

2.3.3 SPE Process

SPE, as Schmietendorf and Scholz (2001) pointed out, reuses and enlarges concepts and methods from many others disciplines such as: Performance management, performance modeling, software engineering, capacity planning, performance tunning and software quality assurance.

In this work, we emphasize the use of the SPE in the early phases of the life-cycle effectively avoiding the *"fix-it later"* approach, as well as in (Smith, 1990), that proposes evaluating performance of software systems in the early stages of the development process. Thus, if performance problems are detected, it will be easier and less expensive to take the appropriate design decisions to solve them. Although our proposal focuses only in these uses of the SPE, in our opinion the use of good SPE practices must be extended along the complete development process, then facilitating to meet performance objectives in each stage. In this way the best choices of software architecture, design and implementation can be considered.

The SPE process that appears in Figure 2.1 was formerly proposed by Smith (1990) and still remains as reference for a very general proposal to establish the basic steps that a methodology based on SPE principles should consider.

Firstly, it must be defined which goals or quantitative values are of interest, obviously it changes from one stage of the software life cycle to other and also among different kind of systems. Business systems define performance objectives in terms of responsiveness as seen by the system users while reactive systems take into account event responses or throughput. The *concept* for the life cycle product also depends on the stage of the life cycle and it refers to the software architecture, the design, the algorithms, the code and so on. Responsive architectures and designs are conceived by applying SPE principles, patterns and antipatterns. Subsequently, data *gathering* is accomplished by defining the proper scenarios interpreting how the system will be typically used and its possible deviations (defining upper and lower bounds when uncertainties). The construction and evaluation of the performance model associated with the system is one of the fundamentals in SPE. Later in this section we explore common types of performance models proposed in the literature.

SPE relies on models to forecast the performance of the proposed software. Nevertheless, due to the fact of the multiplicity of performance models, the interoperability among them is sometimes difficult.

Performance by Unified Model Analysis

Performance by Unified Model Analysis (PUMA) approach tries to bridge the gap between design models and performance models in terms of performance attributes (Woodside et al., 2005; Petriu et al., 2012; Woodside et al., 2013). It is a methodology for the performance evaluation of software systems. PUMA was designed as a framework to obtain different kinds of performance models (queuing networks, layered queuing networks, Petri nets, among others), as targets, from different kind of design models (first and foremost UML diagrams), as sources.

PUMA uses a common intermediate performance model, called *Core Scenario*

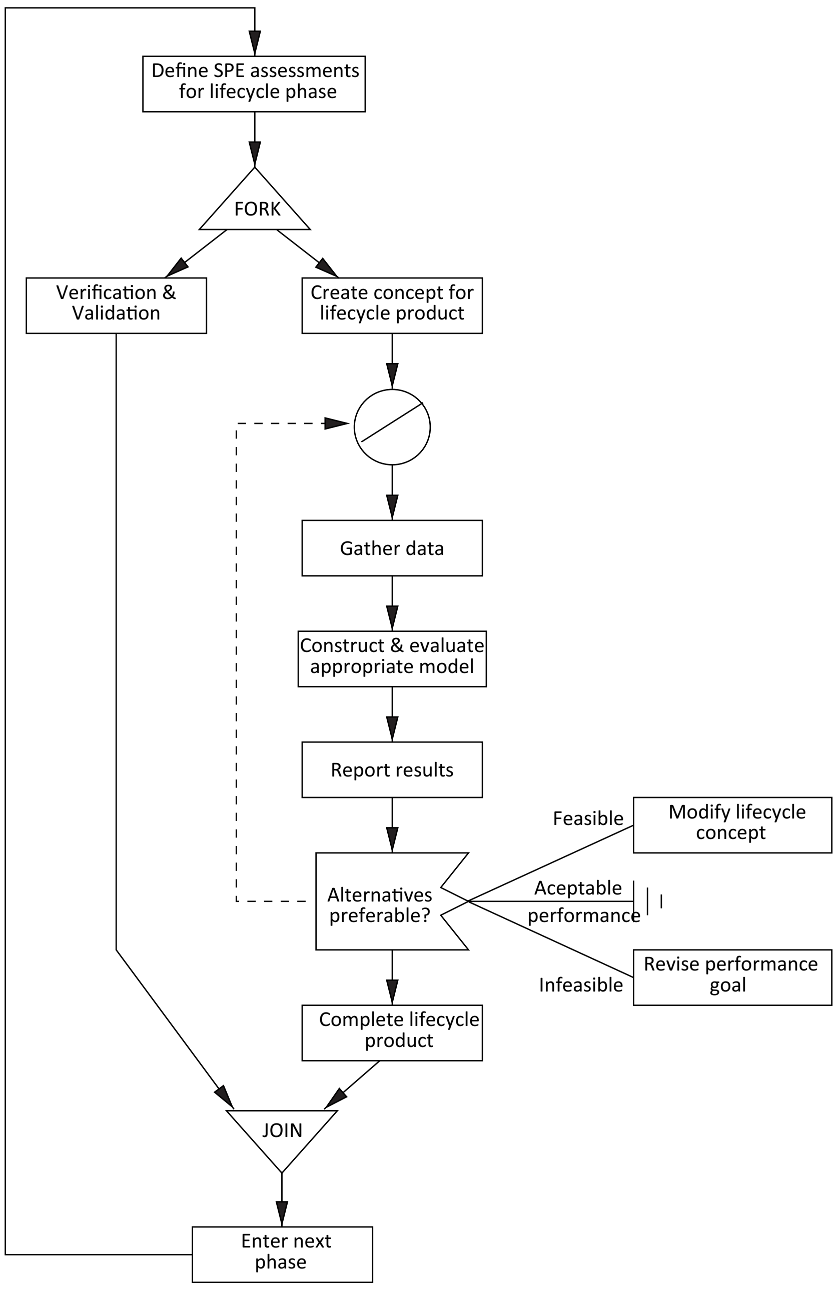

Define SPE assessments for lifecycle phase
FORK
Verification & Validation
Create concept for lifecycle product
Gather data
Construct & evaluate appropriate model
Report results
Alternatives preferable?
Feasible
Modify lifecycle concept
Aceptable performance
Infeasible
Revise performance goal
Complete lifecycle product
JOIN
Enter next phase

Figure 2.1: Software Performance Engineering Process taken from Smith (1990).

Model (CSM), as a bridge among sources and targets, then smartly solving the problem of translating N sources into M targets (Petriu and Woodside, 2007). Figure 2.2 summarizes the PUMA methodology.

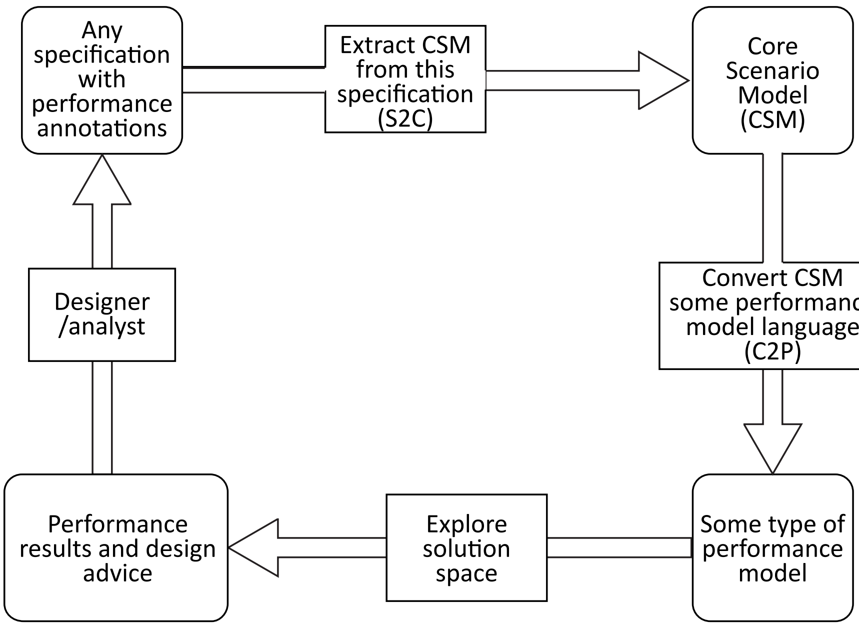

Any specification with performance annotations
Extract CSM from this specification (S2C)
Core Scenario Model (CSM)
Convert CSM some performance model language (C2P)
Some type of performance model
Explore solution space
Performance results and design advice
Designer /analyst

Figure 2.2: Architecture of the PUMA toolset taken from Woodside et al. (2005, 2013).

The CSM is based on the domain model of the UML-SPT profile. CSM is focused on describing performance scenarios. A scenario is a sequence of *Steps*, linked by *Connectors* that include sequence, branch/merge, fork/join and *Start* and an *End* points, where it begins and finishes. A step is a sequential piece of execution. A start connector is associated with a *Workload*, which defines arrivals and customers, and may be open or closed. There exist two kinds of *Resources*: *Active*, which execute steps, and *Passive*, which are acquired and released during scenarios by special *ResAcquire* and *ResRelease* steps. Steps are executed by (software) *Components* which are passive resources. A primitive step has a single host processor, which is connected through its component.

Regarding automation, it is worth noticing that PUMA offers tools to translate a UML-SPT annotated model into CSM models (Petriu and Woodside, 2007), and also a tool to translate from CSM models into GSPN (CSM2PN, 2013).

In the first stages of our work, we use PUMA as starting point to analyse how to evaluate performance of complex distributed architectures in order to develop an assessment methodology, and we start addressing some simple samples.

2.4 Petri Nets

Petri nets (PN) are a graphical tool for the formal description of the flow activities in complex systems. They are particularly suited to represent in a natural way logical interactions among parts or activities in a system. Typical situations that can be modelled by PN are synchronisation, sequentiality, concurrency and conflict.

The theory of Petri nets originated from doctoral thesis of C.A. Petri in 1962. From then, several authors have enriched the basic model with temporal interpretation for the quantitative analysis of performance and reliability of systems. The time variables associated to Timed Petri net (TPN) can be either deterministic or random variables, leading to the class of models called Stochastic Petri Nets (SPN). Generalized Stochastic Petri Nets (GSPN) extend TPN to include immediate transitions used to model conflicts with specific routing rates.

In this section, we briefly outline some basic concepts and definitions concerning Petri nets. For a more extended introduction, we invite to see (Peterson, 1981; Silva, 1985; Murata, 1989; Vernadat et al., 1993).

2.4.1 Nets and Net Systems

A Petri net model of a dynamic system consist of:

- a *net structure*: a bipartite directed graph, that represents the static part of the system.
- a *marking*: representing a distributed overall state on the structure.

According to the definitions and annotations proposed by Murata (1989), a Petri net (PN) is characterized as a four-tuple $mathcalN = (P, T, \mathbf{Pre}, \mathbf{Post})$, where:

$\mathbf{P} = \{p_1, p_2, ..., p_{np}\}$ is the set of *np places* and is drawn as circles in the graphical representation;

$\mathbf{T} = \{t_1, t_2, ..., t_{nt}\}$ is the set of *nt transitions* and is drawn as bars;

Pre is the $|P|\times|T|$ sized, natural valued, *pre-incidence* matrix or *input* function. It is represented by arcs directed from places to transitions;

Post is the $|P|\times|T|$ sized, natural valued, *post-incidence* matrix or *output* function. It is represented by arcs directed from transitions to places.

For instance, $\mathbf{Post}[p, t] = w$ means that there is an *arc* from t to p with *multiplicity* w. When all weights are one, the PN is *ordinary.* A PN is said to be *pure* if it has no self-loops. This kind of PN is characterized by the single *incidence matrix* of the net $\mathbf{C} = \mathbf{Post} - \mathbf{Pre}$. For pre- and postsets of a transition we use the conventional dot notation, that can be extended to set of nodes, ${}^\bullet t = \{p \in P : \mathbf{Pre}[p, t] \geq 1\}$.

The state of a PN is distributed and is defined by the number of tokens (drawn as dots) in each place. A net system of marking Petri net, $\mathcal{S}$, is a Petri net $\mathcal{N}$ with a *initial marking* $\mathbf{M_0} = \{m_1, m_2, ..., m_{np}\}$. $\mathbf{M_0}$ is a $|P|$ sized, natural valued, vector.

The generic entry m_i is the number of tokens in place p_i in marking $\mathbf{M_0}$. The number of tokens at a place represents the *local state* of that place.

The structure of a net is something static. The dynamic of a PN is obtained by moving the tokens in the places by means of the following execution rules, informally named *token game*:

- A transition is *enabled* in a given marking $\mathbf{M}$ if all its input places carry at least one token, i.e., $\mathbf{M}$ iff $\mathbf{M} \geq \mathbf{Pre}[P, t]$.
- An enabled transition fires by removing one token per arc from each input place and adding one token per arc to each output place. Its *firing*, denoted by $\mathbf{M} \xrightarrow{t} \mathbf{M'}$, yields a new marking $\mathbf{M'} = \mathbf{M} + \mathbf{C}[P, t]$.

Given an initial marking $\mathbf{M_0}$, the *reachability* set denoted as $RS(\mathcal{N}, \mathbf{M_0})$ is the set of all reachable markings that can be obtained by repeated application of the above rules. An *occurrence sequence* from $\mathbf{M}$ is a sequence of transitions $\sigma = t_1 \ldots t_k \ldots$ such that $\mathbf{M} \xrightarrow{t_1} \mathbf{M_1} \ldots \mathbf{M_{k-1}} \xrightarrow{t_k} \ldots$. Given σ such that $\mathbf{M} \xrightarrow{\sigma} \mathbf{M'}$, and denoting by σ the $|T|$ sized firing count vector of σ, then $\mathbf{M'} = \mathbf{M} + \mathbf{C} \cdot \sigma$ is known as the *state equation* of $\mathcal{N}$.

If $\mathcal{N}'$ is the subnet of $\mathcal{N}$, defined by $P' \subseteq P$ and $T' \subseteq T$, then $\mathbf{Pre'} = \mathbf{Pre}[P', T']$, $\mathbf{Post'} = \mathbf{Post}[P', T']$ and $\mathbf{M'_0} = \mathbf{M_0}[P']$. Subnets defined by a subset of places (transitions), with all their adjacent transitions (places), are called P- (T-) subnets.

2.4.2 Stochastic Petri Nets

PN models originally include no notion of time. This concept was intentionally avoided by Petri 1962, because of the effect that timing may have on the behaviour of PNs. The association of timing constraints with the activities represented in PN models or systems may prevent certain transitions from firing, thus destroying the important assumption that all possible behaviours of a real system are represented by the structure of the PN.

A variety of time mechanisms have been proposed in the literature. Influenced by the specific application fields, the distinguishing features of the time mechanisms are whether the duration of the events is modelled by deterministic variables or random variables, and whether the time is associated to the PN places (Sifakis, 1979), transitions (Ramchandani, 1974) or tokens (Merlin and Farber, 1976). In the case of performance evaluation, based on stochastic process analysis, the time associated with the PN transitions is an exponentially distributed random delay.

A Stochastic Petri net (SPN) is a PN with a temporal interpretation which permits to model timing constraints in a system (Molloy, 1982; Natkin, 1980). The activities of a process are modelled by means of firing rates with transitions. The temporal interpretation must include *conflict resolution policy*, associating a routing rates to each subset of transitions in conflict.

The firing of a transition is an atomic operation. Tokens are removed from its input places and deposited into its output places with one indivisible operation. A firing delay is associated with each transition. This specifies the amount of time that must elapse before transition can fire.

Formally, a SPN is a pair $(\mathcal{S}, w)$ where $\mathcal{S}$ is a PN system and $w : \mathbf{T} \rightarrow (0, \infty)$ is a function which associates to each transition a random variable with negative exponential probability density function (pdf) with w, its random firing time or firing delay.

A Stochastic Marked Graph (SMG) is a Stochastic Petri net whose underlying Petri net is a Marked Graph. A Stochastic Process Petri Net (SPPN) system is a Stochastic Petri net system whose underlying Petri net is a Process Petri net.

By definition, all the places of a SPPN are covered by p-semiflows, and therefore it is structurally bounded.

There exist different semantics for the firing of transitions, being infinite and finite server semantics the most frequently used. Given that infinite server semantics is more general (finite server semantics can be simulated by adding self-loop places), we will assume that the timed transitions work under infinite server semantics.

The average behaviour of the PN in the limit of time is the existence of a *steady state behaviour*. Ergodicity, introduced by Ross (1983), allows to speak about the average behaviour estimated on the long run of the system, but it is valid only for very strong assumptions on the probability distribution functions (PDF) defining the timing of the model (Campos et al., 1991).

The average marking vector, $\overline{m}$, in an ergodic Petri net system is defined as (Florin and Natkin, 1989):

$$\overline{m}(p) =_{AS} \lim_{\tau \rightarrow \infty} \frac{1}{\tau} \int_0^{\tau} \mathrm{m(p)_u}\, \mathrm{d}u$$

where $\overline{m}(p)_u$ is the marking of place p at time u and the notation $=_{AS}$ means *equal almost surely.*

Similarly, the steady state throughput, χ, in an ergodic Petri net is defined as (Florin and Natkin, 1989):

$$\chi(t) =_{AS} \lim_{\tau \rightarrow \infty} \frac{\sigma(t)_{\tau}}{\tau}$$

where $\chi(t)_{\tau}$ is the firing count of transition t at time τ.

Markov process that describes the time evolution(Ajmone Marsan et al., 1995) of these nets is ergodic(Campos et al., 1991), i.e., when the observation period tends to infinite, the estimated values of average marking and steady state throughput tend to a certain value, what implies the existence of the above limits.

2.4.3 Generalized Stochastic Petri Nets

Generalized Stochastic Petri Nets (GSPN) are a graphical and mathematical modeling tool for describing concurrent systems (Ajmone Marsan et al., 1995). They extends SPN which include immediate transitions in order to model conflicts with specific routing rates. They are very useful in practical modeling, without losing, except for some special cases, the possibility of performing quantitative and qualitative studies.

Formally, a GSPN is a tuple $\mathcal{G} = (\mathcal{N}, \mathbf{\Pi}, \mathbf{\Lambda}, \mathbf{r})$, where $\mathcal{N}$ is a PN system and the set of transitions T is partitioned in two subsets T_t and T_i of timed and immediate transitions, respectively. Timed transitions are depicted as thick white bars, immediate ones are depicted as thin black bars,

$\mathbf{\Pi}$ is a natural valued, $|\,T\,|$ sized, vector that specifies a priority level of each transition; timed transitions have zero priority, immediate transitions have priority greater than zero. A transition $t \in T$, enabled in marking $\mathbf{M}$, can fire if no transition $t' \in T : \mathbf{\Pi}[t'] > \mathbf{\Pi}[t]$ is enabled in $\mathbf{M}$.

Immediate transitions fire in zero time. Instead, the firing of a timed transition is a random variable, distributed according to a negative exponential probability distribution function PDF with rate parameter λ (i,e., mean $\frac{1}{\lambda}$). Then $\mathbf{\Lambda}$ is the non negative real valued, $|T_t|$ sized, vector associated to the transition firing rates (accordingly, the transition firing delay is the inverse of the corresponding firing rate). The positive real valued vector $\mathbf{r}$ is $|T_i|$ sized, and specifies the weights of the immediate transitions for probabilistic conflict resolution. A marking of a GSPN is called *vanishing* if at least one immediate transition is enabled in the marking and *tangible* otherwise.

It has been proved that exactly one Continuous-Time Markov Chain (CTMC) corresponds to a given GSPN under condition only a finite number of transitions can fire in a finite time with no-zero probability (Ajmone Marsan et al., 1984).

In this work, to obtain performance measures we translate UML diagrams into GSPNs by means of ArgoSPE, an ArgoUML plugin, crf. Chapter 6, which implements translation approach proposed by Bernardi et al. (2002); Merseguer (2003); López-Grao et al. (2004); Bernardi and Merseguer (2007).

2.4.4 Petri Nets Analysis

Performance models can be used to estimate some quantifiable *performance measures* in the first stages of the life-cycle to evaluate alternatives for some system parameters. With this purpose a performance model can be simulated or analysed, both approaches have been followed in the examples developed in this work.

Responsiveness and utilization performance measures can be calculated, either operationally or stochastically, see Campos (1998) for a survey. As an example, it can be calculated for places, the average steady-state marking; for transitions, the average steady-state enabling degree, utilization or the throughput.

Traditionally, techniques for the analysis (computation of performance measures or validation of logical properties) of Petri nets are classified in three complementary groups (Colom et al., 1998): enumeration, transformation, and structural analysis:

- *Enumeration* methods are based on the construction of the reachability graph (coverability graph in case of unbounded models), but they are often difficult to apply due to their computational complexity, the well-know *state explosion problem.*

- *Transformation* methods obtain a Petri net from the original one belonging to a subclass easier to analyse but preserving the properties under study, see (Berthelot, 1987).
- *Structural analysis* techniques are based on the net structure and its initial marking, they can be divided into two subgroups: *Linear programming* techniques, based on the state equation and *graph based* techniques, based on "ad hoc" reasoning, frequently derived from the firing rule.

A complementary classification of the techniques for the quantitative analysis of the SPNs based on the quality of the results obtained can be: exact, approximation and bounds.

- *Exact* techniques are mainly based on algorithms for the automatic construction of the infinitesimal generator of the isomorphic Continuous Time Markov Chain (CTMC). Refer to Colom et al. (1998) for numerical solutions of GSPN systems. In general, these techniques suffer the referred state explosion problem. In Donatelli and Franceschinis (1996) the model is decomposed for the exact computation of the steady state distribution of the original model. Others techniques based on *tensor algebra* to obtain exact solutions are proposed in Campos et al. (1999).
- *Approximation* techniques do not obtain the exact solution but an approximation. Some of them substitute the computation of the isomorphic CTMC by the solution of smaller components (Campos et al., 1994). In Pérez-Jiménez (2002) techniques based on *divide and conquer* strategies are presented for the approximated computation of the throughput.
- Finally, *bounds* are techniques that offer the further results from the reality. Nevertheless, they can be useful in the early phases of the software life-cycle.

Performance Analysis of GSPN Models

The analysis of a GSPN model requires the solution of a system of linear equations comprising as many equations as the number of reachable *tangible* markings (Ajmone Marsan et al., 1984). The steady state solution of the model is obtained by solving the system of linear equations:

$$\Pi Q = 0$$
$$\sum_{M \in RS} \Pi[M] = 1.$$

Q is the infinitesimal generator matrix, whose elements outside the main diagonal are the rates of the exponential distributions associated with the transitions from state to state, while the elements on the main diagonal make the sum of the elements of each row equal to zero.

Π is the equilibrium probability mass function (pmf) over the reachable markings; being $\Pi[M]$ the steady-state probability of a given marking M.

The transient solution of the model is instead obtained solving the set of differential equations:

$$\frac{d\Pi(t)}{dt} = Q\Pi(t)$$

where $\Pi(t)[M]$ is the probability of the system being in state M at time t.

Some of the most commonly computed aggregate results obtained from steady-state distributions over reachable markings are (Ajmone Marsan et al., 1995; Reisig and Rozenberg, 1998):

- The pmf of the number of tokens at steady-state in a place, say p, can be obtained by computing the individual probabilities in the pmf as probabilities of the event "place p contains k tokens".
- The average number of tokens in a place can be computed from the pmf of tokens in that place.
- The frequency of the firing of a transition (throughput) can be computed as the weighted sum of the transition firing rate.

In this dissertation thesis, performance measures are calculated or simulated from GSPNs obtained by ArgoSPE, an ArgoUML plugin, cfr. Chapter 6. ArgoSPE internally calls GreatSPN (Chiola et al., 1995) and/or TimeNET (Zimmermann et al., 2000) to analyse or simulate GSPNs. Both tools are software packages for the modeling, validation, and performance evaluation of distributed systems using Petri Nets. We compute all the measures (response time, utilization and scalability) under steady state assumption. Steady state means that the system reaches an equilibrium, so, measures obtained will continue in the future, which is a more general assumption than transient state. In a GSPN, steady state analysis can be carried out when the net is cyclical.

2.5 Conclusions

This chapter examines the background for this dissertation. It introduces and formalizes a set of terminology and annotations for software architecture concepts, software performance engineering, UML, including their SPT and MARTE profiles, and some concepts of Petri nets. These four research fields are the foundations of our work.

Chapter 3

Foundations of the Methodology

As a step previous to develop the performance assessment methodology described in Chapter 4, we started addressing some sample applications. The work presented in this chapter paved the development of the methodology, which is completely established in the industrial case study. Applying SPE techniques to different software technological environments permitted us to detect potential shortcomings and improvements in the methodology to assess software architectures.

We started following the SPE principles and applying the Performance by Unified Model Analysis (PUMA) approach (Woodside et al., 2005, 2013) with GSPNs as formal method to analyse performance issues in three case studies from different domains: a Web-based application, named CDSS, a mobile agent tracking approach called SPRINGS and a gateway oriented architecture for devices controllers, UCH.

CDSS service is recalled from literature (Catley et al., 2004) and we analysed it in Gómez-Martínez and Merseguer (2006b). Thus, we have studied middleware performance issues and proposed different technological alternatives and tunning for determining the optimal system configuration.

Our analysis of SPRINGS, a mobile tracking approach, was presented in Gómez-Martínez et al. (2007) and allowed us to validate the experimental results previously obtained by Ilarri et al. (2006), and to evaluate a mobile agents platform in a variety of hypothetical situations without the burden of real experimentation.

The UCH (Zimmermann and Vanderheiden, 2007) is a realization of the Universal Remote Console (URC) that acts as a gateway for communicating devices. We analysed three implementations of UCH from the performance point of view to achieve the optimal system configuration in Gómez-Martínez and Merseguer (2010).

This chapter is organized as follows. Section 3.1 evaluates performance aspects of the CDSS software system. Section 3.2 analyses SPRINGS tracking approach for mobile agents. Then, Section 3.3 studies the performance of the UCH interoperable architecture. Finally, some conclusions are given in Section 3.4.

3.1 CDSS: Web Service Application

A web service is a collection of protocols and standards used for exchanging of XML messages between applications (Alonso et al., 2004). Unlike other middleware technologies, web services allow to communicate heterogeneous environments deployed on the network, offering flexibility and interoperability. This interoperability is due to the use of open standards such as SOAP (W3C, 2007), Web Service Description Language (WSDL) (W3C, 2001) and Universal Description Discovery and Integration (UDDI) (OASIS, 2005). A more extended description of these standards can be found in Section 2.1.1.

Performance is one of the key aspects and probably the Achilles' heel of web services and in general of services offered over the Internet (Woodside and Menascé, 2006). In this case study, we try to overcome some aspects of this lack by accomplishing an in-depth study of different key aspects of web services performance at the middleware layer: the Simple Object Access Protocol (SOAP) implementations and the eXtensible Markup Language (XML) parsers.

Our study is based on an interesting performance case study of a web service developed, also under the SPE principles, in Catley et al. (2004). We rearchitect this case study following the PUMA approach to get a Generalized Stochastic Petri Nets (GSPN) (Ajmone Marsan et al., 1995). The GSPN, properly analysed with the TimeNET tool (Zimmermann et al., 2000), allows us to offer interesting results about performance middleware key aspects and to contrast them with the results obtained from pragmatic (non-formal) studies.

In this section, firstly we address key issues concerning performance of web services at middleware layer. Then, we recall the web service under study and we obtain the GSPN that models the target system. Such net will be useful to accomplish the study proposed in this section. Therefore, the impact of SOAP implementations and XML parsers is studied by means of that formal model. Finally, we revise the state of the art and places our proposal for study performance of web services in the current scene.

3.1.1 Performance Issues of Web Services

Web service technology has not been developed with performance as a goal. Performance issues affect several aspects: the XML protocols, such as discovering using UDDI (Menascé and Almeida, 2001), transporting using usually HTTP (Elfwing et al., 2002), the latency of SOAP implementations (Davis and Parashar, 2002) or the use of an XML parser (Head et al., 2005). Furthermore, web services can be provided with dynamic composition of web services, affecting performance in any way (Chandrasekaran et al., 2003; Menascé, 2004). The software infrastructure is other significant factor (Liu et al., 2004).

Although all these issues are relevant, we focus on those that being closer to the middleware layer can be parameterized in a UML design. Among them, SOAP implementation is one of the key factors that have influence on performance, as previous studies have shown (Davis and Parashar, 2002; Elfwing et al., 2002; Head et al., 2005). Therefore, it is important to determine which particular SOAP toolkit can meet the performance requirements of an application. These studies remark the following topics:

- **Serialization** is the process to convert an in-memory object into an XML stream. This includes to pack the XML message in the SOAP envelope and to build the message which will be sent by the corresponding transport protocol, mainly HTTP (Head et al., 2005).

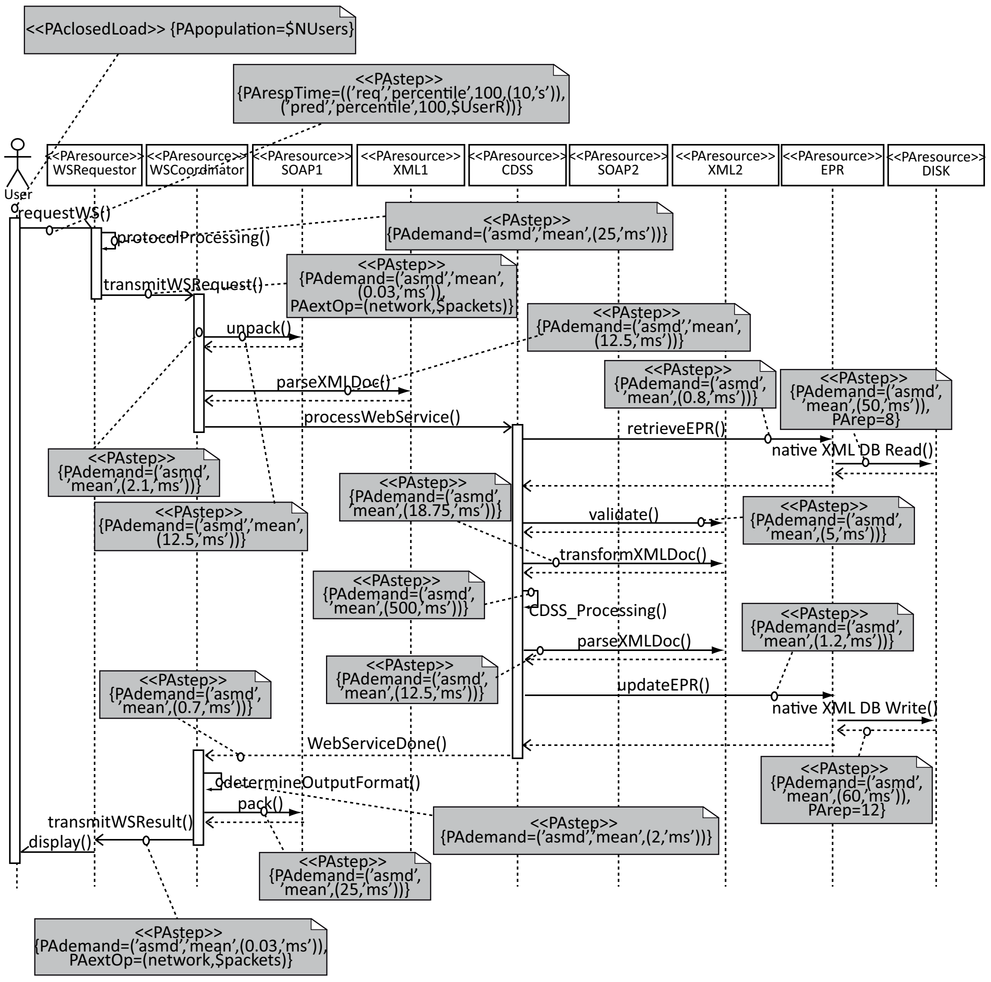
- **Deserialization** converts XML streams in wire-format objects in memory. In this process two phases must be emphasized: (1) unpacking the SOAP envelope and (2) parsing and interpreting the XML document. The most widely models used for parsing are *Document Object Model* (DOM) (W3C, 2009), *Simple API for XML* (SAX) (SAX, 2004) and *XML Pull Parser* (XPP) (Slominski, 2004). DOM parsers are suitable for small documents which must be validated and/or modified. SAX parsers are better for large documents. XPP is optimized when the XML elements are processed in succession and do not need to be visited again. The parser process has a great impact in the performance of SOAP implementation, as previous works have studied (Elfwing et al., 2002; Head et al., 2005). Note that XML native parsers and those embedded in SOAP have to be differentiated, since they exhibit different features and performance characteristics (Sosnoski, 2002; Slominski, 2004).

However, not only these processes affect the performance of a SOAP toolkit, others such as data structure support, optimizations to handle scientific data or algorithms implementation and protocols have influence too (Davis and Parashar, 2002; Head et al., 2005). These topics will not be addressed in this work, since they are out of scope of the case study which guides it. Other significant factors that may impact in performance are the service processing time, i.e., the *business logic*, and the XML file size (Catley et al., 2004).

The goal is to study the impact of the following aspects in web service performance: (**G1**) XML parsers and (**G2**) SOAP toolkits. Furthermore, other factors that may impact in performance will be studied, such as (**G3**) the sensibility of a web service with respect to the document file size exchanged and (**G4**) the service processing time. The implementations of XML parsers under consideration are: Xerces (Apache Software Foundation, 2010a), Xerces2 (Apache Software Foundation, 2010b), Crimson (Crimson, 2005) and XML Pull Parser (XPP) from (Slominski, 2004). The SOAP toolkits considered are AxisJava and .NET, since they are widely used. We have also included XSUL for its excellent performance for large documents (Slominski, 2004). In order to study the impact of the previous goals, we recall a performance case study taken from (Catley et al., 2004), first we summarize it and show its results. Then, we apply the PUMA approach to obtain a GSPN model from the UML system description.

3.1.2 CDSS Web Service

Catley et al. proposed in (Catley et al., 2004) *"an infrastructure to support artificial intelligence-based clinical decision systems (CDSSs). The system processes multidomain medical data in high-risk medical environments in order to reduce medical errors and alert detection systems"*. It integrates and accesses CDSSs and distributed databases from different medical domains in order to predict medical outcomes. These CDSSs are offered as web-services. The paper models a representative subset of this infrastructure, which invokes a CDSS as a web service and accesses the patient's Electronic Patient Record (EPR).



<<PAclosedLoad>> {PApopulation=$NUsers}
<<PAstep>> {PArespTime=(('req','percentile',100,(10,'s')), ('pred','percentile',100,$UserR))}
User
<<PAresource>> WSRequestor
<<PAresource>> WSCoordinator
<<PAresource>> SOAP1
<<PAresource>> XML1
<<PAresource>> CDSS
<<PAresource>> SOAP2
<<PAresource>> XML2
<<PAresource>> EPR
<<PAresource>> DISK
requestWS()
protocolProcessing()
<<PAstep>> {PAdemand=('asmd','mean',(25,'ms'))}
transmitWSRequest()
<<PAstep>> {PAdemand=('asmd','mean', (0.03,'ms')), PAextOp=(network,$packets)}
unpack()
<<PAstep>> {PAdemand=('asmd','mean', (12.5,'ms'))}
parseXMLDoc()
<<PAstep>> {PAdemand=('asmd', 'mean',(0.8,'ms'))}
<<PAstep>> {PAdemand=('asmd', 'mean',(50,'ms')), PArep=8}
processWebService()
retrieveEPR()
native XML DB Read()
<<PAstep>> {PAdemand=('asmd', 'mean',(2.1,'ms'))}
<<PAstep>> {PAdemand=('asmd', 'mean',(18.75,'ms'))}
<<PAstep>> {PAdemand=('asmd','mean', (12.5,'ms'))}
validate()
<<PAstep>> {PAdemand=('asmd', 'mean',(5,'ms'))}
transformXMLDoc()
<<PAstep>> {PAdemand=('asmd', 'mean',(500,'ms'))}
CDSS_Processing()
<<PAstep>> {PAdemand=('asmd', 'mean',(1.2,'ms'))}
parseXMLDoc()
<<PAstep>> {PAdemand=('asmd', 'mean',(12.5,'ms'))}
<<PAstep>> {PAdemand=('asmd', 'mean',(0.7,'ms'))}
updateEPR()
native XML DB Write()
WebServiceDone()
<<PAstep>> {PAdemand=('asmd', 'mean',(60,'ms')), PArep=12}
determineOutputFormat()
pack()
<<PAstep>> {PAdemand=('asmd','mean',(2,'ms'))}
transmitWSResult()
display()
<<PAstep>> {PAdemand=('asmd', 'mean',(25,'ms'))}
<<PAstep>> {PAdemand=('asmd','mean',(0.03,'ms')), PAextOp=(network,$packets)}

Figure 3.1: UML Sequence Diagram taken from (Catley et al., 2004) describing the CDSS system invocation process.

Figure 3.1 depicts the SD of such CDSS invocation process, that proposes an *initial system configuration* made of one instance per hardware and software resource.

The system parses XML documents using the Xerces parser through a DOM interface. The *required* response time for 50 users requesting the system should not exceed 8 seconds.

Catley et al. applied the SPE techniques developed in Petriu and Shen (2002) to assess the required metric. Then they modeled the system by means of deployment and UML Sequence Diagrams annotated according to the UML-SPT profile, cfr. Section 2.2.2, and translated them into an LQN model (Woodside et al., 1995).

This model was solved with the *initial configuration*, determining that the system can not meet the performance target, see Figure 3.2(a) where the response time for 50 users is 39.9 seconds. They identified system bottlenecks and proposed a *new configuration* that replicates processors and threads (10 `WSCoordinator`, 10 `CDSS`, 3 `AppCPU` and a variable number of `EPR`). Figure 3.2(b) depicts the results of multithreading the `EPR` task when the system is executed by 50 concurrent users. They determined that the target is achieved in this new configuration with 8 threads of `EPR`.

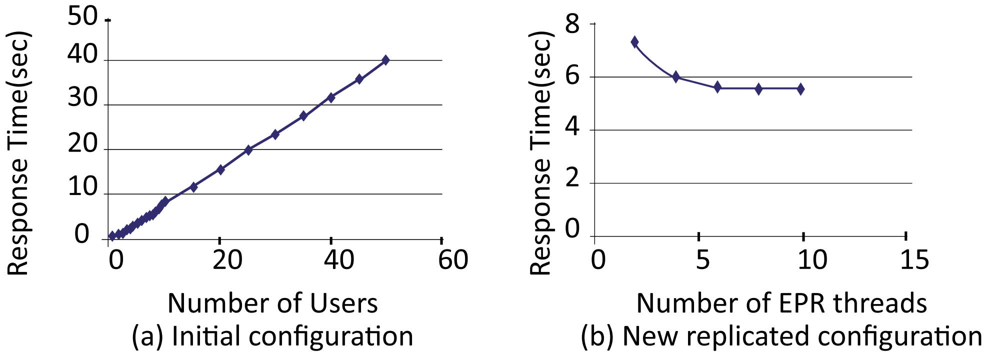


Figure 3.2: Performance results obtained by Catley et al. (2004).

3.1.3 Applying SPE to the CDSS

As mentioned in Section 2.3.3, PUMA was designed as a framework to obtain performance models from design models (Woodside et al., 2005, 2013). Therefore, we used PUMA to obtain a GSPN model from the SD in Figure 3.1, which models the CDSS. The GSPN model aims for validating the CDSS results obtained in (Catley et al., 2004) and for dealing with the performance goals previously given.

PUMA offers a translation process to get a Core Scenario Model (CSM) from a UML SD annotated with UML-SPT. Figures 3.3 and 3.4 depict the resulting CSM for our target SD. Observe that this CSM is made of two scenarios, the one corresponding to the CDSS invocation process in Figure 3.3 and the CDSS processing in Figure 3.4. The CDSS processing scenario comprises the messages from `processWebService()` to

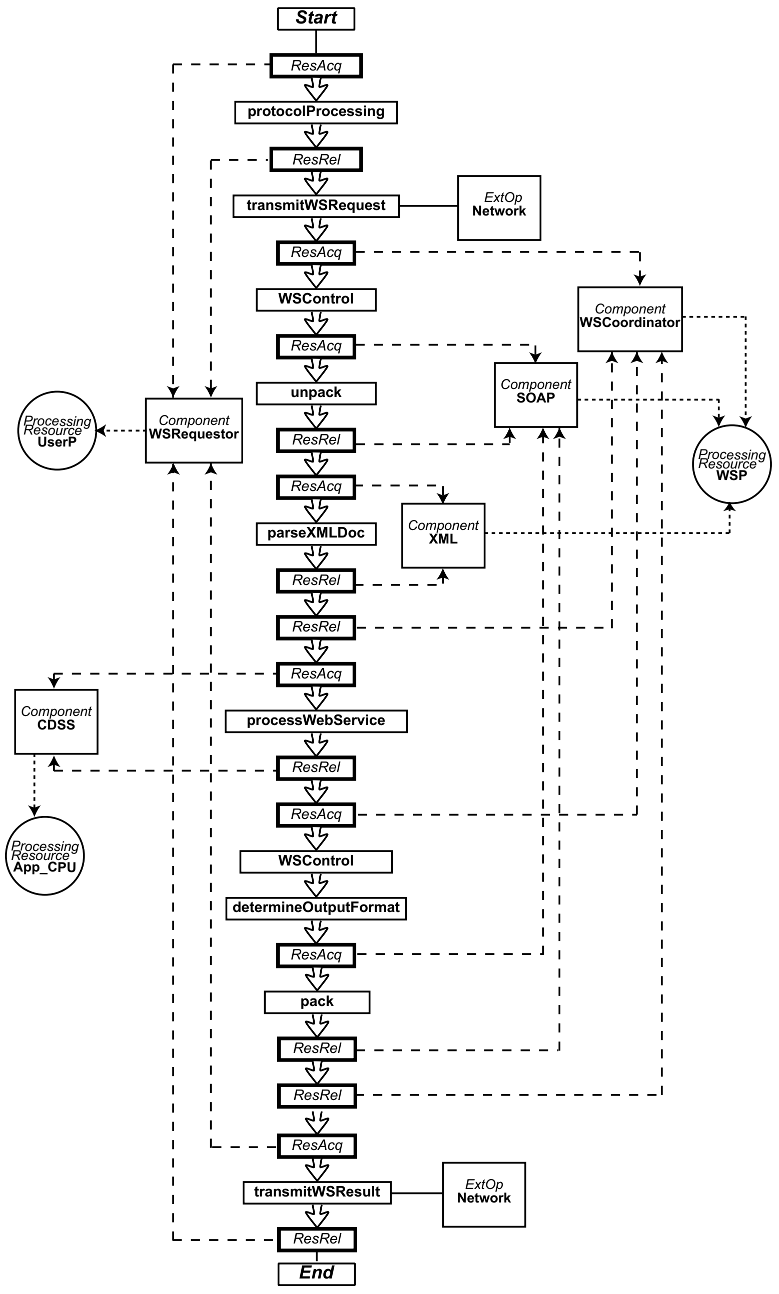


Figure 3.3: Core Scenario Model for the SD of Invocation Scenario.

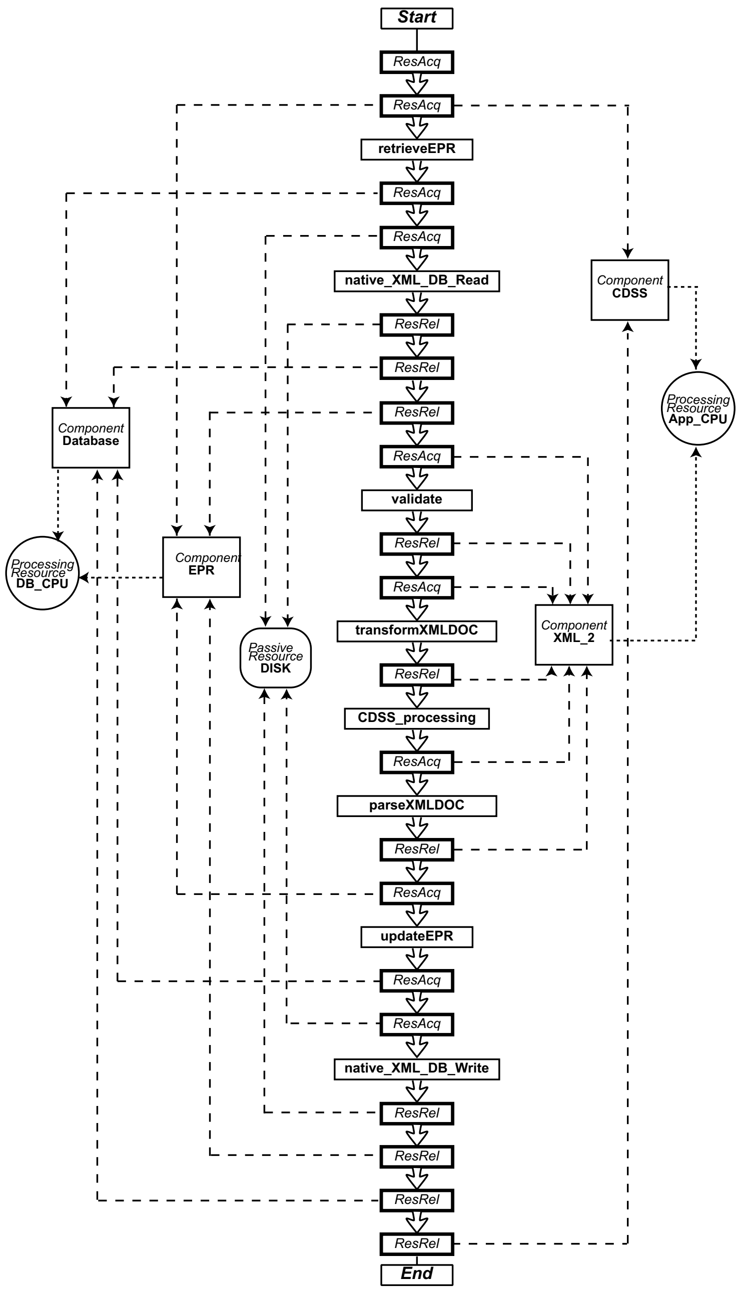


Figure 3.4: Core Scenario Model for the SD of Process Scenario.

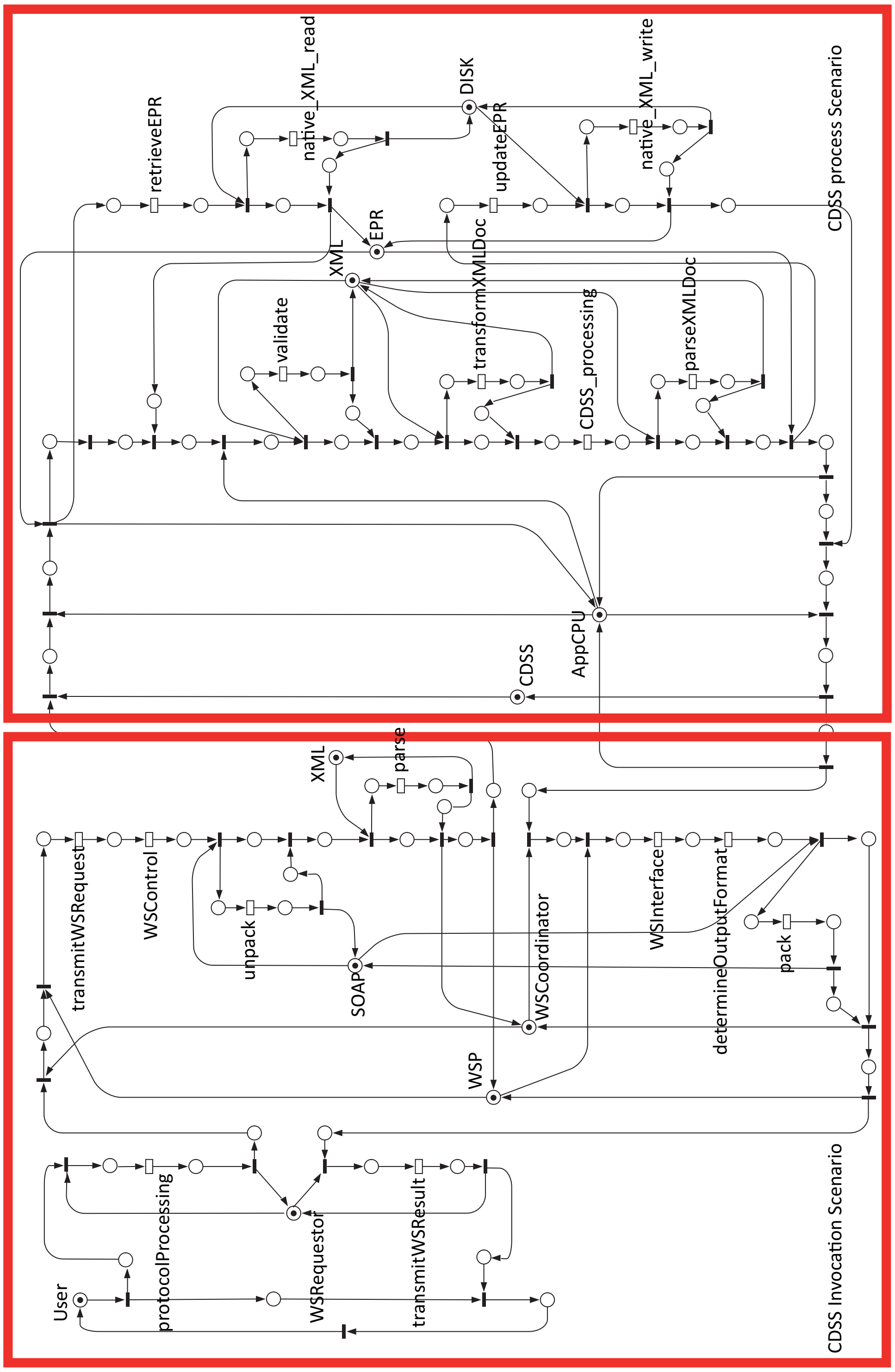


Figure 3.5: CDSS: GSPN representing the whole scenario

`WebServiceDone()`. The other messages of the SD correspond to the CDSS invocation process.

The CSMs in Figures 3.3 and 3.4 are translated into a GSPN, see Figure 3.5, by means of a extraction process developed in (Woodside et al., 2005, 2013). So, each class of the CSM corresponds with a GSPN pattern. For instance, a step is translated into a timed transition with an input place, where its delay is the demand attribute of the step. All of the GSPN patterns are composed until the GSPN representing the whole scenario is built.

Validation of Results

Performance metrics can be obtained using TimeNET, developed by Zimmermann et al. (2000) to solve this GSPN by means of simulation techniques. Figure 3.6(a) and Figure 3.6(b) present the same experiments as Figure 3.2(a) and Figure 3.2(b), respectively, i.e., the response times for the initial and the new replicated configuration. In Figures 3.6(b) and 3.2(b), the response time for 50 users is stated in 9.23 seconds and 5.4 seconds, respectively. The response times obtained with the GSPN are greater than those given by the LQN since PNs introduce synchronization in the model. However, both the order of magnitude and the tendency of the results are kept.

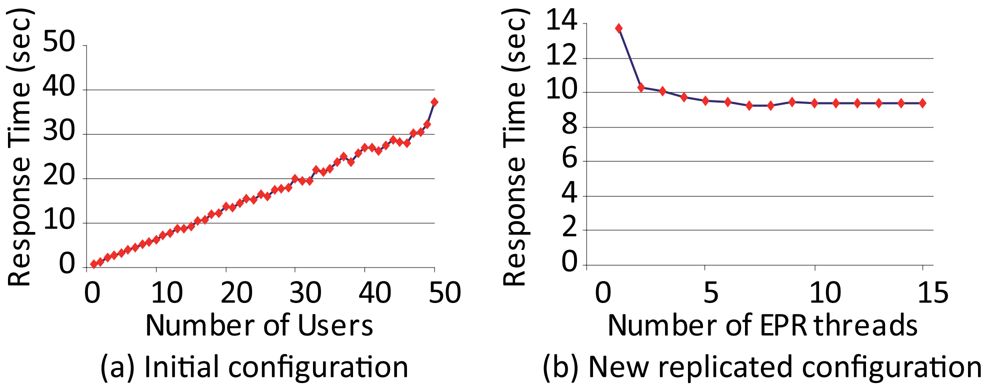


Figure 3.6: Response times obtained from the GSPN model.

Once it has been verified that the results, obtained by the GSPN, are similar to those obtained in (Catley et al., 2004) with LQN, the following step is to study, using this GSPN, the impact of XML parsers and SOAP implementations.

3.1.4 Performance Improvements for the CDSS Web Service

In this section we exploit the CDSS case study to deal with the goals **G1**, **G2**, **G3** and **G4** proposed in Section 3.1.1. The final objective is to extract conclusions about those key aspects of web service performance from the case study.

Impact of XML Parsers

Since XML parsers affect web service performance (Elfwing et al., 2002; Davis and Parashar, 2002; Head et al., 2005), we explore different alternatives in order to study their impact in the CDSS web service.

We realized that some of the CDSS parameters in (Catley et al., 2004) should be changed for the following considerations:

A *Document build time* is the time to scan and interpret the XML document (Sosnoski, 2001), but in Catley et al. (2004) is assigned to the packing operation. In our experiments, we will assign this value for parsing operations, see Table 3.1.

B *Document modify time* is the time required to systematically modify the constructed document representation (Sosnoski, 2001), but in Catley et al. (2004) is assigned to the parsing operations. We do not assign this value to an operation, since we consider that EPR file is not updated.

C *Document walk time* is the time required to walk the constructed document representation (Sosnoski, 2001). As Catley et al. (2004), we will assign it to validate the XML document, see Table 3.1.

D *Text generation time* is the time required to output document representations as text XML documents (Sosnoski, 2001). As Catley et al. (2004), we will assign it to transform the XML document, see Table 3.1.

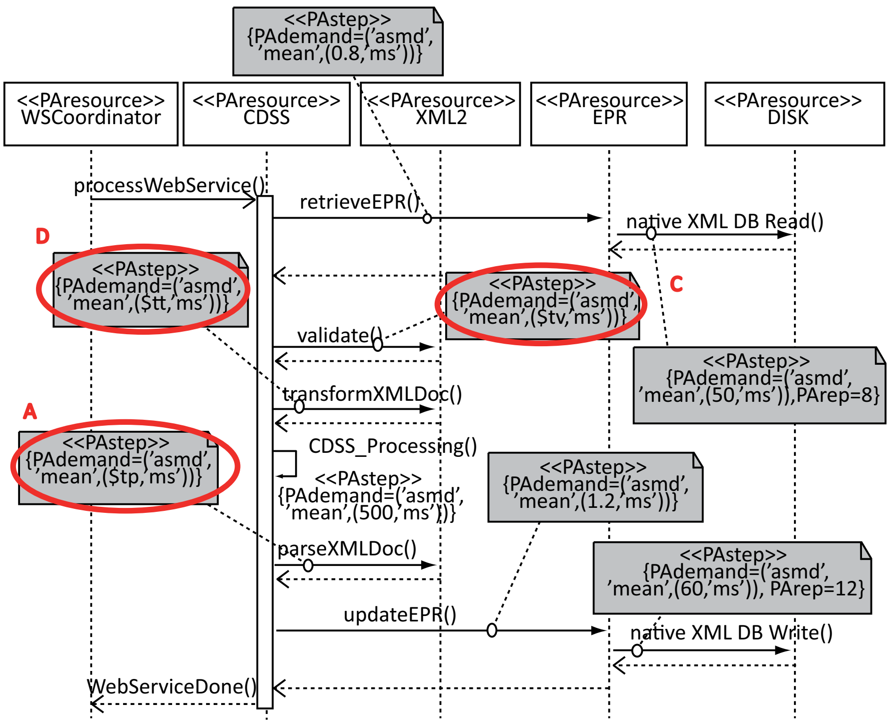
Table 3.1 gives the new values taken from an updated benchmark (Sosnoski, 2002). Figure 3.7 depicts the part of the SD that has been changed to consider the new values in the model.

Table 3.1: Performance parameters for XML operations from Sosnoski (2002)

Operation	**Parameter in SD**	**Mean Execution Time (ms)**			
		Xerces	**Xerces2**	**Crimson**	**XPP**
(A) `parseXMLDoc()`	\$tp	6.957	2.898	9.856	1.159
(C) `validate()`	\$tv	≃0	≃0	≃0	≃0
(D) `transformXMLDoc()`	\$tt	1.055	1.231	1.231	0.703

Figure 3.8(a) and (b) depict the response times when the parameters in Table 3.1 are applied. These results can be compared with those in Figure 3.2(a) and (b), as well as with those in Figure 3.6(a) and (b), being similar in all cases. Therefore, it does not matter which parser is used.

However, according to Elfwing et al. (2002) and Sosnoski (2002), the response times for Xerces parsers are worse than the ones obtained for Crimson or XPP parsers.



<<PAstep>> {PAdemand=('asmd', 'mean',(0.8,'ms'))}
<<PAresource>> WSCoordinator
<<PAresource>> CDSS
<<PAresource>> XML2
<<PAresource>> EPR
<<PAresource>> DISK
processWebService()
retrieveEPR()
native XML DB Read()
D
<<PAstep>> {PAdemand=('asmd', 'mean',($tt,'ms'))}
<<PAstep>> {PAdemand=('asmd', 'mean',($tv,'ms'))}
C
validate()
<<PAstep>> {PAdemand=('asmd', 'mean',(50,'ms')),PArep=8}
transformXMLDoc()
A
<<PAstep>> {PAdemand=('asmd', 'mean',($tp,'ms'))}
CDSS_Processing()
<<PAstep>> {PAdemand=('asmd', 'mean',(500,'ms'))}
<<PAstep>> {PAdemand=('asmd', 'mean',(1.2,'ms'))}
parseXMLDoc()
<<PAstep>> {PAdemand=('asmd', 'mean',(60,'ms')), PArep=12}
updateEPR()
native XML DB Write()
WebServiceDone()

Figure 3.7: UML Sequence Diagram with changes in CDSS w.r.t. the original proposal

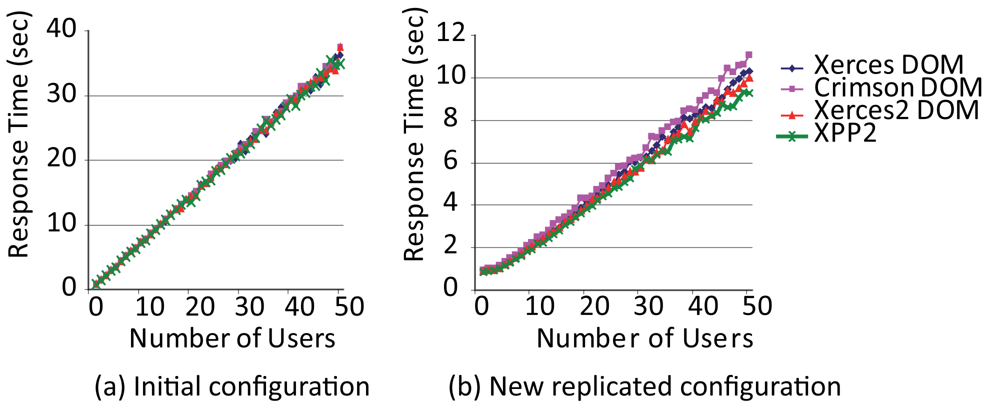


Figure 3.8: Response times for different XML parsers.

Slightly best results are obtained by XPP. The reason for our results is the small size of the EPR file, only 5 KBytes. So, in this case, the XML parser significantly does not affect the performance of the CDSS web service. But in Section 3.1.4 we try to validate the conclusions in (Elfwing et al., 2002; Sosnoski, 2002) by varying the EPR file size.

Impact of SOAP Implementations

Currently, several implementations of SOAP are emerging and their performance differs to a great extent (Davis and Parashar, 2002; Head et al., 2005). Therefore, it is profitable to determinate what toolkit meets performance objectives in the CDSS invocation web service.

We guess that in (Catley et al., 2004) the SOAP parameters are taken from (Sosnoski, 2001), but we consider more appropriate to use specific SOAP benchmark, taken from (Head et al., 2005). Table 3.2 provides the values of SOAP operations, (F) deserialization and (G) serialization and the overhead that the SOAP toolkit imposes, (E) the latency. Note that they have been calculated assuming that most of the content of the EPR file are strings. Figure 3.9 depicts the part of the SD changed to include in the CDSS these new parameters.

Table 3.2: Performance parameters for SOAP toolkits from (Head et al., 2005).

Operation	**Parameter**	**Mean Execution Time (ms)**		
	in SD	**AxisJava**	**.NET**	**XSUL**
(E) Latency → `protocolProcessing()`	$tl	8.35	3.5	2.435
(F) Deserialization → `deserialize()`	$td	10.476	4.797	3.935
(G) Serialization → `serialize()`	$ts	16.151	4.481	3.706

Figure 3.10(a) shows that all the SOAP toolkits give similar response time for the CDSS. Only XSUL performs a little better in the replicated configuration, see Figure 3.10(b). Comparing these results with those in Figure 3.2(a), they are alike. We guess that as the SOAP message, which contains the EPR file, is small, the time taken by processing SOAP is negligible with respect to the CDSS processing time. In Section 3.1.4, we try to verify this affirmation by varying this service processing time.

Impact of the EPR File Size

The previous experiments showed that due to the small size of the EPR file, both the XML parser and the SOAP implementations have no relevant impact for the performance of the CDSS web service.

If it would be considered that this file increases in size, the results could be different. Note that the XML-based EPR file is also enveloped in the SOAP message, therefore its size affects both XML and SOAP operations. Table 3.3 provides the new parameters, considering two sizes for the EPR, they are set in SDs of Fig-

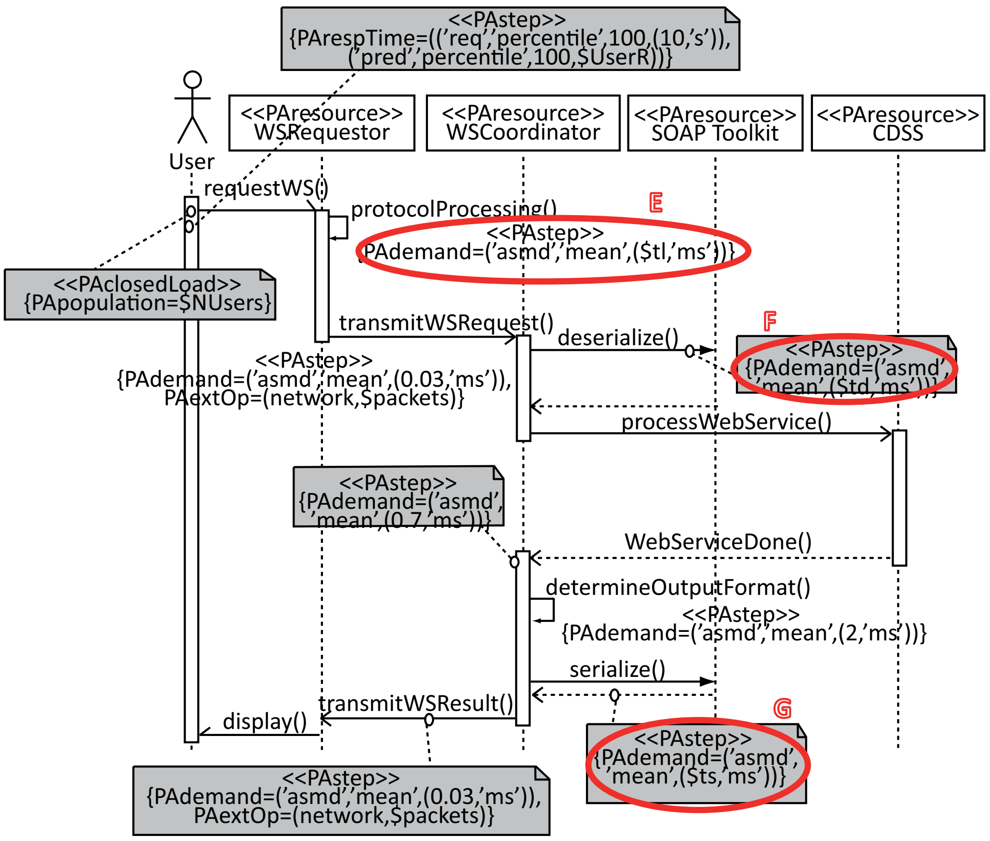


Figure 3.9: UML Sequence Diagram describing the proposed key performance scenario with SOAP toolkit

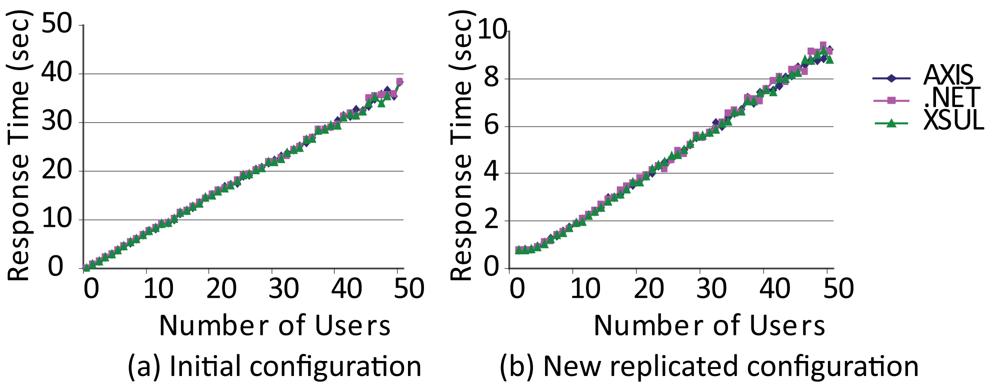


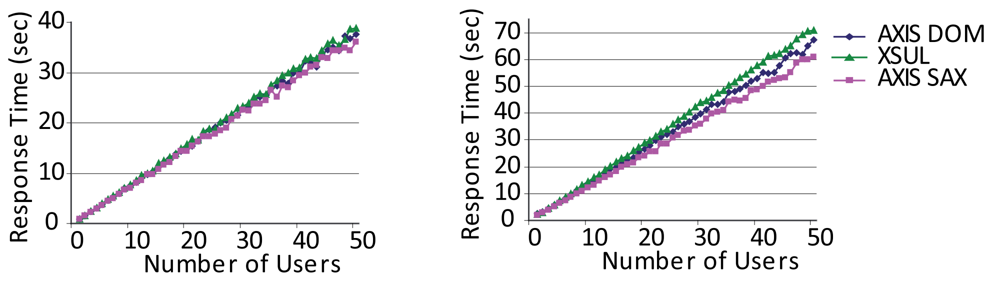
Figure 3.10: Response times for different SOAP implementations.

ure 3.7 and 3.9. We have taken into account that XPP is the native parser for XSUL and for Xerces through DOM or SAX interface for AxisJava.

Table 3.3: Performance parameters from Sosnoski (2002) and Head et al. (2005).

Parameter in SD	Mean Execution Time (ms)					
	100 KBytes			1 MBytes		
	AxisJava			AxisJava		
	DOM	SAX	XSUL	DOM	SAX	XSUL
(A) $tp	27.68	10.19	40.79	297.8	78.37	501.56
(C) $tv	1.37	≃0	0.68	31.34	≃0	15.67
(D) $tt	11.36	≃0	26.51	282.13	≃0	203.76
(E) $tl	8.35	8.35	2.435	8.35	8.35	2.435
(F) $td	44.39	44.39	26.78	917.11	917.11	431.464
(G) $ts	32.00	32.00	35.53	291.82	291.82	265.81

Figures 3.11(a) and 3.11(b) show the response time with the initial configuration when the EPR file size is 100 KBytes and 1 MBytes, respectively. If we observe the results when EPR file size is 100 KBytes, these are similar to those when it is only 5 KBytes, see Figure 3.10(a). However, the response time increases meaningfully with 1 MBytes. As Elfwing et al. (2002) expected, comparing the SOAP implementations, AxisJava through SAX interface outperforms AxisJava through DOM. Surprisingly, in spite of the good results of XSUL presented in Head et al. (2005) for large sizes, it performs poorly in this case. It may be due to the time required by XPP to build the document in memory, as suggested by (Sosnoski, 2002).



(a) EPR file sized with 100 KBytes. (b) EPR file sized with 1 MBytes.

Figure 3.11: Response time for the CDSS web service when EPR file increases in size.

Impact of the CDSS Processing Time

Once studied the impact of EPR file size, we come back to Section 3.1.4 to verify if the time required for processing this EPR file (with SOAP and the XML parser) is irrelevant with respect to the time taken by the `CDSS_Processing()` process. In order to validate this supposition, the service processing time will be modified. See the annotation of the `CDSS_Processing()` message self dispatched by the CDSS in the UML Sequence Diagram depicted in Figure 3.1.

In previous Sections, we guess that XML parser and SOAP toolkits have not influence in CDSS web service, since XML-based EPR file size is small. Therefore, the time required for being processed it by SOAP and XML parser is irrelevant with respect to the time taken by `CDSS_Processing()` process. In order to validate this supposition, the service processing time is modified.

Figure 3.12(a) depicts the response times for 50 users with the initial configuration when the `CDSS_Processing()` service time varies from 0.1 to 2.5 seconds and the EPR file is 5 KBytes; in Figure 3.12(b) the EPR file is 100 KBytes and in Figure 3.12(c), 1 MBytes.

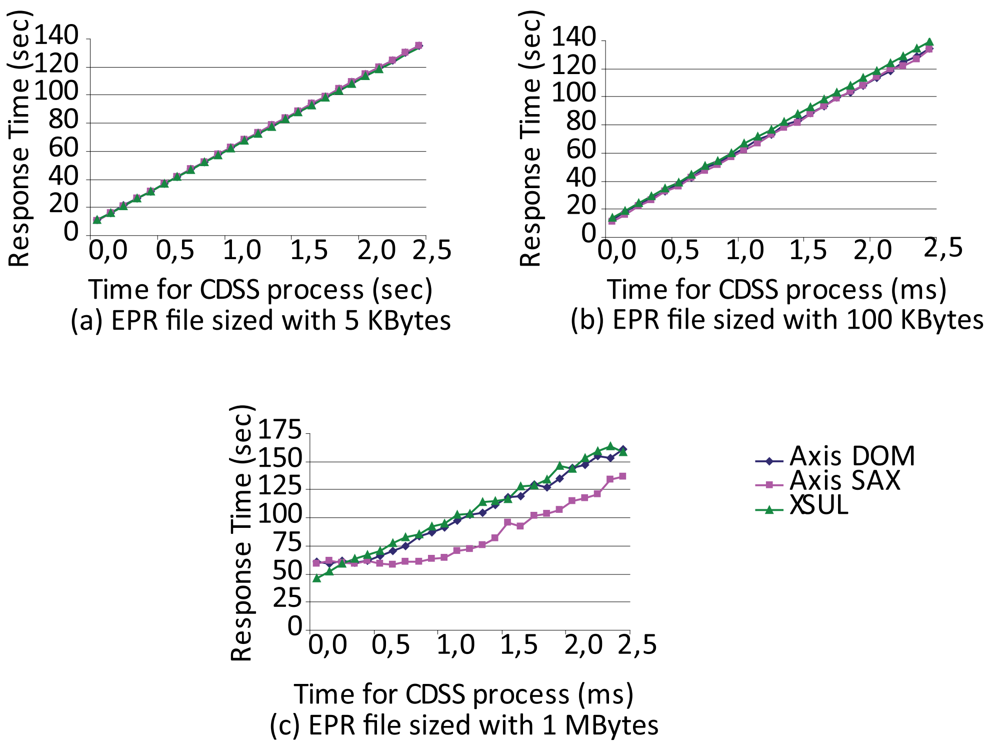


Figure 3.12: Response time when CDSS processing time increases.

The response time of the different SOAP implementations and XML parsers follows the same tendency while sizing EPR file to "*small sizes*", 5 KBytes or 100 KBytes. However, when EPR file is 1 MBytes, "*big sizes*", and `CDSS_Processing()` is less than 0.4 seconds, AxisJava through SAX parser performs poorly compared to AxisJava through DOM and XSUL. But, when service time increases, AxisJava through SAX parser outperforms them. We guess that XSUL performs better when the service time is small because it is oriented to slightly processed scientific data. Similarly, we guess that DOM and SAX outperform XSUL when the processing time is greater than 0.4 seconds since they are conceived to process generic information which may be repeatedly accessed.

This experiment shows that the CDSS processing time (`CDSS_Processing()`) and the EPR file size condition the impact of the XML parser and the SOAP implementation.

3.1.5 Related Work

Performance is an important aspect of web services. Nevertheless, from the best of our knowledge, very few papers focus on performance evaluation of web service-based applications. And a very few of them follow the techniques proposed in the SPE by Smith and Williams (2002b).

Menascé and Almeida (2001) developed a methodology from we have learnt the key issues of performance evaluation of web services. While this methodology is focussed on capacity planning using queuing networks (QN), we aim at its performance prediction using PN and the UML.

Chandrasekaran et al. (2003) proposed a simulation technique for analyzing performance of composite web services in order to obtain efficient web processes. Menascé (2004) studies QoS issues of composite web services. Datla and Goševa-Popstojanova (2005) presented a measurement-based study of performance of e-commerce applications. They study the impact of web services together with other components on integrated applications using benchmark techniques. Ng et al. (2004) evaluated diverse SOAP implementations by means of benchmarks of a simple service with three types of message. In contrast to us, they probe that serialization and deserialization are the primary important bottleneck for this application.

Liu et al. (2004) proposed an approach to predict performance metrics for a middleware-hosted application using QN models. Although, this work is focussed on a J2EE application, their modeling approach is suitable to other middleware technologies, such as CORBA and COM+/.NET.

Verdickt et al. (2005) proposed a Model Driven Architecture (MDA) model transformation for Platform Specific Models (PSM), including middleware performance details. It is based on SPE and the UML-SPT (OMG, 2005) profile. The transformation process is made by a tool which generates LQN models.

Gilmore et al. (2005) proposed a UML-based methodology for analyzing security and performance aspects using PEPA models. This method is implemented in the Choreographer design platform.

3.1.6 Concluding Remarks on CDSS

Our experiments indicate that the XML parser choice slightly affects web services performance when the XML-based file size is small, whereas the SOAP implementation influence is even smaller. However, when the EPR file increases in size, the response times obtained are worst and there exist noticeable differences among XML parsers and SOAP implementations. These differences intensify when the service processing time changes.

We can conclude that the impact of the XML parsers and the SOAP implementations is conditioned by both XML-based file size and service time, i.e., the serialization and deserialization processes are not bottlenecks for large data applications and large service times.

3.2 SPRINGS: Mobile Agents Tracking

Mobile agents have arisen as an interesting paradigm to build distributed applications, due to the unparalleled advantages they offer. However, along with the advantages they also present new challenges. One of the most relevant is that it is not easy to ensure efficient communication among agents that move continually from one computer to another.

In this section, we analyse the performance of the SPRINGS tracking approach developed by Ilarri et al. (2006). Our analysis allows us to validate the experimental results previously obtained by Ilarri et al. (2006), and to evaluate the platform in a variety of other hypothetical situations without the burden of real experimentation.

The structure of this section is as follows. In Section 3.2.1, we introduce basic aspects of mobile agent technology, which is important for this work. In Section 3.2.1, we describe and model the SPRINGS architecture for tracking mobile agents. In Section 3.2.2, the latter model is annotated with performance information. In Section 3.2.3, the PUMA approach is applied in order to get the performance models corresponding to the modeled architecture. Section 3.2.4 exploits the performance models to deal with the proposed analysis goals. Section 3.2.5 revises the related literature.

3.2.1 Mobile Agent Technology

Mobile agents (Milojičić et al., 1999) have stirred up a lot of interest and research efforts. They are programs that can autonomously travel from computer to computer, and present a range of unique advantages, such as autonomy, flexibility, and effective usage of network bandwidth (Lange and Oshima, 1999). Due to their features, the use of mobile agent technology is very attractive in wireless (Spyrou et al., 2004), pervasive, and distributed computing in general.

One of the main challenges in applications based on mobile agents is how to keep track of the current locations of the agents in order to allow an efficient communication among them. If the location of an agent cannot be obtained in a short time, the agent can move to another computer before such location data is used for communication purposes; this situation may occur indefinitely, leading to *livelock* problems from the point of view of the agents that want to communicate with the agent.

The vital importance of designing efficient communication and tracking schemes for mobile agents have been highlighted in many works, such as (Kastidou et al., 2003; Aridor and Oshima, 1999). Moreover, according to the experiments in (Ilarri et al., 2006), this is a key issue to ensure the scalability of a mobile agent platform, especially in highly dynamic contexts.

Several models for tracking agents are conceivable; thus, the work in (Aridor and Oshima, 1999) suggests three methods to locate agents (brute force, logging, and redirection), and in (Milojičić et al., 1998) were proposed four (updating at the home node, registering, searching, and forwarding). A mobile agent platform, called SPRINGS (Ilarri et al., 2006) (Scalable PlatfoRm for movING Software) proposes a new tracking approach; in (Ilarri et al., 2006) has been experimentally shown to be highly scalable and how it outperforms other popular platforms, especially in environments with a high number of mobile agents.

So how can an agent move to another computer and resume its execution there? Mobile agents need a specific execution environment, which we call *context*[1]. Thus, for an agent to travel to another computer, a context must be available there: an agent needs a context in the same way that a web page request needs a web server. Contexts are provided by a specific mobile agent platform (Silva et al., 2001), from which several alternatives are available, e.g., Aglets (Lange et al., 1997), Grasshopper (Bäumer and Magedanz, 1999) or SPRINGS; and provides them with different services, such as *communication and mobility.* The two mentioned services are interrelated. Particularly, mobile agents must be able to communicate among themselves, via remote method invocation or message passing, even if they move across computers.

Contexts and Regions

The architecture, depicted in Figure 3.13, of an agent platform is usually made of agents, contexts and regions.

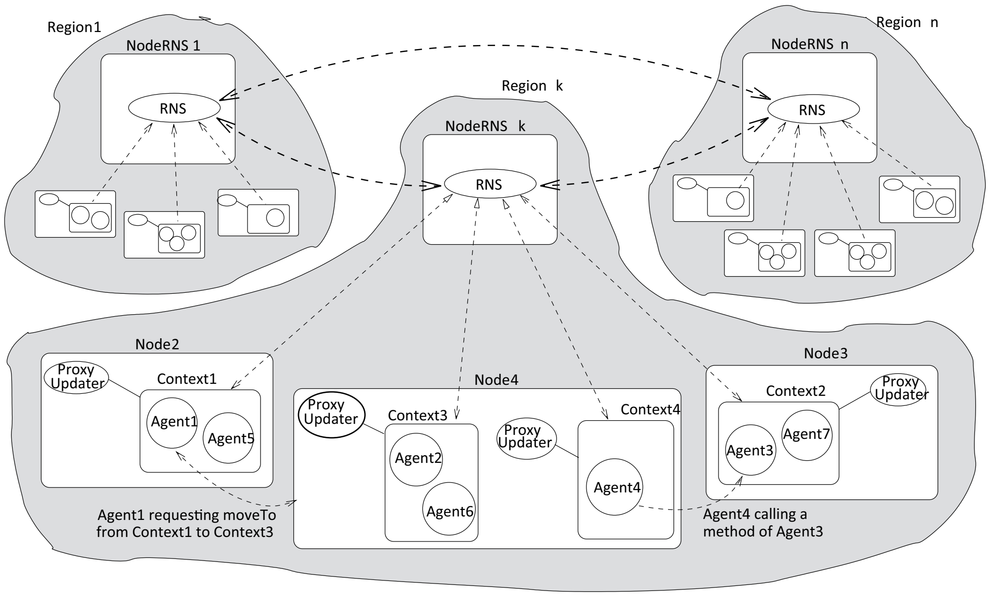


Figure 3.13: Architecture for mobile agent platforms.

- *Contexts* (also called *places* for example in Grasshopper) are the environment where agents execute: a computer can host several contexts, each one assigned to a different communication port and execution process. A context provides

[1]In the literature of mobile agents, the term *place* is frequently used instead.

agents with services such as a *call routing service* irrespective of the target agents' locations[2], and a *transportation service* to move to other contexts.

- A *region* is a set of related contexts. In SPRINGS, for example, the functionality of a region is provided through a remote object called Region Name Server (RNS), which can be located on any computer in the network. An RNS has several functions, such as ensuring the uniqueness of agent/context names, mapping from context names to context addresses, and assigning tracking responsibilities to contexts.

Modeling Tracking

A key functionality of a mobile agent platform is to offer a communication service that allows an agent to communicate with another without the need of knowing its current location. Most agent platforms use the idea of *proxy* as an abstraction to communicate with an agent (if an agent $a1$ wants to communicate with another agent $a2$, it must first obtain a proxy to $a2$); *location transparency* means that the proxy routes the message to its corresponding agent efficiently, wherever it is. SPRINGS hides the proxies to the programmer and stores them in the contexts[3]: an agent can communicate with another one by just specifying the name of the target agent, without the need of using proxies explicitly.

An agent proxy stores the (remote) reference to that agent: its name and current context. If the agent moves to another context, the information contained in the proxy becomes invalid. In SPRINGS, *dynamic proxies* are considered: when an agent arrives at a new context, the *remote proxies* to that agent (i.e., held by other contexts) are updated to reflect the new agent location. These proxies are updated before resuming the agent's execution in order to maximize the probability that another interested agent succeeds in communicating with it. In each context, a *Proxy Updater* thread is in charge of updating the remote proxies to the incoming agents efficiently (see (Ilarri et al., 2006) for more details).

Regarding agent mobility, two important related concepts are considered:

- *Location servers.* A location server of an agent a is a context that stores a dynamic proxy to a because it has been assigned by the RNS to do so.

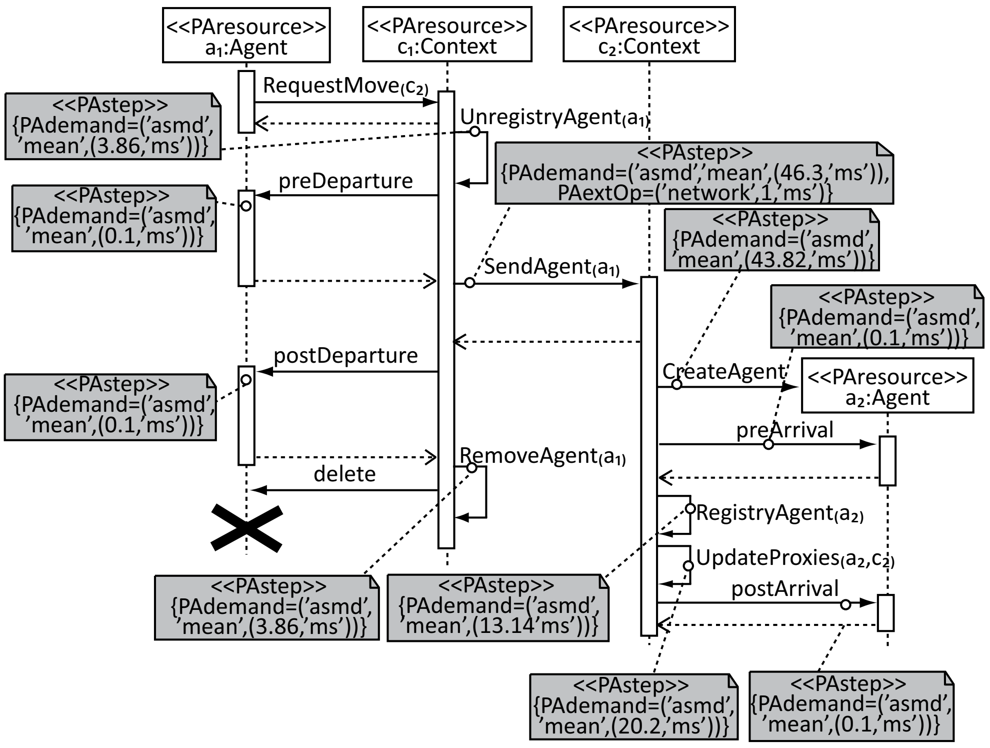
- *Observer contexts.* A context c is an *observer* of a certain agent a when it is interested in knowing the current location of a, which happens when: 1) a local agent has communicated recently with a, which means to include a in its *ProxyList*, or 2) it is a location server for a. An observer of an agent a always stores a dynamic proxy to the agent.

Following the concepts given so far, the SD in Figure 3.14 describes the scenario of how an agent changes its context and how the platform keeps track of it. Firstly,

[2]Not all the existing platforms feature this property.

[3]All the agents in a context that want to communicate with another one use a shared proxy that points to that agent.

an agent a_1 requests to its current context c_1 to travel to a new one c_2. The origin context unregistries the agent. Just before traveling, the agent prepares its departure and when it has finished, the current context sends it to c_2. When it arrives, a new instance a_2 of the agent is created; meanwhile the old instance a_1 at origin ends its departure and it is completely removed from the origin context. Assuming that the agent has successfully arrived, it prepares its arrival and then it is registered. Afterwards, its proxies have to be updated, so the *Proxy Updater* thread, embedded in the Context component, informs all observer contexts of the agent to update its dynamic proxy. After this, the arrival has finished.



<<PAresource>> a1:Agent
<<PAresource>> c1:Context
<<PAresource>> c2:Context
RequestMove(c2)
UnregistryAgent(a1)
<<PAstep>> {PAdemand=('asmd', 'mean',(3.86,'ms'))}
<<PAstep>> {PAdemand=('asmd','mean',(46.3,'ms')), PAextOp=('network',1,'ms')}
preDeparture
<<PAstep>> {PAdemand=('asmd', 'mean',(0.1,'ms'))}
<<PAstep>> {PAdemand=('asmd', 'mean',(43.82,'ms'))}
SendAgent(a1)
<<PAstep>> {PAdemand=('asmd', 'mean',(0.1,'ms'))}
postDeparture
CreateAgent
<<PAresource>> a2:Agent
<<PAstep>> {PAdemand=('asmd', 'mean',(0.1,'ms'))}
preArrival
RemoveAgent(a1)
delete
RegistryAgent(a2)
UpdateProxies(a2,c2)
postArrival
<<PAstep>> {PAdemand=('asmd', 'mean',(3.86,'ms'))}
<<PAstep>> {PAdemand=('asmd', 'mean',(13.14'ms'))}
<<PAstep>> {PAdemand=('asmd', 'mean',(20.2,'ms'))}
<<PAstep>> {PAdemand=('asmd', 'mean',(0.1,'ms'))}

Figure 3.14: Annotated UML Sequence Diagram for agents movement in SPRINGS architecture.

The SD in Figure 3.15 models how agents communicate. Let us assume that an agent a_1 executing on context c_1 wants to communicate with another agent a_2 on context c_2. If any agent on c_1 has recently communicated with a_2, a dynamic proxy to a_2 will be locally available (c_1 is an *observer* of a_2); in this case, the *callAgent* will be directly routed through that proxy, without executing the *locationTransparency* fragment. Otherwise, c_1 must find a_2 in the following way: 1) c_1 obtains from its RNS a location server for a_2; 2) c_1 obtains a proxy to a_2 from that *location server* (ls_2), which registers c_1 as a new *observer* for a_2 (*UpdateProxyList* task); and 3) the retrieved proxy is used to route the call to a_2, and it is stored by c_1 (so the RNS will not need to be contacted the next time). Information about agents not located in the region would be requested by the local RNS to other RNS. In that case, the sequence diagram in Figure 3.15 must be augmented with a lifeline representing the RNS which knows a location server for the called agent.

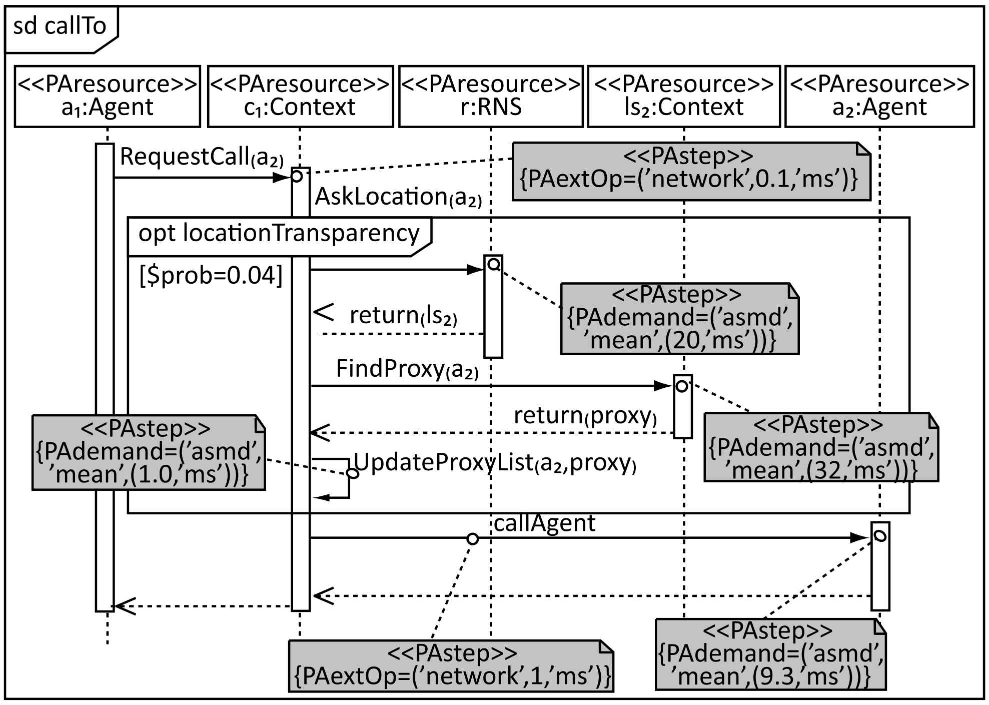

sd callTo
<<PAresource>> a1:Agent
<<PAresource>> c1:Context
<<PAresource>> r:RNS
<<PAresource>> ls2:Context
<<PAresource>> a2:Agent
RequestCall(a2)
<<PAstep>> {PAextOp=('network',0.1,'ms')}
AskLocation(a2)
opt locationTransparency
[$prob=0.04]
return(ls2)
<<PAstep>> {PAdemand=('asmd', 'mean',(20,'ms'))}
FindProxy(a2)
return(proxy)
<<PAstep>> {PAdemand=('asmd', 'mean',(1.0,'ms'))}
UpdateProxyList(a2,proxy)
<<PAstep>> {PAdemand=('asmd', 'mean',(32,'ms'))}
callAgent
<<PAstep>> {PAextOp=('network',1,'ms')}
<<PAstep>> {PAdemand=('asmd', 'mean',(9.3,'ms'))}

Figure 3.15: Annotated UML Sequence Diagram for agents communication in SPRINGS architecture.

Finally, the modeling of the physical structure in Figure 3.13 is provided using a UML Deployment Diagram (DD), see Figure 3.16, which describes resources on the system connected through a network. The DD depicts the system architecture for only one region; new regions with their contexts can be incorporated by duplicating this structure. Since a node may host several contexts and each of them provides services to a number of agents, there may exist several instances of both agents and contexts on each node.

3.2.2 System Performance View

In this section, we present the performance view of the proposed mobile agents tracking approach. We use the UML-SPT profile (OMG, 2005) to annotate both the performance metrics, that characterize the goals of our performance analysis, and the

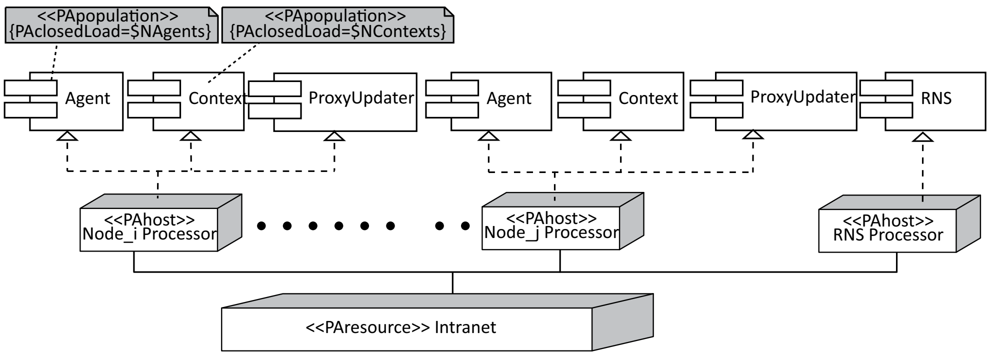


Figure 3.16: UML Deployment Diagram of the SPRINGS architecture.

performance parameters of the system.

Performance Metrics

The first analysis goal is to validate the analytical results obtained from the UML-SPT models against those experimentally obtained in (Ilarri et al., 2006). The valid performance models will be used to determine the *platform optimal configuration*. Finally, this configuration will allow to perform a sensitivity analysis, i.e., to study system response time when the agents size increases or the system is running on slow networks.

The experiments in (Ilarri et al., 2006) present a configuration composed of a single region with 5 contexts residing on 5 computers, one of them executing also the RNS and a variable number of agents ranging from 1 to 1500, each one assigned to a context thread. The analytical experiments, Section 3.2.4, consider the same number of agents and regions as in (Ilarri et al., 2006), but a variable number of context threads running on separate processors, as expressed in the DD annotations, see Figure 3.16.

The Interaction Overview Diagram (IOD) in Figure 3.17 depicts the performance scenario, where the analysis goals will be studied. Considering this IOD, an agent will change its current context (`moveTo`) and immediately will perform a method invocation to another agent (`callTo`). The proposed analysis goals will be studied using as performance metric the one defined in the IOD, i.e., the *scenario response time*.

Performance Parameters

The performance information concerning the actions duration and the messages delay has been taken from (Ilarri et al., 2006), they correspond to the experiment described in Section 3.2.2 when only one agent was executing the platform.

The actions are represented by the stereotype `<<PAstep>>`, where the *PAdemand*

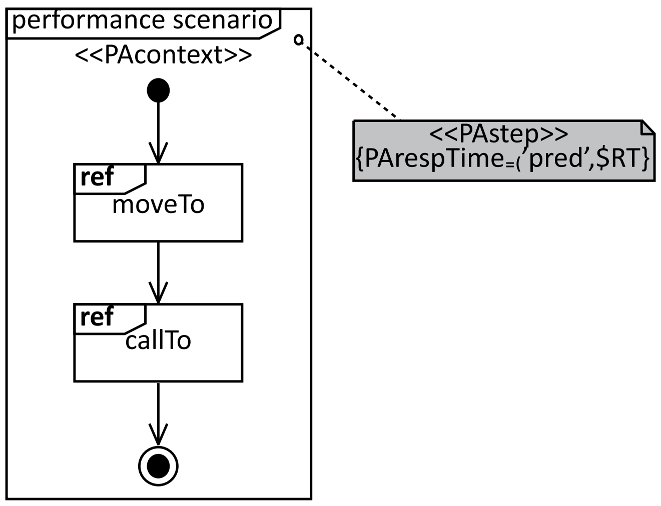


Figure 3.17: UML Interaction Overview Diagram of the performance scenario in SPRINGS.

tag specifies its corresponding execution or delay time as an exponentially distributed random variable. Table 3.4 summarizes the mean execution times that have been annotated along the SDs. By mean execution time we mean that these processing times have been measured by running the system repeating 50 iterations per agent. Experiments in (Ilarri et al., 2006) were developed in this way to ensure accuracy in the experimental tests.

Another parameter that may impact the system performance is the probability of executing the optional fragment `LocationTransparency` in Figure 3.15. Values close to 0 mean that the `ProxyList` of context c_1 owns knowledge enough to solve most of the `RequestCall` messages. Then, values close to 1 are supposed to penalize system performance.

The network is indirectly specified by the $PAextOp$ tagged value, see `SendAgent` message in the SD of Figure 3.14. The $<< PAresource >>$ stereotype annotated in the lifeline of each object defines them as software components.

3.2.3 Performance Models

Once, the performance scenario and its performance parameters have been defined, we apply the PUMA approach to get the performance models where to evaluate the proposed performance metric.

As mentioned in Section 2.3.3, PUMA (Woodside et al., 2005, 2013) is a framework that aims at extracting from a design model (UML or Use Case Maps) an intermediate model, called CSM (Petriu and Woodside, 2007). We translate the CSM into Generalized Stochastic Petri Nets (GSPN) (Ajmone Marsan et al., 1995).

Table 3.4: Mean execution time for system basic operations in SPRINGS.

Operation	**Mean Execution Time (ms)**
`preDeparture()`	0.10
`postDeparture()`	0.10
`preArrival()`	0.10
`postArrival()`	0.10
`CreateAgent()`	43.82
`RemoveAgent()`	3.86
`SendAgent()`	46.30
`RegistryAgent()`	13.14
`UnregistryAgent()`	3.86
`UpdateProxies()`	20.20
`RequestCall()`	0.10
`CallAgent()`	9.30
`AskLocation()`	20.00
`FindProxy()`	32.00
`UpdateProxyList()`	1.00

Building the CSMs

According to PUMA, each SD in the IOD of Figure 3.17 has been translated into a CSM scenario. So, Figures 3.18 and 3.19 illustrate the CSMs that represent the UML SDs in Figures 3.14 and 3.15, respectively.

In order to make clear how PUMA proposes to generate the CSMs, a piece of execution is explained. See the `RequestMove` message in the SD of Figure 3.14, it is straightforward to check that it has its corresponding step in the CSM of Figure 3.18. Furthermore, before executing it, it is necessary to acquire the agent a_1 and the context c_1 software components, which run on the CPU $Node_1$.

Each element in a CSM (e.g., steps, components or resources) has attributes concerning the performance information annotated in the SD. For example, the `RequestMove` step has a demand attribute, its value is taken from the `<<PAstep>>` annotation in the SD. However, we have not shown these performance attributes in our CSM scenarios due to lack of space, but they will be used when parameterizing the performance model.

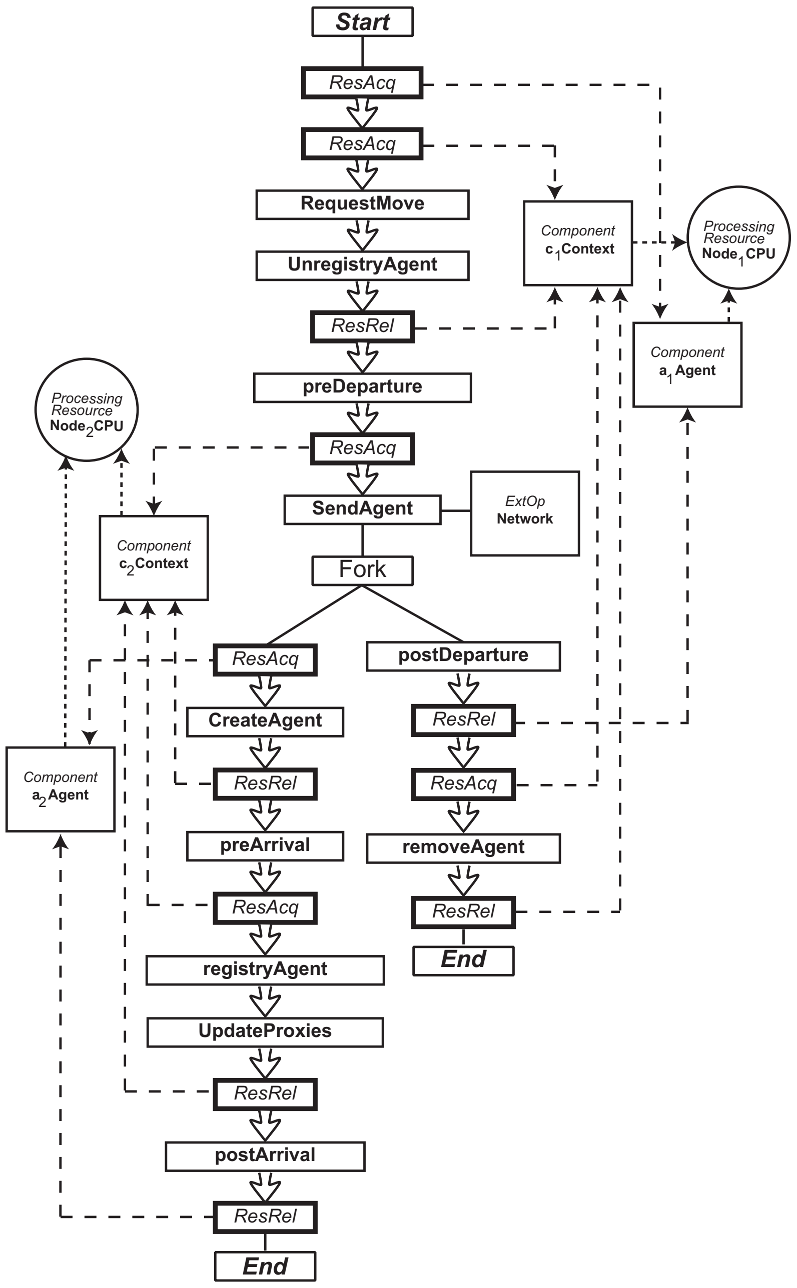


Figure 3.18: Core Scenario Model for moving an agent among contexts in SPRINGS architecture.

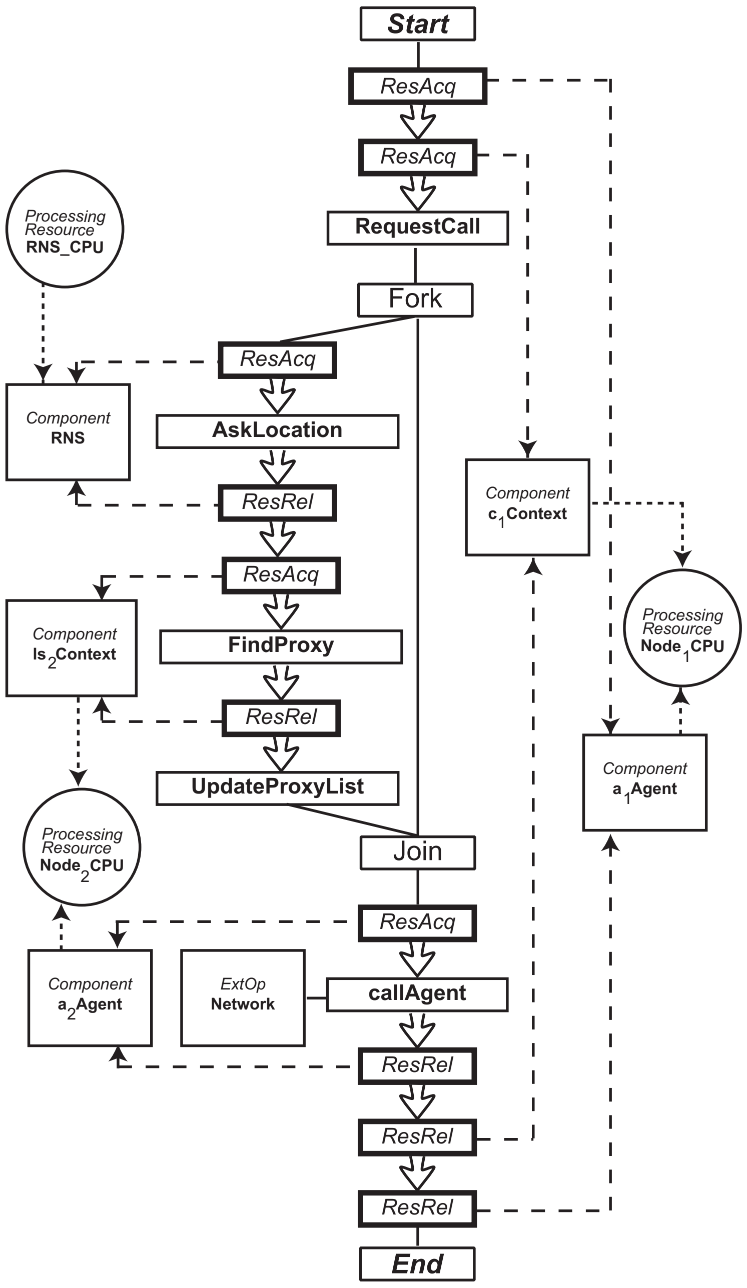


Figure 3.19: Core Scenario Model for agents' communication in SPRINGS architecture.

Building the Performance Models

The next step is to translate the CSMs into GSPNs following the translation process given by PUMA. Figures 3.20 and 3.21 depict the GSPNs that represent the CSMs in Figures 3.18 and 3.19. Just to outline the translation, see the UnregistryAgent step in the CSM of Figure 3.18, it is mapped into a timed transition, see Figure 3.20, being its delay defined as the *demand* attribute of the step. Previously, the agent a_1 and the context c_1 have been acquired, see transition $t\_c_1\_1$. Places representing resources, such as CPUs or software components, are marked with the amount of tokens specified by the corresponding `PAclosedLoad` tag.

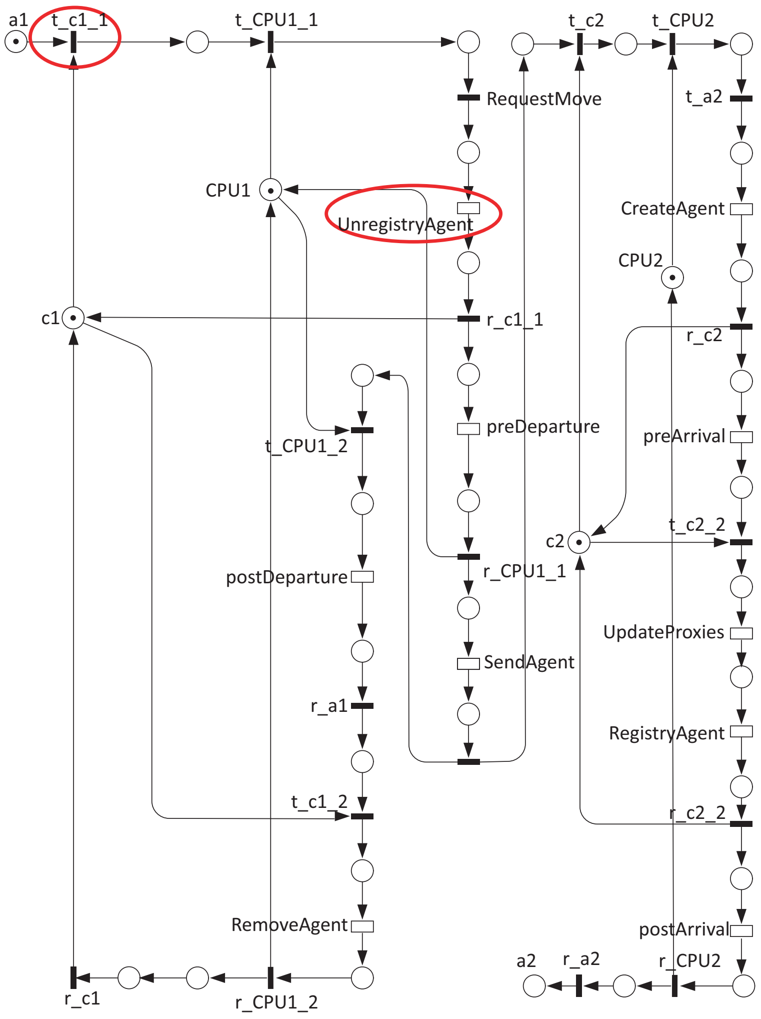


Figure 3.20: GSPN for agent movement in SPRINGS architecture.

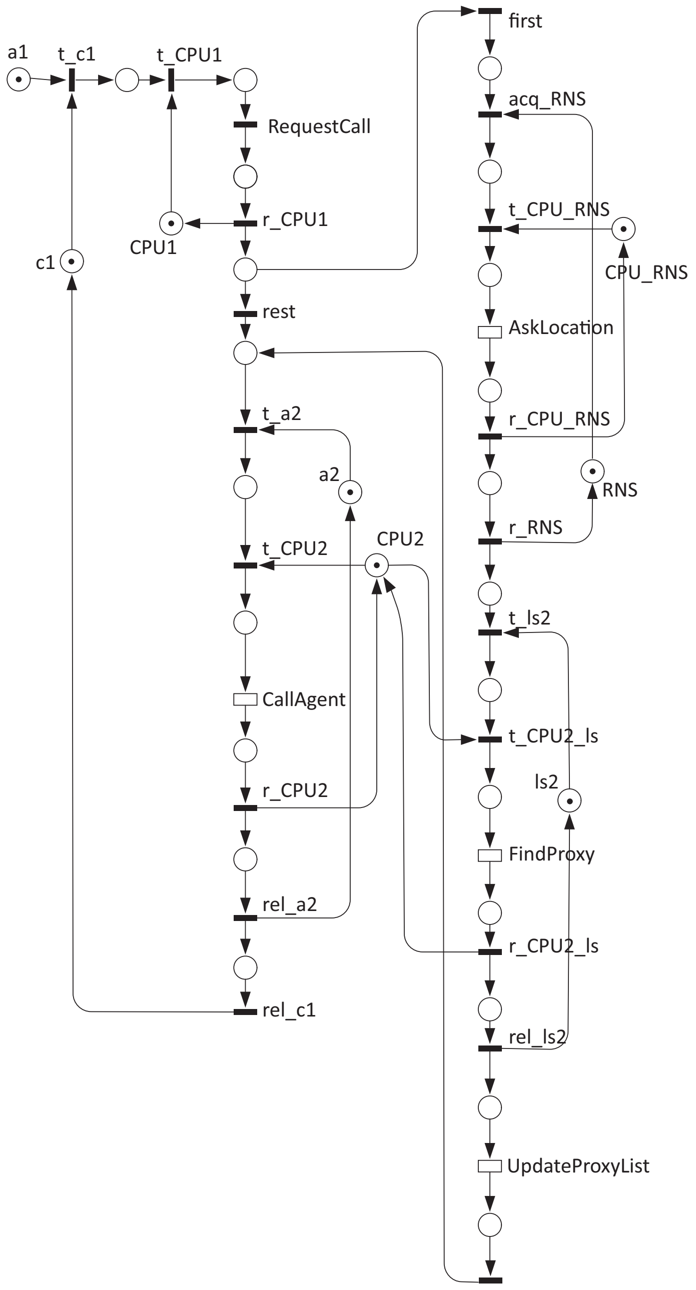


Figure 3.21: GSPN for communication between agents in SPRINGS architecture.

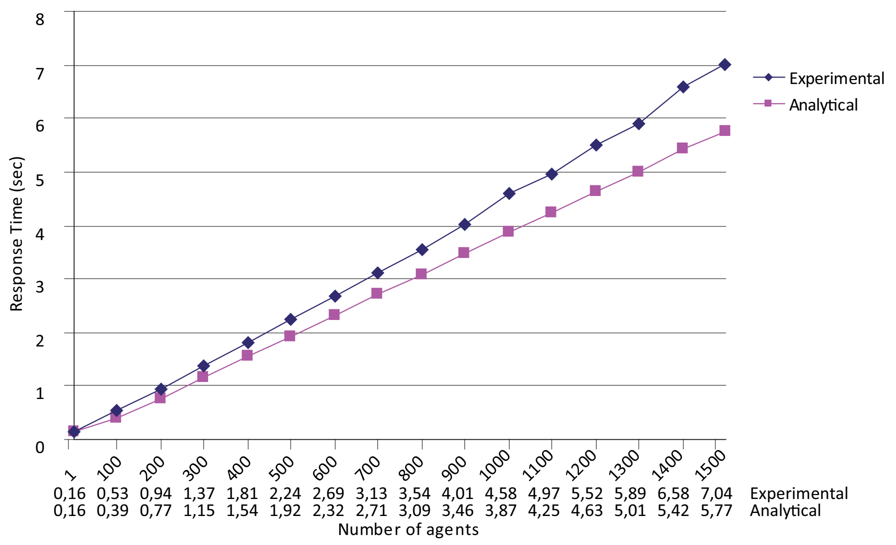
Finally, the GSPNs in Figures 3.20 and 3.21 are composed in order to obtain a performance model, i.e., a new GSPN that models the performance scenario in Figure 3.17. GSPN composition is based on merging the net places that represent common elements in the CSMs, such as resources or components.

3.2.4 Performance Analysis

Once the performance model has been built, we use TimeNET (Zimmermann et al., 2000) to compute the given metric in it by means of simulation techniques.

Validation of the Performance Models

Figure 3.22 depicts the response times given by the experimental test in (Ilarri et al., 2006) and the performance model. The configuration for each one was described in Section 3.2.2. As it can be observed, the results are very similar, and in both cases the response time increases linearly.



Number of agents	1	100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500
Experimental	0,16	0,53	0,94	1,37	1,81	2,24	2,69	3,13	3,54	4,01	4,58	4,97	5,52	5,89	6,58	7,04
Analytical	0,16	0,39	0,77	1,15	1,54	1,92	2,32	2,71	3,09	3,46	3,87	4,25	4,63	5,01	5,42	5,77

Figure 3.22: System response times of SPRINGS architecture.

In Ilarri et al. (2006), linear scalability was considered acceptable, since the experiments were carried out to test the platform in a stressful scenario for highly mobile agents (Murphy and Picco, 2002). Moreover, for a platform to support 1500 agents in such scenario is a real challenge. Mobile agents platforms can get lower response times when agents do not move and communicate so frequently. In the following we describe the stressful scenario.

The goal is to make agents stay on a context for a very short time and continually calling among themselves. To do so, in the experimental configuration described in

Section 3.2.2, each agent has a communication *peer* and performs the following steps: 1) calls its peer; 2) moves randomly to another context; and 3) steps 1 and 2 are repeated without delay until reaching 50 iterations. While the number of agents increases up to 1500, the performance of agent communication decreases; thus, a target agent could move to another context before a message reaches it. So, the *scalability* of the platform is studied in terms of the time that an agent needs to perform one entire iteration (`callTo` + `moveTo`) as the number of agents increases.

Optimal Configuration and Sensitivity Analysis

Optimal configuration for a platform means to minimize the number of context threads needed to execute the agents while keeping the response time.

The analytical experiment in Figure 3.22 was very relaxed in this sense and it considered up to 1500 threads, i.e., a thread per agent.

However, in a new experiment, depicted in Figure 3.23, we drastically reduced the number of threads, in a range from 50 to only 1, while keeping the rest of the parameters. The response time grows exponentially with values under 10 threads.

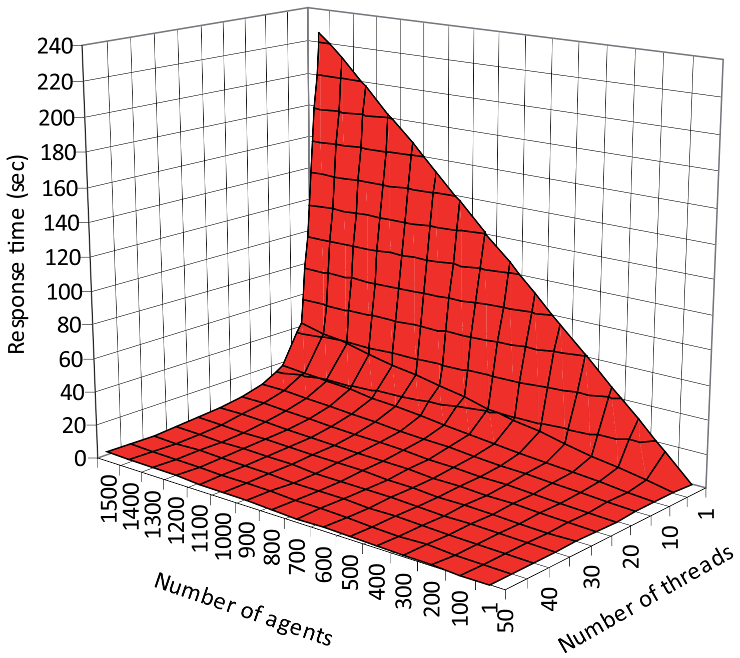


Figure 3.23: Response times when multithreading contexts in SPRINGS architecture.

Finally, Figure 3.24 enlarges Figure 3.23 in the range from 50 to 15 threads. Now, we can observe that the response times obtained in Figure 3.22 are preserved only for the range from 50 to 40 threads. Therefore, 40 threads is the optimal configuration for the agent platform, because values lower than 40 cause response times greater than 6 sec. with 1500 agents.

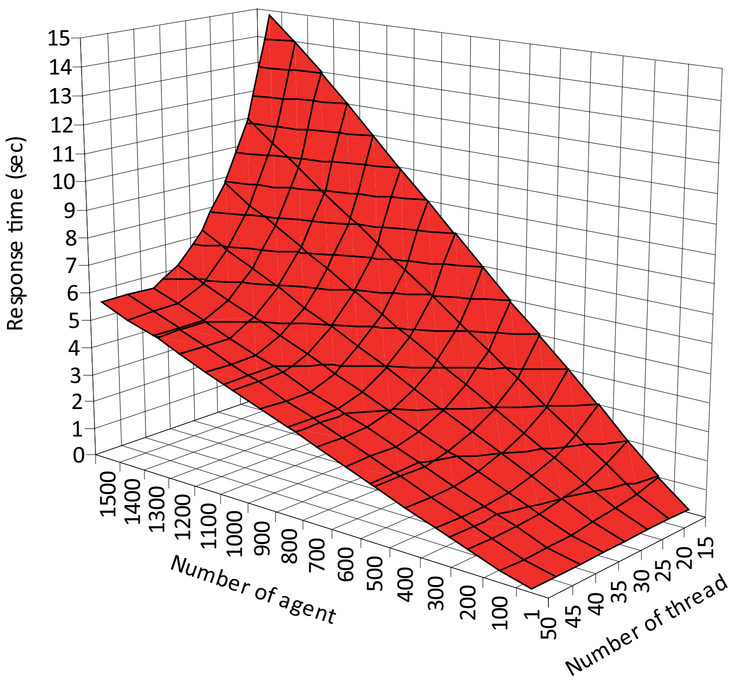


Figure 3.24: Detail of Figure 3.23.

The network speed may also affect the performance of the platform, influencing the time spent by agents traveling to another context, which is captured by the `SendAgent` task.

Using the optimal configuration with 1500 agents and considering that the `SendAgent()` message size is 2 KBytes, Figure 3.25 illustrates the system response time when the delay of the `SendAgent()` message varies. Be aware that a send delay of 10 ms corresponds with a network speed of 200 KBytes per second, while 250 ms corresponds with 8 KBytes per second. The system is sensitive to the network speed; although from 33 KBytes per second (60 ms), it does not perform better.

3.2.5 Related Work

To the best of our knowledge, there is no other work that analyses the performance of mobile agent tracking strategies using a model-based approach; an experimental analysis can be found in (Ilarri et al., 2006). In the following, we study the most relevant works related with the modeling of mobility with performance analysis purposes.

Currently, there is no standard way for modeling mobility, although different pro-

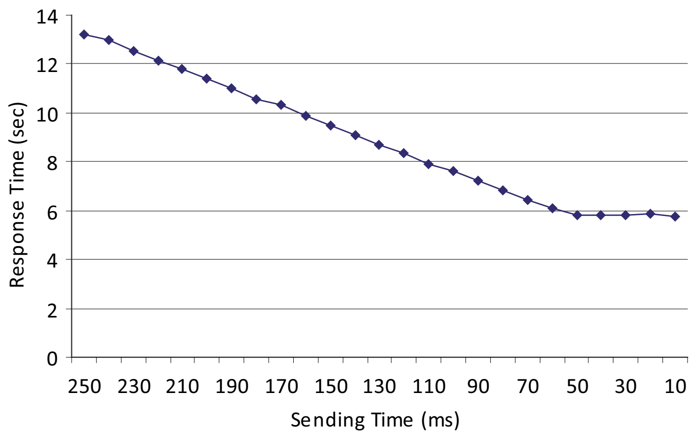


Figure 3.25: Sensitivity analysis of SPRINGS architecture.

posals exist, some of them based on UML (Baumeister et al., 2003; Balsamo and Marzolla, 2003b; Grassi et al., 2004; Bracchi et al., 2004), while others such as (Hillston and Ribaudo, 2004) follow some formalism, in this case PEPA nets.

The work in (Baumeister et al., 2003) presents an extension of the UML class, sequence and UML Activity Diagrams to model mobile systems and performance and security characteristics, but they do not explain how to get any performance model or how to compute metrics. In (Grassi et al., 2004), a non-standard UML profile for modeling mobile systems using activity, deployment and state machine diagrams is proposed, as well as an extension for collecting performance information according to the UML-SPT. The models are automatically translated into queueing networks to analyse performance. Balsamo and Marzolla (2003b) described a UML-based methodology for modeling and evaluating the performance of mobile systems using use case, activity and UML Deployment Diagrams augmented with the UML-SPT. They use simulation techniques to compute metrics.

Bracchi et al. (2004) proposed a framework to model performability for mobile software systems using use case, sequence, collaboration and deployment diagrams, from which Stochastic Activity Networks (SANs) are obtained.

Finally, it is worth noticing some works that use Petri nets for mobility and performance. In (Scarpa et al., 2002), the performance of different communication paradigms is compared using stochastic Petri nets. Merseguer et al. (2003) compared the performance of two software retrieval systems applying SPE techniques using high-level Petri nets.

3.2.6 Concluding Remarks on SPRINGS

In this section we have analysed the performance of the SPRINGS tracking approach.

The most interesting conclusion for the SPE point of view is that it has been possible to analyse one of the key aspects concerning performance of mobile agent platforms without tailoring a SPE methodology for this purpose. Therefore, the paper shows that the PUMA approach is powerful enough to deal with complex performance problems in the mobile agents software domain. Nevertheless, PUMA method lacks assessment proposal about performance improvements.

3.3 UCH: Universal Control Hub

In industrial societies, people massively use electronic devices in everyday life: mobile phones, TV sets or washing machines are some of several examples. Nevertheless, their use may became very complicated, and even impossible, to people with special needs, such as impaired or elderly people. Their user interfaces are not generally designed considering their needs neither *Design for all* principles proposed by Newell (2008).

According to last reports of Eurostat office (2013), the majority of the European countries own more mobile subscriptions than inhabitants. Internal studies of ONCE[4] foundation demonstrate similar trends for people with disabilities. So, the achievement of moving the proper control of electronic devices to adapted devices (e.g., mobile phone) may solve most user interface accessibility issues. Therefore, interoperability is critical to realizing the vision of personalized and pluggable user interfaces for electronic devices and services. An International Standard on pluggable user interfaces has here a key role to play, Universal Remote Console (URC) (ISO/IEC, 2008). Such a standard would facilitate user interfaces that adapt or can be adapted to user's personal needs and preferences. It would allow easy to use interfaces that employ various modalities for input and output.

Limitations in URC advised to develop an architecture called Universal Control Hub (UCH) (Zimmermann and Vanderheiden, 2007) to make URC practical in real scenarios. In short, UCH is a URC realization that acts as a gateway for communicating devices. UCH has been implemented, using different languages and technologies (URC Consortium, 2010c,b), and currently is offering adequate and interoperable service within environments of a reduced number of users. However, for UCH is not only interoperability the critical issue, performance is or will be a must when the amount of plugged devices depletes the infrastructure to the point of exhausting its resources. This may happen in real scenarios such as intelligent buildings where hundreds or thousands of users will concurrently access the deployed architecture. Although UCH and underlying implementations have not been tested in such environments, we propose to assess whether UCH can offer quality of service in these interesting settings. Note that the costs (in budget as well as in technical difficulties) prevent the architecture from testing in the new proposed environments, then in our opinion the use of predictive performance models can play a role in this context. Besides, in case of identifying adversities that turn UCH into a non practicable solution, the testing investment would be a waste in resources. So, the assessment should be also useful to pinpoint where in the UCH architecture are the problems located, again predictive models can offer cheaper solutions than real experimentation.

The study of the performance of UCH in future real situations is then a necessity that we will carry out using the formal model of Petri nets (Ajmone Marsan et al., 1995) in the context of the PUMA approach (Woodside et al., 2005, 2013).

In the following, the URC-UCH architecture is detailed in order to understand potential performance problems it may provoke. We analyse UCH and propose a set

[4]National Organization of blind people in Spain.

of experiments that will allow to compare their results with results obtained from current real implementations, then they will validate the performance model. From the validated model, the system will be tested to assess its usefulness for future necessities above described.

3.3.1 URC-UCH Architecture

The URC is an ISO published as the standard ISO/IEC 24752 in 2008. URC describes an interoperable architecture with a set of elements that allow users to control one or various devices by means of a remote console, in a transparent way for them. So, URC, or remote console, defines a framework of remote access to control devices or services. It can be designed both, as a dedicated hardware (e.g., a universal remote control), or as a URC-complaint software to run on specific devices such as personal computer, PDA or mobile phone. Therefore, it is a device or software architecture (gateway) through which the user accesses other devices, then being capable of rendering its user interface. This fact allows to develop adaptable user interfaces, which can satisfy users with special needs. In the following, the devices or services that the user wants to control are referred as *targets*, and the *controller* may be any user device. For instance, a blind person can control the washing machine, in this case the target device, by means of his/her mobile phone (controller device). So, URC allows to show washing machine functionalities in accessible manner.

ISO/IEC 24752 does not impose how it must be implemented. Besides, it does not assume a specific network protocol between controller and targets, but only network requirements. So, a URC interaction could be implemented on top of existing networking platforms as long as they support device discovery, control and eventing, such as UPnP (universal plug and play), Web services and/or HomePlug (IEEE 1901:2010). Among others, URC defines the following XML documents: *Target Description* (TD) and *User Interface Implementation Description* (UIID). The TD document permits the remote console to learn how to use the target device, locate its functionalities, current status, and other interesting information. The main advantage of UIID is that delivers a generic user interface, so the remote console can implement it under the most adaptive way to the user (optical, audible, tactile), addressing *Design for All* principles (Newell, 2008). Nevertheless, URC presents some issues: lack of devices with URC technology, lack of plugging in several targets and multiplicity of communication protocols.

The Universal Control Hub (UCH)architecture fixes some of the above mentioned problems (Zimmermann and Vanderheiden, 2007) . Indeed, UCH is seen as an “open box” between the target and the controller, acting as gateway between various controllers and various targets, which overcomes communication limitations of URC. Basically, UCH is a manner to implement URC, that focusses on normalizing how the Control Hub works. So, UCH defines APIs and interfaces between internal modules of Control Hub, inheriting the URC XML documents. Figure 3.26 depicts the components in UCH:

- **User Interface Protocol Module** (UIPM): is a “pluggable user interface” that specifies a protocol between the controller and the Socket Layer via an API. URC-HTTP protocol is a UIPM specification based on HTTP.

- **Socket Layer**: is the core part of UCH, hosting the sockets of the targets.
- **Target Adaptor** (TA): synchronizes one or multiple targets with their sockets (running in the Socket Layer). TAs can be dynamically loaded at runtime.
- **Target Discovery Module** (TDM): discovers specific targets, connects to the Socket Layer via API, and to the targets via any protocol. TDMs can also be dynamically loaded at runtime.
- **UIList**: contains a dynamic list of available user interfaces, as given by the currently loaded UIPMs.

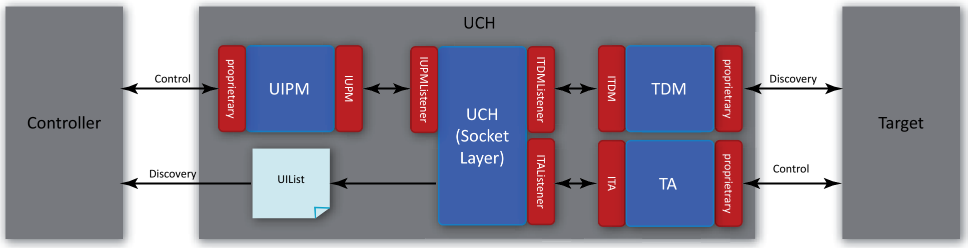


Figure 3.26: Components of UCH architecture taken from (URC Consortium, 2005).

Currently, there are three implementations of UCH, two of them developed under open source: UCHj (2010c) and UCHe (2010b), and another one under proprietary software. UCHj is a Java implementation designed for a closed delimited network, such as an office or home. UCHe is developed in C/C++ for embedded systems. Recently, a UIPM client for iPhone smart phone has been published (URC Consortium, 2010a), however this one does not implement UCH core.

These different implementations could seriously affect performance in a scenario with concurrent users. As URC and UCH are based on exchanging XML messages, they suggest poor performance, as previous studies have observed (Elfwing et al., 2002; Davis and Parashar, 2002; Head et al., 2005). Since both UCHj and UCHe implement UIPM on HTTP, then UIPM performance should be also taken into account. Moreover, dynamic loading of modules (TA, TDM) will impact system performance. Considering that the Socket Layer is the UCH core module, then it will play a decisive role from a performance point of view, since it is attending all system requests.

3.3.2 Applying SPE to UCH

UCH and related implementations comprise a complex software for which, as above described, their performance was considered critical in project. Complexity advised to carry out the evaluation from different points of view, so to allow comparison, gain insights on the products and also validate results. Therefore, it was decided that performance of current implementations should be traced both, experimentally and within a benchmark approach (Catalán and Catalán, 2010), but also it was pointed out the interest of an evaluation with formal methods, hence to be able to test the system not only in its current form but under future variations (mainly concurrent users).

Design Models

For an initial understanding and in order to determine the interactions that mostly affect system performance, we start summarizing the necessary steps to control a *target* device, see UML Sequence Diagram in Figure 3.27.

In a first step, the UCH core is initialized and then it discovers and registers connected target devices by means of TDM module. Targets are listed (UIPM) as accessible devices for users to eventually manage their services. Then UCH waits for requests from user devices. When a request arises and compatibility is checked, the UIPM module opens a session and obtains the target devices list and corresponding services or functionalities which are granted in the form of a list (UIList document), that is eventually shown in the user interface. Hitherto, the system has performed two complex processes, discovery and user interface auto-adaptation, that obviously spend a considerable amount of time and resources. However, we will leave them out of our performance study since they are executed only once, i.e., they are the equivalent to start up the system, and all we understand the need for this process and its implications. So, we assume that from now on, the user is able to control the target device (i.e., to invoke commands through `setValues` message), which also means to modify the device status and variables. Indeed, this is the normal usage of the system and it repeats as many times as invoked commands (as indicated by the loop in the diagram), besides, several concurrent users (all those initialized in the system) will be executing. Then this loop interaction turns to be the performance critical part of the system.

UML Sequence Diagram in Figure 3.28 models how a user requests a *target* by means of `setValue()` operation, i.e., it details the previous critical loop. Firstly, the User Device communicates to UCH core by means of UIPM via URC-HTTP protocol. The Socket Layer module, i.e., UCH core, connects to TA module in order to send a `setValue()` request to the Target Device. Once the request is made, the response is rendered in the User Device in an adaptive way.

The physical structure of the system is necessary to describe the resources where to allocate the modules of the architecture, as well as their connections through a network, which obviously will delay the interchange of messages among modules according to the size of the messages. A UML Deployment Diagram, Figure 3.29, will help to understand these issues.

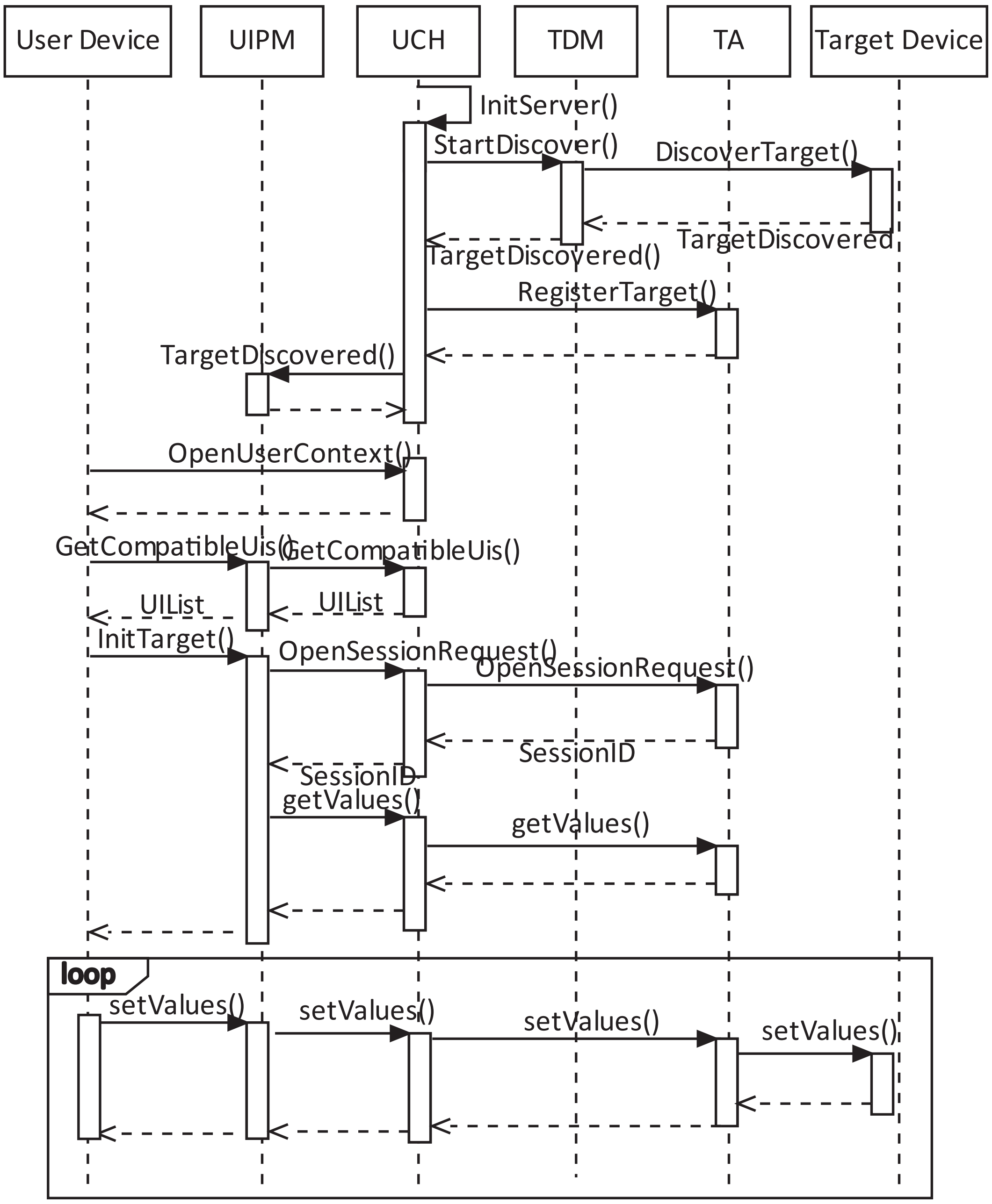
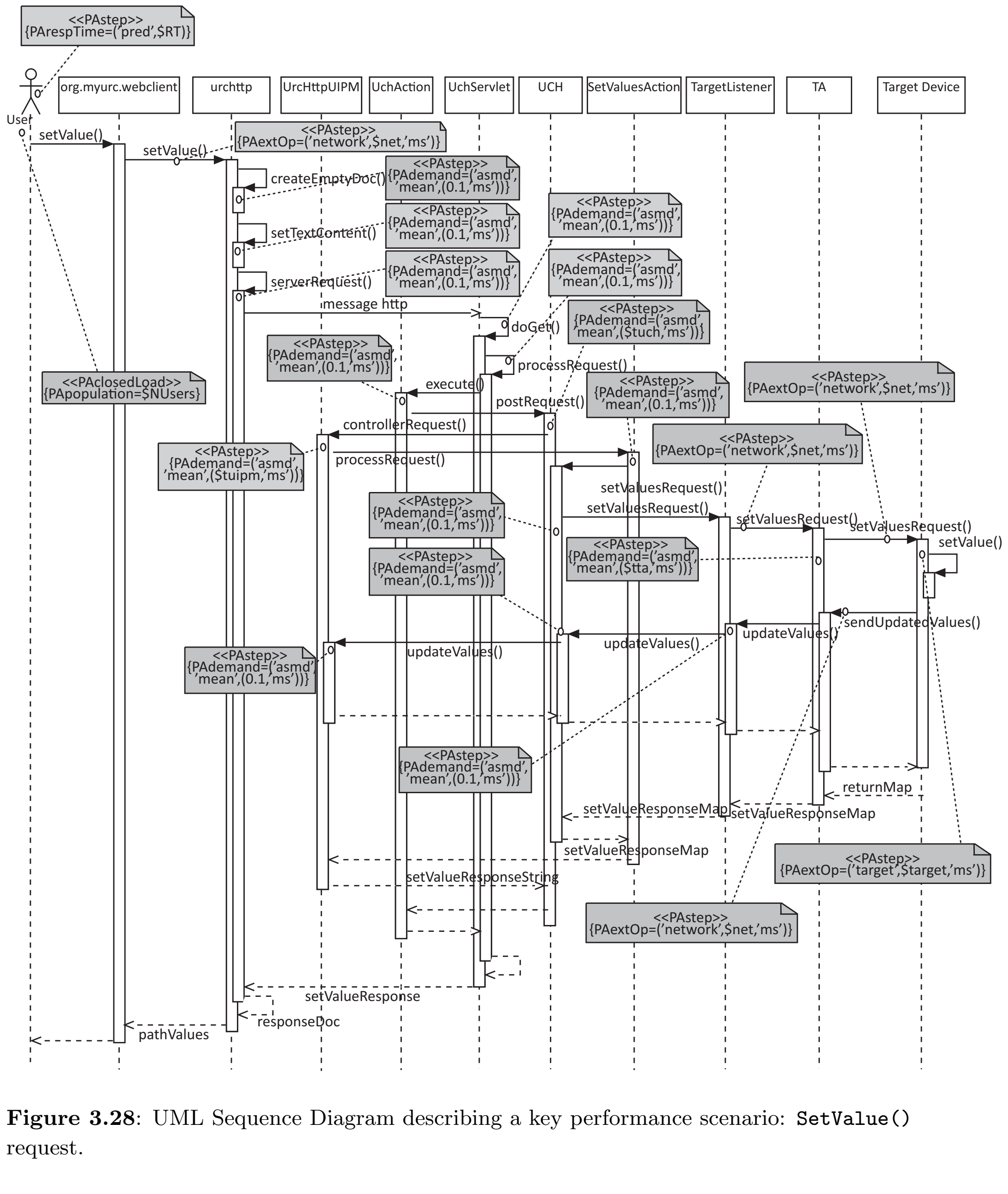


Figure 3.27: UML Sequence Diagram summarizing the target device control process.



<<PAstep>> {PArespTime=('pred',$RT)}
User
org.myurc.webclient
urchttp
UrcHttpUIPM
UchAction
UchServlet
UCH
SetValuesAction
TargetListener
TA
Target Device
setValue()
setValue()
<<PAstep>> {PAextOp=('network',$net,'ms')}
createEmptyDoc()
<<PAstep>> {PAdemand=('asmd', 'mean',(0.1,'ms'))}
setTextContent()
<<PAstep>> {PAdemand=('asmd', 'mean',(0.1,'ms'))}
serverRequest()
<<PAstep>> {PAdemand=('asmd', 'mean',(0.1,'ms'))}
<<PAstep>> {PAdemand=('asmd', 'mean',(0.1,'ms'))}
<<PAstep>> {PAdemand=('asmd', 'mean',(0.1,'ms'))}
message http
doGet()
<<PAstep>> {PAdemand=('asmd', 'mean',($tuch,'ms'))}
<<PAstep>> {PAdemand=('asmd', 'mean',(0.1,'ms'))}
processRequest()
<<PAclosedLoad>> {PApopulation=$NUsers}
execute()
<<PAstep>> {PAdemand=('asmd', 'mean',(0.1,'ms'))}
<<PAstep>> {PAextOp=('network',$net,'ms')}
postRequest()
controllerRequest()
<<PAstep>> {PAextOp=('network',$net,'ms')}
<<PAstep>> {PAdemand=('asmd', 'mean',($tuipm,'ms'))}
processRequest()
setValuesRequest()
<<PAstep>> {PAdemand=('asmd', 'mean',(0.1,'ms'))}
setValuesRequest()
setValuesRequest()
setValuesRequest()
setValue()
<<PAstep>> {PAdemand=('asmd', 'mean',($tta,'ms'))}
<<PAstep>> {PAdemand=('asmd', 'mean',(0.1,'ms'))}
sendUpdatedValues()
updateValues()
updateValues()
updateValues()
<<PAstep>> {PAdemand=('asmd', 'mean',(0.1,'ms'))}
<<PAstep>> {PAdemand=('asmd', 'mean',(0.1,'ms'))}
returnMap
setValueResponseMap
setValueResponseMap
setValueResponseMap
<<PAstep>> {PAextOp=('target',$target,'ms')}
setValueResponseString
<<PAstep>> {PAextOp=('network',$net,'ms')}
setValueResponse
responseDoc
pathValues

Figure 3.28: UML Sequence Diagram describing a key performance scenario: `SetValue()` request.

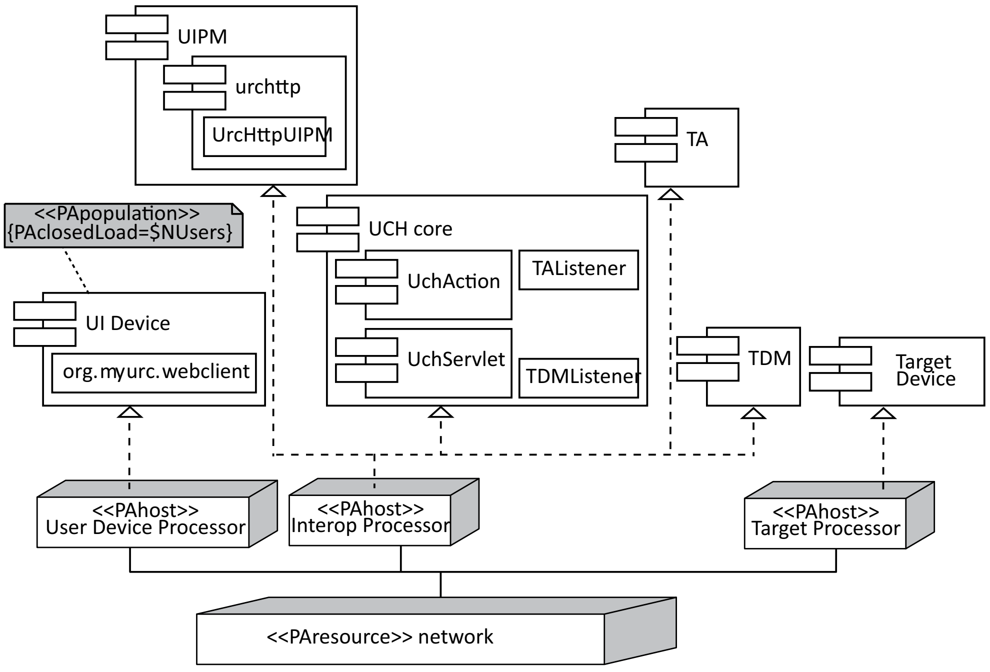


Figure 3.29: UML Deployment Diagram of UCH architecture.

Performance Parameters

Once the design has been carried out, the models of interest (Figures 3.28 and 3.29) have to be annotated as PUMA proposes, i.e., with performance information according to the UML-SPT (OMG, 2005). They will help to introduce input parameters and metrics in the eventual performance model.

Table 3.5: Mean execution time in milliseconds of UCH operations.

Parameter	UCHj	UCHe
`$tuipm`	243.05	81.96
`$tuch`	3.00	1.18
`$tta`	51.45	1.27

Table 3.5 summarizes this performance information concerning atomic actions duration collected by experimental tests, which have considered both UCHj and UCHe implementations. These actions are represented by `<<PAstep>>` stereotype, where `PAdemand` tag specifies its corresponding average execution time as an exponentially distributed random variable.

Other parameter that may affect system performance is the access to the *target* device, which is tagged by the `PAextOp` value. In the following, let us assume that this time is negligible, since it is independent of UCH architecture (e.g., the whole cycle time of a washing machine is very different from a TV set), and obviously we have to take it as an external and non-controlable part of our system.

The metric to be calculated will be the system response time, it has been annotated in the UML Sequence Diagram attached to the first message. The workload of the system is the number of users, annotated in the first life line of the UML Sequence Diagram. The rest of parameters of interest can also be seen in this diagram.

From the UML-SPT models we obtained the corresponding CSM model, depicted in Figure 3.30.

Performance Model

The next step in PUMA advises to transform the CSM into a performance model (GSPN in our case), for which we used the CSM2GSPN translator (CSM2PN, 2013) then to obtain the GSPN in Figure 3.31. The value of `$tuimp` in Table 3.5 corresponds to the duration of transition `controllerRequest` in the GSPN, as indicated by annotation attached to the same message (`controllerRequest`) in Figure 3.28.

In the same way, values for GSPN transitions `postRequest` and `setValuesRequest` correspond to variables `$tuch` and `$tta`. On the other hand, `setValue, sendUpdateValues` and `setValueResponse` are external operations whose duration is given by variables `$net` and `$target` that were set to 0.01 milliseconds. These values were taken from real experimentation, they are low because they depend on the network infrastructure that in this case was the corporate intranet. Finally, transitions `createEmptyDoc, setTextContent, serverRequest, doGet, processRequest, execute` and `updateValues` represent simple operations or calls, being their execution time around 0.01 milliseconds. The accuracy of the latter values is imposed by the system clock function.

Resources are indicated with tokens in corresponding places. So, the number of concurrent users, or system closed workload, is the number of tokens in place `users`, then matching to variable `$NUsers` in the sequence and UML Deployment Diagrams. Tokens in place *p_userDevice* represent the user device (and its corresponding user interface) and hence the concurrent threads, while places *p_uipm*, *p_uch* and *p_ta* represent the UCH modules as resources.

A first glance to the GSPN reveals that the net sequentially executes the activities once resources are acquired step by step, hence the performance will be hampered by the number of concurrent users, place `users`, and alleviated by the number of available threads, *p_userDevice*, *p_uipm*, *p_uch* and *p_ta*.

Performance Analysis

Once the performance model has been built, we used TimeNET (Zimmermann et al., 2000) in order to solve the GSPN by means of simulation techniques. Our first

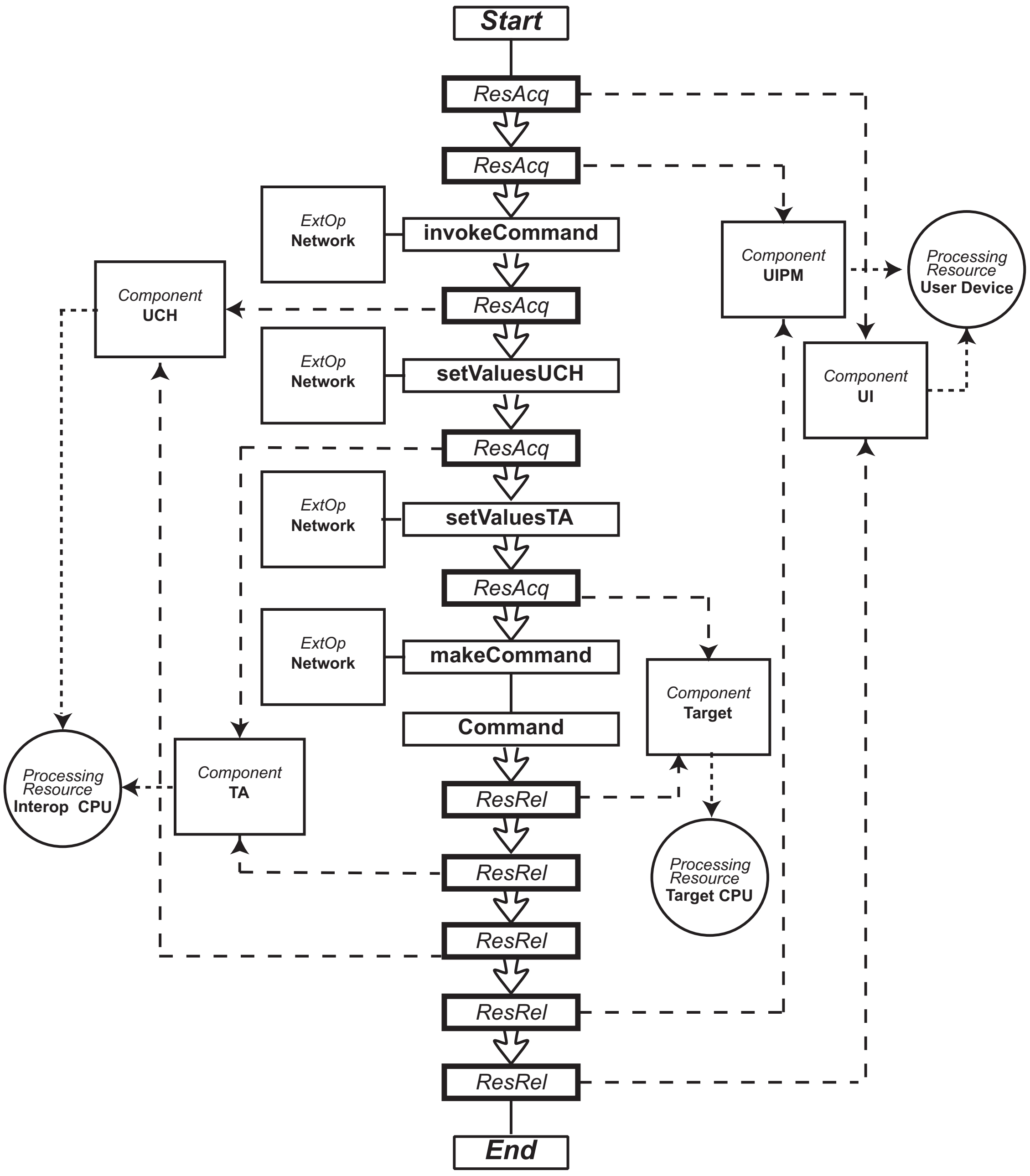


Figure 3.30: Core Scenario Model of UCH architecture.

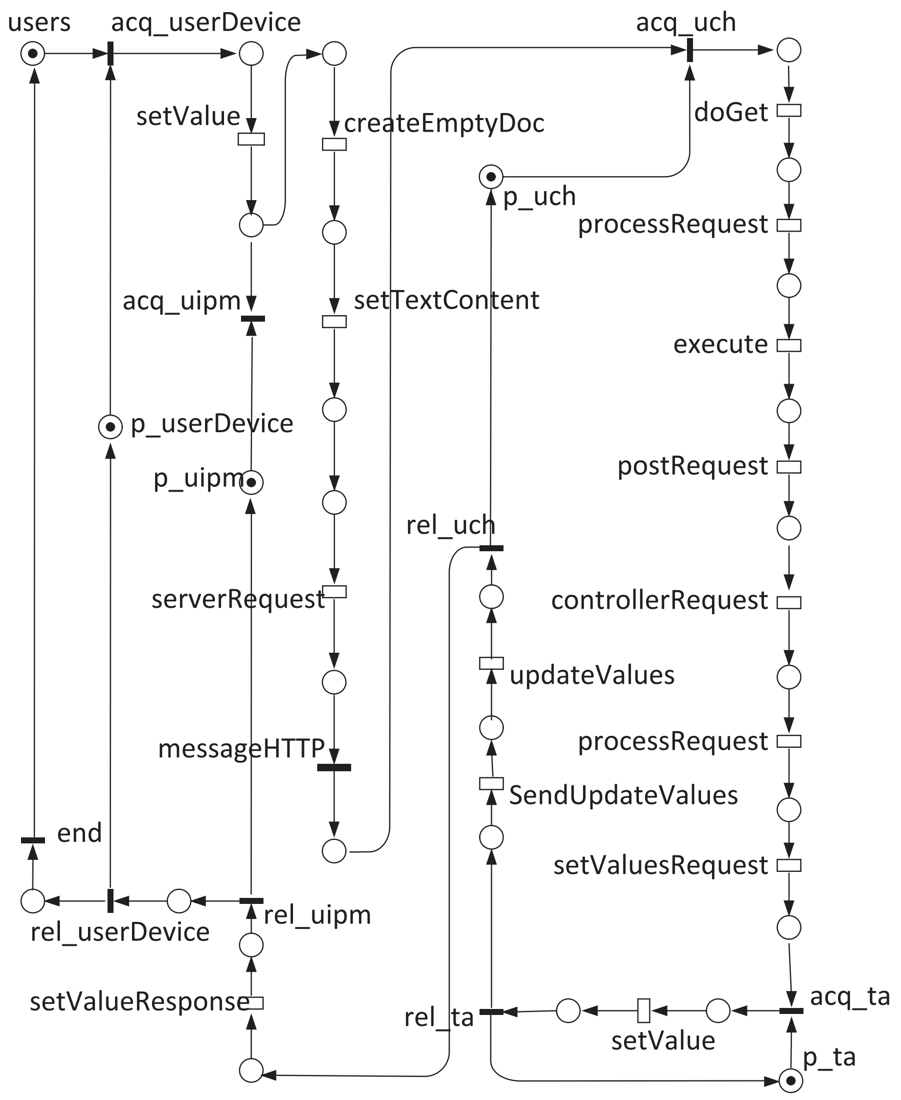


Figure 3.31: Petri net describing the key performance scenario.

analysis goal was to study UCH scalability considering the current open source implementations, UCHj and UCHe. Later, we will try to determine a system "optimal configuration" in a context with several concurrent users.

UCH was initially designed as an interoperable architecture for smart homes, which means that relatively few people will be simultaneously using the system to control different devices. Nevertheless, this architecture may be projected in more complex environments, such as intelligent buildings, business buildings, hospitals or hotels. In this case, the system will have to support requests from several concurrent users.

Firstly, both implementations, UCHj and UCHe, were experimentally tested within the INREDIS project (Catalán and Catalán, 2010; INREDIS Consortium, 2010a). These experiments assumed that each user wanted to control his/her own device, i.e., one user per *target* device. Results regarding response time (Catalán and Catalán, 2010; INREDIS Consortium, 2010a) could be hardly obtained up to forty users due to the difficulties of real experimentation. We reproduced these experiments using our performance model, which meant to put as many tokens as users in places *users* and *p_userDevice* of the GSPN, so to also match one user to one interface, and then we obtained the results in Figure 3.32. Differences in the results between our performance model and the Java and C/C++ real implementations accounted for less than a ten percent, then we assumed our GSPN as a valid performance model and ready to address experiments initially not feasible to carry out with the real implementations.

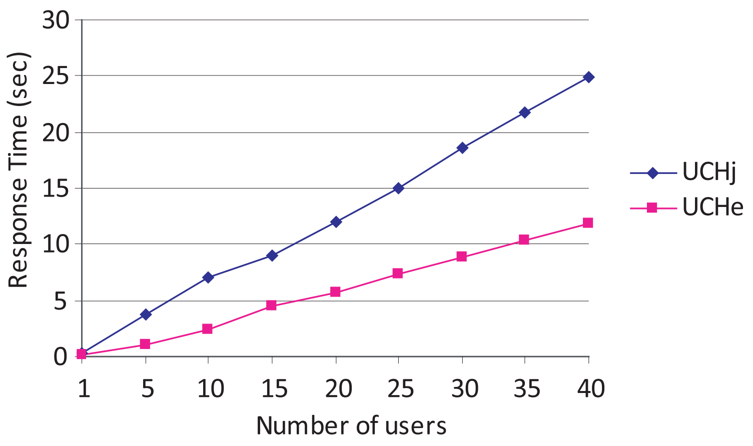


Figure 3.32: Response time of UCHj and UCHe implementations from 1 to 40 concurrent users.

On the other hand, the discussion about what could be considered a good response time is controversial, since besides the times so far considered, it may also depend on the kind of impairment the user has and on the kind of *target* device the user wants to control. For example, elderly people could request commands in their personal telecare device at a rate of few seconds. However for a blind person it could last much more time to operate for instance the washing machine. Pragmatically, we will assume quantities around ten seconds as acceptable response times, according to (Miller, 1968; Nielsen, 1993). This is so because in the experiments (both, real and GSPN) we did not want to consider the time spend by the impaired persons and neither the time to operate the target[5]. Therefore, for concrete scenarios (persons and targets with defined profiles) the response times will be higher.

Our next step, assuming valid the performance model, was to exercise the same for a larger amount of concurrent users. Figure 3.33 extends experiments in Figure 3.32 up to 1000 users, so offering response time of the GSPN w.r.t. both implementations, where we observe that UCH performs poorly, specially Java implementation. Therefore, although UCHe outperforms UCHj, UCH should not be considered as a practicable architecture in a real time environment with hundreds of concurrent users. Now, we will try to get solutions by means of replication.

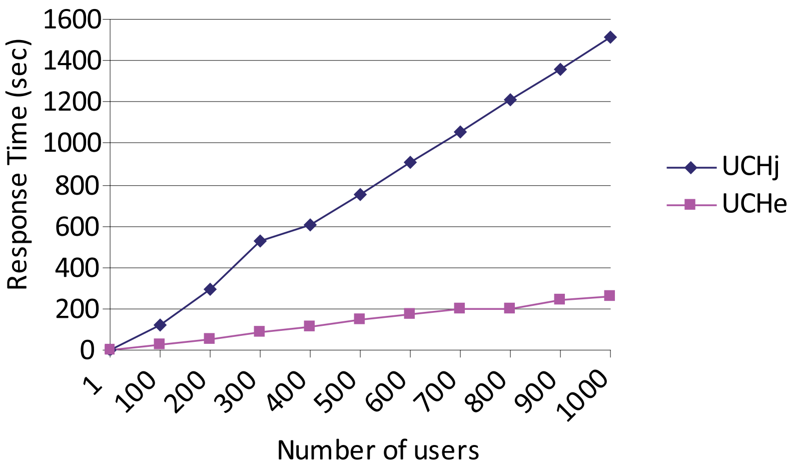


Figure 3.33: Response times of UCHj and UCHe implementations.

UCH specification does not define whether UIPMs, TDMs and TAs modules should be executed as independent processes or threads, or if they should be allocated in different memory spaces, hence these are choices for each specific implementation. In the case of both UCHj and UCHe, all UCH components execute in the same space of memory and are attended by a unique process.

Now, we want to study an "optimal configuration" for the architecture by means of modules replication. In fact, we replicated the two implementations of UCH modules, i.e., UIPM, Socket Layer and TA modules (represented by places p_uipm, p_uch, p_ta in Figure 3.31), which were populated with threads ranging from 1 to 25 in the same space of memory. Figure 3.34(a) shows the effect of adding threads in UCHj implementation and Figure 3.34(b) in UCHe. Although both graphics have similar shape, the order of magnitude is quite different. As expected, UCHe outperforms UCHj. Note that using 15 threads, the response times improve significantly in both cases, but adding more threads they do not perform better. For a few hundreds of users, UCHe may get acceptable values with 15 threads, around 8 seconds, however

[5]Note that this is not a limitation to evaluate the UCH architecture.

UCHj in these cases still is not feasible, around 50 seconds. Figure 3.35 summarizes the response times of both implementations with 15 threads.

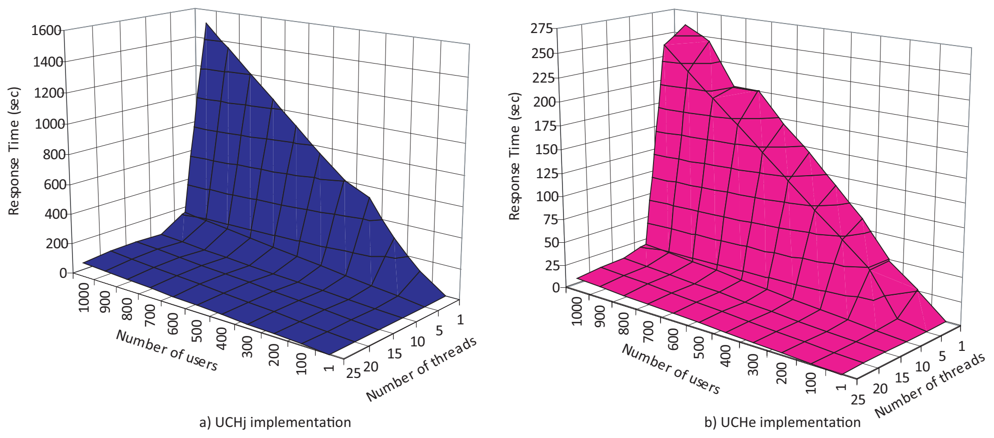


Figure 3.34: Response times of UCH implementations adding threads.

As a conclusion, a solution for an "optimal configuration" for populated environments could be a UCHe implementation of 15 threads, since as it can be observed in the graph, UCHe response times in this setting may be acceptable.

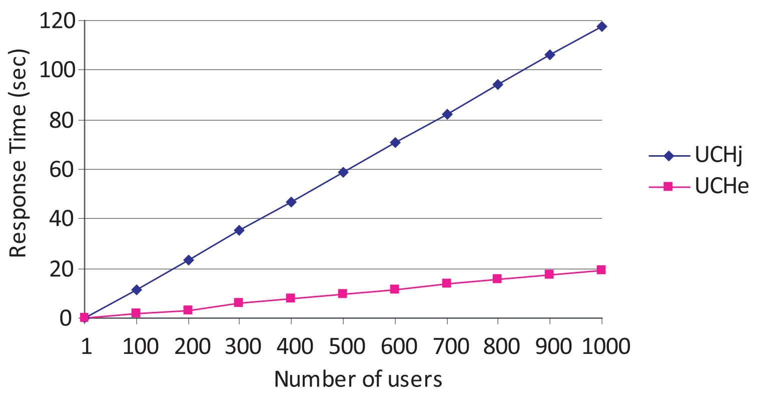


Figure 3.35: Response times of UCHj and UCHe implementations using 15 threads.

3.3.3 Concluding Remarks on UCH

We have analysed the performance of the UCH interoperable architecture through two open source implementations, UCHe and UCHj. The use of GSPNs has made possible to validate experimental results and to analyse scenarios that otherwise could not be afforded with real experimentation.

The performance results demonstrate that current UCH implementations fit in a very delimited context, with very few users. However we assessed that system performance can be improved by adding threads, but also that UCHe will always outperform UCHj, confirming that it is the best option for achieving user requirements.

We think that further analyses of the GSPN can help improving URC architecture and consequently related implementations. The solution explored, i.e., module replication, have to be supplemented with other architectural decisions that indeed we hope could be assessed by the GSPN analysis.

3.4 Conclusions

Taking our models and results as a background, the chief contribution of this chapter is the use of SPE principles, concretely the PUMA approach, in different case studies, with different architectural styles and from different domains, being one of them a real-complex case study in the industrial setting. To the best of our knowledge, PUMA has been applied in examples or academic studies, but not in an industrial setting. We remark that case studies presented along this chapter are delimited, they are studied in isolation, i.e., without taking into account interactions with other components of the complex systems which they are within.

In addition, we carried out the assessment of the performance of these software architectures in hypothetical situations, note the case of UCH where over budget hampers the benefits of the evaluation. The use of GSPNs has made possible to validate experimental results and to analyse scenarios that otherwise could not be afforded with real experimentation.

The final objective of these assessments is to gain insight in closing the "assessment loop" (Design $\rightarrow$ Performance Model $\rightarrow$ Analysis $\rightarrow$ Results $\rightarrow$ new Design). Actually, the first transitions in the loop are well-known today and even some tool support exists for them. However, very different is the case for the last one (from Results to a new Design), and our interest is to further exploit these case studies to gain insight at this regard and then to try to automate some aspects of this transition, i.e., how to automate design decisions based on analysis results.

Chapter 4

A Performance Assessment Methodology

Architecture design is a crucial part of the software development process, where decisions about which software elements will make up the system and their relationships are taken, as pointed out Williams and Smith (2002). Software architectures have emerged in the last years as the cornerstone for early evaluation of qualitative and quantitative properties of the software (QoSA, 2013). In the SPE field, architecture design is recognized as an asset for performance assessment.

In this chapter, we present a scenario-based methodology to apply SPE principles and techniques to assess software architecture at design. Our methodology has been developed in Gómez-Martínez et al. (2013a) taking the insight gained in (Gómez-Martínez and Merseguer, 2006b; Gómez-Martínez et al., 2007; Gómez-Martínez and Merseguer, 2010; Gómez-Martínez et al., 2013b) and makes use of the concepts and notations given in Chapter 2. This methodology allows to assess performance as a "by-product" of the software life-cycle.

Along this dissertation thesis, we describe the iterative process leading to develop this methodology by means of examples. In Chapter 3, we apply SPE principles to different systems: a Web-based service application, a mobile agent tracking system and an interoperability gateway. In Chapter 5, we offer advise by indicating how we actually applied the methodology in an industrial project.

The remainder of the chapter is organized as follows. Section 4.1 outlines the assessment approach. Then, Section 4.2 presents the phase of designing a system architecture. Section 4.3 explains how performance models are obtained. Section 4.4 defines performance objectives. Section 4.5 details the assessment stage, proposing alternatives to meet performance objectives and to improve software architectures. Section 4.6 revises the state of the art and places our proposal in the current scene. Finally, some conclusions are given in Section 4.7.

4.1 Overview of the Methodology

In this chapter, we resort to well-established SPE principles and techniques (Smith, 1990) for performance assessment of software architectures. The methodology we propose, outlined in Figure 4.1, loops to decide whether the proposed performance objectives are met and to obtain the architecture that can meet these objectives. The methodology is inspired by the Performance by Unified Model Analysis (PUMA) approach proposed by (Woodside et al., 2005, 2013).

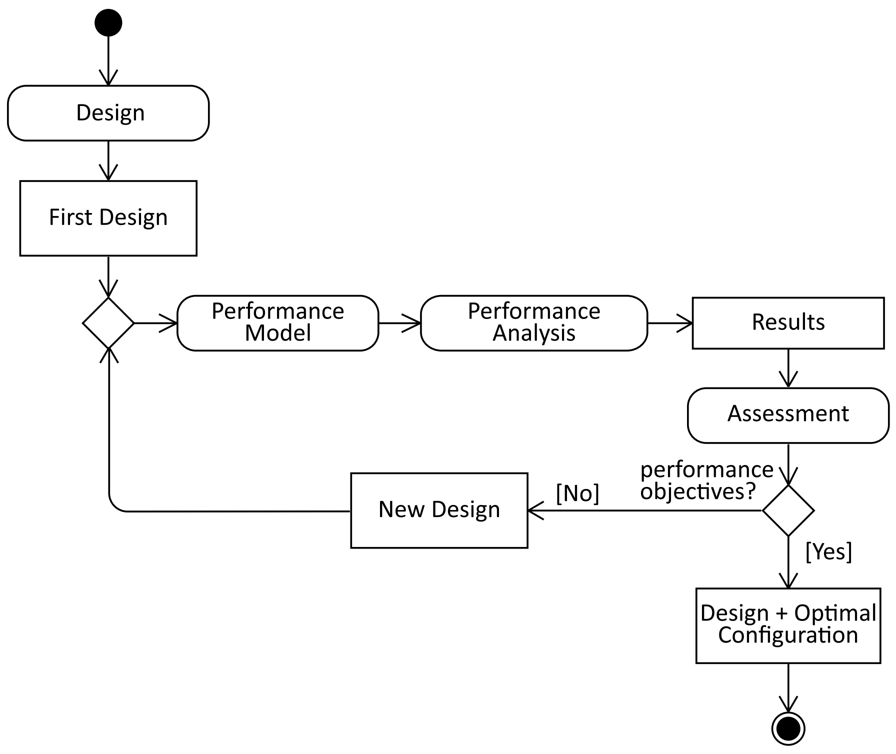


Figure 4.1: Performance assessment methodology loop

We base our work on PUMA for several reasons, among others because PUMA was developed by one of the most experienced performance evaluation groups worldwide[1], and because we have had satisfactory industrial experiences using it in the past (Gómez-Martínez et al., 2007; Gómez-Martínez and Merseguer, 2010). However, PUMA not only presents advantages, we were aware that PUMA is a methodology that demands performance expertise, at this regard we had to simplify PUMA in order to ease its application by software engineers. Unlike PUMA, which was designed for managing various formalisms in the same project, we only focus on Petri nets,

[1]`http://www.sce.carleton.ca/rads/index.html`

concretely GSPN, and we also include an assessment stage, being this phase the chief contribution of our methodology. Although we based on PUMA for the reasons above, there exist other methodologies that could have been used provided that the authors would have had previous industrial experiences with them. Among these proposals it is worth mentioning Q-ImPrESS (2009), PASA by Williams and Smith (2002) or the Palladio Component Model (PCM) by Becker et al. (2009); these methodologies are reviewed in Section 4.6.

Our methodology is then composed of four phases, which are outlined in the following paragraphs:

- *Design.* The methodology begins by modeling the system architecture using UML diagrams. We also address the behaviour of those scenarios of the system critical for performance. Finally, in the design, it is also introduced the *performance view* of the system, which identifies the scenarios that are important from a performance perspective. These scenarios are called *performance scenarios* (Smith, 1990).

- *Performance Model.* For each critical scenario, a performance model is obtained. We use Generalized Stochastic Petri Nets (GSPN) proposed by (Ajmone Marsan et al., 1995) as performance model. Section 2.4.3 offers an introduction to this formalism.

- *Performance Analysis.* According to Smith (1990) measures of interest in our work are response time, scalability and resources utilization. They are computed by analysis or simulation of the performance model. We will carry out sensitive analysis, which means to modify performance parameters to test different system configurations. Sensitive analysis is managed in the assessment step to locate performance issues (e.g., high response times or overused resources) and solutions. In fact, the assessment of these outcomes will help us to get responsive and scalable systems.

- *Assessment.* The assessment stage proposes alternatives to meet performance objectives and to improve the software architecture. Resource replication, threading and improvement of service times are the choices commonly explored. In our work, we also considered performance patterns and performance antipatterns as choices that could improve performance.

This methodology for assessing software architectures is transparent for software designers. By "transparent" we mean that the software designer should be concerned as less as possible to learn new processes since the analysis and design task already implies the use of the aforementioned methodology.

Although goal of this methodology is to apply it at architectural level, we put into practice our methodology to a running example in order to illustrate the process along this chapter. We adapt the classical in concurrency Producer-Consumer problem (Gomaa, 2000). This problem describes two processes: the Producer and the Consumer. The Producer generates a piece of data and puts it into a shared common resource, called Buffer. The Consumer is waiting to receive the data and, then, consume it. In our example, a User calls to the Producer-Consumer.

4.2 Performance-Oriented Design

The first step in our methodology is to describe a software architecture by means of UML diagrams. Thus, software engineers elaborate the UML design of a system; in particular, the architectural design with special emphasis on the behavioural view, which is of primary importance for performance assessment. Therefore, this step includes two activities: software architecture design by means of UML diagrams and performance view using performance information annotated with MARTE, a UML profile.

4.2.1 Software Design: UML diagrams

We use UML as the modeling language for software design. We mainly focus on those UML diagrams which allow us to extract performance information for the next steps of our approach: Use Case Diagram, Deployment Diagram, Interaction Overview Diagram, Sequence Diagram, Activity Diagram and State Machine Diagram. The following paragraphs explain how we use each of these diagrams. For a more extended explanation of these UML diagrams, see Section 2.2.

- **UML Use Case Diagram** (UC) is a behavioural diagram that describes the functionality of a system in a horizontal way. In our approach, UCs model *scenarios*, see Section 2.3.1, as well as the population, that is, the number of concurrent users in each scenario.
- **UML Interaction Overview Diagram** (IOD) is a special and restricted kind of UML Activity Diagram which represents the flow relationships among fragments and UML Sequence Diagrams. IODs represent high-level scenarios, as well as their population.
- **UML Deployment Diagram** (DD) identifies the system software components as well as the hardware nodes in which the former are deployed. DDs are used to have a static view of the software architecture and, for the performance perspective, to show potential communication delays and/or the number of available resources.
- **UML Sequence Diagram** (SD) shows object interactions; more specifically the messages exchanged between the system components arranged in time sequence. It provides useful constructors such as loops, alternatives or parallel execution. SDs represent the performance scenarios with time information, host demands, messages size exchanged and acquisition/release of resources.
- **UML Activity Diagram** (AD) specifies the control flow of a component, subsystem, or system. An activity represents an action in the execution of the

activity. In our methodology, ADs are employed to express execution times of actions within a specific activity, as well as the acquisition/release of concrete resources.

- **UML State Machine Diagram** (SM) describes the lifetime of objects. A state represents a time period in the life of an object during which the object satisfies some condition, performs some action or waits for an event. SMs are mainly used to get information concerning activities duration.

The software engineer decides which of these UML diagrams better express the performance design of the software architecture, as well as the level of detail, i.e., an IOD shows high-level interactions while a SM specifies object level. Nevertheless, from the performance perspective, it is sufficient to model at least one behavioural view (IOD, SD, AD and/or SM).

In the following, we document the Producer-Consumer example. The static view is represented with the DD in Figure 4.2. Thus, the Producer node deploys the Producer package; and the Consumer node, the Consumer package. Both nodes are connected through a communication node, concretely a network.

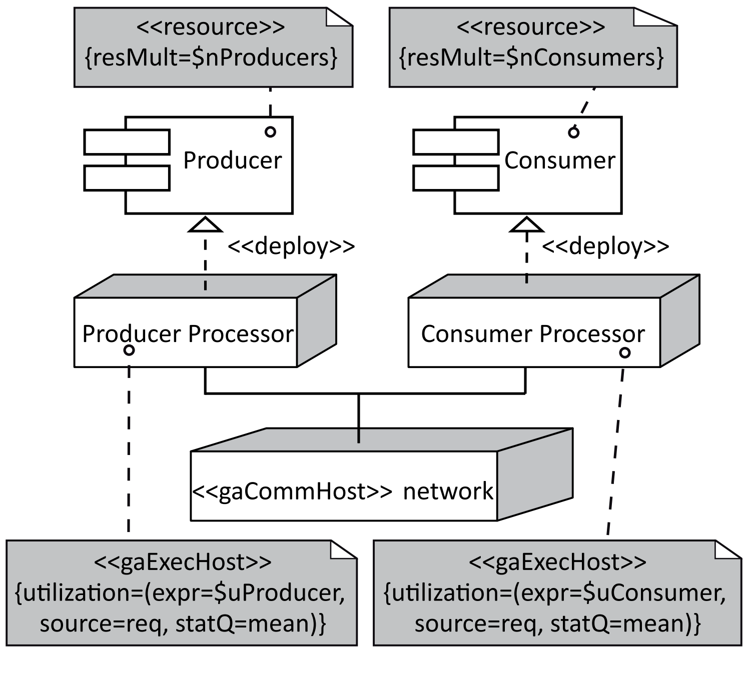


Figure 4.2: Producer-Consumer example: UML Deployment Diagram.

Figure 4.3 depicts the SM of the Producer-Consumer example. The Producer produces an item, which could be an Integer or a String with a probability of 30% and 70%, respectively. Then, it waits for a signal of `IsConsumed` from the Consumer. Meanwhile, the Consumer is waiting to get an item from the Producer to consume.

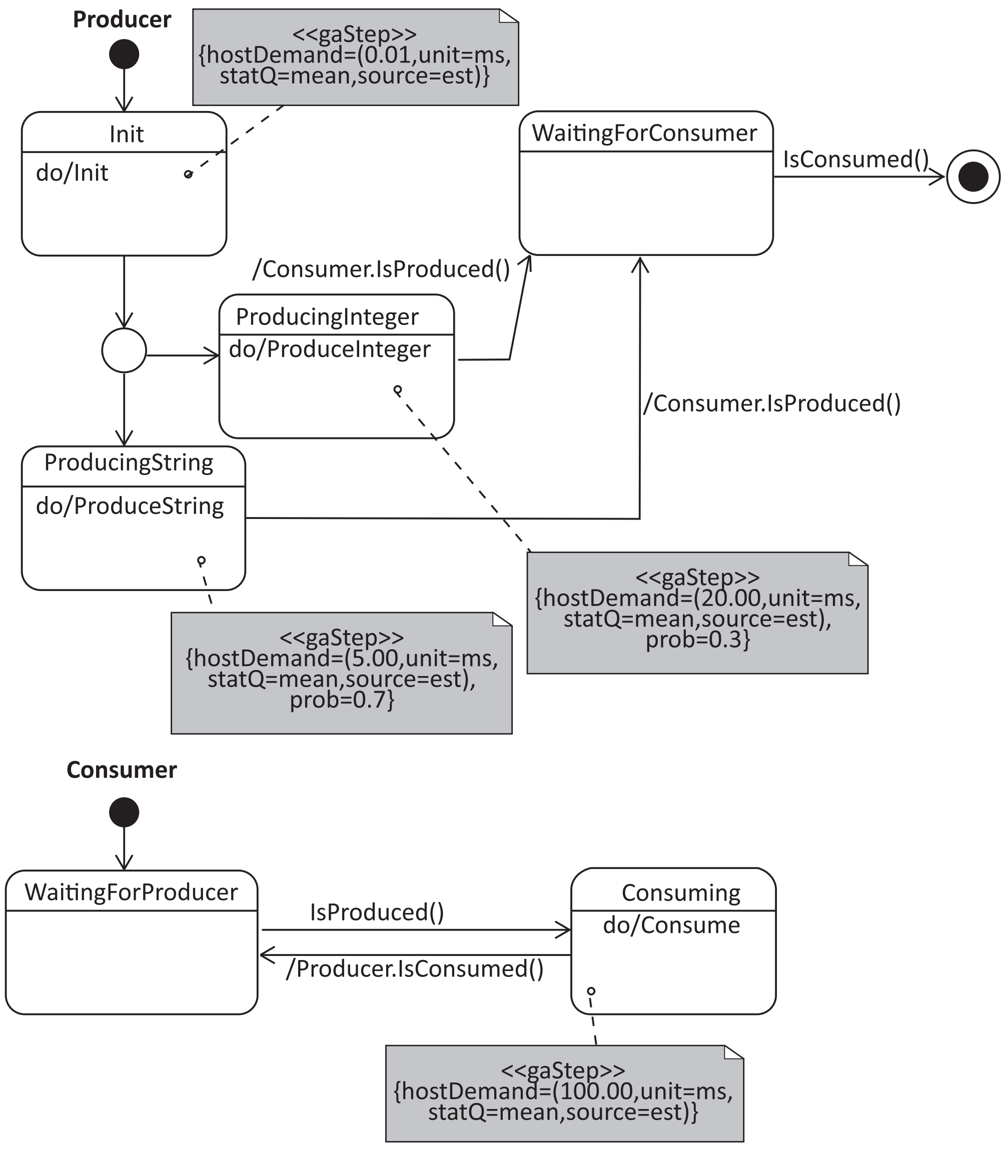
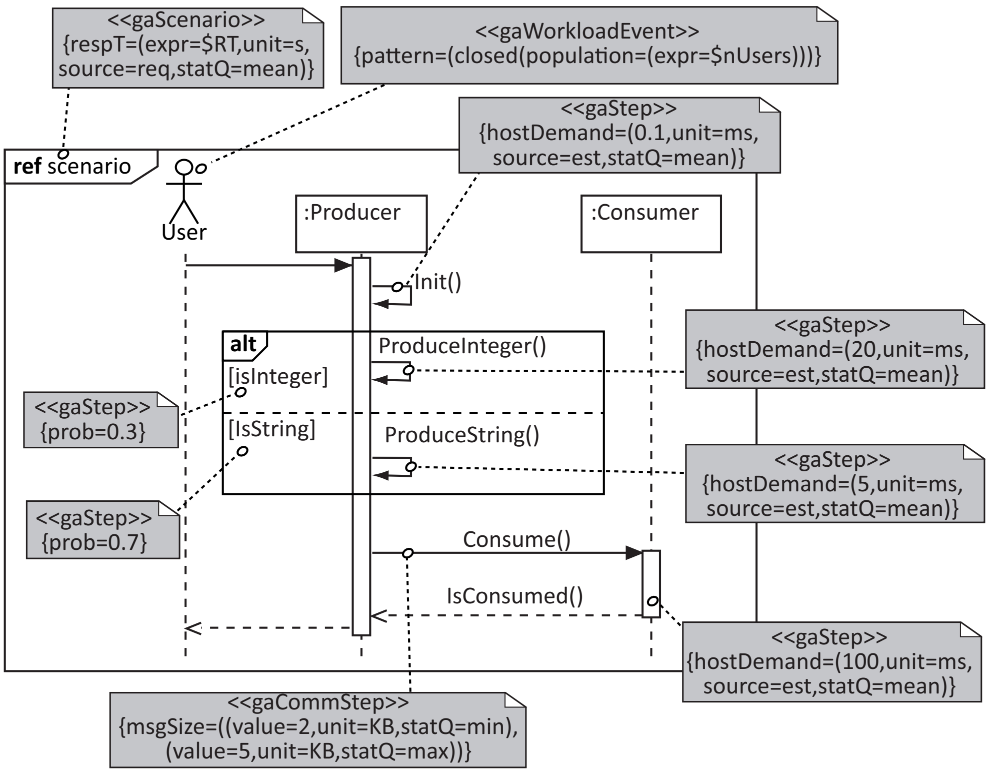


Figure 4.3: Producer-Consumer example: UML State Machine Diagram.

Figure 4.4 represents a SD of Producer-Consumer example scenario. Firstly, a User interacts with the Producer. Then, the Producer produces an item (Integer or String) as output and it sends it to the the Consumer. The Consumer is waiting for consuming and, when it receives the item, it consumes it. The Producer waits for a message communicating that the Consumer has consumed this item.



<<gaScenario>>
{respT=(expr=$RT,unit=s, source=req,statQ=mean)}
<<gaWorkloadEvent>>
{pattern=(closed(population=(expr=$nUsers)))}
<<gaStep>>
{hostDemand=(0.1,unit=ms, source=est,statQ=mean)}
ref scenario
User
:Producer
:Consumer
Init()
alt
ProduceInteger()
[isInteger]
<<gaStep>>
{hostDemand=(20,unit=ms, source=est,statQ=mean)}
<<gaStep>>
{prob=0.3}
[IsString]
ProduceString()
<<gaStep>>
{hostDemand=(5,unit=ms, source=est,statQ=mean)}
<<gaStep>>
{prob=0.7}
Consume()
IsConsumed()
<<gaStep>>
{hostDemand=(100,unit=ms, source=est,statQ=mean)}
<<gaCommStep>>
{msgSize=((value=2,unit=KB,statQ=min), (value=5,unit=KB,statQ=max))}

Figure 4.4: Producer-Consumer example: UML Sequence Diagram.

As it can be observed, diagrams in Figures 4.3 and 4.4 show a similar behavioral information, but from different perspective. Note also that UML diagrams depicted in Figures 4.2, 4.3 and 4.4 have notes, coloured in grey, which are performance annotations explained in the following section.

4.2.2 Performance View: Scenarios and Annotations

Once the system is designed and modelled by means of UML diagrams and, following SPE principles, we now introduce the *performance view* of the system, i.e., information that is relevant from performance perspective: performance scenarios and objectives and how to express this performance information.

According to Smith and Williams (2001), performance scenarios are those scenarios that are executed frequently, that are critical to the user's perception of responsiveness, or that represent a risk that performance goals might not be met. Behavioural UML diagrams, chiefly IODs and SD, augmented with performance information represent performance scenarios.

Performance objectives describe specific and measurable criteria that a software system must meet. These criteria are specified in each performance scenario using performance annotations.

The usual way in SPE for introducing a performance specification is by annotating the design diagrams. Annotations have to specify the inputs and outputs (performance measures) of the evaluation, both of which are statistical in nature. They account for properties such as workload, host demands or routing rates. As commented in Section 2.2, profiling is the mechanism UML offers to enhance a design with specifications beyond the typical structural and behavioural views.

Profiling was introduced by UML to indeed add new capabilities to the language. A UML profile is just an extension of the UML defined in terms of:

- **Stereotypes**: They are concepts in the target domain that will be added to the UML, i.e., they specify the main performance characteristics of the UML model elements.
- **Tagged values**: They specify the attributes of the stereotypes.
- **Constraints**: They are formulae that apply to stereotypes and UML elements to extend their semantics.

MARTE is an Object Management Group (OMG) standard defined using the profiling mechanism to support model-based description of real time and embedded systems, crf. Section 2.2.2. MARTE annotations capture properties, measures and requirements of interest for carrying out our performance analysis. We propose a subset of MARTE profile to annotate performance properties, namely we focus on a subset of Generic Quantitative Analysis Modeling (GQAM) and Generic Resource Modeling (GRM), depicted in Figure 4.5.

According to UML, each stereotype is made of a set of tags which define its properties. For example, *GaAcqStep* stereotype has *acqRes* and *resUnits* as tags. The former is used to specify the *Resource* that this step needs to acquire to be executed, and the latter to define the number of units of that resource that will be acquired by this step. The values assigned to tags are either basic UML types or NFP types expressed using the Value Specification Language (VSL) syntax. In particular, for complex NFP different values can be set: a value or variable name prefixed by the dollar symbol (*value* property); the origin of the NFP (*source*), e.g., a requirement (*req*), an calculated parameter (*calc*), an estimated (*est*) or measured (*mea*) value; the type of statistical measure (*statQ*), e.g., a mean or a variance. Instead, the VSL enables the specification of variables and complex expressions according to a well-defined syntax. Figure 4.6 shows the excerpts of NFP used in our approach.

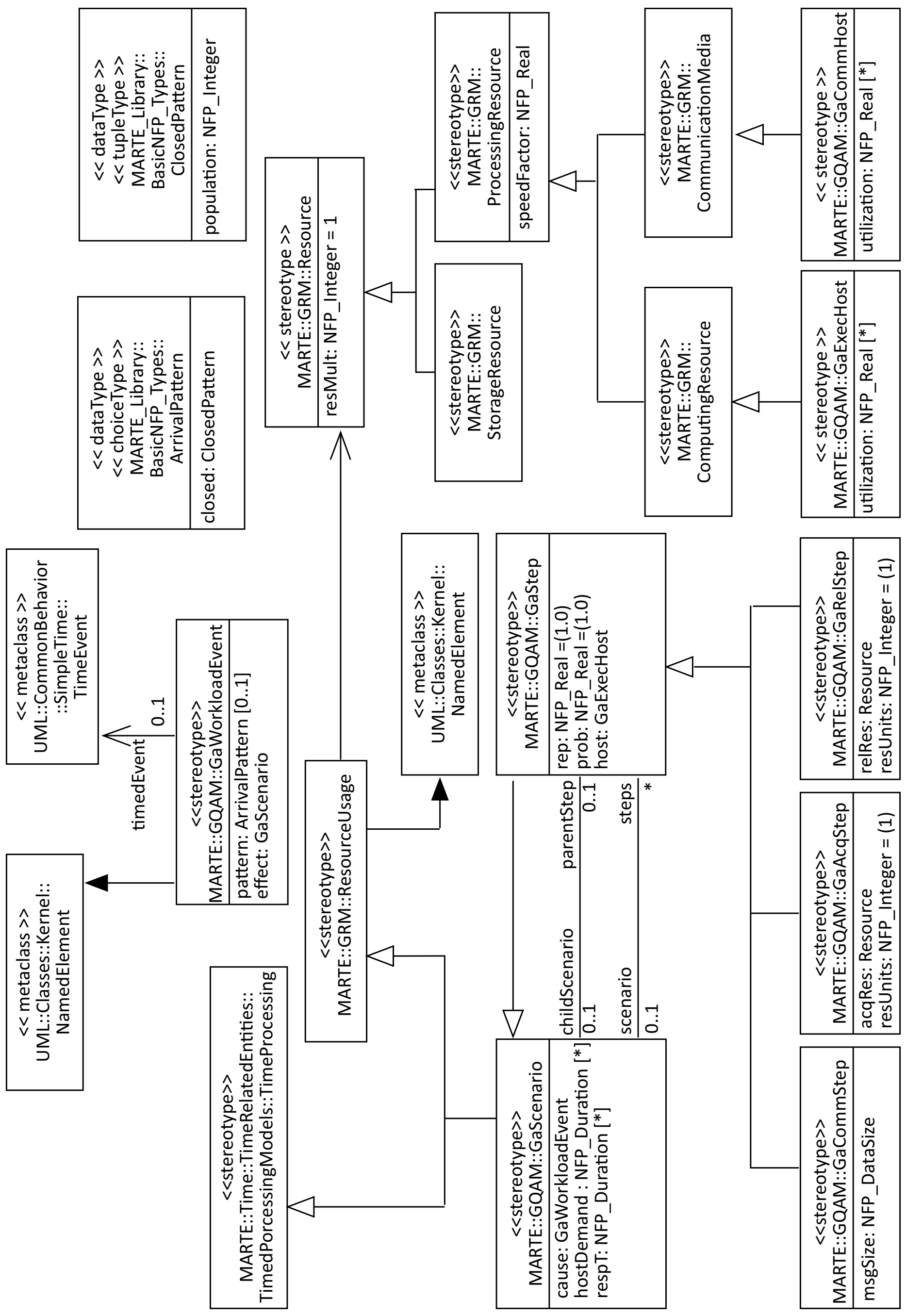
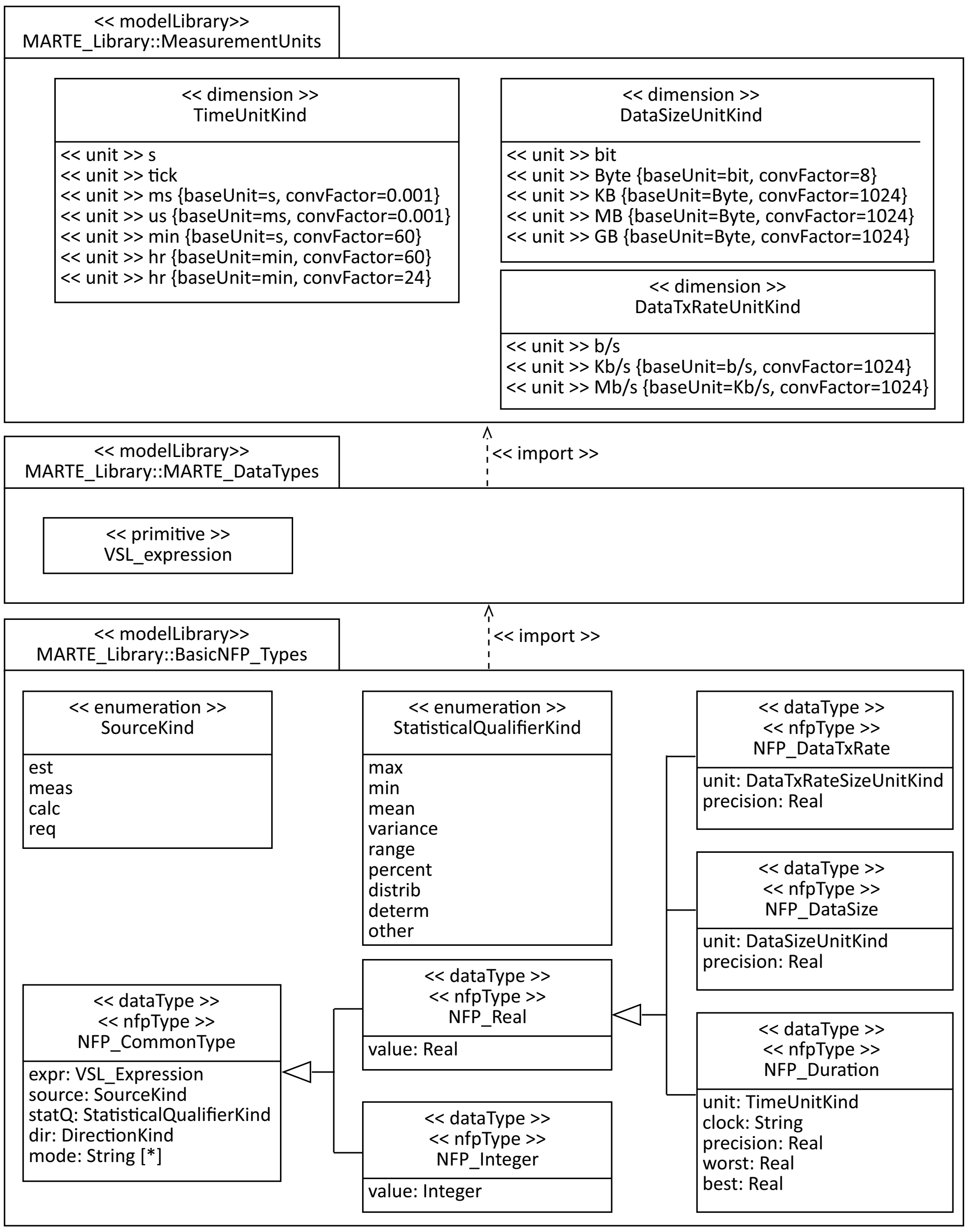


Figure 4.5: Subset of GQAM and GRM of MARTE.



<< modelLibrary>> MARTE_Library::MeasurementUnits
<< dimension >> TimeUnitKind
<< unit >> s
<< unit >> tick
<< unit >> ms {baseUnit=s, convFactor=0.001}
<< unit >> us {baseUnit=ms, convFactor=0.001}
<< unit >> min {baseUnit=s, convFactor=60}
<< unit >> hr {baseUnit=min, convFactor=60}
<< unit >> hr {baseUnit=min, convFactor=24}
<< dimension >> DataSizeUnitKind
<< unit >> bit
<< unit >> Byte {baseUnit=bit, convFactor=8}
<< unit >> KB {baseUnit=Byte, convFactor=1024}
<< unit >> MB {baseUnit=Byte, convFactor=1024}
<< unit >> GB {baseUnit=Byte, convFactor=1024}
<< dimension >> DataTxRateUnitKind
<< unit >> b/s
<< unit >> Kb/s {baseUnit=b/s, convFactor=1024}
<< unit >> Mb/s {baseUnit=Kb/s, convFactor=1024}
<< import >>
<< modelLibrary>> MARTE_Library::MARTE_DataTypes
<< primitive >> VSL_expression
<< import >>
<< modelLibrary>> MARTE_Library::BasicNFP_Types
<< enumeration >> SourceKind
est
meas
calc
req
<< enumeration >> StatisticalQualifierKind
max
min
mean
variance
range
percent
distrib
determ
other
<< dataType >> << nfpType >> NFP_DataTxRate
unit: DataTxRateSizeUnitKind
precision: Real
<< dataType >> << nfpType >> NFP_DataSize
unit: DataSizeUnitKind
precision: Real
<< dataType >> << nfpType >> NFP_Duration
unit: TimeUnitKind
clock: String
precision: Real
worst: Real
best: Real
<< dataType >> << nfpType >> NFP_Real
value: Real
<< dataType >> << nfpType >> NFP_Integer
value: Integer
<< dataType >> << nfpType >> NFP_CommonType
expr: VSL_Expression
source: SourceKind
statQ: StatisticalQualifierKind
dir: DirectionKind
mode: String [*]

Figure 4.6: Excerpt of pre-declared NFP types and measures units from MARTE.

In the following, we describe the performance annotations used by our approach. Table 4.1 summarizes these annotations. Note that ArgoSPE tool supports a subset of UML-SPT annotations, see Chapter 6.

- The **workload** is defined in behavioural and interaction diagrams, such as IOD and SD, using *GaWorkloadEvent* stereotype. We specify the number of concurrent users that populates the system by means of a closed workload with *pattern=closed* and through a variable in VSL, generally named $\$nUsers$. This variable allows to parameterize the performance model with values to carry out system sensitivity analysis. An example of this annotation can be observed in Figure 4.4: *<<gaWorkloadEvent>>* {*pattern=(closed(population=$nUsers))*}

- The **response time** is specified in behavioural diagrams, such as IOD and SD, in this case using *GaScenario* stereotype. Response time is a measure to be predicted during analysis as indicated by *source=calc*. We generally gather the result in a variable $\$RT$. The *unit* of measurement is seconds (s) or other derived unit. The *statistical measure* is a mean. Figure 4.4 offers an example of this annotation: *<<gaScenario>>* {*respT=(expr=$RT,unit=s,staQ=mean,source=calc)*}

- **Host demands** and **size of the messages** are requirements needed to compute system duration activities. They are provided by the software engineer during analysis. They are specified in behavioural diagrams, such as SD, AD and SM. Figure 4.3 and 4.4 depict several examples, as the following: *<<gaStep>>* {*hostDemand=(20,unit=ms,statQ=mean,source=est)*} In this case, *GaStep* stereotype indicates an input parameter (*source=est*) with an execution duration of 20 milliseconds and the *statistical measure* is a mean. Concerning size of the messages Figure 4.4 illustrates an example: *<<gaCommStep>>* {*msgSize=((value=2,unit=KB,statQ=min),(value=5,unit=KB,statQ=max))*}. Thus, a message with a size between 2 and 5 KB is sent.

- UML Sequence Diagrams and Activity diagrams also capture system **routing rates**, in the alternative fragments or flows. They are specified by *prob* annotation in the *GaStep* attached to the alternative fragments. For instance, in Figure 4.3, *<<gaStep>>*{*prob=0.7*}, a rate of 0.7 means a probability of 70% to execute this step (`ProduceString()`).

- System **resources** are expressed in MARTE as lifelines in the SDs or as Nodes in DDs. The resource type is specified in the UML Deployment Diagram with the GRM package; for instance *StorageResource* for specifying a storage resource, *GaExecHost* for specifying an execution host, *GaCommStep* for specifying a communication host or *Resource* for a general resource.

  The utilization of a resource is usually specified using a variable $\$U$ in VSL to be computed in the *utilization* tag, which will be predicted during analysis.

  Annotations *acqRes* and *relRes* attached to a resource specify their acquisition and release. In the case of communication resources, transmission speed for a connection or a network between communication nodes is tagged with *speedFactor*.

For specifying the number of system available resources, annotation *resMult* in the deployment is used. Figure 4.2 illustrates this annotation: *<<resource>>*{*resMult=$nProducers*} These variables allow to perform sensitive analysis parameterizing the system with different number of resources.

In addition, to allocate a class to a particular physical node the tagged value *host* in *GaStep* stereotype is needed.

As it can be observed in Figures 4.2, 4.3 and 4.4, the corresponding UML diagrams of the Producer-Consumer example are enriched with performance information.

4.3 Performance Model

According to our methodology, in this step, we need to obtain a performance model for each critical scenario that we have previously annotated.

As above mentioned, performance models are formal models that help to obtain measures of interest (e.g., system response time) by analysis or simulation. There are different kinds of performance formalisms widely accepted in SPE: queuing networks (Lazowska et al., 1984), stochastic process algebras (Hermanns et al., 2002) and stochastic Petri nets (Ajmone Marsan et al., 1995). There exist SPE methodologies that translate performance-annotated UML models into the aforementioned formalisms. For example, the work in (Petriu and Woodside, 2002) to obtain queuing networks, the work in (Tribastone and Gilmore, 2008) to obtain process algebras or (Bernardi and Merseguer, 2007; Distefano et al., 2011) to obtain Petri nets. Some of these methodologies have associated tools that automate the translation process.

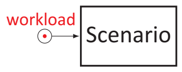
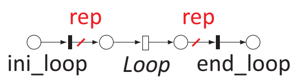
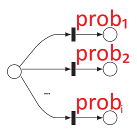
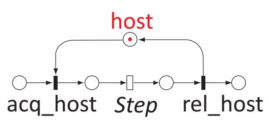
We propose to use stochastic Petri nets (SPN) and concretely generalized (GSPN). Section 2.3.2 gives a brief introduction of Petri nets formalism and some techniques of analysis. In the following, we assume that the reader is familiar with this formalism. This choice has been driven by two main factors: (i) GSPNs provide a formal notation which avoids any source of ambiguity while representing the stochastic behaviour of systems, such as their capacity to represent routing rates, competition for shared resources, stochastic duration of the host demands, parallel executions and forks and joins; (ii) GSPNs have a clear graphical notation and several tools have been developed for analysis. Moreover, the transformation from UML to GSPN can be carried out using well-established tools, such as ArgoSPE (see Chapter 6), ArgoPN (Delatour and de Lamotte, 2003) and ArgoPerformance (Distefano et al., 2011).

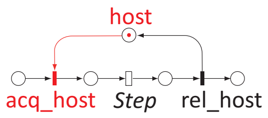
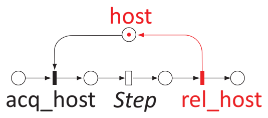
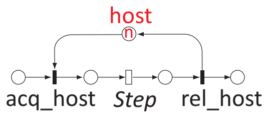
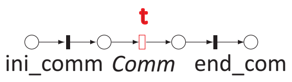
We translate UML models into GSPNs according to the algorithms proposed by Bernardi et al. (2002); Merseguer et al. (2002); López-Grao et al. (2004) and Bernardi and Merseguer (2007). Nevertheless, we introduce some changes, with respect these algorithms, needed to support our subset of the performance annotations in UML-MARTE, shown in Table 4.1. The translation rules of the performance annotations into GSPNs is summarized in Table 4.2.

Table 4.1: Subset of performance annotations in MARTE.

Stereotype	**Tag**	**Value**	**Diagram**	**Element**	**Annotation**
GaWorkload-Event	pattern	(*closed* (*population*= `NFP_Integer`))	IOD, SD, AD	Scenario	Closed population that executes a scenario.
GaScenario	respTime	`NFP_Duration`	IOD, SD, AD	Scenario	Required response time for a Scenario.
	hostDemand	`NFP_Duration`	SD, AD, SM	Scenario	The sum of all demands for all its steps, executed in the same host.
GaStep	hostDemand	`NFP_Duration`	SD, AD, SM	Step	CPU demand on the host of the process that executes the Step.
	rep	`NFP_Integer`	SD, AD, SM	Step	Repetition count for a Step or loop.
	prob	`NFP_Real`	SD, AD, SM	Step	The probability that the Step (subpath) is executed.
	host	*GaExecHost*	SD, AD, SM	Step	To deploy a step to a particular execution host.
GaCommStep	msgSize	`NFP_DataSize`	SD	Messages	Size of exchanged messages.
GaAcqStep	acqRes	*Resource*	DD	Step	Resource to be acquired within the step.
GaRelStep	relRes	*Resource*	DD	Step	Resource to be released within the step.
Resource	resMult	`NFP_Integer`	DD	Node	Number of an specific resource.
StorageResource		-	DD	Node	A storage resource.
GaExecHost	utilization	`NFP_Real`	DD	Node	The fraction of time that an execution resource is busy.
GaCommHost	utilization	`NFP_Real`	DD	Node	The fraction of time that a communication resource is busy.
	speedFactor	`NFP_DataTxRate`	DD	Node	Speed of a communication resource.

Table 4.2: Translation rules of the performance annotations in MARTE into GSPN models.

Stereotype	Tag	Annotation	GSPN
GaScenario	respTime	Required response time for a Scenario.	Variable to be computed.
	hostDemand	The sum of all demands for all its steps, executed in the same host.	Firing delay rate of a timed transitions.
GaWorkload-Event	pattern	Closed population that executes a scenario.	Number of tokens in the place which represents the initial state of each scenario. 
GaStep	hostDemand	CPU demand on the host of the process that executes the Step.	Firing rate of a timed transitions.
	rep	Repetition count for a Step or loop.	The arcs are weighted with the number of repetitions. 
	prob	Probability rate that a Step (subpath) is executed.	Routing rate of subpaths is specified by the firing rate of a immediate transitions.  being, $$\sum_i prob_i = 1$$
	host	To deploy a step to a particular execution host.	The host is acquired before the step execution and released after the completion of the step. 

Stereotype	Tag	Annotation	GSPN
GaAcqStep	acqRes	Resource to be acquired within the step.	The resource is acquired before the step execution. 
GaRelStep	relRes	Resource to be released within the step.	The resource is released after the completion of the step. 
GaCommStep	msgSize	Size of exchanged messages.	See below speedFactor stereotype.
Resource	resMult	Number of specific resources.	By populating the place representing a resource with as many tokens as resources indicated. 
GaExecHost	utilization	The fraction of time that an execution resource is busy.	Variable to be computed.
GaCommHost	utilization	The fraction of time that a communication resource is busy.	Variable to be computed.
	speedFactor	Speed of a communication resource.	The transmission of a message through a communication node is modeled with a timed transitions, being delay firing time: $t = \frac{size}{speed}$ 

Furthermore, we developed ArgoSPE, an ArgoUML (2013) plug-in which implements the aforementioned algorithms. A detailed description of ArgoSPE is found in Chapter 6.

Thus, we model the system using ArgoSPE and translate each critical scenario identified in the former step. For each critical scenario, we obtain the corresponding structure of a GSPN. Nevertheless, the translation using ArgoSPE, although automatic and, in principle, transparent for the software architect, requires some additional effort as we remark in Chapter 6, since ArgoSPE only supports a subset of performance annotations in UML-SPT. Thereby, we need to introduce some of the latter performance parameters in the GSPN models manually.

Recalling the Producer-Consumer example, if we translate the UML diagrams depicted in Figures 4.2, 4.3 and 4.4 into a GSPN according to the aforementioned algorithms and rules, we obtain the GSPN shown in Figure 4.7.

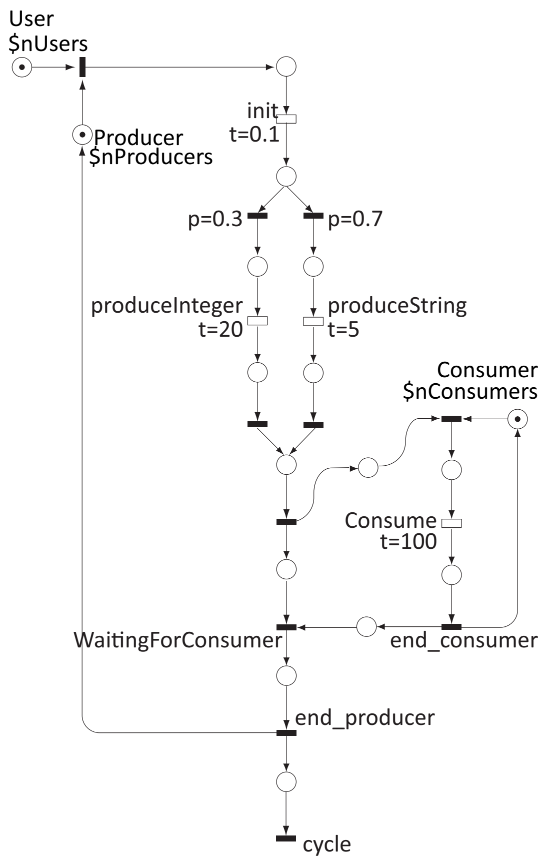


Figure 4.7: Producer-Consumer example: GSPN.

4.4 Performance Analysis

Once obtained the performance models of the key performance scenarios, the software engineer reviews the performance objectives, which were defined during the design step, and carries out the analysis of the performance model.

According to Smith and Williams (2002b), performance objectives are quantitative measures that can be computed in the performance models. We chiefly focus on: responsiveness and scalability. *Responsiveness* is the ability of a system to meet its objectives for response time (or throughput). *Scalability* is the ability of a system to continue to meet its response time as the demand for the software function increases. The response time of a software architecture is the time required to response a user request or an external request from other system. The scalability of a software architecture is the ability to support a very large amount of concurrent users or external requests.

Beyond these objectives, that are generally established by the software architect during design step, we also determine software resource *utilization* as a performance objective, due to its relation to scalability. Utilization appropriately measures the effect of software as it scales in usage (Smith and Williams, 2002b). Furthermore, in the case of a software architecture would be deployed in a "pay-per-use" cloud infrastructures, the resources utilization affects directly with the economic cost.

Although the most important task in this step is to validate that the software architecture meets the proposed performance objectives by analysing the performance models of its critical scenarios for obtaining results, previously it is needed to compute these performance objectives by means of performance measures.

Next, we discuss implications of performance objectives and how performance measures are computed in the GSPN models. Section 2.4 offers an introduction of Petri nets concepts, as well as some techniques of analysis.

Computation of Measures in the GSPN Models We compute all the measures (response time, utilization and scalability) under steady state assumption. Transient state is the probability of the system of being in state i at time t and steady state is the probability of being in state i. Therefore, steady state means that the system reaches an equilibrium. Thus, measures obtained will continue in the future, which is a more general assumption than transient state. In consequence, transient solution is more meaningful than steady state solution when the system under investigation needs to be evaluated with respect to its short-term behaviour (Bolch et al., 2001). Using steady state measures instead of transient measures could lead to substantial errors in this case.

From the software architecture point of view, steady state assumption represents a typical run execution of an application or component. Transient state could be interpreted as a snapshot of its execution. Although transient analysis can be interesting to study reliability assessment (Gokhale and Trivedi, 2002), we only focus on steady state analysis, since we propose to obtain mean response times of running software architectures.

In order to calculate the steady state, i.e., the values of the average behaviour, all the places of a GSPN must be covered by p-semiflows, and therefore it must be structurally bounded (Ross, 1983). Thus, in a GSPN, steady state analysis can be carried out when the net is cyclical.

However, the translation of a UML behavioural diagram, such as a UML Sequence Diagram, produces an acyclical GSPN. It starts with a place representing an initial state (see in Figure 4.7, place $\$np$) and ends with a transition for the last scenario message, (see in Figure 4.7, transition t_{cycle}). Therefore, we need to add an arc from this last transition to the starting place, then achieving a cyclical net and all the places are covered by p-semiflows. Observe the red arc depicted in Figure 4.8, which illustrates the GSPN with the closed cycle of the Producer-Consumer example. Now, the scenario can be analysed under steady state assumption.

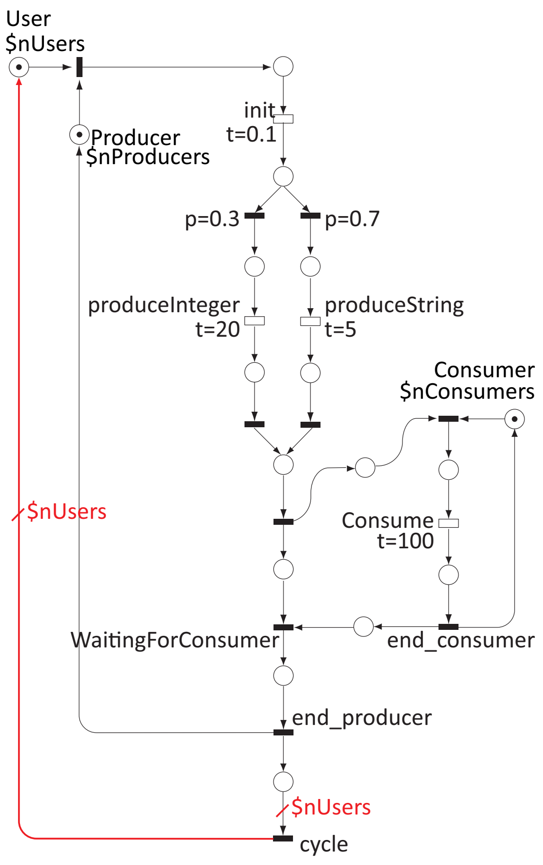


Figure 4.8: Producer-Consumer example: Closed-cycle GSPN.

In addition, since we only consider closed workloads, we assume that the entire execution cycle is finished when all the population finishes its cycle, i.e., the first token must “wait” until the last one completes its cycle. We model this behaviour by adding weight to the aforesaid arc, being this weight equals to the number of tokens that populates the initial state. Figure 4.8 illustrates this weighted arc in red (from transition cycle to place $User$).

4.4.1 Responsiveness

Responsiveness is the ability of a system to meet its objectives for response time or throughput (Smith and Williams, 2002b). The *response time* of scenario is the time required to response a user request or an external request from other system. Therefore, it is important to determine who or what interacts with each critical scenario of the software architecture under analysis. We only consider average response time of critical performance scenarios.

In software systems with Human Computer Interaction (HCI), the response time is defined from the users’s perspective as the number of seconds required to response a user request. Basic advice regarding response times has been studied by Miller (1968) and Card et al. (1991), among others. The Usability Engineering principles, proposed by Nielsen (1993), establish the following intervals:

- 0.1 second is about the limit for having the user feel that the system is reacting instantaneously.
- 1.0 second is about the limit for the user’s flow of thought to stay uninterrupted, even though the user will notice the delay.
- 10 seconds is about the limit for keeping the user’s attention focused on the dialogue. Users should be given feedback indicating when the computer expects to be done.

Furthermore, in software architectures with human interaction, it would be desirable to analyse its target audience, as we demonstrate in the case study described in Section 5.3.3. For instance, if in our target audience are people with special needs, we should study the kind of impairment the user has and the kind of devices or services they use; or if the potential users are kids, the response times should be less, since they need quick responses. Generally, in any software architecture, all the expected response times of HCI scenarios should be within these intervals.

For real-time systems, response time is the amount of time required to respond to a particular external event or the number of events that can be processed in a given time (Smith and Williams, 2002b). In these cases, the response times depend on the purpose of the system and the application domain. Consequently, software engineers must define the objective of response times in those cases.

In the presented methodology, we specify the response time of a scenario using <<*GaScenario*>> stereotype and the annotation *respT*. Since the average response time is calculated (*source=calc*), a variable $\$RT$ in VSL gathers the following computation result. The *unit* of measurement is a time unit, usually seconds s, and the *statistical measure* is a *mean*.

Computation of Responsiveness in GSPN Models The average response time RT of a scenario is calculated as the inverse of the throughput χ of the transition t_{cycle} that closes the entire execution cycle.

$$RT = \frac{1}{\chi_{t_{cycle}}}$$

As mentioned, the result of this computation is gathered in the variable $\$RT$ and translated to the UML behavioural diagram that represents the scenario.

4.4.2 Scalability

A software architecture is scalable if, when there is a significant increase in the number of resources and the number of users, it will remain (Coulouris et al., 2005). Then, the software architecture *scalability* can be defined as the ability to support a very large amount of concurrent users.

Scalability of a software architecture depends on the usage context as well as its deployment. The same software architecture can be deployed in very different environments, e.g., building automation, urban, leisure or financial. The number of concurrent users can vary considerably, even for the same kind of environment it changes by orders of magnitude, for example, in the building automation case we could have smart homes, asylums, hospitals or hotels. Considering that a software architecture has to be the same for all environments, it needs to scale accordingly. In an embedded system, the scalability is also conditioned for its own internal demand of resources and services. As such, the execution context, not only users, is of crucial importance for scaling up the system.

In consequence, the number of potential users (or internal demands) of a software architecture should be studied taking into account the context of use. If this number is large, the scalability must be analysed. Otherwise, this step can be skipped.

As mentioned previously, we model the scalability of a software architecture parameterizing the number of concurrent users that populates the system, which is specified through a variable $\$nUsers$ in VSL in the <<*GaWorkloadEvent*>> stereotype. The annotation *pattern=closed* shows that we only consider closed workloads. This population is represented in the GSPN with the initial marking of the first place of the net.

Computation of the Scalability in GSPN Models We determine the scalability of the system by calculating the response times using future workload intensities (Smith and Williams, 2002b). To study the system scalability, we compute the system response time varying the parameter that represents the population. For each population value we obtain its corresponding response time.

Figure 4.9 depicts the response times for the Producer-Consumer example with concurrent users.

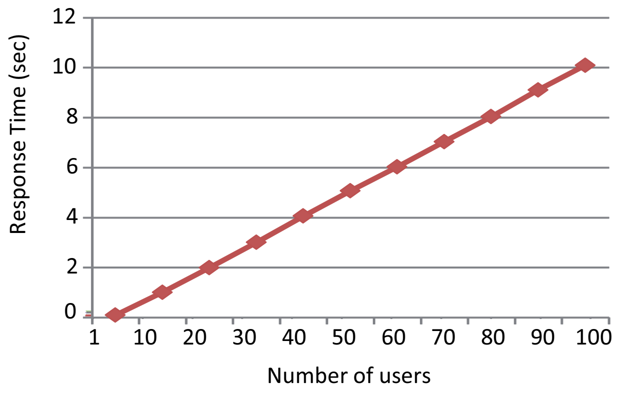


Figure 4.9: Producer-Consumer: Response times for concurrent users.

4.4.3 Utilization

Lazowska et al. (1984) defined the utilization of a resource as the proportion of time the resource is busy, or, equivalently, as the average number of customers in service.

From the SPE perspective, Smith and Williams (2002b) denoted the determination of software resource utilization to appropriately measure effect of software as it scales in usage. Therefore, resource utilization analysis detects resource saturation and potential bottlenecks when the system is highly populated and consequently, it permits to tune up the resource configuration.

The optimal resources configuration directly defines resource optimization. When resources are not well-dimensioned, it may happen that either the throughput is constrained by lack of available resources (then performance is lower than it could be), or there are idle resources (then money has been squandered) (Goldratt and Cox, 1992). In addition, the resource utilization relates with the power consumption.

Accordingly, apart from detecting bottlenecks, it is crucial an appropriate resource utilization, since it determines how to deploy the software architecture into an infrastructure. For instance, if the architecture is planned to be deployed into cloud-based infrastructure, which imply pay-per-use services. Being each new instance of a thread independently invoiced, resource utilization must be optimized. If the case of infrastructure such as datacenter, the deployment of a new software architecture directly impacts in the economic cost for the acquisition of new hardware.

As pointed out, resources are represented in the UML Sequence Diagrams by lifelines or in the UML Deployment Diagrams as nodes. The number of copies of a resource is modelled with the <<*Resource*>> stereotype annotated with *restMult*. The utilization of a resource is specified through the annotation *utilization* of the <<*GaExecHost*>> and the <<*GaCommHost*>> stereotype in the variable U, which is computed.

Computation of the Utilization in the GSPN Models According to the rules described in Table 4.2, each resource, software or hardware, is represented by a place p in the GSPN. The number of tokens M in the place p represents the available copies of that resource. Sereno and Balbo (1997) defined that the utilization of a place is given by the steady state probability that the place is non-empty. On the other hand, the average number of tokens n in steady state represents the mean occupancy of the place. Therefore, the utilization U of a place p that represents a resource can be calculated as:

$$U_p = \frac{M(p) - n(p)}{M(p)}$$

Finally, the variable U in the corresponding UML-DD gathers the utilization rate of the resource in the software architecture.

A resource is saturated when its utilization ratio is closed to 1. Nevertheless, both Lazowska et al. (1984) and Smith and Williams (2002b) advice that percentages higher than 80% should be analysed.

Figure 4.10 plots the utilizations of the Producer-Consumer example. As it can be observed, Producer resource is high saturated.

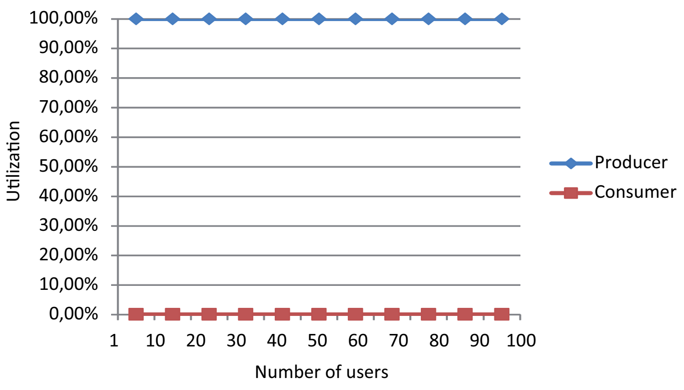


Figure 4.10: Producer-Consumer: Utilization of each resource.

4.5 Performance Assessment

In the light of the analysis results, the aim of the assessment is to introduce changes in the system for getting the best possible software architecture configuration. For each assessment iteration, we consider to apply at least: resource replication, performance patterns and performance antipatterns. Next, we describe to which extend we use these techniques.

4.5.1 Resource Replication

Utilization resource analysis detects saturated resources which would be potential system bottlenecks. In both cases, the performance of software architecture can degrade since their resource can not deal with all the request with low response time.

Firstly, we must identify potential saturated resources or bottlenecks by means of their utilization. Thus, the resource utilization rates, gathered in variable U is reviewed. As aforementioned, utilization rates higher than 80% should be analysed, according to Lazowska et al. (1984) and Smith and Williams (2002b).

We then adapt the theory proposed by Goldratt and Cox (1992): once the bottleneck is identified, the capacity of the associated resource is increased. Thus, we increase the capacity by means of resource replication. Software replication relies on multithreading to serve multiple requests in parallel. Nevertheless, this solution does not always work, since the bottleneck can be in the hardware resources, such as I/O devices or CPU capacity. In the latter case, the solution will be to add more CPU capacity (e.g., adding additional computational nodes).

We recall that resources, both software and hardware, are represented: a) in the UML Sequence Diagrams by life-lines or in the UML Deployment Diagrams as nodes, b) in the GSPN by shared places. Thus, from the architectural point of view, the resource replication means that the parameter value of the *restMult* tag in the <<*Resource*>> stereotype is increased. Then, we recalculated the response times for the performance scenario. Since the resource is modeled by a place in the GSPN, the replication is modeled in the GSPN by populating the resource place with new tokens, as same as the new value of the parameter *restMult*.

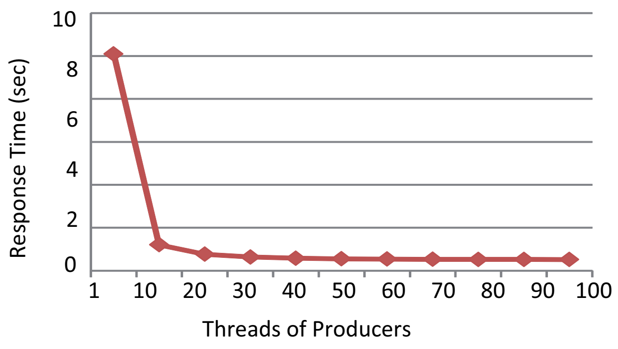


Figure 4.11: Producer-Consumer: Response times for 100 concurrent users when multithreading Producer.

In the Producer-Consumer example, the Producer resource is saturated, since its utilization is higher than 80%, as shown Figure 4.10. If we increase the number of threads, we obtain the response times in Figure 4.11. We observe that, from 20 threads on, the example does not perform better. Figure 4.12 depicts the utilization of each resource when multithreading.

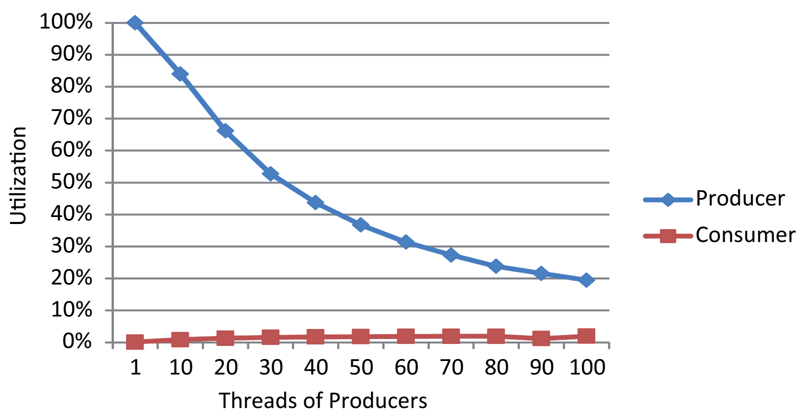


Figure 4.12: Producer-Consumer: Utilization of resources for 100 concurrent users when multithreading Producer.

4.5.2 Performance Patterns

A *pattern* (or design pattern) is a general solution to a design problem that recurs repeatedly in many different contexts (Gamma et al., 1995). Thereby, patterns provide generic solutions for many architectural, design and implementation problems. Patterns use a formal approach to describing a design problem, its proposed solution, and any other factors that might affect the problem or the solution. Therefore, patterns capture expert knowledge about "best practices" in software design. The presented methodology assumes that the software engineer is familiar with the patterns language.

Gamma et al. (1995) identified twenty three design patterns, that solve a wide range of software design problems. This patterns language includes the following types of information:

- Pattern name that describes the pattern.
- Problem to be solved by the pattern.
- Context, or settings, in which the problem occurs.
- Forces that could influence the problem or its solution.

- Solution proposed to the problem.
- Context for the solution.
- Rationale behind the solution (examples and stories of past successes or failures often go here).
- Known uses and related patterns.
- Author and date information.
- References and keywords used or searching.
- Sample code related to the solution, if it helps.

Since design patterns and architectural styles are similar (Gamma et al., 1995) in that the latter capture recurring solutions, at the level of the overall system organization, recurring solutions to common problems in structuring software systems. Therefore, performance characteristics of the style can be used to reason about the performance of that instance (Williams and Smith, 2002).

Smith and Williams (2002b) proposed *performance patterns*, which are inspired by design patterns and describe best practices for producing responsive and scalable software. Performance patterns are a higher level of abstraction than design patterns, since they are applied at design level and implementation details are not considered. Each pattern is characterized by its name, the problem description, the potential solution and the consequences to apply this pattern, as well as the principle on which is based. Performance patterns detected by Smith and Williams (2002b) are outlined in the following, other important design patterns can be found in (Grand, 1998, 2001; Lea, 1999; Schmidt et al., 2000). Table 4.3 summarizes these performance patterns.

- **Fast Path** pattern is based on the *Centering* principle proposed by Gamma et al. (1995), which tries to minimize the processing for dominant workload functions, thus *Fast Path* creates an alternative streamlined path and it also uses the *Locality Principle* by combining information likely to be needed together.
- **First Things First** pattern extends the *Centering* principle proposed by Gamma et al. (1995). This pattern prioritizes important processing tasks to ensure that they complete. When overloading conditions arise, the system gracefully degrades and improves as the conditions improve.
- **Coupling** pattern indicates that the responsibilities of each entity should be correctly assigned in order to combine information likely to be needed together. It is very effective in distributed systems to reduce the high communication costs. It is based on *Centering*, *Locality*, *Processing versus Frequency* principles.
- **Batching** pattern extends *Processing versus Frequency* principle in order to reduce the total amount of processing required of all tasks by combining frequent requests for services to save the overhead of initialization, transmission and termination processing for the request.

- **Alternate routes** pattern spreads the demand for high-usage objects spatially to reduce contention delays for the objects. It is based on the *Spread-the-Load* principle.
- **Flex Time** pattern spreads the demand for high-usage objects temporally. This pattern complements the *Alternate routes* pattern by spreading the load temporally rather than spatially. It is based on the *Spread-the-Load* principle.
- **Slender Cyclic Functions** pattern based on *Centering* principle minimizes the amount of work that must execute at regular intervals. This kind of processing is very common in embedded real-time systems.

Table 4.3: Performance patterns

Pattern	**Description**	**Principle**
Fast Path	Identify dominant workload functions and streamline the processing to do only what is necessary.	Centering
First Things First	Focus on the relative importance of processing tasks to ensure that the least important tasks will be the ones omitted if everything cannot be completed within the time available.	Centering
Coupling	Match the interface to objects with their most frequent uses.	Centering, Locality, Processing versus Frequency
Batching	Combine requests into batches so the overhead processing is executed once for the entire batch instead of for each individual item.	Processing versus Frequency
Alternate Routes	Spread the demand for high-usage objects spatially, that is, to different objects or locations.	Spread-the-Load
Flex Time	Spread the demand for high-usage objects temporally, that is, to different periods of time.	Spread-the-Load
Slender Cyclic Functions	Minimize the amount of work that must execute at regular intervals.	Centering

In the presented methodology, performance patterns are used to obtain the best design of the software architecture from the performance viewpoint. We then adapt the algorithm proposed by (Bergenti and Poggi, 2000) and applied it throughout the architecture. Figure 4.13 represents this algorithm. The pattern-detection algorithm has a sequential structure comprising four steps. Firstly, detection of classes groups that represent a pattern realization is made. Then, UML Sequence Diagrams are used to refine this detection matching with pattern-specific interactions. The third step consolidates the obtained results gathering the pattern realizations that are detected more than once, as the rules used for the detection can find a single pattern realization many times. The last step of the pattern-detection algorithm presents the obtained results to the engineer. Unlike the Bergenti and Poggi's proposal, which detects any design pattern, we only focus on performance patterns.

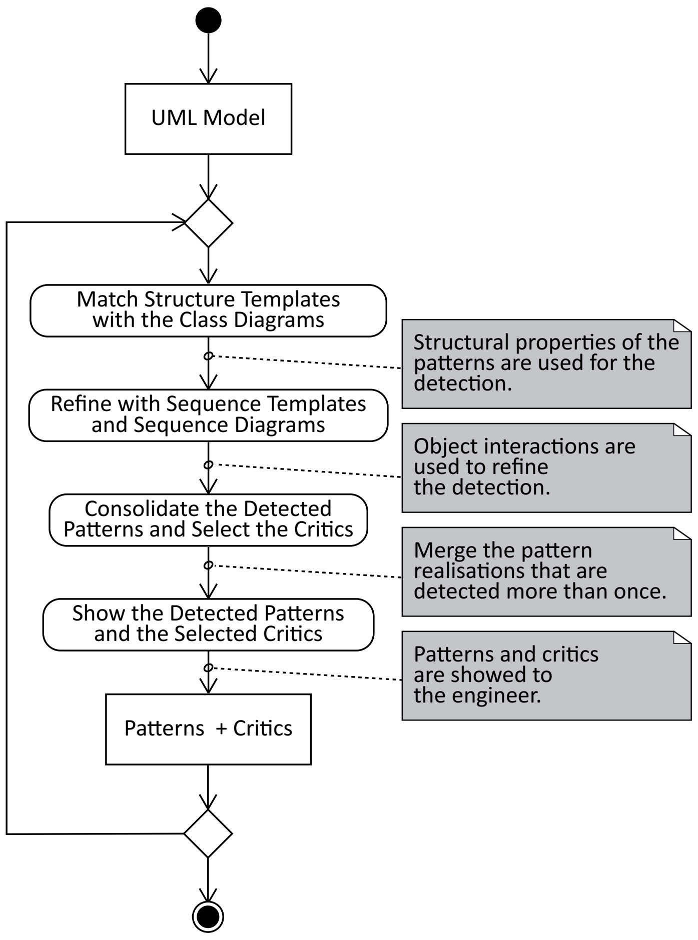

UML Model
Match Structure Templates with the Class Diagrams
Structural properties of the patterns are used for the detection.
Refine with Sequence Templates and Sequence Diagrams
Object interactions are used to refine the detection.
Consolidate the Detected Patterns and Select the Critics
Merge the pattern realisations that are detected more than once.
Show the Detected Patterns and the Selected Critics
Patterns and critics are showed to the engineer.
Patterns + Critics

Figure 4.13: Adapted pattern detection algorithm from (Bergenti and Poggi, 2000).

Nevertheless, the impact of changes made by design patterns on the design model is not always expected from the performance perspective, as remark Smith and Williams

(2002b) and Mani et al. (2011). Furthermore, some of these patterns force major changes in the overall software application, and consequently in its behaviour.

Recalling the Producer-Consumer example, which is based on the Producer/Consumer design pattern, which is based on the Master/Slave pattern in turn. We observe that no performance pattern is detected.

4.5.3 Performance Antipatterns

Antipatterns extend the notion of patterns to capture design errors and their solution (Brown et al., 1998). Their use (or misuse) produces negative consequences. Therefore, they describe common mistakes and their solutions: what to avoid and how to solve the problems. Antipatterns are *refactored* (restructured or reorganized) to overcome their negative consequences. A *refactoring* is a correctness-preserving transformation that improves the quality of software (Smith and Williams, 2002a).

Smith and Williams (2000) defined *performance antipatterns* as "bad practices" that affect software performance in a negative way. And thereby, they describe recurring software performance problems and their solution (Smith and Williams, 2003). Performance antipatterns identifies a problem, i.e., the bad practice that negatively affects the software performance, a solution, i.e., a set of refactoring actions that can carried out to remove it.

The following paragraphs summarize some of the performance antipatterns gathered and analysed in (Smith and Williams, 2000, 2001, 2002a, 2003).

- **Blob or "god" Class** (Smith and Williams, 2000) occurs when a single class or component either 1) performs all of the work of an application or 2) holds all of the applications data. Either manifestation results in excessive message traffic that can degrade performance. The solution will be to refactorize the design to distribute intelligence uniformly over the application's top-level classes, and to keep related data and behaviour together.

- **Concurrent Processing Systems** (Smith and Williams, 2002a) occurs when processing cannot make use of available processors. To solve this antipattern is necessary to restructure software of change scheduling algorithms to enable concurrent execution.

- **"Pipe and Filter" Architecture** (Smith and Williams, 2002a) occurs when the slowest filter in a "pipe and filter" architecture causes the system to have unacceptable throughput. The solution will be to break large filters into more stages and combine very small ones to reduce overhead.

- **Extensive Processing** (Smith and Williams, 2002a) occurs when extensive processing in general impedes overall response time. The solution will be to move extensive processing so that it does not impede high traffic or more important work.

- **Empty Semi Trucks** (Smith and Williams, 2003) occurs when an excessive number of request is required to perform a task. It may be due to inefficient use of available bandwidth, and inefficient interface, or both. The Batching performance pattern combines items into messages to make better use of available bandwidth is the best solution.

- **Roundtripping** (Smith and Williams, 2003) occurs when many fields in a user interface must be retrieved from a remote system. The solution will be to buffer all the calls together and make them in one trip. The Facade design pattern and the distributed command bean accomplish this buffering.

- **Tower of Babel** (Smith and Williams, 2003) occurs when processes excessively convert, parse, and translate internal data into a common exchange format such as XML. To detect this antipattern, the Fast Path performance pattern identifies paths that should be streamlined. The solution will be to minimize the conversion, parsing, and translation on those paths.

- **One-Lane Bridge** (Smith and Williams, 2000) occurs at a point in execution where only one, or a few, processes may continue to execute concurrently. Other processes are delayed while they wait for their turn. To alleviate the congestion, use the Shared Resources Principle to minimize conflicts.

- **Excessive Dynamic Allocation** (Smith and Williams, 2000) occurs by the overhead required when an application unnecessarily creates and destroys large number of objects during its execution. The two solutions are: 1) “Recycle” objects (via an object “pool”) rather than creating new ones each time they needed. 2) Use the Flyweight pattern to eliminate the need to create new objects.

- **The Ramp** (Smith and Williams, 2003) occurs when processing time increases as the system is used. The solution will be to select algorithms or data structures based on maximum size or use algorithms that adapt to the size.

- **Traffic Jam** (Smith and Williams, 2000) occurs when one problem causes a backlog of jobs that produces wide variability in response time which persists long after the problem has disappeared. To solve this antipattern is necessary to begin by eliminating the original cause of the backlog. If this is not possible, provide sufficient processing power to handle the worst-case load.

- **More is Less** (Smith and Williams, 2002a) occurs when a system spends more time “thrashing than accomplishing real work because there are too many processes relative to available resources. The solutions will be to Quantify the thresholds where thrashing occurs (using models or measurements) and determine if the architecture can meet its performance goals while staying below the thresholds.

- **Sisyphus Database Retrieval** (Dugan-Jr. et al., 2002) occurs when performing repeated queries that need only a subset of the results. The solutions will be to use advanced search techniques that only return the needed subset.

- **Falling Dominoes** (Smith and Williams, 2003) occurs when one failure causes performance failures in other components. To solve this antipattern is necessary to make sure that broken pieces are isolated until they are repaired.

- **Unnecessary Processing** (Smith and Williams, 2002a) occurs when processing is not needed or not needed at that time. The solution will be to delete the extra processing steps, reorder steps to detect unnecessary steps earlier, or restructure to delegate those steps to a background task.

As we can observe, each antipattern is characterized by its name, problem and textual solution description. Cortellessa et al. (2010, 2012) formalized this textual description, by means of logical predicates, in order to systematize their identification. Figure 4.14 illustrates the algorithm for the antipattern detection process that we adopt in our methodology.

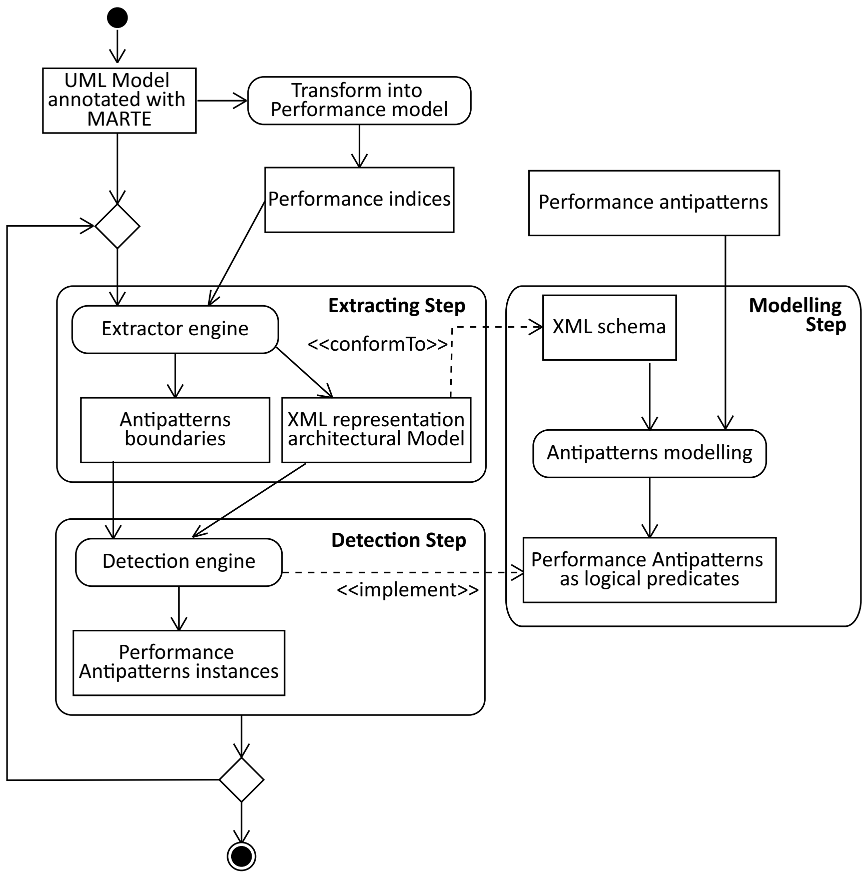


Figure 4.14: Adapted antipattern detection algorithm from (Cortellessa et al., 2012).

Cortellessa et al. (2012) thus partitioned performance antipatterns in two categories: *single-value* performance antipatterns detectable by single values of performance indices (such as mean, min or max) and *multiple-value* performance antipatterns, that require the trend or evolution of the performance indices during the time to capture performance problems. The logical representation of performance antipatterns is based on the following rationale: an antipattern identified unwanted software or hardware properties, hence an antipattern is formulated as predicate on the software architectural model elements. Thereby, an antipattern is composed of a set of basic predicates, which are first described as semi-formal natural language and then formalized by means of first-order logics; and three different views of the software architectural model elements: *Static View* (software elements), *Dynamic View* (interactions) and *Deployment View* (hardware elements). The formalization of the elements that they proposed provides only the concepts that antipatterns require to be automatically detected. The basic predicates are XML Schema elements. The detection process identifies the system elements encountered in performance antipatterns organized in XML schemas and bounded antipatterns indices in order to identify performance antipatterns in software architecture.

In addition, Cortellessa et al. built an engine to automatically detect performance antipatterns and to refactorize them by generating a feedback step at the software architecture level. The detection engine returns as output the specific model elements that participate in the occurrence of the antipattern, and if any device causes it.

We follow Cortellessa's approach to automatically identify performance antipatterns in software architectures. Like our methodology, they also model software architecture by means of UML diagrams annotated with MARTE as input. However, they transform the software architecture model to Queueing Network (QN) using PRIMA-UML tool (Cortellessa and Mirandola, 2002) in order to obtain performance measures, while we propose to use GSPN models. Moreover, we adapt their XML schemas which represent the logical predicates of the performance antipatterns, to only consider our subset of performance annotations, which slighly differs that Cortellesa's approach.

Since the Producer-Consumer example is very simple, no performance antipattern is detected.

4.5.4 Optimal Configuration

Finally, the final step in the assessment phase is the obtaining of the optimal configuration of the software architecture. In other words, we apply all the improvements proposed in the previous steps to our initial software architecture design in order to verify if our software architecture meets the performance objectives.

Once, all the proposed improvements are applied, the assessment loop of our methodology is closed when we validate that the software architecture under study meets performance objectives.

In the case of the Producer-Consumer example, the optimal configuration is the following: the increasing of the number of Producer threads.

4.6 Related Work

Software architecture assessment constitutes an important stage in the software design process, in order to guarantee non-functional requirements. Nevertheless, to the best of our knowledge, there are very few initiatives to assess architectures based on SPE principles at industrial level. An exception is the PASA (Performance Assessment of Software Architectures) method, proposed by Williams and Smith (2002).

PASA, focused on performance scenarios, is a performance-based software architecture analysis method that provides a framework for the whole assessment process. PASA inspired us in order to automatically systematize the process to detect performance issues, as well as to propose the corresponding potential solutions. As in PASA, our methodology carries out performance analysis considering responsiveness, but also resource utilization and scalability. Moreover, we have included automatic detection of performance patterns and antipatterns, by considering the work of Cortellessa et al. (2012).

Pooley and Abdullatif (2010) defined Continuous Performance Assessment of Software Architecture (CPASA). This method adapts PASA to the agile development process. To the best of our knowledge, CPASA has not been applied to an industrial case yet.

Liu et al. (2005) developed a methodology for component-based applications to predict their performance under various workloads. As in our approach patterns play an important role, but at architectural level in this case. Patterns are modeled by means of UML Activity Diagrams, while system scenarios using UML Sequence Diagrams, as in our approach. Queuing networks were used for prediction. To verify the approach they implemented different systems and established errors of prediction around 11 and 15 percent.

Basing on component-based software engineering, Becker et al. (2009) proposed a meta-model to predict extra-functional properties component-based software architectures designed with Palladio Component Model. They modeled parametric context dependencies to system resources and dependencies to parameter usages. Following this work, Huber et al. (2010) described an industrial case study where they applied the Palladio Component Model to a storage system. A model was firstly implemented, next they conducted several experiments on a prototype to derive the resource usage of each model component and finally, the model was calibrated with realistic resource demands and validated. The approach presented by Trubiani and Koziolek (2011) aimed to identify flaws in Palladio Component Models basing on performance antipatterns. They suggested design alternatives to solve these antipatterns.

The Software Engineering Institute (SEI) created some well-known scenario assessment methods: the Scenario-Based Analysis of Software Architecture (SAAM) and the Architecture Tradeoff Analysis Method (ATAM). SAAM tries to measure the software's quality through scenarios, rather than the general and inaccurate quality attributes description (Bass et al., 2005). ATAM is a method for evaluating software architectures relative to quality attribute goals (Clements et al., 2002). ATAM evaluations expose architectural risks that potentially inhibit the achievement of an organization's business goals. Nevertheless, neither of these two approaches follow SPE principles.

As above stated, PUMA (Woodside et al., 2005; Petriu et al., 2012; Woodside et al., 2013) also guided our work. PUMA is not a methodology, but a model transformation chain. It translates behavioral UML diagrams annotated with performance attributes into different formalisms using intermediate performance models, concretely Core Scenario Models (CSM). Unlike PUMA, which is focused on performance analysis, we go one step beyond taking performance results of software architectures at design phase and exploring different alternatives to meet performance objectives and, consequently, improve the system. Nevertheless, we only use GSPN as performance model.

4.7 Conclusions

In this chapter, we have presented a scenario-based methodology for assessing software architectures at design level. This methodology is based on SPE techniques and principles and inspired by the PUMA approach (Woodside et al., 2005, 2013). The use of SPE in the early phases of the life-cycle avoids the *fix-it later approach* (Smith, 1990).

This methodology closes the "assessment loop" (Design → Performance Model → Analysis → Results → new Design), since it automates some design decisions based on performance analysis results.

The proposed methodology is a part of the software development life-cycle in early stages, being the performance assessment a "by-product". Therefore, software engineer and practitioner does not need an additional effort to learn about software architecture assessment from performance viewpoint, as well as GSPNs or other performance models.

Chapter 5

Industrial Case Study: An Interoperable Architecture

Once we have gained insight in the performance assessment of software architectures by means of the case studies in Chapter 3, we have applied our proposal of the performance assessment methodology, developed in Chapter 4, to a case study.

This chapter is an *industrial experience* since we report results regarding the application of the performance assessment methodology to a real-complex industrial project, which we exhibited in (Gómez-Martínez et al., 2013a). This case study is an interoperable architecture to automatically adapt interfaces for people with disabilities. Moreover, some of the components and functionalities of the system were presented in Gómez-Martínez and Merseguer (2010); Iglesias-Pérez et al. (2010); Murua et al. (2011) and Gómez-Martínez et al. (2013b).

We have developed this chapter following recommendations from Runeson and Höst (2009) about case study research methodology for software engineering. Thus, the objective of the chapter is twofold. First, we want to describe how we applied SPE in the project, with special attention to performance patterns and antipatterns. Therefore, this chapter tries to be a blueprint for practitioners needing to evaluate performance in a software project. On the other hand, we want to assess the software architecture for performance, which is of interest not only for the project engineers but also for designers of accessible user interfaces.

The rest of the chapter contains the following sections. Firstly, Section 5.1 summarizes the industrial project and its objectives. The software architecture is outlined in Section 5.2, where chief software components are described. Section 5.3 applies the assessment approach to the aforementioned architecture. Section 5.4 discusses obtained results. Section 5.5 covers related work. Finally, Section 5.6 gives some conclusions of the chapter.

5.1 Overview

Universal Access continues being a critical quality target for Information and Communications Technology (ICT), as Stephanidis (2001) stated, especially in industrial societies where there is a growing number of people with functional diversity, including those with aging-related conditions. Indeed, ICTs may require particular skills and abilities to interact with platforms, the plethora of wireless communication systems and smart devices such as kiosks or ATMs. The inexistence of these skills and abilities extends in some cases the traditional concept of disabled people towards people with functional diversity or special needs. The growing gap between their abilities and access to ICT is called the *digital divide*. Interoperable software architectures that support universal designed user interfaces and Assistive Software (AS from now on) are two approaches to bridge this gap, e.g., (Margetis et al., 2012; Cabrera-Umpiérrez et al., 2011; Zimmermann and Vanderheiden, 2008).

The INREDIS project (INterfaces for RElations between Environment and people with DISabilities) (INREDIS Consortium, 2010b) aimed to develop environments that enable the creation of communications and interaction channels between people with some kind of special need and their context, where the targets are a set of auto-discoverable devices. More than 200 researchers from 14 Spanish companies and 19 research organizations collaborated to carry out this project during 48 months and a budget of €23.6 millions.

In the context of this project, an interoperable architecture, capable of adapting different types of interfaces to users needs and preferences, was designed and developed. In addition, we also developed an autonomous AS selection mechanism that makes the environment able to automatically select the most suitable AS according to user's profile, user's device capabilities and target device services.

Although goal of the INREDIS project was to completely develop an accessibility architecture for disabled people, here we only focus on the analysis and design steps of the project, in particular in the performance assessment carried out. The basis for the assessment is to explore the feasibility of deploying this architecture in environments with a large number of concurrent users. Early performance assessment for the system architecture is highly desirable to prevent underperformance during system deployment.

For achieving this performance assessment of the software architecture, we have followed our proposed methodology, detailed in Chapter 4, based on the principles of Software Performance Engineering (SPE) (Smith, 1990). Nevertheless, the performance assessment was intricate, due to several reasons:

- INREDIS is a very large system, various developing teams of tens of people were involved.
- Technologies were new for these teams. So, it was unknown how to capitalize these technologies for system performance maximization.
- Being the product targeted to people with special needs, performance requirements may differ from the habitual ones.

- We expected to deploy the system in various settings, most of them not yet completely defined. For example, hotels or facilities where hundreds of users could leverage the system.

Performance evaluation of software systems has been traditionally accomplished after deployment. This is the well-known *fix-it later approach* and it has well-known problems (Smith, 1990). For example, the cost of re-architecting, re-implementing and re-deploying the system when performance goals are not fulfilled. Also the over-budget for being out of schedule as Woodside et al. (2007) described. Moreover, our project had specific reasons for rejecting the fix it later approach:

- Although the operative versions of the system should be deployed at the very end of the project, we needed to deploy prototypes at the beginning of the project, for users experimentation.

- We needed to experiment with the potential environments previously referred. So, to gain some insight about their potential system performance. Otherwise, successful implementations in real deployments could not be reused in potential environments, which could imply to start a new project for each new deployment.

We have applied the methodology at software architecture design level. In fact, architecture design is a crucial part of the software design process, where decisions about which software elements will make up the system and their relationships are taken. In the SPE field, architecture design is recognized as an asset for performance assessment.

5.2 An Interoperable Architecture

The INREDIS architecture further develops the idea of Universal Control Hub (UCH) proposed by Zimmermann and Vanderheiden (2007), see Section 3.3 for its description and performance analysis. Its rationale is that a person with its adapted device (e.g., smartphone, PDA or universal controller) should be able to interact and control different devices (television, door locks, ATMs, and a long etcetera), as well as external software services. For instance, a blind person can control the washing machine (target device), by means of his/her mobile phone (controller device). The controller device allows to introduce assistive technologies to bridge the gap between the user and the target device.

The INREDIS architecture was conceived as a universal solution capable to provide disabled and elderly people with accessible and personalized interfaces according to their preferences and needs. Consequently, the architecture was designed for a general purpose context of use. Nevertheless, some running prototypes were built for different environments, covering a wide range of real world scenarios, among them leisure services (location and purchase tickets for events), smart home (Sainz et al., 2011), urban networking (Giménez et al., 2012), social networks (Murua et al., 2011), eGoverment (Alvargonzález et al., 2010) and banking services (ATMs) (Pous et al., 2012). While users with functional diversity are able to fully exploit the architecture capabilities, "any" user should be able to obtain benefits when using the system (e.g., using their mobiles as universal remote controllers in the smart environment).

The most important components of the INREDIS architecture are depicted in the UML Deployment Diagram in Figure 5.1. Recall that the grey notes in the UML diagrams are performance annotations which were explained in Chapter 4.

- **Knowledge Base** (KB in Figure 5.1). It stores ontologies and instances sets that provide formal descriptions of the elements in the INREDIS domain (e.g. user, assistive software instances, devices, software, etc.). The KB also stores the terminology and a collection of rules. It also provides mechanisms for reasoning with each of these type of knowledge and allows querying all the instances set using SPARQL (Prud'hommeaux and Seaborne, 2006).

- **Adaptive Modelling Server** (AMS in Figure 5.1). It keeps updated the KB content using information from different and heterogeneous sources (application context, user interaction logs or complex events processing).

- **Assistive Technology Server** (ATS server in Figure 5.1). It provides automatic discovery and configuration of assistive technologies, in a smart and transparent fashion reducing the existing accessibility gap that may exist between the users and their universal controller device.

- **Interface Generator** (IG in Figure 5.1). It adapts interfaces expressed in a generic and abstract language, a subset of the User Interface Markup Language (UIML) (Phanouriou, 2000), into concrete utilizable and accessible ones (implemented in XHTML (W3C, 2010)). This activity is made in terms of the user characteristics, the device capabilities and the context. All this is possible using the reasoning capabilities provided by the KB.

The main processes performed by the INREDIS architecture are pictured by the UML Interaction Overview diagram (IOD) in Figure 5.2. In the following we summarized them.

- **First Interaction.** It consists in the creation of the initial interface that acts as the access medium to the environment for the user. In the generation of such interface the system must take into consideration the relevant set of devices and services for the user (the INREDIS perimeter) and their state (without forgetting the special needs of the user). This process involves an interface generation subprocess, for building an accessible XHTML interface, and the determination of the set of assistive software instances that permit the user to interact which such interface.

- **Navigation.** Once the user has selected the device or service to interact with, the navigation process starts. Devices and services are defined by complex multi-staged interface descriptions that users can navigate. Through navigation, we

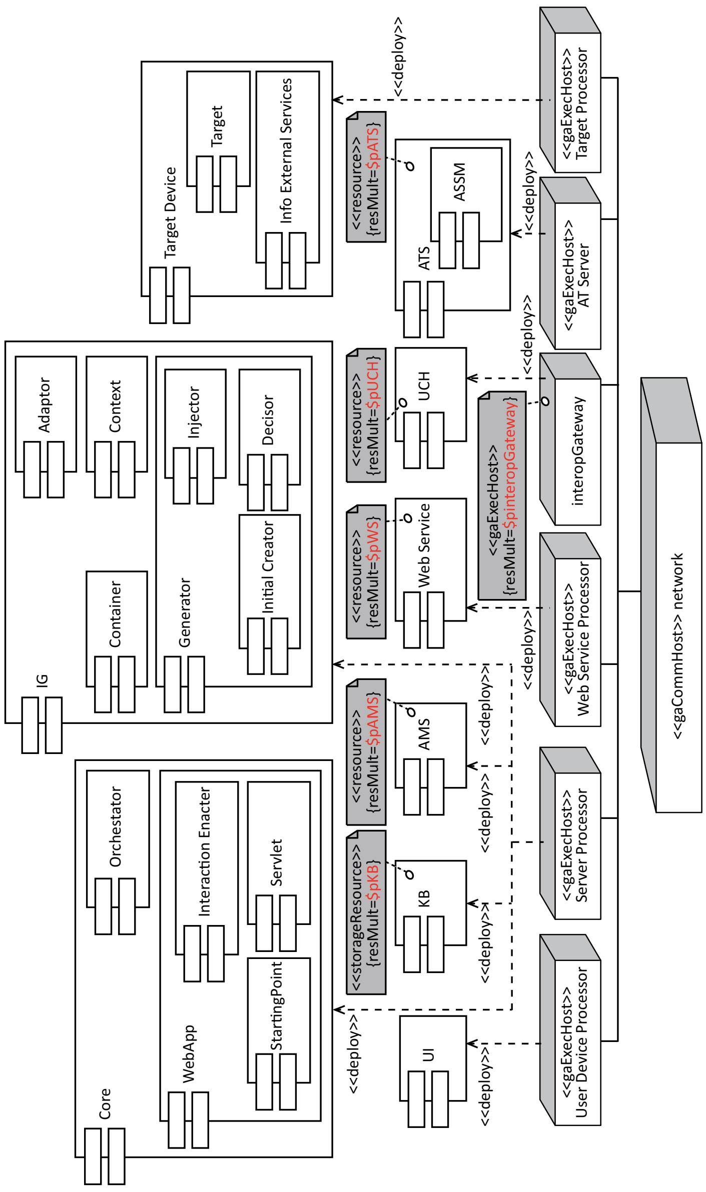


Figure 5.1: UML Deployment Diagram of the INREDIS Interoperable Architecture

simplify the information offered at a time to the user and we allow complex conversations with the device.

- **Device interaction.** When user navigation ends, or when the user performs certain actions in the device interface, interactions with the end device occur. The architecture supports interactions with devices either as a UCH Target or as a Web service transparently.
- **Back to top.** The user can at any moment reset its interaction with the device, going back to the first interface that the device offers. An updated initial interface of the device must be rendered again.

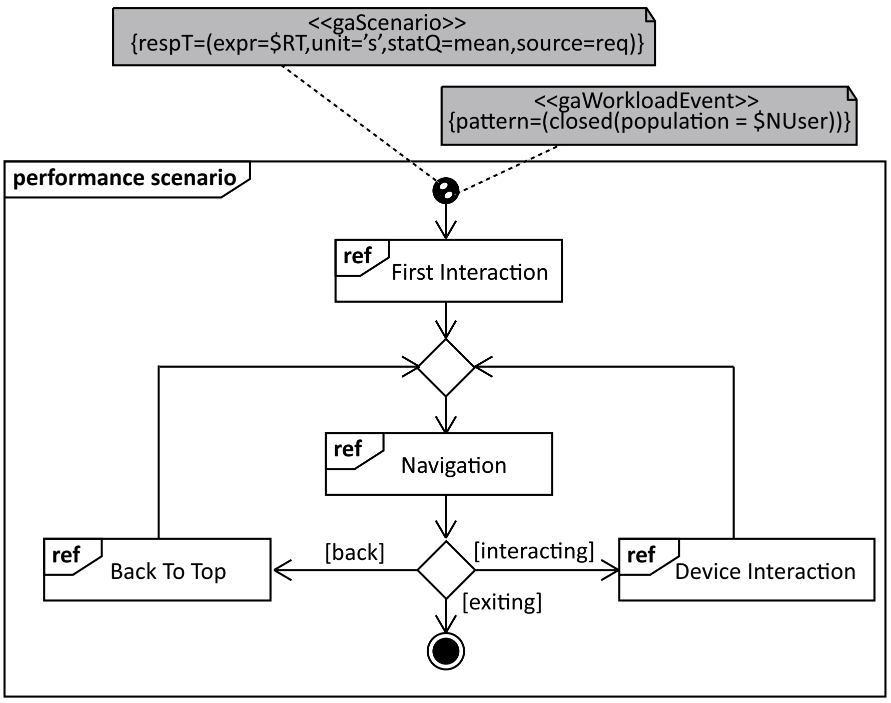


Figure 5.2: UML Interaction Overview Diagram of the main processes of the architecture

These processes can be summarized with the following example: A user wants to turn the TV at home on. Firstly, the user logins with his/her nickname and password using his/her mobile phone (*device controller*). A screenshot with the available devices and services, grouped by environment, is displayed (**First Interaction**), e.g. it appears “Smart Home”, “Products and Services” and “Health Care”, among others. These devices and services depend on the user's location. The user navigates through the screenshots until s/he identifies the device or service that s/he wants to control (**Navigation**); for instance, in the “Smart Home” display, s/he selects “TV set” (*target device*) and "Turn on/Turn off" options. S/he turns on the TV (**Device Interaction**) and waits for the notification of the new status. Finally, s/he comes back to the first screenshot in order to interact with other device or service (**Back to top**). Obviously, all the screenshots must be accessible and adapted to the specific needs and preferences of this user.

Besides of these processes, special attention deserves the Assistive Software Selection Mechanism (ASSM). The ASSM makes the environment able to automatically select the most suitable assistive technologies provided by the ATS component. It considers possible discrepancies between the user and the environment, namely in the case of functional diversity. The ASSM can work as an integrated component of the INREDIS architecture, as well as a stand alone service.

Each of these four processes and the ASSM are carefully explained in the following sections, which describe the behaviour of the system and make up the design of the INREDIS architecture. Since the design of the architecture is the cornerstone for performance evaluation, the performance engineer needs to use these diagrams in his/her work, as we later describe.

Finally, although we exclusively focus on the analysis and design stages of the INREDIS architecture, it is worth mentioning that the actual architecture implementation and the development of some target devices and services was carried out by all the INREDIS Consortium (2010b) partners cooperatively.

5.2.1 First Interaction Scenario

First Interaction, depicted in the UML Sequence Diagram (SD) in Figure 5.3, consists in the creation of the INREDIS initial interface, which acts as the access medium to the environment for the user. It lists all the available devices and services along with their current state and related information; and allows the user to select which one she wants to interact with.

Its creation involves two processes detailed in the following sections:

- The calculation of the INREDIS parameter for that concrete user (Perimeter Calculation).
- The generation of the initial interface in terms of this newly calculated perimeter (Initial Interface Generation).

Perimeter Calculation Process

The user perimeter represents the list of devices and services available to the user in a given moment. This kind of information is stored in the KB, but its calculation is made by the AMS. This module makes the necessary updates in the KB, keeping updated the situation of the user, the state of the surrounding devices, and the current state and information of the available services. Figure 5.4 shows the SD describing this process.

This task involves the following steps: First it must update the current location of the user. It starts with the `setAbsoluteLocation()` method that updates the

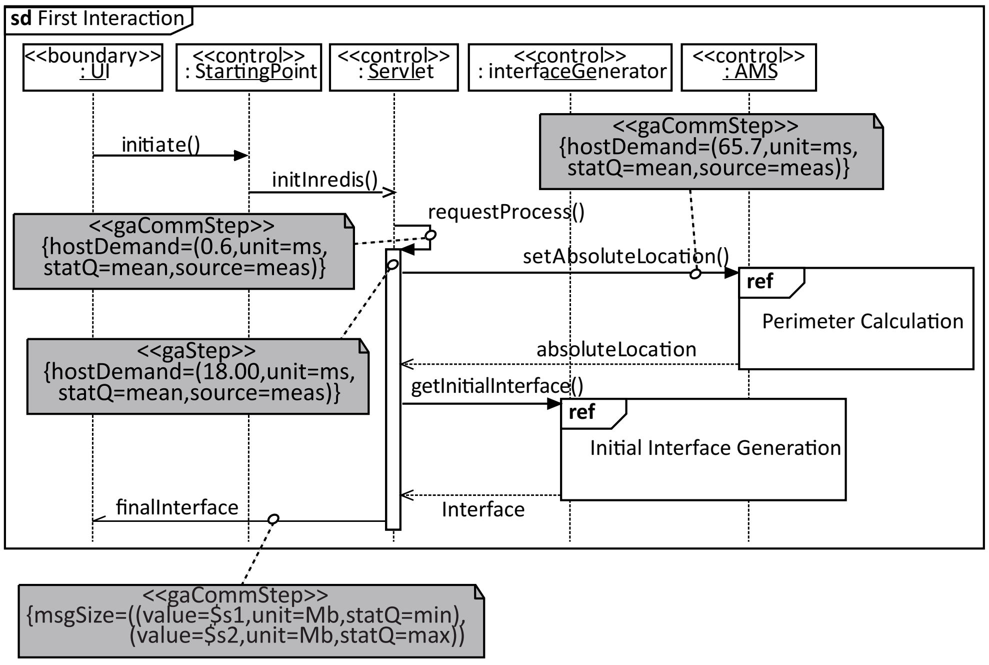

sd First Interaction
<<boundary>> : UI
<<control>> : StartingPoint
<<control>> : Servlet
<<control>> : interfaceGenerator
<<control>> : AMS
initiate()
initInredis()
requestProcess()
setAbsoluteLocation()
absoluteLocation
getInitialInterface()
Interface
finalInterface
ref Perimeter Calculation
ref Initial Interface Generation
<<gaCommStep>> {hostDemand=(65.7,unit=ms, statQ=mean,source=meas)}
<<gaCommStep>> {hostDemand=(0.6,unit=ms, statQ=mean,source=meas)}
<<gaStep>> {hostDemand=(18.00,unit=ms, statQ=mean,source=meas)}
<<gaCommStep>> {msgSize=((value=$s1,unit=Mb,statQ=min), (value=$s2,unit=Mb,statQ=max))}

Figure 5.3: UML Sequence Diagram representing the user's first interaction

information about the user in the KB. After setting the current location of the user, the AMS updates the current status of each device in the user's INREDIS perimeter. It first requests the list of device and services in the user's perimeter (the KB `getUserPerimeterServices()` method) and for each of these devices:

- It requests to the Interoperability Gateway module the current state of each device (the `getState()` method). The Interoperability Gateway obtains this information no matter whether the device is exposed as a Web service or for UCH Target in a transparent fashion.

- It updates their current state on the KB accordingly (the KB `setState()` method).

A similar process is performed for the services in the user's INREDIS parameter:

- It requests to the services in the perimeter information about their current state (the `getServiceInfo()` method).

- It updates the current state of the KB accordingly (the KB `setState()` method).

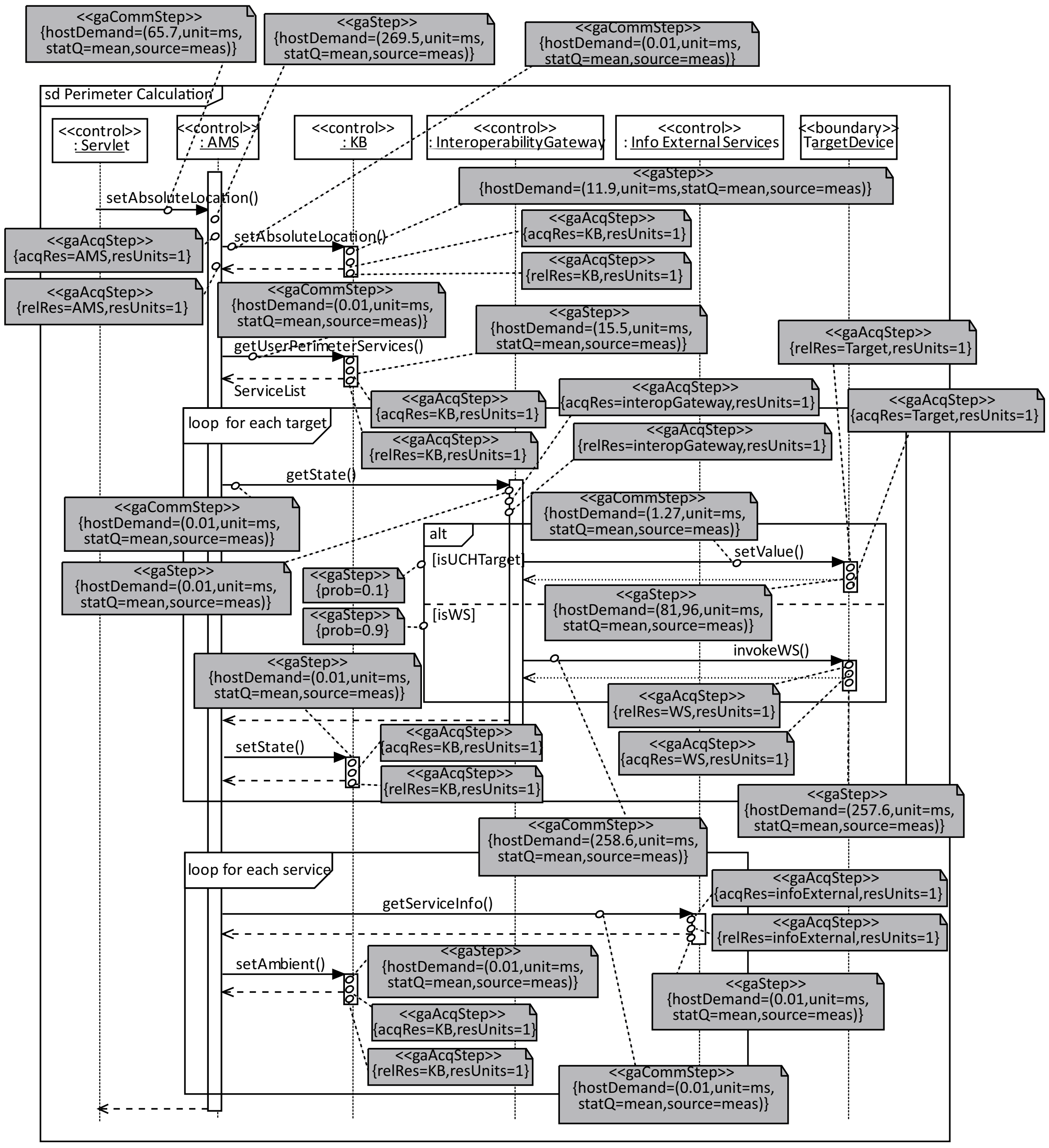

sd Perimeter Calculation
<<control>> : Servlet
<<control>> : AMS
<<control>> : KB
<<control>> : InteroperabilityGateway
<<control>> : Info External Services
<<boundary>> TargetDevice
<<gaCommStep>> {hostDemand=(65.7,unit=ms, statQ=mean,source=meas)}
<<gaStep>> {hostDemand=(269.5,unit=ms, statQ=mean,source=meas)}
<<gaCommStep>> {hostDemand=(0.01,unit=ms, statQ=mean,source=meas)}
<<gaStep>> {hostDemand=(11.9,unit=ms,statQ=mean,source=meas)}
setAbsoluteLocation()
<<gaAcqStep>> {acqRes=AMS,resUnits=1}
<<gaAcqStep>> {relRes=AMS,resUnits=1}
setAbsoluteLocation()
<<gaAcqStep>> {acqRes=KB,resUnits=1}
<<gaAcqStep>> {relRes=KB,resUnits=1}
<<gaCommStep>> {hostDemand=(0.01,unit=ms, statQ=mean,source=meas)}
<<gaStep>> {hostDemand=(15.5,unit=ms, statQ=mean,source=meas)}
<<gaAcqStep>> {relRes=Target,resUnits=1}
getUserPerimeterServices()
ServiceList
<<gaAcqStep>> {acqRes=KB,resUnits=1}
<<gaAcqStep>> {acqRes=interopGateway,resUnits=1}
<<gaAcqStep>> {acqRes=Target,resUnits=1}
loop for each target
<<gaAcqStep>> {relRes=KB,resUnits=1}
<<gaAcqStep>> {relRes=interopGateway,resUnits=1}
getState()
<<gaCommStep>> {hostDemand=(0.01,unit=ms, statQ=mean,source=meas)}
<<gaCommStep>> {hostDemand=(1.27,unit=ms, statQ=mean,source=meas)}
alt
setValue()
[isUCHTarget]
<<gaStep>> {hostDemand=(0.01,unit=ms, statQ=mean,source=meas)}
<<gaStep>> {prob=0.1}
<<gaStep>> {prob=0.9}
[isWS]
<<gaStep>> {hostDemand=(81,96,unit=ms, statQ=mean,source=meas)}
invokeWS()
<<gaStep>> {hostDemand=(0.01,unit=ms, statQ=mean,source=meas)}
<<gaAcqStep>> {relRes=WS,resUnits=1}
<<gaAcqStep>> {acqRes=WS,resUnits=1}
setState()
<<gaAcqStep>> {acqRes=KB,resUnits=1}
<<gaAcqStep>> {relRes=KB,resUnits=1}
<<gaStep>> {hostDemand=(257.6,unit=ms, statQ=mean,source=meas)}
<<gaCommStep>> {hostDemand=(258.6,unit=ms, statQ=mean,source=meas)}
loop for each service
<<gaAcqStep>> {acqRes=infoExternal,resUnits=1}
getServiceInfo()
<<gaAcqStep>> {relRes=infoExternal,resUnits=1}
setAmbient()
<<gaStep>> {hostDemand=(0.01,unit=ms, statQ=mean,source=meas)}
<<gaStep>> {hostDemand=(0.01,unit=ms, statQ=mean,source=meas)}
<<gaAcqStep>> {acqRes=KB,resUnits=1}
<<gaAcqStep>> {relRes=KB,resUnits=1}
<<gaCommStep>> {hostDemand=(0.01,unit=ms, statQ=mean,source=meas)}

Figure 5.4: UML Sequence Diagram showing the perimeter calculation process.

Initial Interface Generation Process

Once the system has ensured that the interaction is possible, the first interface is created, see Figure 5.5.

Before creating the initial interface the system has firstly to guarantee that the user is able to interact with its controller device. In consequence, it is necessary to determine the assistive technology that is necessary to enable such interaction. The ATS is the module responsible of such task; and also of determining how this software should be configured (method `AskAT()`). Using the user URI (Uniform Resource Identifier) the ATS makes queries to the KB to obtain user's profile, which is according to CAP (ISO/IEC, 2009). With such profile, the ATS creates the list of the necessary assistive technology along with its configuration. The next step is the creation of the interface generator context where the variables are stored, such as the user URI, the controller device URI, navigation graph and its variables. Now the initial interface is created. As we have stated, in order to define interfaces we use a set of UIML interfaces. The case of the initial interface is no exception, but instead of having a static UIML document, in the case of the first interaction the UIML interface definition is generated, a UIML document that contains all the available devices and services, allowing the user to choose among of them. The `Generator` module, the module that generates the UIML documents, delegates the creation of such interface to the `Initial Creator` module.

The `Initial Creator` creates what we refer as an abstract interface. It is a UIML document that still includes some context-dependent variables that have not been substituted, and a set of initialization rules that have not been performed. Such interfaces are made concrete by the Injector module (method `concretizeInterfaze()`). This module executes the initialization rules and retrieves context related values from the IG context.

Once the UIML interface has been made concrete, it is time to determine the process that transforms this UIML document into an accessible XHTML user interface. For that we use a collection of XSLT transformations that address different UIML components and users special needs, which after being applied to the UIML document translate it into an XHTML document tailored to user concrete needs. The `Decisor` module of the Interface Generator in charge of determining the set of transformations (`chooseAdaptationTransformation()` method). It does so by communicating with the KB (`getTransformations()` method) that given a user URI and the user's controller URI determines which is the proper transformation to be applied. The selection of this transformation takes into account many orthogonal aspects, such as user's special needs, preferences and the controller interaction capabilities, see (González-Cabero, 2010).

Once the concrete UIML interface has been generated and the proper XSLT transformations have been selected, there are a set of parameters that are needed to tailor the transformations. We call them the adaptation parameters, and they include the final interface language and other lower-level implementation topics. They are determined in an analogous manner to what we did for selecting the XSLT transformation.

The Decisor module (`chooseAdaptationParameters()` method) gathers such in-

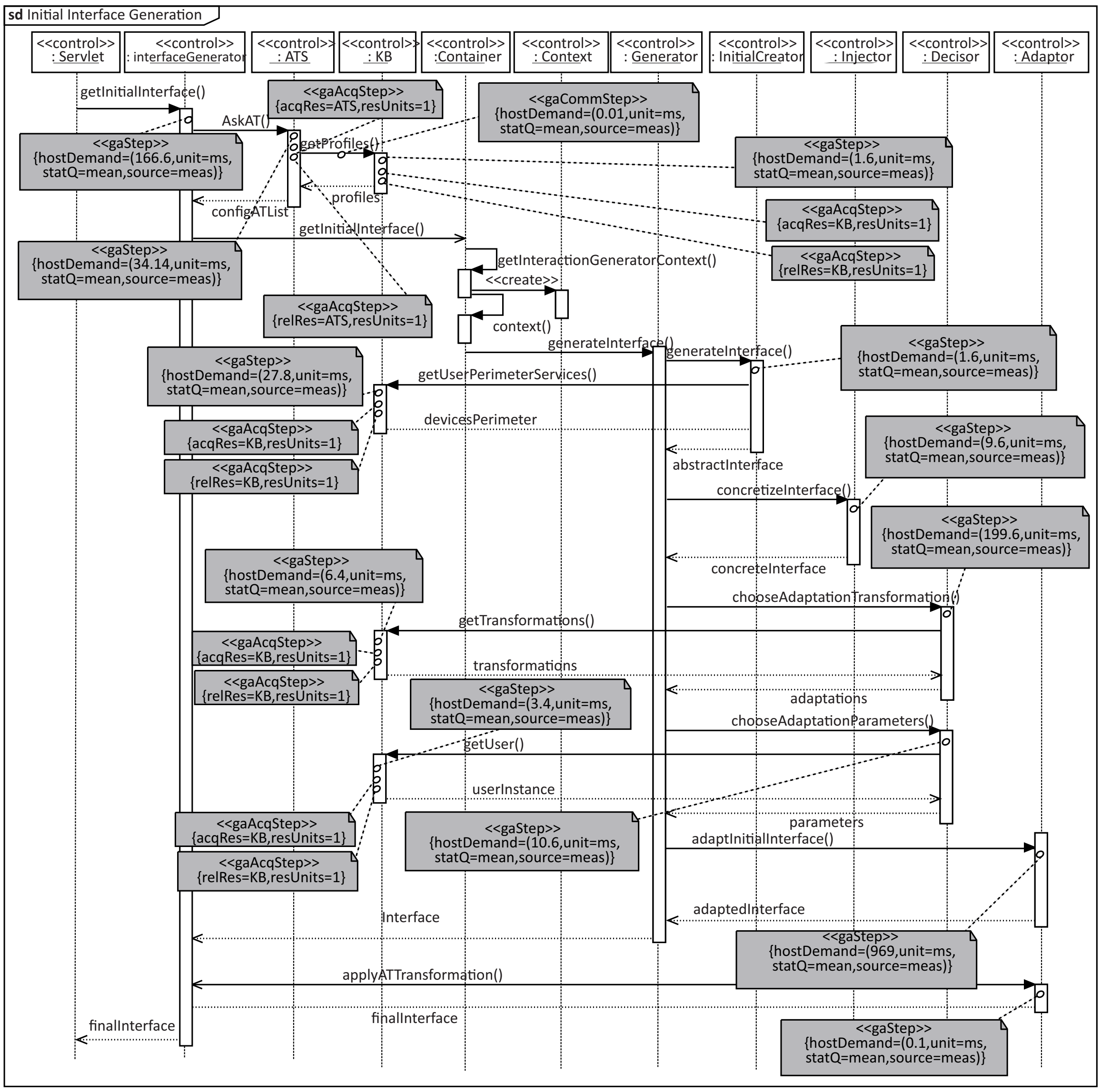

sd Initial Interface Generation
<<control>> : Servlet
<<control>> : interfaceGenerator
<<control>> : ATS
<<control>> : KB
<<control>> :Container
<<control>> : Context
<<control>> : Generator
<<control>> : InitialCreator
<<control>> : Injector
<<control>> : Decisor
<<control>> : Adaptor
getInitialInterface()
AskAT()
getProfiles()
profiles
configATList
getInitialInterface()
getInteractionGeneratorContext()
<<create>>
context()
generateInterface()
generateInterface()
getUserPerimeterServices()
devicesPerimeter
abstractInterface
concretizeInterface()
concreteInterface
chooseAdaptationTransformation()
getTransformations()
transformations
adaptations
chooseAdaptationParameters()
getUser()
userInstance
parameters
adaptInitialInterface()
adaptedInterface
Interface
applyATTransformation()
finalInterface
finalInterface
<<gaAcqStep>> {acqRes=ATS,resUnits=1}
<<gaCommStep>> {hostDemand=(0.01,unit=ms, statQ=mean,source=meas)}
<<gaStep>> {hostDemand=(166.6,unit=ms, statQ=mean,source=meas)}
<<gaStep>> {hostDemand=(1.6,unit=ms, statQ=mean,source=meas)}
<<gaAcqStep>> {acqRes=KB,resUnits=1}
<<gaAcqStep>> {relRes=KB,resUnits=1}
<<gaStep>> {hostDemand=(34.14,unit=ms, statQ=mean,source=meas)}
<<gaAcqStep>> {relRes=ATS,resUnits=1}
<<gaStep>> {hostDemand=(1.6,unit=ms, statQ=mean,source=meas)}
<<gaStep>> {hostDemand=(27.8,unit=ms, statQ=mean,source=meas)}
<<gaAcqStep>> {acqRes=KB,resUnits=1}
<<gaAcqStep>> {relRes=KB,resUnits=1}
<<gaStep>> {hostDemand=(9.6,unit=ms, statQ=mean,source=meas)}
<<gaStep>> {hostDemand=(199.6,unit=ms, statQ=mean,source=meas)}
<<gaStep>> {hostDemand=(6.4,unit=ms, statQ=mean,source=meas)}
<<gaAcqStep>> {acqRes=KB,resUnits=1}
<<gaAcqStep>> {relRes=KB,resUnits=1}
<<gaStep>> {hostDemand=(3.4,unit=ms, statQ=mean,source=meas)}
<<gaAcqStep>> {acqRes=KB,resUnits=1}
<<gaAcqStep>> {relRes=KB,resUnits=1}
<<gaStep>> {hostDemand=(10.6,unit=ms, statQ=mean,source=meas)}
<<gaStep>> {hostDemand=(969,unit=ms, statQ=mean,source=meas)}
<<gaStep>> {hostDemand=(0.1,unit=ms, statQ=mean,source=meas)}

Figure 5.5: UML Sequence Diagram modeling the Initial Interface Generation process.

formation asking to the KB for information about the user, and it also takes into account information contained in the IG context. Once the IG posses all the necessary information (i.e., the initial interface as a concrete UIML interface, the set of transformations, and the adaptation parameters) it invokes the `adaptInitialInterface()` method of the Adaptor module. It returns the XHTML document that represents the initial user interface.

Finally, there may be some transformations that must be applied to the initial interface XHTML document that stem from the set of assistive software provided and configured by the ATS. They are applied by the `Adaptor` module (the `applyATTransformation()` method). After these transformations have been applied, the final version of the interface has been created and can be delivered to the user's controller device.

5.2.2 Navigation Scenario

As we have already stated users interaction with a device often implies navigating through different atomic interfaces. Figure 5.6 shows the SD of the Navigation process.

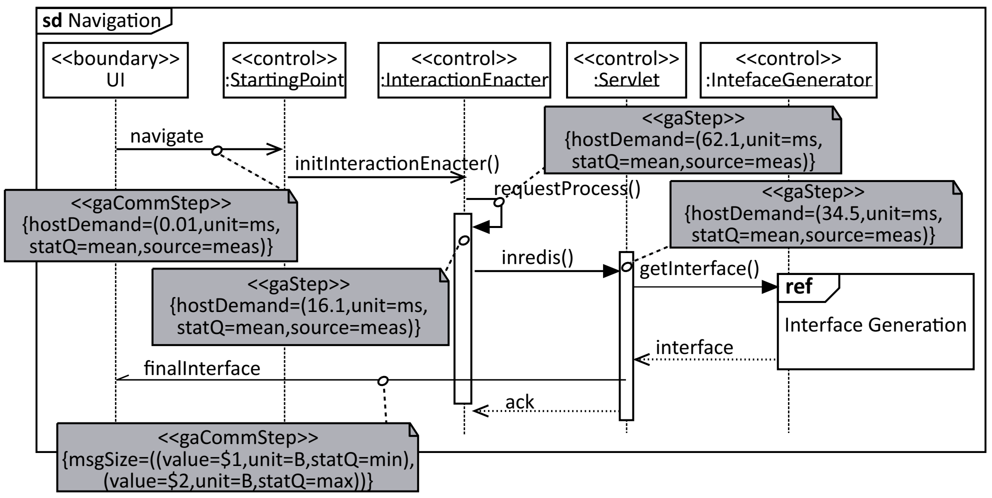

sd Navigation
<<boundary>> UI
<<control>> :StartingPoint
<<control>> :InteractionEnacter
<<control>> :Servlet
<<control>> :IntefaceGenerator
navigate
initInteractionEnacter()
requestProcess()
inredis()
getInterface()
ref
Interface Generation
interface
finalInterface
ack
<<gaStep>> {hostDemand=(62.1,unit=ms, statQ=mean,source=meas)}
<<gaStep>> {hostDemand=(34.5,unit=ms, statQ=mean,source=meas)}
<<gaCommStep>> {hostDemand=(0.01,unit=ms, statQ=mean,source=meas)}
<<gaStep>> {hostDemand=(16.1,unit=ms, statQ=mean,source=meas)}
<<gaCommStep>> {msgSize=((value=$1,unit=B,statQ=min), (value=$2,unit=B,statQ=max))}

Figure 5.6: UML Sequence Diagram modeling a simple navigation

The interaction starts with the interface requesting a navigation to the `Starting Point` that acts as a gateway between the user interface and the system. The Interaction Enacter is the module that handles the navigation between interfaces. It is so because the Navigation activity is considered a subclass of the Device Interaction activity, as from a user perspective, the kind of buttons that perform device interaction activities are the same as those that allow the user navigate within the complex interface. In the request Interaction Enacter accesses to the navigation graph of the complex interface and determines which is the next interface that should be generated. This information is stored in the context of the IG. Finally, a new interface generation process starts. As the context of the IG has been updated with the next interface to be rendered, this is the one that is rendered.

Interface Generation Process

INREDIS devices UIML interfaces are composed of two types of documents:

- Views, a set of UIML documents that describe structure and its interaction elements of each atomic interface. As described in Abrams et al. (1999), the use of UIML allows the abstract and platform-independent definition of user interfaces.

- Navigability graph, which defines the how and on what conditions the complex interfaces navigates throw the different views. Only one view at a time is shown, we refer to it as the current view.

Generating an interface for a user means to transform the current view of an interface into an accessible XHTML document taking into account the characteristics of the user and the needed assistive technology. Most part of the process, depicted in Figure 5.7, is identical to the one defined for the first interaction. The difference is that this process does not adapt the initial interface, but it transforms the current view of the device interface that the user is using at present (which like in the case of the initial interface created by the Initial Creator is an abstract UIML interface).

The first part of the diagram, the one related with the detection of accessibility gaps and the determination of the necessary assistive technology, is the same as the one defined for the first interaction with the INREDIS system. When the Generator module receives the petition of generating an interface by means of the `generateInterface()` module the first step is to determine which is the current view of the interface. This information is stored in the context of the IG (`getCurrentView()` method). The current view is a URL that points to the location of the abstract UIML document that should be used as the starting point of the final user interface. The `Generator` module invokes the `retrieveXMLSource()` method of the helper class `Resource Manager`, and retrieves an abstract UIML interface.

The rest of the steps of the generation of the interface are exactly as the ones described for the first interaction. Instead of using the abstract UIML interface created by the Initial Creator, they use the abstract UIML interface retrieved by the `Resource Manager` from the interface current view URL.

5.2.3 Device Interaction Scenario

The interactions with devices, and services are realized by the Interaction Enacter, see the SD in Figure 5.8.

This module once initialized executes the action involved in the device/service interaction. In order to do so it invokes the `executeAction()` method of the

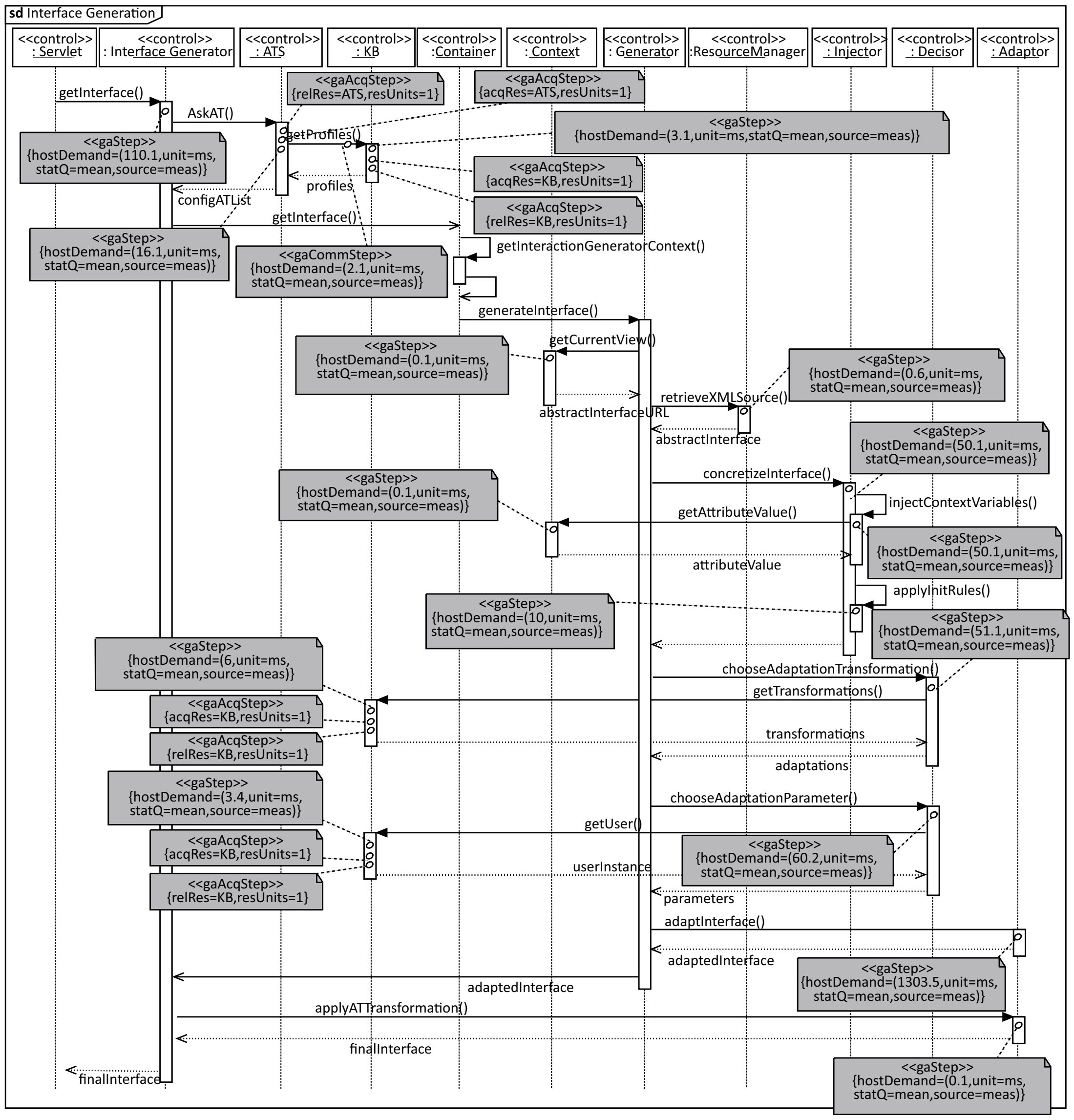

sd Interface Generation
<<control>> : Servlet
<<control>> : Interface Generator
<<control>> : ATS
<<control>> : KB
<<control>> :Container
<<control>> : Context
<<control>> : Generator
<<control>> :ResourceManager
<<control>> : Injector
<<control>> : Decisor
<<control>> : Adaptor
<<gaAcqStep>> {relRes=ATS,resUnits=1}
<<gaAcqStep>> {acqRes=ATS,resUnits=1}
getInterface()
AskAT()
<<gaStep>> {hostDemand=(3.1,unit=ms,statQ=mean,source=meas)}
getProfiles()
<<gaStep>> {hostDemand=(110.1,unit=ms, statQ=mean,source=meas)}
<<gaAcqStep>> {acqRes=KB,resUnits=1}
profiles
configATList
<<gaAcqStep>> {relRes=KB,resUnits=1}
getInterface()
<<gaStep>> {hostDemand=(16.1,unit=ms, statQ=mean,source=meas)}
getInteractionGeneratorContext()
<<gaCommStep>> {hostDemand=(2.1,unit=ms, statQ=mean,source=meas)}
generateInterface()
<<gaStep>> {hostDemand=(0.1,unit=ms, statQ=mean,source=meas)}
getCurrentView()
<<gaStep>> {hostDemand=(0.6,unit=ms, statQ=mean,source=meas)}
retrieveXMLSource()
abstractInterfaceURL
abstractInterface
<<gaStep>> {hostDemand=(50.1,unit=ms, statQ=mean,source=meas)}
concretizeInterface()
<<gaStep>> {hostDemand=(0.1,unit=ms, statQ=mean,source=meas)}
injectContextVariables()
getAttributeValue()
<<gaStep>> {hostDemand=(50.1,unit=ms, statQ=mean,source=meas)}
attributeValue
applyInitRules()
<<gaStep>> {hostDemand=(10,unit=ms, statQ=mean,source=meas)}
<<gaStep>> {hostDemand=(51.1,unit=ms, statQ=mean,source=meas)}
<<gaStep>> {hostDemand=(6,unit=ms, statQ=mean,source=meas)}
chooseAdaptationTransformation()
getTransformations()
<<gaAcqStep>> {acqRes=KB,resUnits=1}
<<gaAcqStep>> {relRes=KB,resUnits=1}
transformations
adaptations
<<gaStep>> {hostDemand=(3.4,unit=ms, statQ=mean,source=meas)}
chooseAdaptationParameter()
getUser()
<<gaAcqStep>> {acqRes=KB,resUnits=1}
<<gaStep>> {hostDemand=(60.2,unit=ms, statQ=mean,source=meas)}
userInstance
<<gaAcqStep>> {relRes=KB,resUnits=1}
parameters
adaptInterface()
adaptedInterface
<<gaStep>> {hostDemand=(1303.5,unit=ms, statQ=mean,source=meas)}
adaptedInterface
applyATTransformation()
finalInterface
<<gaStep>> {hostDemand=(0.1,unit=ms, statQ=mean,source=meas)}
finalInterface

Figure 5.7: SD modeling the Interface Generation process

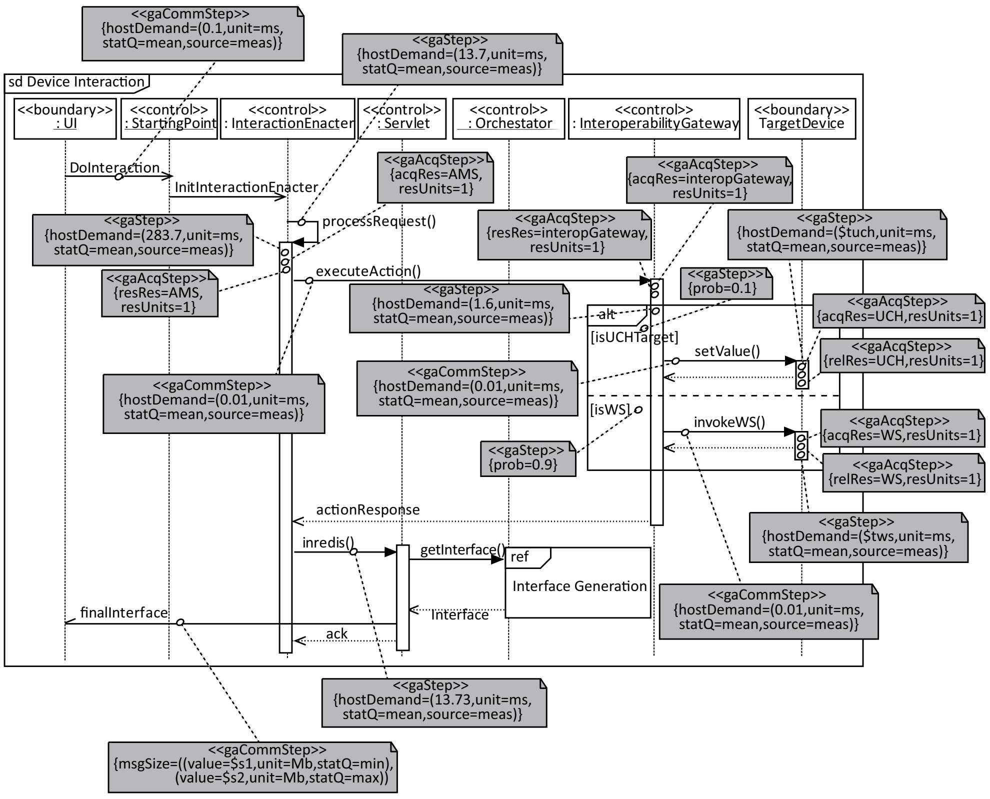

sd Device Interaction
<<boundary>> : UI
<<control>> : StartingPoint
<<control>> : InteractionEnacter
<<control>> : Servlet
<<control>> : Orchestator
<<control>> : InteroperabilityGateway
<<boundary>> TargetDevice
<<gaCommStep>> {hostDemand=(0.1,unit=ms, statQ=mean,source=meas)}
<<gaStep>> {hostDemand=(13.7,unit=ms, statQ=mean,source=meas)}
DoInteraction
InitInteractionEnacter
processRequest()
<<gaAcqStep>> {acqRes=AMS, resUnits=1}
<<gaAcqStep>> {acqRes=interopGateway, resUnits=1}
<<gaStep>> {hostDemand=(283.7,unit=ms, statQ=mean,source=meas)}
<<gaAcqStep>> {resRes=interopGateway, resUnits=1}
<<gaStep>> {hostDemand=($tuch,unit=ms, statQ=mean,source=meas)}
<<gaAcqStep>> {resRes=AMS, resUnits=1}
executeAction()
<<gaStep>> {prob=0.1}
<<gaStep>> {hostDemand=(1.6,unit=ms, statQ=mean,source=meas)}
alt
[isUCHTarget]
<<gaAcqStep>> {acqRes=UCH,resUnits=1}
setValue()
<<gaAcqStep>> {relRes=UCH,resUnits=1}
<<gaCommStep>> {hostDemand=(0.01,unit=ms, statQ=mean,source=meas)}
<<gaCommStep>> {hostDemand=(0.01,unit=ms, statQ=mean,source=meas)}
[isWS]
invokeWS()
<<gaAcqStep>> {acqRes=WS,resUnits=1}
<<gaStep>> {prob=0.9}
<<gaAcqStep>> {relRes=WS,resUnits=1}
actionResponse
<<gaStep>> {hostDemand=($tws,unit=ms, statQ=mean,source=meas)}
inredis()
getInterface()
ref
Interface Generation
<<gaCommStep>> {hostDemand=(0.01,unit=ms, statQ=mean,source=meas)}
finalInterface
interface
ack
<<gaStep>> {hostDemand=(13.73,unit=ms, statQ=mean,source=meas)}
<<gaCommStep>> {msgSize=((value=$s1,unit=Mb,statQ=min), (value=$s2,unit=Mb,statQ=max))}

Figure 5.8: UML Sequence Diagram representing a user's device interaction

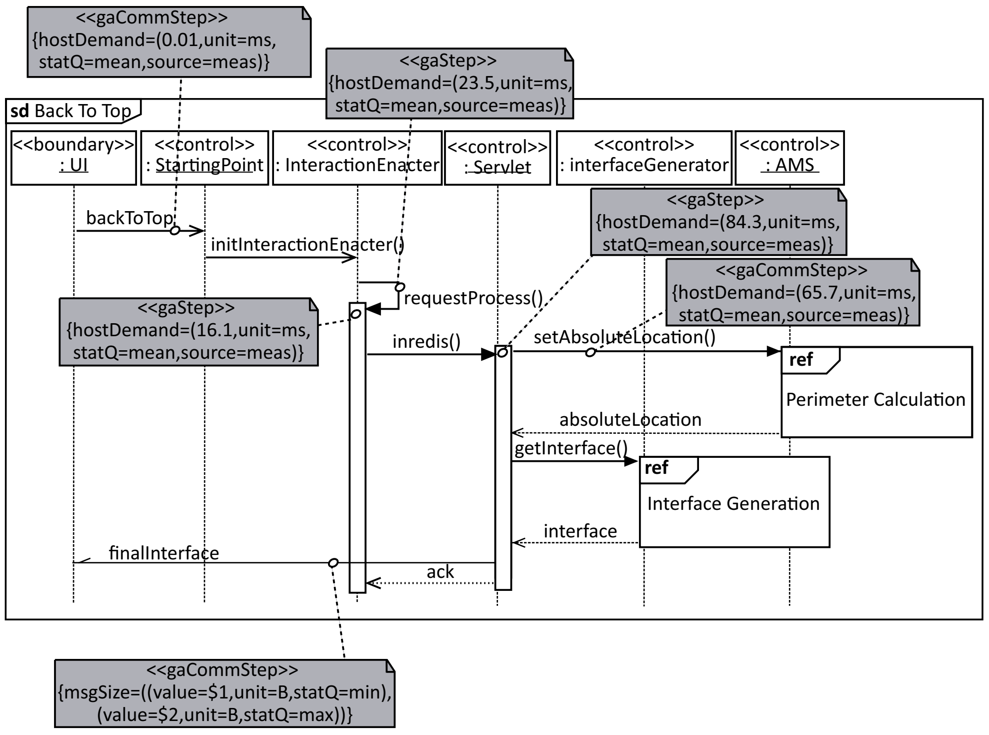
`Interoperability Gateway`, which is a class that decouples the Interaction Enacter from the underneath technology used to interact with the device. The `executeAction()` method may result in:

- `setValue()` method invocation, in case that the device is exposed as a UCH target. The user interaction is translated into the change of one or more values of the UCH Target.

- `invokeWS()` method invocation, in case that the device is exposed as a Web service (or when there is no device and we are dealing with a Web service invocation)

The result is stored in the context of the IG, for later use in case of need. Once the interaction has been carried out, a new interface is generated (invoking the `Orchestrator` method `getInterface()`). This new interface is generated to make sure that it reflects the changes and latest state after the interaction with the device.

5.2.4 Back to Top Scenario



<<gaCommStep>> {hostDemand=(0.01,unit=ms, statQ=mean,source=meas)}
<<gaStep>> {hostDemand=(23.5,unit=ms, statQ=mean,source=meas)}
sd Back To Top
<<boundary>> : UI
<<control>> : StartingPoint
<<control>> : InteractionEnacter
<<control>> : Servlet
<<control>> : interfaceGenerator
<<control>> : AMS
<<gaStep>> {hostDemand=(84.3,unit=ms, statQ=mean,source=meas)}
<<gaCommStep>> {hostDemand=(65.7,unit=ms, statQ=mean,source=meas)}
backToTop
initInteractionEnacter()
requestProcess()
<<gaStep>> {hostDemand=(16.1,unit=ms, statQ=mean,source=meas)}
inredis()
setAbsoluteLocation()
ref Perimeter Calculation
absoluteLocation
getInterface()
ref Interface Generation
interface
finalInterface
ack
<<gaCommStep>> {msgSize=((value=$1,unit=B,statQ=min), (value=$2,unit=B,statQ=max))}

Figure 5.9: UML Sequence Diagram representing the back-to-top process.

This process, illustrated in the SD of Figure 5.9, means going back to the device initial interface.

As in the case of the navigation, the Interaction Enacter is the module that handles the back to top process. It is so because this activity is considered a subclass of the Device Interaction activity, as from a user perspective, the kind of buttons that perform device interaction activities are the same as those that allow going back to the device top interface. In order to keep all the information in the KB up-to-date we begin updating the location of the device and we recalculate the information about the user's INREDIS perimeter. The next step is to generate top interface of the device, which is made using the Interface Generation process that we have already described in Section 5.2.2.

5.2.5 Assistive Software Selection Mechanism

Assistive technology is the hardware or software that is added or incorporated within a system than increases accessibility for an individual, as defined in ISO (2011).

Assistive Software (AS) is understood as a piece of software used to increase our ability to manage some kind of information in a digital device. The Assistive Software Selection Mechanism (ASSM) makes the environment able to automatically select the most suitable AS for a given interaction with a specific electronic target device taking advantage of the user's context (user, controller device and target device) and considering the possible discrepancies between the user and the environment, namely in the case of functional diversity.

The ASSM uses different knowledge based on ontologies to achieve this goal, so this process consists of five main activities, one of them split into six, see the UML Activity Diagram in Figure 5.10. The complete AS selection mechanism (ASSM) is described by Gómez-Martínez et al. (2013b). The following is a summarized description of each activity.

Detecting Discrepancies The first activity detects any accessibility issues that might prevent the user from being able to use a controller. In order to detect discrepancies we use a set of specific rules stored in the `KB` that compare the characteristics of the interaction that the user is able to perform with those that the controller is able to emit/receive. The complete catalogue of rules is specified by González-Cabero (2010).

Checking Feasibility Each discrepancy found in the previous step, is analyzed to determine whether mediation by the AS can enable the interaction. The following activities are intended to ascertain which AS is most appropriate.

Matching by History Log When the user has already employed the system to interact with the same target using the same context, it is possible to retrieve the most suitable AS without further reasoning, just by querying the `KB` and retrieving the matching set from the AS History.

Matching by Score This activity triggers the reasoning process where four subsets of concepts are simultaneously queried in the `KB` using parallel activities. This activity is divided into the next activities:

- *Retrieve Standard Fulfillment.* This activity performs an evaluation where the best scoring AS will be those that follow worldwide *accessibility standards* established by *recognized accessibility entities.*

- *Retrieve Privacy.* This activity checks that the AS complies with the *data protection measures* issued by security bodies. It is important to note that, according to many laws in different countries, when an *AS* complies with a *data protection act* level, it also complies some *data protection measures.* This is taken into account here via rules to assert those facts in the `KB`. This is the case for e.g., Federal Data Protection and Information Commission of Switzerland or Ley Orgánica de Protección de Datos in Spain.

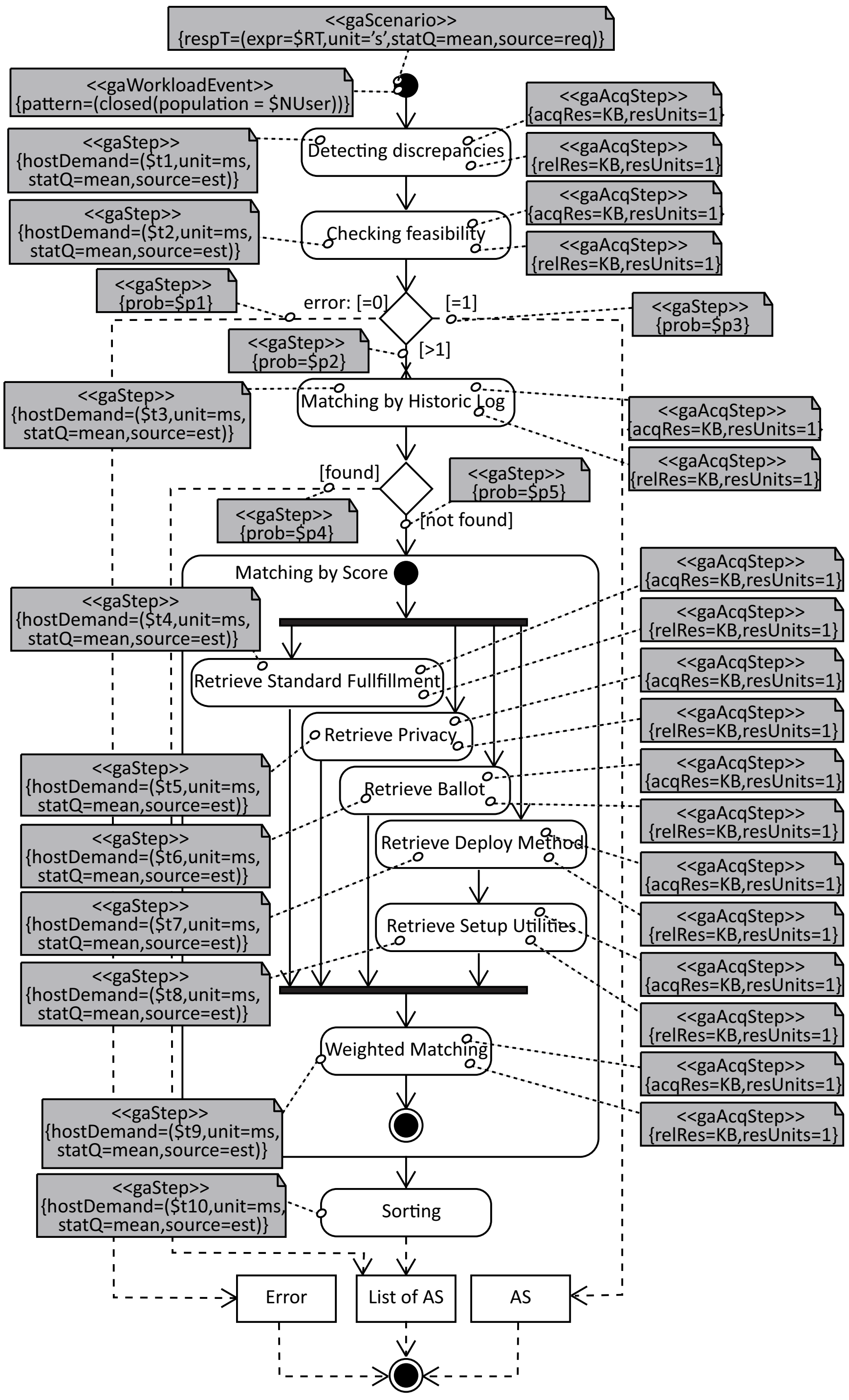


Figure 5.10: UML Activity Diagram of the AS selection process

- *Retrieve Ballot.* This activity increases the score for those AS with the best user reviews. These reviews are drawn from all the system's users but they are not linked to any individual user, to maintain privacy about users' functional diversity.
- *Retrieve Deploy Method.* The scoring is the simplest, just to foster the use of *AS* deployed as *SaaS* (*Software as a Service*).
- *Retrieve Setup Utilities.* This activity needs the output of *Retrieve Deploy Method*, so it is not executed in parallel with the others. All of the concepts are scored in this activity to take into account the ease of access and use of the AS.
- *Weighted Matching.* This is the final activity of matching by score. It focuses on adapting the matching to the user's preferences and the Domain Experts' assumptions.

  The user has been previously asked to state its level of importance by means of the user profile stored in the `KB`. With the weighting system, all the roles involved in the selection are taken into account (i.e., domain experts, the user, and all system users at once using the reviews).

Sorting This is the final activity of the whole process. This activity orders the set of AS products/services of the weighted matching activity in descending order.

The system is prepared to automatically provide the first AS in the list (i.e. the most suitable AS for the given context). However, when the user rejects the highest-scoring AS, the full set is presented to the user formatted as a descending-order list. The weights used in the ASSM can evolve based on experience, while the overall of the proposal remains fixed.

5.3 Applying the Assessment Methodology

In this section, we pursue performance results for assessing and eventually improving the INREDIS architecture. For achieving this objective, we have followed the methodology proposed in Chapter 4 based on SPE principles (Smith, 1990).

In the following, we aim to apprise practitioners of the use of the methodology. We offer advise by indicating how we actually applied the methodology in the INREDIS project and which were our choices (e.g., which languages, performance models or tools we used).

5.3.1 Performance-Oriented Design

The methodology begins by modeling the system architecture using UML diagrams. We also address the behaviour of those scenarios of the system critical for performance. Section 5.2 reported these two first tasks of this step for our software architecture describing the UML design of the system. In particular, the architectural description has a focus on the behavioural view, which is of primary importance for performance assessment.

The overall architecture was presented in the UML Deployment Diagram in Figure 5.1. The IOD in Figure 5.2 defines a general system scenario made of four subscenarios, each one describing a part of the system behaviour.

Following the methodology based on SPE principles, we now introduce the *performance view* of the system. The usual way in SPE for introducing a performance specification is by annotating the design diagrams. As mentioned in Chapter 4, we annotate the UML diagrams with MARTE profile, and use the VSL to define data-types characterized by several properties. MARTE annotations appear as gray notes in the UML diagrams we have presented. In the following, the most interesting annotations are commented. They capture properties, measures and requirements of interest for carrying out performance analysis. Section 4.2 details the performance annotations considered in our methodology. In the following, we summarize some of them in the INREDIS architecture:

- The **response time** has been specified in the IOD in Figure 5.2, in this case using *GaScenario* annotation. In this case, response time is a measure to be calculated during analysis as indicated by *source=calc*. The result will be gathered in variable $\$RT$. The `unit` of measurement are seconds and the *statistical measure* is a mean.
- **workload** has been specified in the IOD in Figure 5.2 using *GaWorkloadEvent* annotation. This is a closed workload, then specifying the number of concurrent users in the system through variable $\$NUsers$ in VSL.
- **Host demands** represents the system duration of the activities. In our architecture, they have been measured in the testing phase by the software engineers. Figure 5.3 offers some examples.
- An example of **routing rates** is depicted in Figure 5.4 by means of the *prob* annotation in the *GaStep* attached to the alternative fragments.
- System **resources** can be expressed in MARTE as lifelines in the UML Sequence Diagrams. Annotations *acqRes* and *relRes* attached to *GaCommStep* specify their **acquisition** and **release**. Figure 5.4 depicts several examples, see one of them attached to the `KB` lifeline. For specifying the number of system available **resources**, annotation *resMult* in the deployment (Figure 5.1) is used. Variables (for example $\$pKB$ or $\$pAMS$) will allow to perform sensitive analysis parameterizing the system with different number of resources.

5.3.2 Performance Model

Following the methodology, we need to obtain a performance model for each critical scenario that we have previously annotated. As formalism, we use Generalized Stochastic Petri Nets (GSPN) proposed by Ajmone Marsan et al. (1995). See Section 2.4 for an introduction of Petri nets.

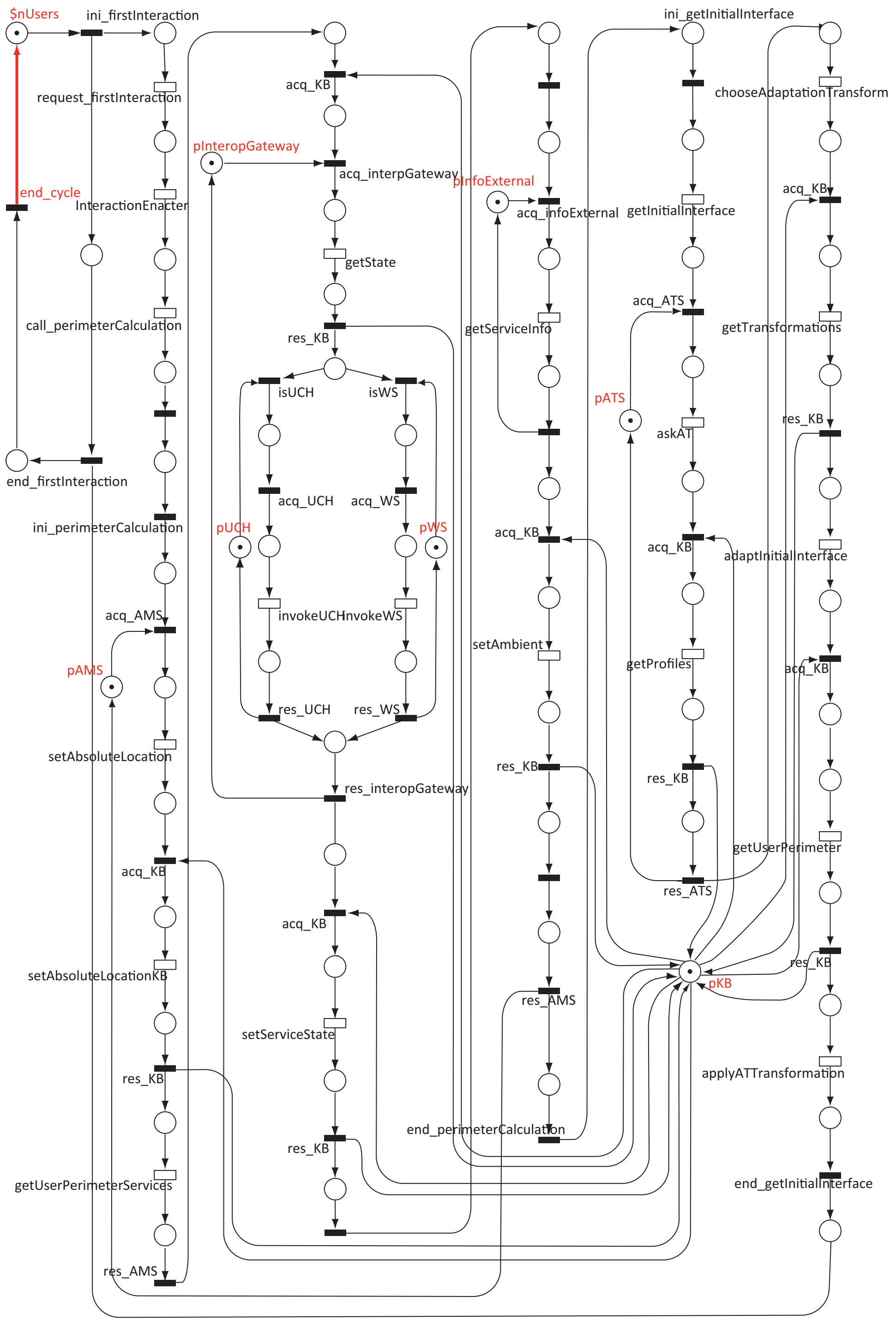


Figure 5.11: GSPN representing First Interaction scenario.

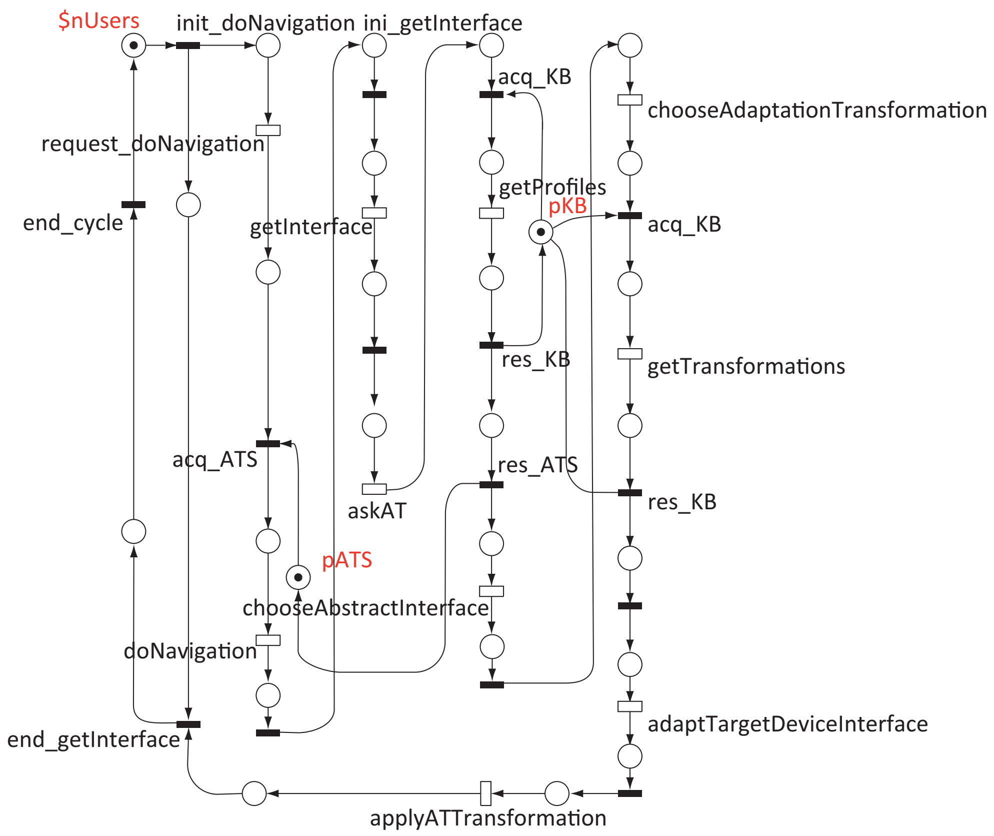


Figure 5.12: GSPN representing Navigation scenario.

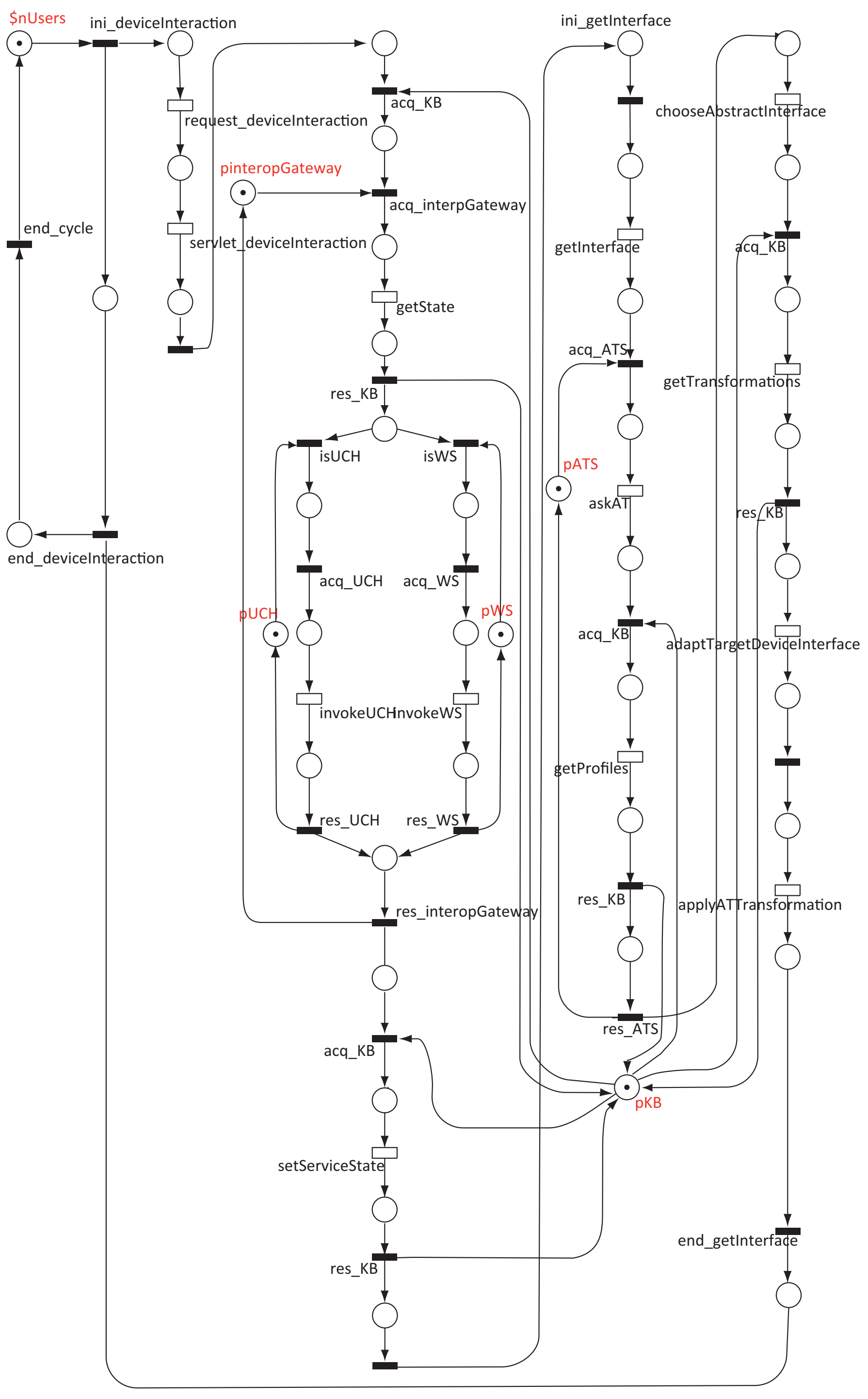


Figure 5.13: GSPN representing Device Interaction scenario.

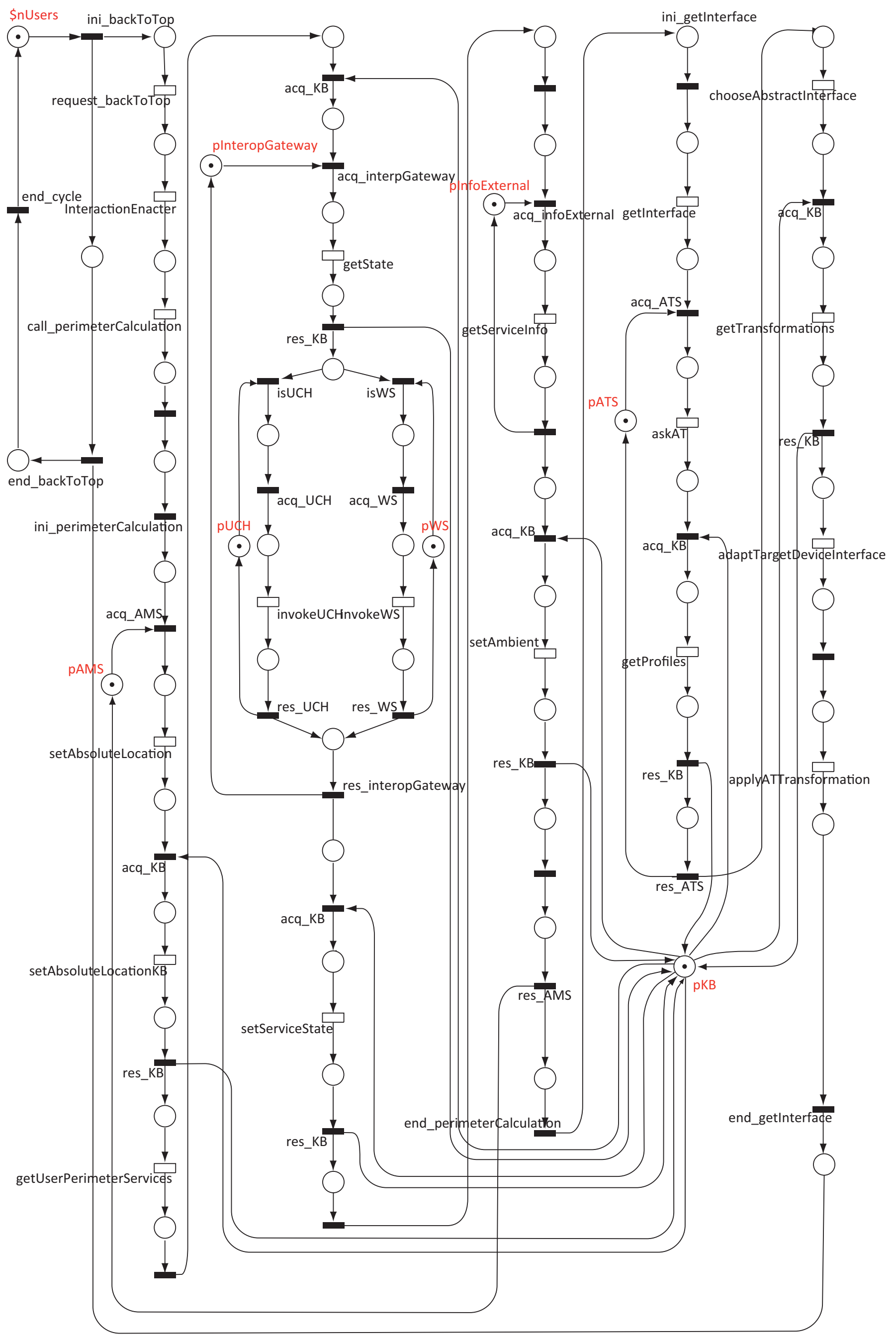


Figure 5.14: GSPN representing Back To Top scenario.

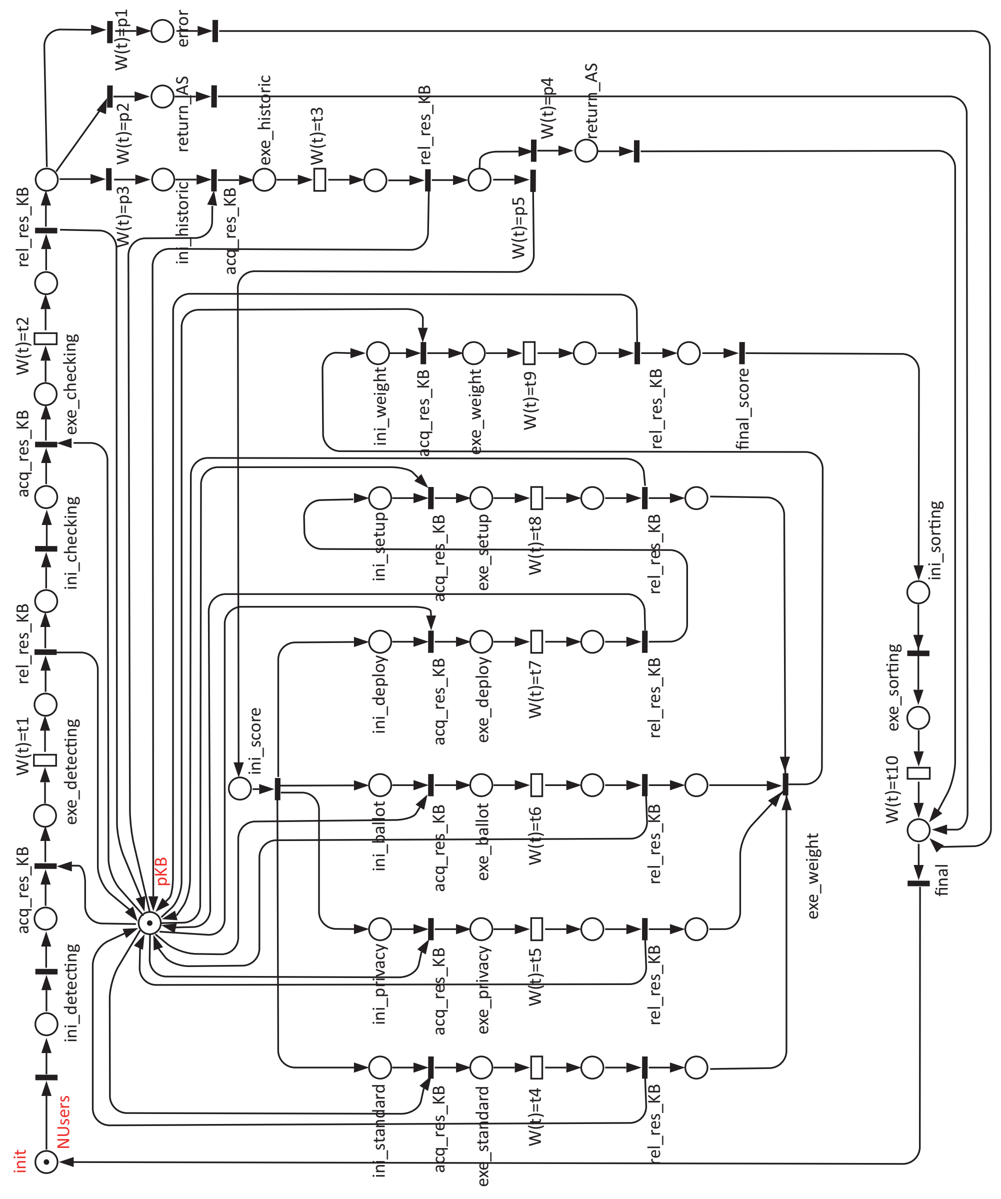


Figure 5.15: GSPN representing the ASSM.

To translate performance-annotated UML models into GSPN, we use ArgoSPE which translated UML diagrams into GSPNs following the methods proposed by Bernardi and Merseguer (2007) and Distefano et al. (2011). We translated each critical scenario and obtained the structure of a GSPN, Figure 5.11 depicts the GSPN for the First Interaction scenario. The rest of the GSPN models of Navigation, Device Interaction and Back To Top scenarios, and the ASSM obtained from the design, appear in Figure 5.12, Figure 5.13, Figure 5.14 and Figure 5.15, respectively. The translation, although automatic, required some additional effort. So, we had to introduce the representation of some performance parameters manually. Section 5.4 discusses these issues. ArgoSPE internally calls GreatSPN tool (Chiola et al., 1995) to analyse or simulate these GSPNs in order to obtain the performance metrics in the following step of our methodology.

5.3.3 Performance Analysis

In this step, the software engineer reviews the performance objectives, which were defined during the design step, and carries out the analysis of the performance model. Performance objectives are quantitative measures that can be computed in the performance models. There were two important objectives in the INREDIS project: responsiveness and scalability.

Responsiveness is a property of primary importance for software systems, so we can build the right system but if it does not meet the expected response time it will not be useful from the user perspective. In general, people with special needs demand software with response times equal to people without those needs, see discussion below. However, there is a group not so demanding, those with intellectual disabilities, for which INREDIS is also intended. In any case, being the scope of INREDIS people with all kind of special needs, the adapted interfaces have to be timely created, for the user not to lose the focus.

Regarding scalability INREDIS has to be deployed in very different environments, e.g., building automation, urban, leisure or financial. The number of concurrent users can vary considerably, even for the same kind of environment it changes by orders of magnitude, for example, in the building automation case we could have smart homes, asylums, hospitals or hotels. Considering that the architecture has to be the same for all environments, it needs to scale accordingly.

Moreover, in our methodology, we include to analyse the utilization as performance metric. Utilization appropriately measures the effect of software as it scales in usage (Smith and Williams, 2002b).

Therefore, the important task in this step is the analysis of the performance models, according to the performance objectives, for obtaining results. Next, we discuss implications of performance objectives in the project and how these measures are computed in the GSPN models.

Computation of Measures in the GSPN Models. We compute all the measures (response time, utilization and scalability) under steady state assumption. Steady state means that the system reaches an equilibrium, so, measures obtained will continue in the future, which is a more general assumption than transient state. In a GSPN, steady state analysis can be carried out when the net is cyclical. However, the translation of a UML Sequence Diagram produces an acyclical GSPN, it starts with a resource place (see in Figure 5.11, place $\$nUsers$) and ends with a transition for the last scenario message (see in Figure 5.11, transition *end_cycle*). Therefore, we need to add an arc from this last transition to the starting place, then achieving a cyclical net (see the red arc in Figure 5.11). Now, the scenario can be analyzed under steady state assumption.

Responsiveness

In this step, firstly we should study who/what interacts with our system. In the case of INREDIS project, it has a Human-Computer Interaction (HCI) system, being potential users people with special needs. Therefore, from the user's perspective, the response time is the number of seconds required to response to a user request. The *Usability Engineering* principles, proposed by (Nielsen, 1993) establish the response time intervals for HCI systems. They are outlined in Section 4.4.1.

Although target audience in our system has special needs, the response times must be similar to users without those needs[1] if we do not consider the time spent by the disabled people to operate the target device. Then, in our architecture, all the expected response times should be within these intervals. Pragmatically, we will assume quantities around ten seconds as acceptable response times. Nevertheless, we know that response time may also depend on the kind of impairment the user has and on the kind of *target* device or service the user wants to control. For example, elderly people could request commands in their personal telecare device at a rate of few seconds. However, for a blind person it could last much more time to operate for instance the washing machine. On the other hand, it is important to note that slow response times could prove frustrating for a person with cognitive disabilities, it also has serious consequences for the system usability.

Scalability

As discussed at the beginning of Section 5.3.3, depending on the environment, the number of potential users of the INREDIS architecture could be small (e.g. smart home) or large (e.g. intelligent buildings, public banking). Furthermore, in an embedded system, as our architecture, the scalability is not only conditioned by the number of users but also for the internal demand of resources and services. As such, the execution context is also crucial for scaling up the system. Note that differently from other INREDIS modules, the ASSM is exclusively used by impaired people when it works as a stand-alone service; otherwise, the ASSM is executed only once in the context of INREDIS architecture.

[1]For example, blind people interact with tactile interfaces by means of an immediate audible feedback.

Although implementations in large environments are not accomplished yet in INREDIS, we strive for evaluating the scalability of the architecture also in these contexts. On the other hand, our architecture considers not only physical devices, but also software services available on the Internet. Therefore, to cover all these cases, the system must support requests from a large number concurrent users. We will parameterize such number through the system workload, taking into account that currently around 10 per cent of the total world population live with a disability, according to United Nations[2].

Utilization

As mentioned in Section 4.4.3, resource utilization analysis detects resource saturation and potential bottlenecks when the system is highly populated and consequently, it permits to tune up the resource configuration. In our project, apart from detecting bottlenecks, it is crucial an appropriate resource utilization, since the architecture is planned to be deployed into cloud-based infrastructures, which imply pay-per-use services. Being each new instance of a thread independently invoiced, resource utilization must be optimized.

Empirical Results and Validation of the Performance Models

In order to validate the architecture for running prototypes, some pivotal pieces (modules) were implemented and tested within the INREDIS project. Tests considered diverse users disabilities, preferences and profiles. Catalán and Catalán (2010) tested the architecture experimentally, by using a set of user controlled tests. The main challenge was to measure the satisfaction of the user experience with diverse interaction modes of services and devices for people with special needs. This level of satisfaction included usability aspects as well as performance objectives. Additional experimental results can be found in INREDIS Consortium (2010a).

As described in Section 5.2, the main modules making up the INREDIS architecture are: Interface Generator, Knowledge Base, Assistive Technology Server (ATS) and Adaptive Modelling Server (AMS). For experimental evaluation, only the Interface Generator module was completely implemented. The Knowledge Base module, which stores ontologies, was very low populated, only with basic knowledge mechanisms, a minimum for experimentation. The ATS and the ASSM were implemented to support only the users profiles and interaction modes required for the tests. Finally, the basic functionality of the AMS was implemented. These modules were executed to carry out the four system scenarios (First Interaction, Navigation, Back To Top and Device Interaction), depicted in Figure 5.2. The experimental tests were targeted to analyse the responsiveness and scalability of the system. The tests were mainly focused on interoperability and usability. Figure 5.16 shows the measured average response times of these key performance scenarios in this user testing phase.

However, the burden of real experimentation with complex interoperable architectures, elderly people, and people with special needs, raised quickly and it greatly

[2]http://www.un.org/disabilities/default.asp?id=18

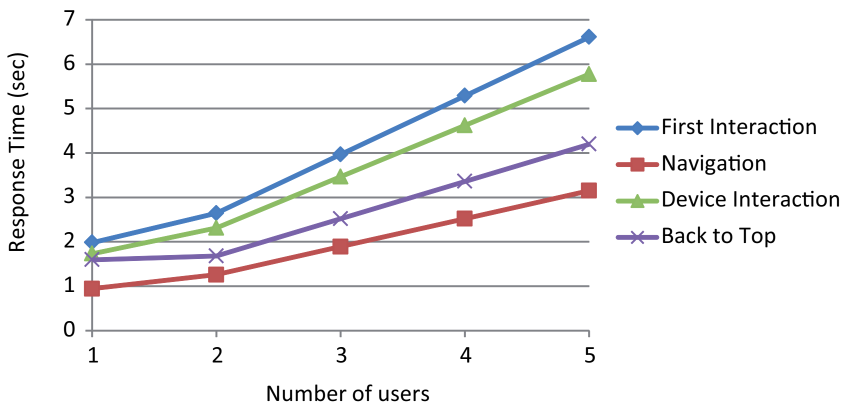


Figure 5.16: Empirical response times for a set of test users.

limited the evaluation. In this user testing phase, the number of concurrent users never exceeded 5 due to the logistical difficulties of real experimentation, as Sainz et al. (2011) described. On one hand, some of the users needed caregiver or additional assistive products. On the other hand, some tests were made individually to specifically study user interactions. Finally, the cost of the team for supporting the experiments and the facilities (e.g. smart home) were also important issues for carrying out more complex experimentations.

Hence, experimentation problems and limitations of real implementations advocated the use of models, specially in these initial phases of the system life-cycle. Models can represent the system in a variety of hypothetical situations and can perform analysis at a lower cost. SPE, as summarized in Section 4.5, offers techniques and tools that can overcome these problems.

We then reproduced these experiments using the performance models obtained by the second step of the methodology. We got the results in Figure 5.17. Note that using models we obtained results for one hundred users, which was enough for our purposes. We could have obtained results for larger populations using the same GSPN models by changing the workload. Table 5.1 compares for each scenario the results obtained in real experimentation (*Real* rows) with those obtained by our GSPN models (*Model* rows). We appreciate that differences (*Var.* rows) between our models and real experimentation are around a five percent in most cases, differences never went beyond ten percent, except for the Device Interaction scenario in the case of three concurrent users. In this latter case, we assume that the variation might be caused by the accuracy level in the computation of the GSPN models. Moreover, we observed that tendencies in the graphs were similar. So, we can assume that our performance models can be useful to address experiments initially not feasible to carry out with the real implementations.

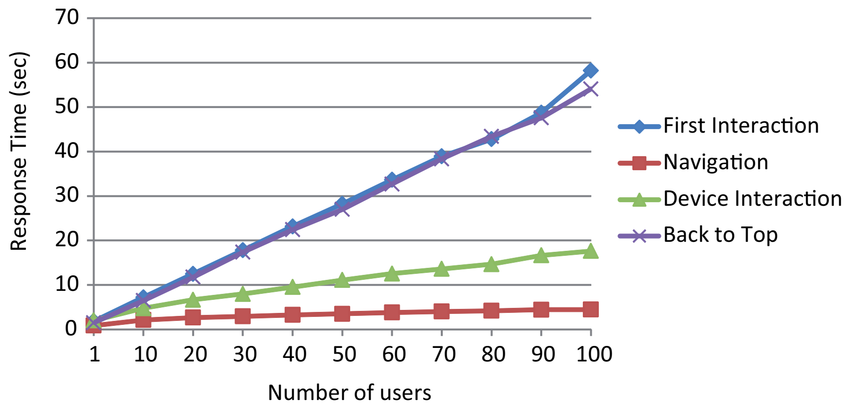


Figure 5.17: Response times for concurrent users using models.

Table 5.1: INREDIS: Results in seconds for each performance scenario

Scenario		**Number of users**							
		1	2	3	4	5	10	50	100
First Interaction	Real	1.983	2.644	3.966	5.287	6.609	NE	NE	NE
	Model	1.841	2.674	3.776	4.912	6.210	7.212	28.229	58.231
	Var.	$\simeq 5\%$	$< 5\%$	$\simeq 5\%$	$> 5\%$	$> 5\%$	-	-	-
Navigation	Real	0.945	1.260	1.891	2.521	3.151	NE	NE	NE
	Model	0.875	1.130	1.684	2.338	2.793	3.109	3.520	4.470
	Var.	$> 5\%$	$\simeq 10\%$	$\simeq 10\%$	$\simeq 10\%$	$\simeq 10\%$	-	-	-
Device Interaction	Real	1.732	2.310	3.465	4.619	5.774	NE	NE	NE
	Model	1.808	2.351	2.994	3.937	3.380	4.752	11.114	17.648
	Var.	$\simeq 5\%$	$\simeq$	$> 10\%$	$\simeq 10\%$	$\simeq 10\%$	-	-	-
Back to Top	Real	1.596	1.679	2.519	3.359	4.199	NE	NE	NE
	Model	1.497	2.121	2.745	3.369	3.993	6.488	27.013	54.093
	Var.	$\simeq 5\%$	$\simeq 10\%$	$> 5\%$	$\simeq$	$< 5\%$	-	-	-

NE - The experiment could not be carried out

Due to the fact that ASSM can work independent of the rest of the INREDIS architecture, responding external requests, we also validate the analysis of its response times before its implementation, examining the behaviour when the number of users increased. Table 5.2 summarizes the probability rates, which are also annotated in the UML Activity Diagram shown in Figure 5.10.

Table 5.2: Probability rates in the UML Activity Diagram of ASSM.

	Parameter	**Probability**
Checking feasibility $= 0$	p_1	0.1
Checking feasibility $= 1$	p_2	0.3
Checking feasibility > 1	p_3	0.6
Matching by History Log $= found$	p_4	0.7
Matching by History Log $= notfound$	p_5	0.3

Figure 5.18 depicts the response times for the ASSM obtained in the user testing phase. Remark that the knowledge base of the ASSM is populated with only ten AS, we analyse this issue later in Section 5.3.4.

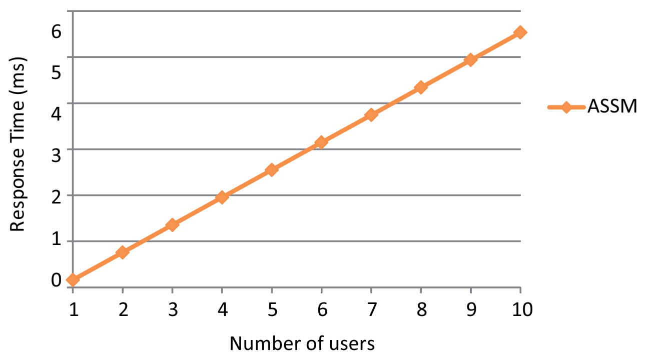


Figure 5.18: Empirical response times for the ASSM for a set of test users

The results of the simulation showed that for a large number of impaired people concurrently using the ASSM the design behaves extremely well, responding in less than two milliseconds. Furthermore, the results for one hundred users can be considered a success since the response time is around six milliseconds. Figure 5.19 summarizes these results graphically. It is important here to remember that the ASSM is executed only once in the context of INREDIS architecture. On the other hand, these results directly depend on the underlying KB system, as observed by (Liang et al., 2009).

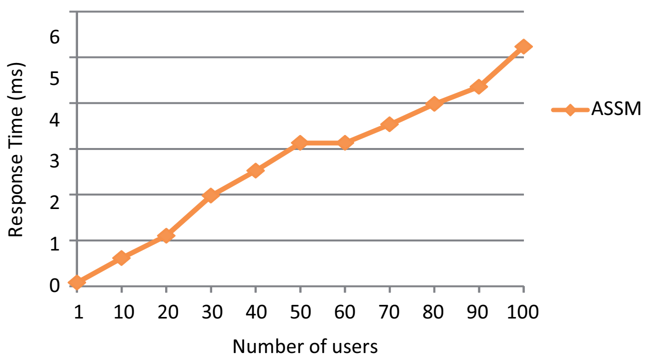


Figure 5.19: Response times of the ASSM using performance models

These experimental results were later compared with those obtained with the real implementation. As occurred with other INREDIS modules, differences accounted for less than ten percent. Hence, we considered our performance model to be validated and so it can be used to test the system in hypothetical situations in which it can be difficult or expensive to carry out experiments. For example, now we can forecast the behaviour of the ASSM when used by 1000 users, which could be a very difficult experiment with the real system. Indeed, the deployed ASSM has been only proved by 8 people simultaneously. For this case the response time obtained with GreatSPN was 48 milliseconds.

Discussion of the Results: Performance View

First, we note that, as requested in Section 5.3.3, the experiments (both, real and GSPN) did not consider the time spent by the disabled people, neither the time to operate the target device or service. Note that this issue is not a limitation to evaluate the architecture.

The discussion about what could be considered a good response time was introduced in Section 5.3.3 from the Usability Engineering point of view (Nielsen, 1993). Pragmatically, we decided that quantities around ten seconds could be considered as acceptable response times. From results in Figure 5.17, we observe that both Device Interaction and Navigation scenarios have acceptable response times. The Navigation scenario never goes beyond six seconds, while the Device Interaction scenario is below ten seconds until it reaches forty concurrent users. However, Back To Top and First Interaction scenarios perform poorly. The reason is that both of them must calculate the user context perimeter, which depends on the number of devices or services, and their corresponding available operations. Therefore, our assessment loop, developed in Chapter 4, concentrates on how to decrease response in these scenarios mainly.

5.3.4 Performance Assessment

Alternatives discussed in our methodology, in Section 4.5, for improving responsiveness were: resource replication (using utilization resource analysis) and application of performance patterns and antipatterns. In the following, we conduct the study following these alternatives for getting an "optimal system configuration" for the INREDIS Interoperable Architecture.

Resource Replication

Resource utilization analysis detects those resources/tasks, which would be potential system bottlenecks. We recall that resources are represented: a) in the UML Sequence Diagrams by life-lines, b) in the GSPN by shared places, highlighted in red in Figures. Section 4.4.3 explained how we compute utilization in the GSPNs. Our analysis considered that some resources were shared between several scenarios (e.g., the Adaptive Modelling Server (AMS), which appears in the Back To Top and in the First Interaction scenarios).

Then, for each of the four key scenarios, we obtained the utilization of all resources involved. However, Figure 5.20 depicts only the utilization of some resources, to avoid cluttering. As we can observe, in the First Interaction and Back To Top scenarios, both Interoperability Gateway and AMS resources are highly saturated, with maximum utilizations of 94% and 98% respectively.

We replicated resources (added threads) for the AMS and the Interoperability Gateway and computed response times for the Back To Top scenario. Figure 5.21 presents results obtained for the case of 50 users (which is representative of all the experiments we performed). We can observe that the response time does not improve, it is around 30 seconds, same as in Figure 5.17 where no replication was introduced. We thought that saturation could be caused not only because of these resources. Therefore, we computed resource utilizations, for all the possible multithreading situations, in the Back To Top scenario.

Figure 5.22 presents only a representative part of these results. It depicts the case of 50 users, with a variable number of threads of the Interoperability Gateway and the AMS, for the rest of the resources it considers one thread only. As observed, the AMS and the Interoperability Gateway are no longer saturated. However, the Target Service/Device becomes saturated. This resource, although not initially considered, appears in different scenarios, e.g. Device Interaction (in Figure 5.8) or Perimeter Calculation (in Figure 5.4).

We performed all the experiments again, from one to one hundred users, in the Back To Top scenario. In this case, replicating threads for the AMS, the Interoperability Gateway and the Target Service/Device. Figure 5.23 shows the results for the case of 50 concurrent users. Now, the response time has reached an acceptable threshold according to the usability principles, around 5 seconds in the best situations. We perform experiments, although not depicted in the figure, and observed that, from 30 threads on, the system did not perform better.

Finally, once we had identified all critical resources (AMS, IG and Target Ser-

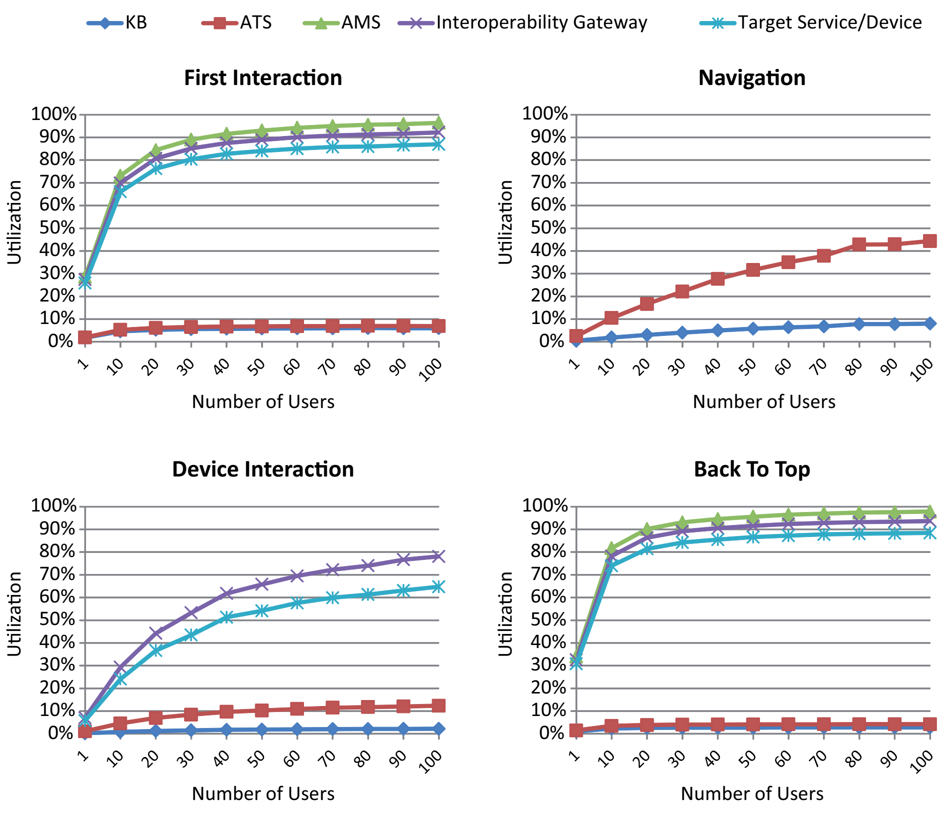


Figure 5.20: Resource utilization of each key scenario.

vice/Device), we replicated them, according to our investigations, and computed response times in all the scenarios. Figure 5.24 presents these results, it shows that the response times have significantly improved.

Performance Patterns

Although the results obtained satisfied the usability principles, our objective, at this stage, was to discover whether we could improve system responsiveness and scalability. We then aim at applying some performance patterns to the architecture design. We used the algorithm proposed by (Bergenti and Poggi, 2000) and applied it throughout the architecture. As a result, we found that the Fast Path performance pattern could be applied for improving the Perimeter Calculation process, one of the processes most used in the system. In fact, this pattern caters to the centering principle, which means to focus attention on the performance of the scenarios that are exercised the most or have large performance impact. The Fast Path pattern was summarized in

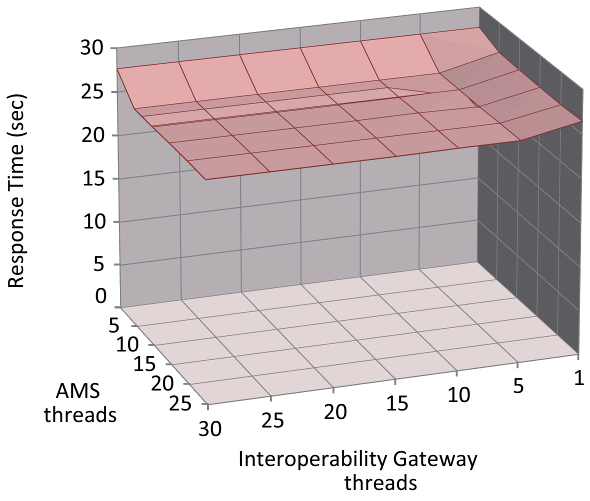


Figure 5.21: Response times when multithreading AMS and Interoperability Gateway.

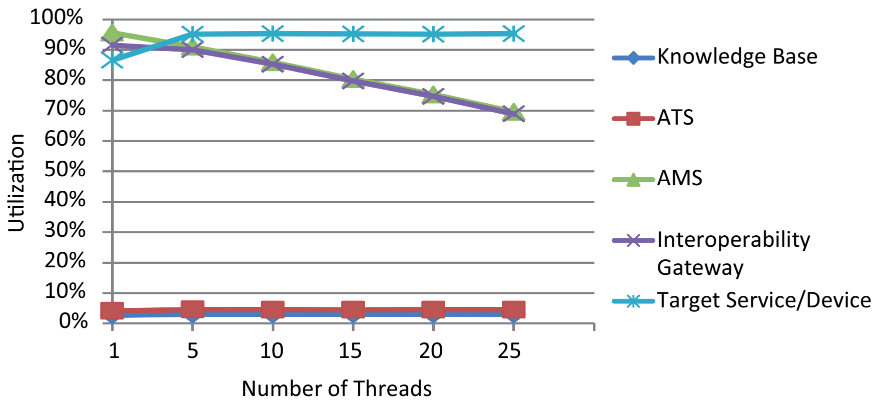


Figure 5.22: Resource utilization when threading Interoperability Gateway and AMS.

Section 4.5.2.

Perimeter Calculation, depicted in Figure 5.4, is a process used by the First Interaction and Back To Top scenarios, which were compromising system responsiveness. The *perimeter* represents the list of devices and services available and this process updates the status of each device and service by consulting the Interoperability Gateway. This is shown by the two consecutive loops in the UML Sequence Diagram in

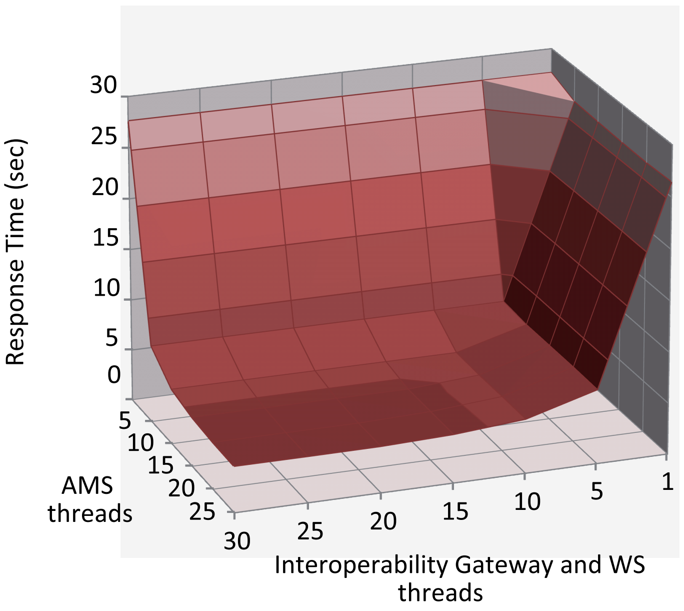


Figure 5.23: Response times when multithreading AMS, Interoperability Gateway and Target Service/Device.

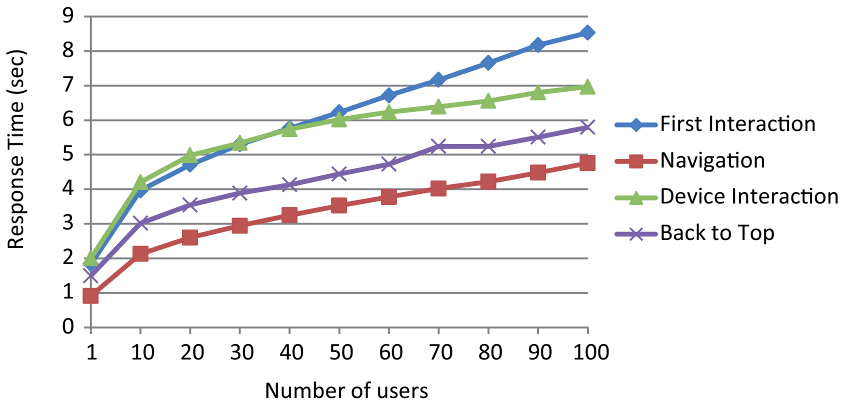


Figure 5.24: Response times when multithreading AMS, Interoperability Gateway and Target Service/Device for concurrent users.

Figure 5.4. Therefore, if the number of available devices is x and each of them has f functionalities on average, the second loop is executed $x \times f$ times.

The Fast Path can be applied here for providing an alternative execution path which minimizes the steps of execution or dedicates more resources here than to other scenarios. Consequently, response time should improve.

One manner to apply the Fast Path pattern in the INREDIS architecture is to request only those functionalities that will be displayed (e.g., if a user would like to control the air conditioning, it would be not necessary to load leisure services). In other words, the user's context (or locality) must be taken into account in order to calculate the perimeter. According to the deployment tested, which had a total of 30 target devices or services and each of them had about 2 or 3 functionalities, if the Fast Path pattern is applied, then the number of times that loops are executed is reduced around 60% on average, which significantly reduces the response time.

We applied the Fast Path pattern in our architecture design and obtained new performance models for the First Interaction and Back To Top scenarios.

We carried out the whole set of experiments with the new performance models, but without taking into account the multithreading discussed in previous section, since we wanted to know how much, by itself, the performance pattern could improve system responsiveness. Figure 5.25 shows these results. If we compare them with those in Figure 5.17, we observe that response times, although not fitting the usability principles yet, are less than half.

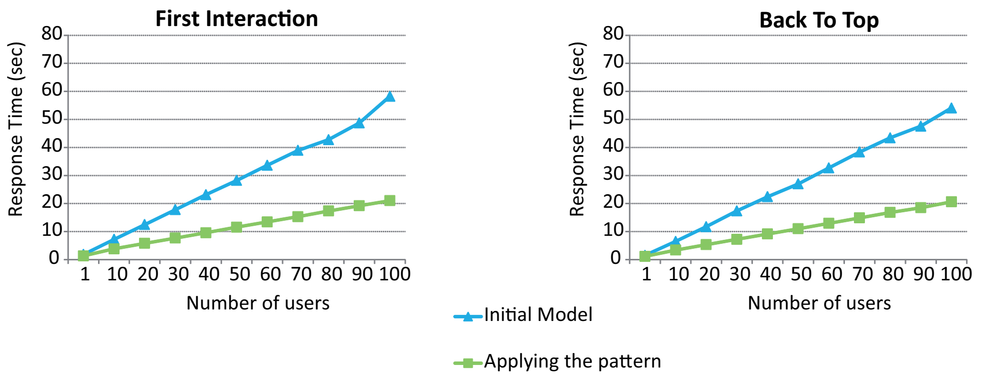


Figure 5.25: Comparison of response times when applying Fast Path pattern: baseline and new results.

Performance Antipatterns

As mentioned in Section 4.5.2, we used the logical predicates defined by Cortellessa et al. (2012) in order to systemize the identification of performance antipatterns in our architecture. After applying all these logical predicates, we detected The Ramp antipattern in the ASSM, detailed in Section 5.2.5. The problem arises since the ASSM searches incrementally in the Knowledge Base. The ASSM was completely developed in the INREDIS project, however it was populated with only ten AS in the testing phase[3]. Figure 5.18 depicts the response times obtained in the testing phase.

As mentioned in Section 4.5.3, the Ramp antipattern *occurs when processing time increases as the system is used.* (Cortellessa et al., 2012) formalized it as follows:

$$\exists OpI \in \mathbb{O} \mid \frac{\sum_{1 \leq t \leq n} |F_{RT}(OpI, t) - F_{RT}(OpI, t-1)|}{n} > Th_{OpRtVar} \wedge \frac{\sum_{1 \leq t \leq n} |F_T(OpI, t) - F_T(OpI, t-1)|}{n} > Th_{OpThVar} \tag{5.1}$$

where:

- OpI is an operation instance whose response time increases along n time slots,
- $\mathbb{O}$ represents the set of all operation instances in the system,
- F_{RT} and F_T are functions that respectively compute the mean response time and throughput of an operation instance observed in a time slot,
- $Th_{OpRtVar}$ and $Th_{OpThVar}$ are thresholds for the response time and throughput, respectively.

The Ramp occurs when the average response time and throughput of the operation increases in n consecutive time slots and the increments overmatch some predefined thresholds.

The critical operation, in the ASSM, is the incremental search in the Knowledge Base. The ASSM process affects all four key performance scenarios since it is called by two subscenarios, Initial Interface Generation and Interface Generation. The former subscenario belongs to the First Interaction scenario, while the latter is present in the Navigation, Device Interaction and Back To Top scenarios.

On the other hand, (Smith and Williams, 2002a) determined the following relation in The Ramp:

$$RT = \frac{i \cdot \frac{ds}{dt} \cdot s}{1 - X \cdot (i \cdot \frac{ds}{dt} \cdot s)} \tag{5.2}$$

where:

- RT is the response time of the operation,
- i is the number of items in the data set of the operation,
- s is the amount of service time required to process a single item,
- $\frac{ds}{dt}$ is the slope of the ramp,
- X is the arrival rate of queries to the operation.

[3]According to EASTIN (`www.eastin.eu`), the principal Assistive Technology Information Network in Europe, the number of Assistive Products available in the EU increased to more than 39.221 products in 2009.

Combining equations 5.1 and 5.2, we get the response time for the operation:

$$RT_{OpI} = \frac{i \cdot (F_{RT}(OpI, t) - F_{RT}(OpI, t-1)) \cdot F_{RT}(OpI, 1)}{1 - X \cdot (i \cdot (F_{RT}(OpI, t) - F_{RT}(OpI, t-1)) \cdot F_{RT}(OpI, 1))} \tag{5.3}$$

Taking the experimental results obtained for the ASSM in Figure 5.18 and applying equation 5.3, we calculated response times, in the four key scenarios, for 100 users. As it can be observed in Figure 5.26, The Ramp antipattern greatly impacts in the response times. The reader should note that estimated response times in Figure 5.17 did not take into account the effect of The Ramp, since the Knowledge Base was very few populated, only with ten Assistive Software (AS) products. Therefore, we detected the impact in this phase since the Knowledge Base was populated with ten thousand AS products[4]. Consequently, the response times in Figure 5.26 are so different from those in Figure 5.17.

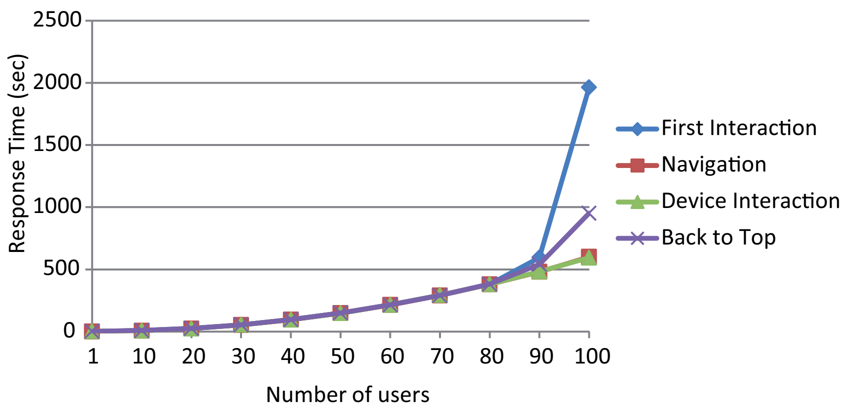


Figure 5.26: Impact analysis of The Ramp antipattern in the response times of key scenarios

To solve this antipattern, both (Smith and Williams, 2002a) and (Dugan-Jr. et al., 2002) propose to select another search algorithm more appropriate for large amount of data. The ASSM is based on a simple filtered search in SPARQL. This search can be improved by changing the recommender process and using a specific "recommend" operator, as (Levandoski et al., 2011) suggest. Thus, the response time for a search performs better in 33%, independently of the number of users. We then recalculate response times for the scenarios considering the improvement in the search algorithm. Figure 5.27 shows the results, which considerably improve those in Figure 5.26.

[4]From all the Assistive Products in the marketplace, we considered those that can be integrated into the architecture, i.e., Assistive Software products.

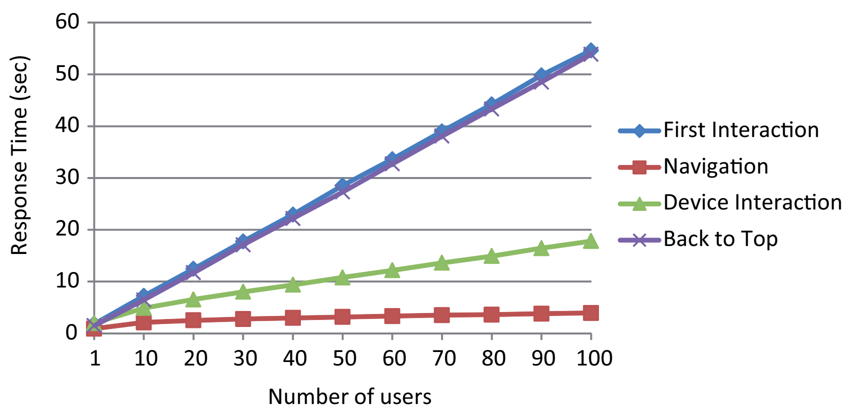


Figure 5.27: Response time applying specific "recommend" operator in search algorithm of ASSM.

5.3.5 Optimal Configuration

Once all the alternatives for improvement were analysed, we applied them to the original configuration of the INREDIS architecture, in order to achieve an optimal configuration. These improvements are summarized in the following:

- *Utilization and Multithreading*: We detected the AMS, the Interoperability Gateway and the Target Service/Device resources as bottlenecks. They were mitigated by adding threads as indicated in Figure 5.24.
- *Performance patterns*: Applying performance patterns helps to improve the software design and the system performance. We identified the Perimeter Calculation subscenario as candidate for the Fast Path pattern, then we refactorized this scenario in order to apply the pattern.
- *Performance antipatterns*: We detected The Ramp antipattern using the logical predicates in (Cortellessa et al., 2012). Then, we analysed its potential consequences and changed the search algorithm in ASSM process.

Figure 5.28 depicts the response times when we applied all the aforementioned alternatives for improvement. As it can be observed, all these improvements help to meet performance objectives based on Usability Principles. Otherwise, another iteration of the assessment loop would have been necessary.

5.3.6 Validation of the Performance Assessment

Once our assessment has produced an optimal design of the architecture, we are committed to apply the improvements to our initial prototype. By doing so we want

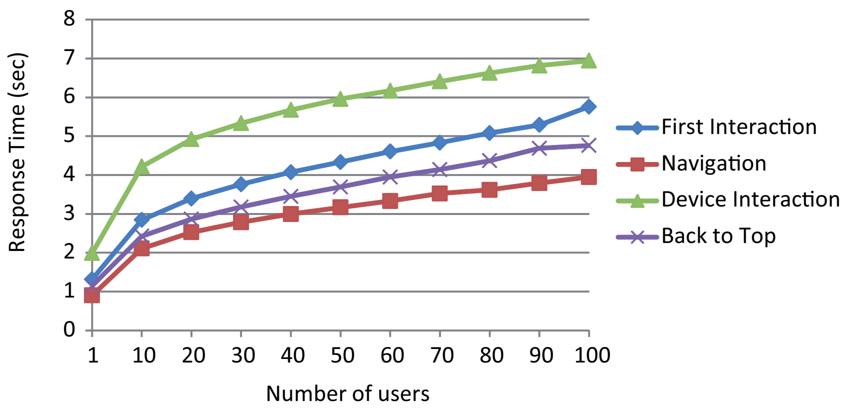


Figure 5.28: Response time for the optimal configuration of the INREDIS Interoperable architecture.

to assess whether the results of the model match the results of the architecture. Next we detail the important issues in the new prototype implementation:

- *Resource replication*: We fully applied to the initial prototype all the proposed improvements. We replicated the resources and applied multithreading to overcome all detected bottlenecks.

- *Performance patterns*: We satisfactorily applied the Fast Path pattern for perimeter calculation, then modifying the location of services and target devices. Hence, this assessment was also completely applied.

- *Performance antipatterns*: We could not apply this assessment in the prototype since the update of the search algorithm was very complex and affected other processes (out of scope of the INREDIS project). However, the effects of the assessed antipattern were almost negligible in the results of the initial prototype because the Knowledge Base was populated with only ten Assistive Products, as previously explained.

Figure 5.29 (last line in the caption) plots the response times of this optimal prototype. The prototype was deployed in the same servers as the initial one and the experiments replicated in the same facility (automation house). As described in Section 5.3.3, the experiments were very difficult to carry out due to several issues (legal, logistic or user selection among others). We could involve three concurrent users, hence the results were extrapolated for five users through a linear function. We observe in Figure 5.29 that the results of our optimal model and those of the optimal prototype are very similar. In some scenarios our model is slightly more optimistic but slightly more pessimistic in others. As mentioned in Section 5.3.3, these low variations between empirical and predicted results might be mainly caused by the accuracy in the computation of the GSPN models.

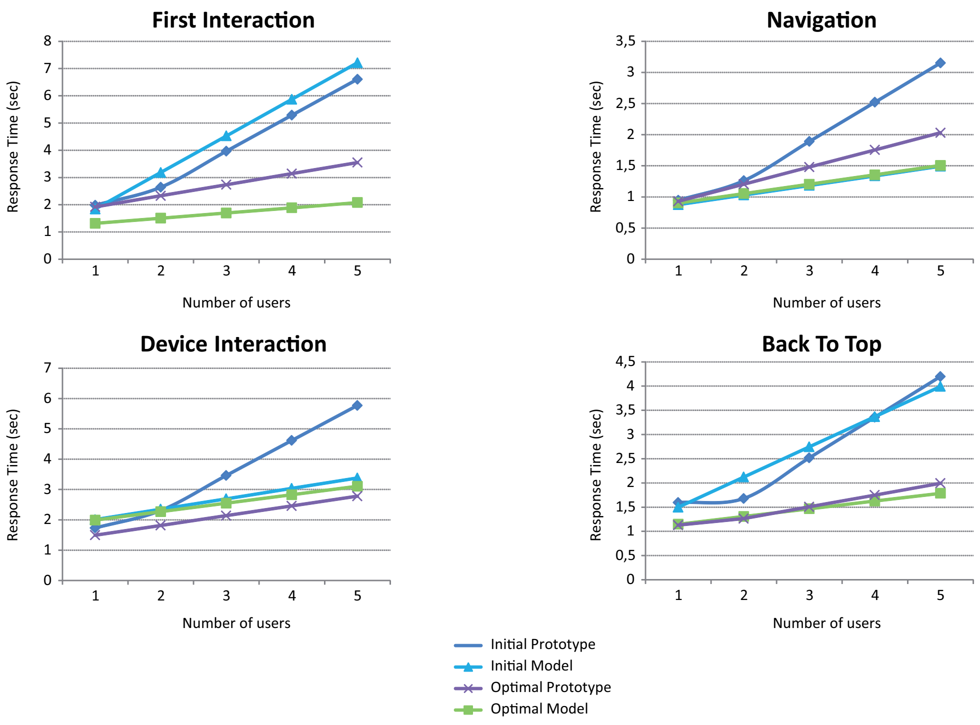


Figure 5.29: Comparison between initial results (prototype and model) and results of the optimal configuration (prototype and model).

5.3.7 Comparison of Results

Figures 5.29 and 5.30 have been introduced to depict a throughout comparison of all the results obtained so far. Figure 5.29 plots for each performance scenario the following information:

- The empirical results obtained with the initial prototype, i.e., it replicates the information presented in Figure 5.16.
- The results we obtained using the initial model, i.e., it replicates the information presented in Figure 5.17, but for five users only, due to the aforementioned limitations of experimental user tests.
- The results we obtained using the optimal model -the model of the optimal

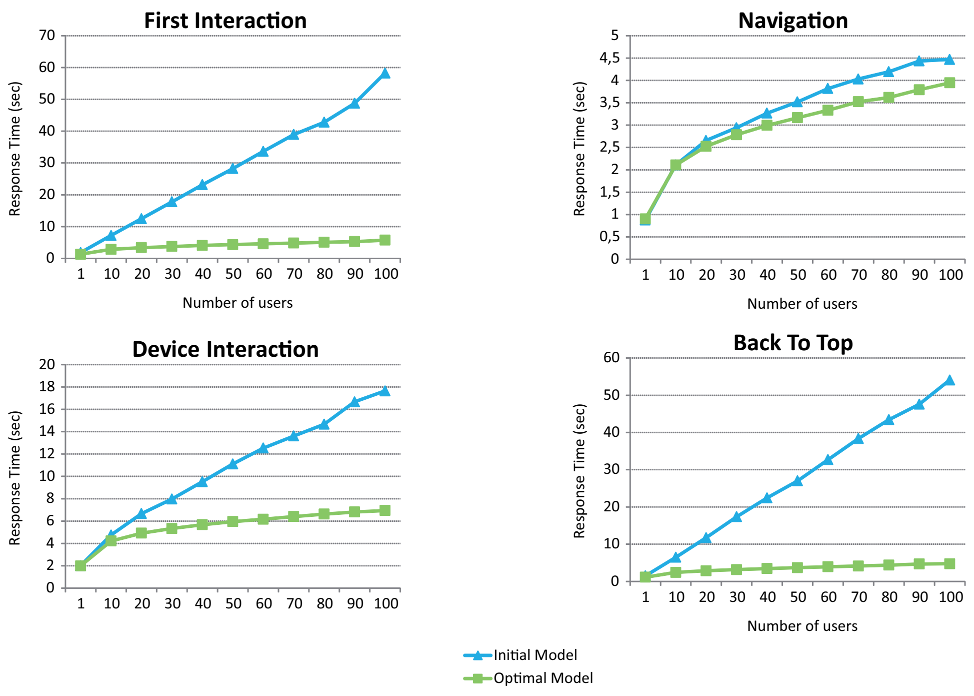


Figure 5.30: Comparison between results of the initial and optimal models.

configuration-, i.e., it replicates the information presented in Figure 5.28, but for five users only.

- The results obtained with the optimal prototype. The optimal prototype is the initial prototype plus the improvements obtained by our assessment. Subsection 5.3.6 explained how we developed the optimal prototype.

Figure 5.29 shows, for all the four scenarios, that the results given by the optimal model improves both, the initial empirical results and the initial model prediction.

The low differences between empirical and predicted results might be mainly caused due to the accuracy in the computation of the GSPN models. Nevertheless, although empirical and predicted data have similar trend for almost all graphs, it is observed that they differ for the Navigation and Device Interaction with the initial prototype, particularly in the latter case. However, for the optimal prototype the trend is the same in all scenarios. We guess that this could be due to imprecisions in gathering data during the testing phase session, -corresponding to the initial prototype-, since there were great variabilities in all actions carried out by users. However, for getting data from the models we do not express such variability since we only used average execution times. All in all, in the first user testing phase, the number of users involved was small but they had not any restriction to interact with the system. However, tests with the optimal prototype were more controlled and precise. Therefore, the quality, more than the quantity, of the empirical data used determined the worth of the predicted results.

Figure 5.30 extends results in Figure 5.29 for 100 users, the information empirically obtained with both prototypes is missing since it could be obtained only for 5 users. Figure 5.30 shows that the results of the optimal model are far better than those of the initial model. These figures clearly demonstrate that the changes, due to the application of the SPE approach, have improved the models to the degree of compliance with the performance requirements for a large number of users, in fact it was our initial objective.

5.4 Discussion

The assessment of software architectures is a process that is acquiring increasing importance in industrial practice. The work carried out allowed us to determine an optimal configuration and, therefore, to improve the final product. In the following, as suggested by (Runeson and Höst, 2009), we discuss the limitations of the results obtained, the lessons learned and the issues disclosed while applying the assessment process, and we also explain some of the consequences of all these matters.

The outcomes of this research can be interpreted from two perspectives. First, we discuss the outcomes explicitly related to the assessment methodology. Second, we analyse the collected data obtained by applying the methodology in order to achieve an optimal configuration.

5.4.1 Concerning the Methodology

Our first objective was to carefully revise each step of the methodology to teach practitioners how we applied these steps, but also to offer a blueprint of the INREDIS project that could be used as a guide for practitioners in future applications.

We discuss issues of the methodology according to the evaluation criteria proposed by Isa and Jawawi (2011), which consider: process related aspects and modeling related aspects. Finally, we also consider issues related to the tools used.

Concerning the process for performance assessment, the SPE methodology is intended to support general-purpose domains. The methodology explicitly influences the development process by focusing on performance properties. In particular, the assessment process manages the system performances from the requirements and analysis phases until the design phase by analyzing a set of key performance scenarios. The methodology systematically defines all the steps needed to discover potential performance problems and how to mitigate them.

Another aspect related to the process concerns to the tradeoffs that the engineer needs to consider for achieving performance. Bass et al. (2005) defend that quality attributes can never be achieved in isolation, the achievement of any one will have an effect, sometimes positive and sometimes negative, on the achievement of others. In fact this happened when we applied the SPE methodology, the improvement of system performance influenced other quality attributes, such as maintainability or cost, and in the worst case, the improvement of performance decreased other quality attributes. In particular, for improving performance we introduced performance patterns, in this case the tradeoff positively influenced the maintainability of the system since it is widely recognized the benefit of using design patterns for this quality of the system. Regarding antipatterns we can say the same. They capture design errors, therefore, by using them we not only gain in system performance but eventually in maintainability and system testability. On the other hand, when we replicated resources, we incurred in a cost, i.e., the influence of improving performance was negative. For example, replication of CPU capacity implies a monetary cost, while multithreading implies a software more difficult to test and maintain.

Concerning the modeling criterion, it analyses how the performance requirements and system functionalities are specified and developed. Being the approach centered on the architectural level, it perfectly captures the system structure model definition and the behavioural issues, then allowing assessment of the complete software architecture. As mentioned, we used a simplified version of the PUMA methodology (Woodside et al., 2005, 2013), which addresses a systematic performance modeling with the support of UML, which allows annotations with MARTE profile for the performance properties. Nevertheless, the application of the complete PUMA approach we did of some performance modeling aspects is limited, mainly due to the characteristics of the INREDIS project. One of these characteristics is the discrete number of users and resources, thus, the workload identification only considers closed ones with discrete values. A similar situation occurs with resources, since we did not model some low level issues, such as random access memory, disk storage or cache memory.

Concerning the tools, we used ArgoSPE (Gómez-Martínez and Merseguer, 2006a), an ArgoUML[5] plugin, to partly automate the assessment process in a transparent way for software architects. Unfortunately, performance annotations supported by ArgoSPE are not in MARTE, but in (OMG, 2005) profile format. Thereby, we had the choice of translating the annotations into UML-SPT or to introduce some performance parameters in the GSPN manually. For example, the number of system resources, we solved it looking at the GSPN places representing resources, such as *pATS*, *pWS* or *pKB* in Figure 5.11, then populating them with as many tokens as resources indicated in the MARTE annotation *resMult* (Figure5.1).

ArgoSPE internally calls GreatSPN (Chiola et al., 1995) to analyse or simulate GSPNs. We used the simulation programs since the large size of the models prevented the analysis programs. The problem stems from the reachability graph of the GSPN. Simulation outcomes can be obtained almost immediately for simple samples. However, the computation times for some of the results obtained in this chapter consumed long time (several hours), even some of them lasted for a couple of weeks. Concretely, those for which the number of concurrent users was greater than 50 and some resources were multithreaded. To reduce the computation times, we decreased

[5]http://argouml.tigris.org/

the simulation default accuracy, i.e., the precision of the approximation in the parameters estimation. This reduction affected the response times, i.e, the results we obtained, in the order of ±10 milliseconds. Although this significantly decreased the computation times, few of the experiments lasted for two or three hours yet.

Furthermore, ArgoSPE lacks other plugins, such as tools for identifying performance patterns and antipatterns automatically. Thus, we had to use external applications manually, as those developed by (Cortellessa et al., 2012). Therefore, ArgoSPE is still a very limited tool for performance assessment, specially for complex case studies such as the INREDIS architecture. Consequently, we have detected the need for developing a new framework which integrates all these functionalities: UML modeling, GSPN simulation and analysis, patterns and antipatterns detection in a transparent way to the user and efficient computation times. This framework could also include assessment of other functional and non-functional properties, such as dependability, security or model checking.

5.4.2 Concerning the Optimal Configuration

Our second objective tried to reveal how good the architecture proposed by the INREDIS software engineers was, from the performance point of view exclusively. The performance results demonstrated that the original architecture and configuration fitted for limited environments with very few concurrent users. This usage scenario might occur, for example, when users interact with electronic devices at smart homes or students in a classroom. Nevertheless, our results disclosed that this configuration performs poorly in contexts with several concurrent users. Thus, we systematically assessed system performance by changing the software design and the configuration, concretely adding threads and refactoring some components. These improvements helped to meet performance objectives as well as to scale the system in more challenging performance usage scenarios, such as web services, urban networking, hospital and/or retirement homes, where multiple users with different capabilities can simultaneously access the system.

However, some limitations are still unsolved. First of all, in real implementations, the Knowledge Base had not been fully populated with users preferences and capabilities, Assistive Software products in the ASSM and interaction modes. Therefore, a specific sensitivity performance analysis of the Knowledge Base and ASSM would be desirable.

A more detailed study of the target devices or services would also be desirable, since both their usage and their corresponding functionalities can affect the architecture. In this chapter, we have assumed that this time is negligible, since it is independent of the architecture (e.g., the whole cycle time of a washing machine is very different from a TV set), and obviously we have to take it as an external and non-controllable part of our system. However, real implementations did not consider increments in the number of devices or services in the user perimeter. Thus, as noted in Sec. 5.3.4 through our experiments, the number of nested iterations in the Perimeter Calculation depends on the amount of available target devices and services and their functionalities.

Finally, as above mentioned, the architecture was implemented considering most flexible and cutting-edge technologies at that moment. However, some of the communication protocols used had poor performance, such as the ESB (Enterprise Service Bus) architecture designed by Chappell (2004) combined with services implemented in SOAP (Liu et al., 2007). A similar situation can be found in the `Knowledge Base` component, as observed by (Liang et al., 2009). Consequently, our performance models considered the measured times of these prototype implementations. In particular, we used them as host demands for the Petri net transitions. However, it would be feasible to include an additional stage in our performance assessment proposal, which carries out sensitivity analysis to assess technological alternatives for implementation.

5.5 Related Work

Software architecture assessment constitutes an important stage in the software design process, in order to guarantee non-functional requirements. Nevertheless, to the best of our knowledge, there are very few initiatives to assess architectures based on SPE principles at industrial level. An exception is the PASA (Performance Assessment of Software Architectures) method, proposed by Williams and Smith (2002).

PASA, focussed on performance scenarios, is a performance-based software architecture analysis method that provides a framework for the whole assessment process. PASA inspired us in order to automatically systematize the process to detect performance issues, as well as to propose the corresponding potential solutions. As in PASA, our methodology, carries out performance analysis considering responsiveness, but also resource utilization and scalability. Moreover, we have included automatic detection of performance patterns and antipatterns, by considering the work of Cortellessa et al. (2012). As above stated, PUMA (Woodside et al., 2005, 2013) also guided our work.

Pooley and Abdullatif (2010) defined Continuous Performance Assessment of Software Architecture (CPASA). This method adapts PASA to the agile development process. To the best of our knowledge, CPASA has not been applied to an industrial case yet.

Regarding industrial experience reports that assess performance at architectural level, we have found a few.

Kauppi (2003) conducted a case study using PASA for analyzing mobile communication software systems. They used Rate Monotonic Analysis and layered queuing networks (LQN) (Woodside et al., 1995) instead of Petri nets for system analysis. Results of improvements were not explicitly given due to the confidentiality of the project.

Kozoliek et al. (2012) reported their experience on performance and reliability analysis in a large-scale control system. They applied the method Q-ImPrESS (2009) (Quality Impact Predictions for Evolving Service-oriented Systems), which is supported by an IDE that combines tools for creating and editing models, performing predictions, and conducting a tradeoff analysis. LQNs were used for performance prediction, results were impressive for throughput estimation since they deviated only around 0.2 percent. The authors explain that such good results were obtained because resources were not saturated.

Huber et al. (2010) described an industrial case study where they applied the Palladio Component Model (Becker et al., 2009) to a storage system. A model was firstly implemented, next they conducted several experiments on a prototype to derive the resource usage of each model component and finally, the model was calibrated with realistic resource demands and validated.

Kounev (2006) modelled a new industry-standard benchmark for measuring the performance and scalability of J2EE hardware and software platforms. In this work, Petri nets are used as performance model. The methodology is also based on SPE principles, however they did not explicitly use patterns and antipatterns, as we do. They could implement the system and the models accurately reflected the real system performance.

The work of de Gooijer et al. (2012) re-architects a legacy system, for remotely diagnose industrial devices, in ABB company. The goal was to improve system performance and scalability. The problems addressed to re-architect a system for performance are very different, although not easier, to those to design for performance from scratch, as it was our case. They could start from real system measurements to calibrate their models, which were constructed using the Palladio Component Model and translated into LQNs. They used the PerOpteryx (Koziolek et al., 2011) tool to find new architectural candidates, in contrast we used patterns and antipatterns.

Jin et al. (2007) developed an approach that combines benchmarking, monitoring and performance modeling for database-centric legacy information systems. As in the work previously analysed, important challenges relate to measuring the production system to calibrate the model. They could not match their predictions with the planned system since the implementation was not ready, but established an accuracy of their models within 8%. This work uses a performance model different to ours, in particular they use LQNs. Also different is the application domain, concretely they target the approach towards “legacy systems” in the database field.

Concerning related work about adaptive interfaces for people with special needs, as we have stated, the INREDIS architecture further develops the idea of Universal Control Hub (UCH) proposed by Zimmermann and Vanderheiden (2007)(which is also aligned with the initial ideas that Llinás et al. (2009) propose on how disabled people can take advantage of adaptative interfaces from the ubiquitous computing perspective).

Regarding to practical approaches for automatizing the interface adaptation and assistive software selection processes, initially, the approach that we propose bares similarity with the work done by Kadouche et al. (2009), namely the SMF (Semantic Matcher Framework). But even though we share the use ontologies for representing the elements (both implemented in the OWL (W3C, 2012) representation language), on the one hand our approach makes the reasoning at a class level to reason with taxonomies of concepts and relationships; and the other hand, we take into account assistive software in our process whereas the SMF does not. Also related to the presented work, since they use different Artificial Intelligence techniques for similar tasks, Chi et al. (2012) provide a solution just for the problem of assistive software selection based on a decision; Cortés et al. (2003) propose the use of a Multi-Agent System to controlling and configuring a very specific assistive technology instance, an electric wheelchair, and the intelligent environment that surrounds it; and finally, Woodcock et al. (2012) propose a decision support system developed to assist in the planning and evaluation of assistive technology, but not like our approach for end users in usage scenarios, but for assistive technology market stakeholders decision support.

Finally, to the best of our knowledge, there is no literature concerning the performance of such architectures that realize adaptive interfaces for impaired people.

5.6 Conclusions

As conclusion, we have assessed the INREDIS architecture for performance. This software architecture tries to provide a global solution for universal access to disabled and elderly people with special needs. It automatically adapts user interfaces for both UCH devices and web services, according to users' needs and preferences, improving their accessibility. In addition, an AS selection mechanism enabling to automatically select the most suitable AS has been embedded.

The results of the external objective lead us to conclude that the SPE methodology effectively helps the software architect to improve designs. Besides, UML and MARTE are languages that can address performance specification challenges, however, tools for specification and analysis are not mature enough. The integration of the specification and analysis tools, to carry out the whole cycle, is also a weak point. The results of the internal objective helped to improve the system response time by refactoring extensive parts of the design. The initial design, proposed by INREDIS software engineers, only met performance requirements in one out of the four main system scenarios. After the refactoring process, all system scenarios met the required response time. For the case of one hundred users, the better results were obtained in the First Interaction and Back to Top scenarios. The former was reduced from 60 seconds to 6, while the latter from 55 to less than 5 seconds.

We believe that the results gathered in this chapter are relevant for both, researchers and SPE practitioners. From the research viewpoint, we provide evidence that the ideas and theory behind SPE can be applied for assessing and improving a large software architecture in an industrial project. SPE practitioners, which are the target of our work, can use this industrial report as a blueprint, it can help them to develop a strategy, for assessing the performance of a software architecture, according to the needs of their projects. Furthermore, other target audience can be interested in this paper, such as accessibility experts or user experience designers.

Chapter 6

The ArgoSPE Tool

A step forward for the application of the methodology, presented and applied in previous chapters, should be the development of tools that support it. The availability of these tools is a necessary condition for the industry in order to apply the aforementioned methodology and to discover the feasibility of the SPE research field. The UML-SPT (OMG, 2005), being concerned about the problem, proposed the main steps and modules that any SPE tool should follow.

In this chapter, we present ArgoSPE in (Gómez-Martínez and Merseguer, 2006a), a tool for the performance evaluation of software systems in the first stages of the development process. It is based and implements most of the features given in (Bernardi and Merseguer, 2007; Distefano et al., 2011). The system is modeled as a set of UML diagrams, annotated according to the UML-SPT, which are translated into GSPN (Ajmone Marsan et al., 1995).

ArgoSPE has been used to model and analyse several software systems (Bernardi and Merseguer, 2006; Gómez-Martínez and Merseguer, 2006b; Gómez-Martínez et al., 2007; Gómez-Martínez and Merseguer, 2010; Gómez-Martínez et al., 2013a,b), as well as the samples presented in Chapter 3 and the industrial case study described in Chapter 5.

The rest of the chapter is organized as follows. Section 6.1 presents the most interesting features of ArgoSPE, while Section 6.2 focuses on its software architecture. Section 6.3 surveys those tools developed to analyse performance of software systems, based on the UML-SPT. Finally, Section 6.4 gives some conclusions.

6.1 ArgoSPE Features

From the user viewpoint, ArgoSPE is driven by a set of "performance queries" that s/he can execute to get the quantitative analysis of the modeled system. We understand that a performance query is a procedure whereby the UML model is analysed to automatically obtain a predefined performance index. The steps carried out in this procedure are hidden to the user. Each performance query is related to a UML diagram where it is interpreted, but it is computed in a GSPN model obtained by ArgoSPE automatically. The UML diagrams used to obtain a performance model by means of ArgoSPE are some of those considered in our process: statemachines (SM), UML Activity Diagrams (AD) and interaction diagrams. The use case diagram is taken into account in the process, but it has not been considered in ArgoSPE yet. The class and the implementation diagrams (components and deployment) are used to collect some system parameters (system population or network speed).

Moreover, the performance analyst, that has expertise in Petri net modeling and analysis, can use the GreatSPN (Chiola et al., 1995) tool to compute domain specific metrics using the GSPN models, that ArgoSPE generates automatically.

6.1.1 Queries in the Statechart Diagram

A statechart models the behaviour of a class. Figure 6.1 depicts an example of statechart that models a Consumer class and a very simplified version of its GSPN translation.

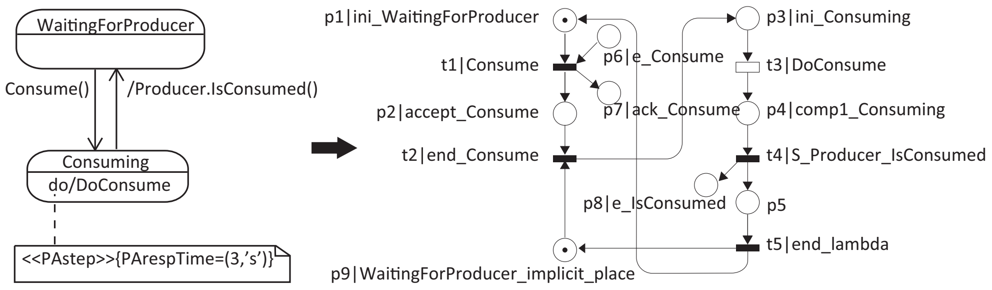


Figure 6.1: Statechart and its corresponding GSPN.

The queries in the statechart diagram supported by ArgoSPE are the following:

- **State population**. This query computes the percentage of objects in each state. For example, in Figure 6.1, the 40% of the objects could be `Consuming` and the others `WaitingForProducer`.

  The query can be useful to detect saturated software processes or to know how an agent shares out its execution among different tasks (states). The state population is obtained by dividing the number of objects in the state among the mean number of objects that populate the class.

  For instance, in the state `WaitingForProducer`, the *State population* is computed by dividing the mean marking of place `p9` among the initial marking of the net in place `p1`.

- **Stay time**. Represents the mean time that the objects of a class spent in each state. For each state, this value is computed by dividing the mean number of objects in it among its throughput, therefore, applying the Little's Law.

In the example of Figure 6.1, the *Stay time* of the state `WaitingForProducer` represents the mean time that a `Consumer` spends waiting for consuming, and it is computed by dividing the mean marking of place `p9` among the throughput of the transition `t5`.

- **Message delay**. When the sender and the receiver of a message reside in different physical nodes, this query calculates the time spent by the message to reach the receiver's node. This value is straightforward calculated by dividing the size of the message among the network delay (see Section 6.1.2).

6.1.2 Queries in the Deployment and Collaboration Diagrams

The UML Deployment Diagram specifies the execution architecture of a system. Hardware resources are represented as nodes where software components can be deployed. Moreover, the physical network connections are modeled as relationships between nodes. A query in the UML Deployment Diagram is supported by ArgoSPE:

- **Network delay**. Calculates the network delay (bit rate) between two non adjacent hardware resources (nodes). Given a system configuration, the network delay is useful to find the node where a new software component could be deployed, i.e., the node that minimizes the delay of the component's messages.

The collaboration diagram is an interaction diagram that focuses on how objects exchange messages. It describes the behaviour of the system in a specific context (scenario). The following query in the collaboration diagram is supported by ArgoSPE:

- **Response time**. For a given collaboration diagram (system scenario), this query computes its mean response time, i.e., the mean duration of a certain system execution.

6.1.3 Performance Annotations

ArgoSPE uses as input a UML-SPT annotated model, i.e., UML models have to explicitly include performance characteristics. These performance annotations, defined in the UML-SPT, are made by means of the UML extension mechanisms: *stereotypes* and *tagged values*.

The *stereotypes* specify the main performance characteristics of the UML model elements, while the *tagged values* specify the attributes of the stereotypes. As an example, see the performance annotation in Figure 6.1, where the stereotype `PAstep` means that `DoConsume` is a computation step, while the tagged value `PArespTime` models its response time, three seconds.

The UML-SPT defines a *Tag Value Language* (TVL), a subset of the Perl language, that allows to specify complex and parameterized expressions in the tagged values.

The annotations supported by ArgoSPE are those necessary to compute the proposed performance queries, see Table 6.1.

Table 6.1: Performance annotations in ArgoSPE.

Annotation	**Stereotype**	**Tagged value**	**Unit**	**Model elements and Diagrams**
Activity duration	PAstep	PArespTime	ms, s, m, h	Activities in the SC and AD
Probability	PAstep	PAprob	-	Transitions in SC and AD. Messages in the Coll.
Size	PAstep	PAsize	b, B, kb, kB, Mb, MB	Messages in the SC and Coll.
Network speed	PAcommunication	PAspeed	bps, Bps, kbps, kBps, mbps, MBps	Deployment
Population	PAclosedLoad	PApopulation	-	Class in Class diagram
Initial state	PAinitialCondition	PAinitialState	$true or $false	State in the SC and AD.
Resident classes	GRMcode	-	-	Deployment

6.2 Software Architecture

ArgoSPE has been implemented as a set of Java modules, that are plugged into the open source ArgoUML CASE tool (ArgoUML, 2013). It follows the architecture proposed in the UML-SPT (OMG, 2005), see Figure 6.2.

The ArgoUML works as the Model Editor, while the ArgoSPE modules implement and coordinate the Model Configurer and the Model Processor (Model Convertor, Model Analyzer and Results Convertor) functions. Figure 6.3 depicts the ArgoSPE menu inside the menu bar of ArgoUML.

6.2.1 Model Editor

The Model Editor is used to create and modify performance-annotated UML diagrams.

ArgoUML permits to model and to annotate the UML diagrams involved in the translation process (Merseguer et al., 2002; Bernardi et al., 2002; López-Grao et al., 2004). ArgoUML, as most CASE tools, exports UML models into XMI (OMG, 2011c) files, allowing the standard exchange of information with another tools.

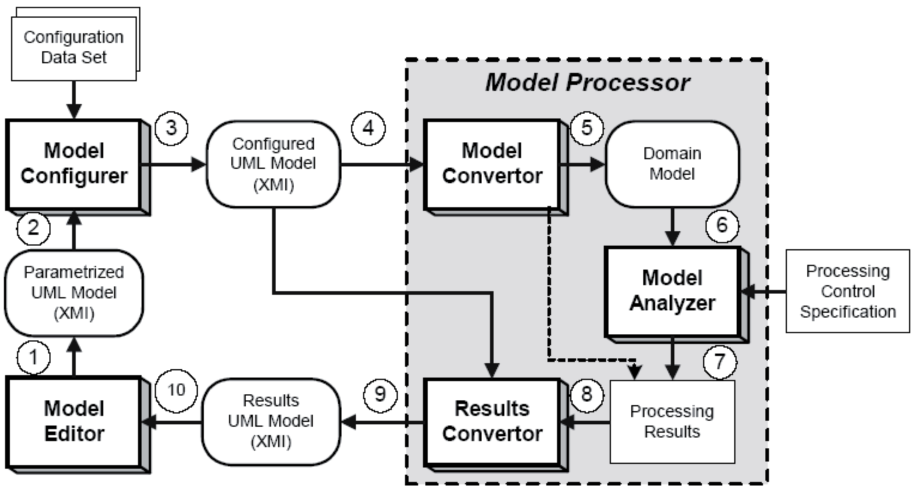


Figure 6.2: Architecture proposed in the UML-SPT OMG (2005).

From the performance-annotated UML diagrams, ArgoSPE creates a parameterized XMI file, that will be an input for the Model Configurer. This XMI file contains the modeling and performance information of: the statecharts, describing the behaviour of the classes in the system; the UML Activity Diagrams, specifying the activities of the statecharts; the UML Deployment Diagram, that gathers information about physical nodes location and network transmission speed; the class diagram with information about the system workload and the collaboration diagram.

6.2.2 Model Configurer

The Model Configurer functionality, see Figure 6.2, consists in converting a parameterized UML model in XMI format, into a configured UML model using a configuration data set. The main target is to substitute in the XMI file, the tagged values written in TVL with parameterized expressions that represent the performance annotations, for the equivalent evaluated expressions.

The first task is to parse the XMI file, obtaining a tree structure, called Document Object Model (DOM) (W3C, 2009). This one is visited recursively from its root node into their children nodes to search for performance annotations. So, a list of XMI identifiers with known stereotypes is extracted. For each element in that list, a TVL expression is obtained, some of them with variables, that will be evaluated, then modifying the tree. At the same time, a symbol table is created containing the performance annotations. Finally, the tree is serialized to an XMI file.

Since the TVL expressions can contain variables, ArgoSPE prompts the user to choose a configuration file containing a configuration data set. An example of this

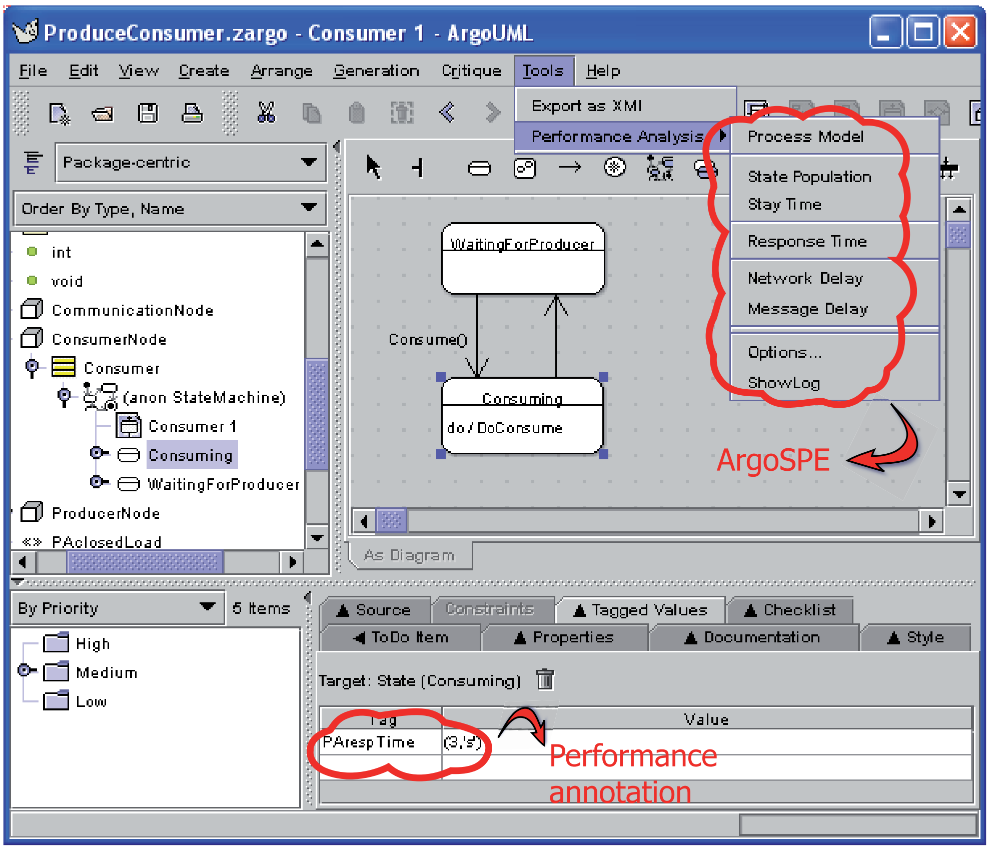


Figure 6.3: ArgoSPE menu inside ArgoUML.

kind of file is depicted in Figure 6.4.

ArgoSPE evaluates this file by invoking a Perl interpreter to get the actual value for variables and expressions. Then, for a performance annotation like `<<PAstep>>{PArespTime=($value,'s')}` the variable `$value` will be evaluated and replaced according to the configuration file (e.g. the one in Figure 6.4), so the evaluated expression will be `<<PAstep>>{PArespTime=(5,'s')}`. Multi-valued expressions are supported by ArgoSPE.

6.2.3 Model Processor

The Model Processor turns the configured model, obtained from the Model Configurer, into an analyzable model (GSPN model), analyses it and returns the results. These tasks are respectively addressed by the Model Convertor, the Model Analyzer and the Results Convertor.

```
#A very simple configuration file written in Perl
$value=5;
$value2=10;
$value3=($value<40)?100-$value2:100;
```

Figure 6.4: A configuration file example for ArgoSPE.

Model Convertor

The Model Convertor module encapsulates the translation process from the configured model into the target performance formalism. Therefore, in ArgoSPE this is a heavy process that implements the translation theory proposed in (Merseguer et al., 2002; Bernardi et al., 2002; López-Grao et al., 2004; Merseguer, 2003). The GSPN models are obtained in the GreatSPN file format (Chiola et al., 1995).

The high-level algorithm implemented in the tool for the Model Convertor is illustrated in Algorithm 1. Note that it needs as inputs not only the configured XMI, as proposed in the UML-SPT, also the symbol table, that allows to speed up the translation process, but it increases the module coupling.

Lines 2 to 14 correspond to the translation process of the statecharts and its associated UML Activity Diagrams. Then, a GSPN, called `SysGSPN`, that models the whole system behaviour is obtained by merging the Petri nets of the classes (line 26). The translation of the collaboration diagrams is described from lines 17 to 24. The result is a set of GSPNs, $\texttt{Scenario}_i\texttt{GSPN}$, each one modeling the scenario specified by the collaboration diagram i.

Model Analyzer

The Model Analyzer implements the performance queries described in Sections 6.1.1 and 6.1.2. Concretely, the queries for the statecharts are computed using the `SysGSPN` net. While the *Response time* query for a collaboration diagram i is computed in the $\texttt{Scenario}_i\texttt{GSPN}$ net.

The Model Analyzer invokes the GreatSPN programs to get the answers to the queries. ArgoSPE currently uses the GreatSPN programs that implement analytical/numerical techniques. The use of GreatSPN simulation techniques will be considered to complement the results.

Results Convertor

The main function of the Results Convertor is to convert the results of the analysis back to the UML Model Editor, in a way that a software engineer can interpret them easily.

In the current version of ArgoSPE, the returned results are directly displayed by the Model Editor in a simple message window, then not directly in the UML models.

Algorithm 1 Model Convertor

```
Require: A configured XMI file and a symbol table
Ensure: A set of GreatSPN models (SysGSPN, Scenario$_i$GSPN)
 1: UML diagrams ← XMI file
 2: for all class c ∈ Class diagram do
 3:   if c has Statechart then
 4:     node ← Locate node in UML Deployment Diagram(c)
 5:     A ← Activity diagrams associated with c
 6:     for all Activity ac ∈ A do
 7:       pa ← Annotations associated with ac ∈ symbol table
 8:       acGSPN[j] ← TranslateToGSPN(ac,pa)
 9:     end for
10:     sc ← Statechart associated with c
11:     pa ← Annotations associated with sc ∈ symbol table
12:     scGSPN ← TranslateToGSPN(sc,pa,node)
13:     clGSPN[k] ← Merge(scGSPN, acGSPN[])
14:   end if
15: end for
16: SysGSPN ← Merge(clGSPN[])
17: for all class c ∈ Class diagram do
18:   C ← Collaboration diagrams associated with c
19:   for all Collaboration co ∈ C do
20:     pa ← Annotations associated with co ∈ symbol table
21:     coGSPN[i] ← TranslateToGSPN(co,pa)
22:     Scenario$_i$GSPN ← Merge(SysGSPN,coGSPN[i])
23:   end for
24: end for
```

6.3 Related Work

A number of performance evaluation tools based on Petri nets have been developed in the last decade, such as Möbius (Clark et al., 2001), GreatSPN (Chiola et al., 1995) or TimeNET (Zimmermann et al., 2000). But in this chapter, we only revise and compare those tools that focus on the SPE field.

DSPNexpress-NG (Lindemann, 1995), proposed by Lindemann et al., constitutes a framework that can evaluate both discrete-event systems specified as Petri nets and UML system models. It uses UML statecharts which are not annotated according to the UML-SPT, and transforms them into deterministic and stochastic Petri nets (DSPNs) to obtain numerical solutions.

Distefano et al. (2004) developed a performance plug-in for ArgoUML. Following the UML-SPT, they focus on use cases, deployment and UML Activity Diagrams and introduce an intermediate model, which is used to gather performance information. This intermediate model is transformed into SPN and analysed with a web-based non-markovian Petri net tool.

Using formalisms different from Petri nets, Petriu and Shen (2002) proposed an algorithm to transform UML Activity Diagrams into LQN models. They obtained the XML files from existing UML tools, and changed them by hand in order to add performance annotations to the different model elements. The tool of Gilmore and Kloul (2003) uses ArgoUML to compile statecharts and collaboration diagrams through a process algebra language. D'Ambrogio (2005) introduced a framework to automatically translate LQN models from annotated activity and UML Deployment Diagrams. Cortellessa et al. (2004) proposed a tool that in two phases gets a parameterized QN from use cases, UML Sequence Diagrams and UML Deployment Diagrams. Marzolla and Balsamo (2004) transformed annotated use cases, deployment and UML Activity Diagrams into a discrete-event simulation model.

6.4 Conclusions

Petri nets are recognized as a useful modeling paradigm for the performance evaluation of a wide range of systems. Nevertheless, most software engineers do not feel comfortable far from their pragmatic (non formal) modeling languages, such as UML. Moreover, engineers find easier and more productive to use only one modeling paradigm for all the project stages. Since ArgoSPE obtains GSPNs as a "by-product" of the software life-cycle, software engineers can use their UML models to assess system performance properties.

A number of new features can improve the tool: First, the more system properties assessed the more useful the tool become. New performance queries have to be implemented. Second, a standard format, PNML (ISO/IEC, 2011a), could be the target file format, then gaining the possibility to use other Petri net analysers.

Chapter 7

Final Conclusions and Future Work

At the beginning of our efforts, it was demonstrated the necessity to systematically analyse performance of distributed software architectures in order to obtain the foundations for the early detection of potential performance issues and the subsequent assessment to improve those architectures. In addition, the performance assessment should be obtained as a "by-product" of the development life-cycle and their analysis results back to the design model.

To the best of our knowledge, there are very few initiatives to assess architectures based on SPE principles hitherto. An exception is PASA (Williams and Smith, 2002). Nevertheless, this methodology relies some steps of the process, such as architectural style or detection of performance antipatterns, on skills and experience of the software analyst. Moreover, the interpretation of results of performance analysis and the feedback generation is hard, since they are represented in the performance model. CPASA (Pooley and Abdullatif, 2010) is an adaptation of PASA to the agile development process, but it does not solve these issues. Moreover, these methodologies have been applied in examples or academic studies, but not in an industrial setting. Thus, the results and conclusions could be limited.

7.1 Achievements

The aim of this dissertation thesis is to devise a methodology for systematically enhancing performance assessment of software architectures. In order to accomplish this objective, we have analyzed diverse software architectures using SPE techniques and interpreted the obtained performance results, as well we have studied the impact of some technological decisions. Once we achieved a deep insight in the performance analysis of software architectures and a certain knowledge on closing the "assessment loop" by interpreting the performance results, we have devised our methodology for assessing performance of software architectures. A list of the main scientific contributions of this thesis is given in the following.

- **The application of SPE techniques for the performance analysis of diverse software architectural styles**. We have applied SPE techniques, concretely the PUMA approach (Woodside et al., 2005, 2013), for the performance analysis of diverse software architectural styles: mobile agents platforms, web service-based applications and a remote console framework. The interpretation of the obtained performance results, as well as the detection of performance issues, have allowed us to gain insight in the performance evaluation of hypothetical situations, beyond the real implementation. Moreover, the performance analysis allows us to exploit potential improvements in the aforementioned software architectures.

  These performance analyses are detailed in Chapter 3. The outcome of this acquired knowledge has permitted us to devise the methodology proposed in Chapter 4. This research has been published in (Gómez-Martínez and Merseguer, 2006b; Gómez-Martínez et al., 2007; Gómez-Martínez and Merseguer, 2010).

- **A methodology for performance assessment of software architectures**. We have developed a scenario-based methodology to apply SPE principles and techniques to assess software architectures. The proposed methodology is a part of the software development life-cycle in early stages, being the performance assessment a "by-product". It tries to automatize the expertise achieved in the previous contribution regarding software performance: evaluation, issues detection and assessment to attain an optimal configuration.

  This methodology is described in Chapter 4 and it is obtained as a deep insight in the performance analysis of diverse case studies presented in Chapter 3. The methodology is applied for the performance assessment of an industrial case study in Chapter 5. This research has been accepted to publication (Gómez-Martínez et al., 2013a).

- **An interoperable architecture for people with special needs**. We have designed and implemented an interoperable architecture to adapt interfaces (device controllers or web services) according to the capabilities or preferences of people with special needs.

  The main components of this industrial case study are described in Chapter 5. This research has been accepted to publication (Gómez-Martínez et al., 2013a) and submitted to (Gómez-Martínez et al., 2013b) and some of the components have been published in (Gómez-Martínez and Merseguer, 2010; Iglesias-Pérez et al., 2010; Murua et al., 2011).

- **The application of the methodology in an industrial case**. We have systematically applied the above mentioned methodology for the performance assessment of the previous interoperable architecture in order to obtain its optimal configuration. Moreover, the theoretical performance results have been validated against the experimental ones obtained in the user testing phase. In addition, other hypothetical situations have been studied.

Chapter 5 details the application of the methodology for performance assessment of this interoperable architecture. This research has been accepted to publication (Gómez-Martínez et al., 2013a).

- **Development of a tool for performance evaluation based on SPE principles**. A tool for the automation of the performance evaluation of software systems in the first stages of the development process has been developed. It is based and implements most of the features given in (Bernardi and Merseguer, 2007; Distefano et al., 2011). This tool permits to expedite the performance analysis of software systems.

  The tool, named ArgoSPE, has been implemented as a set of Java modules, that are plugged into the open source ArgoUML CASE tool (ArgoUML, 2013). It follows the architecture proposed in the UML-SPT (OMG, 2005).

  ArgoSPE was used for the performance evaluations carried out in Chapter 3 and Chapter 5 and it is detailed in Chapter 6. This research has been published in (Gómez-Martínez and Merseguer, 2005, 2006a).

This thesis also reported the first performance evaluation based on SPE techniques and patterns to analyse and assess an industrial case study, concretely an interoperable architecture for impaired people. In addition, SPE practitioners, which are the target of our work, can use this industrial report as a blueprint, it can help them to develop a strategy, for assessing the performance of a software architecture, according to the needs of their projects in future applications.

7.2 Scientific Results

The work of this thesis has produced the following papers, being the author of the present dissertation thesis the principal contributor in all of them (except for 5).

1. *Journal paper*:
   Elena Gómez-Martínez, Rafael González-Cabero and José Merseguer. Performance Assessment of an Architecture with Adaptative Interfaces for People with Special Needs. *Empirical Software Engineering. Accepted for publication.* 2013

2. *Journal paper*:
   Elena Gómez-Martínez, Marino Linaje, Fernando Sánchez-Figueroa, Andrés Iglesias-Pérez, Juan Carlos Preciado, Rafael González-Cabero and José Merseguer. Interacting with inaccessible smart environments: Conceptualization and evaluated recommendation of Assistive Software. *Submitted.* 2013

3. *Conference paper*:
   Ane Murua, Igor González and Elena Gómez-Martínez. Cloud-based Assistive Technology Services. *Proc. of the 3rd Workshop on Software Services:*

Semantic-based Software Services (WoSS'11) at the Federated Conference on Computer Science and Information Systems (FedCSIS 2011). Pages 985-989. 2011

4. *Conference paper*:
   Elena Gómez-Martínez and José Merseguer. Performance modeling and analysis of the Universal Control Hub. *Proc. of the 7th European Performance Engineering Workshop (EPEW 2010).* Lecture Notes in Computer Science (Vol. 6342). Pages 160-174. 2010.

5. *Conference paper*:
   Andrés Iglesias-Pérez, Marino Linaje, Juan-Carlos Preciado, Fernando Sánchez-Figueroa, Elena Gómez-Martínez, Rafael González-Cabero, José A. Martínez-Usero. A Context-Aware Semantic Approach for the Effective Selection of Assistive Software. *Proc. of the 4th Symposium of Ubiquitous Computing and Ambient Intelligence (UCAmI 2010)* Garceta. Pages 51-60. 2010.

6. *Conference paper*:
   Elena Gómez-Martínez, Sergio Ilarri and José Merseguer. Performance Analysis of Mobile Agent Tracking Approaches. *Proc. of the 7th International Workshop on Software and Performance (WOSP 2007).* ACM. Pages 181-188. 2007.

7. *Conference paper*:
   Elena Gómez-Martínez and José Merseguer. Impact of SOAP Implementations in the Performance of a Web Service-based Application. *Proc. of the Workshop on Middleware and Performance (WOMP 2006) at the International Conference on Frontiers of High Performance Computing and Networking (ISPA 2006).* Lectures Notes in Computer Science (Vol. 4331). Pages 884-896. 2006

8. *Conference paper*:
   Elena Gómez-Martínez and José Merseguer. ArgoSPE: Model-based software performance engineering. *In Proc. of the 27nd International Conference On Application And Theory Of Petri Nets And Other Models Of Concurrency (ICATPN 2006).* Lectures Notes in Computer Science (Vol. 4024). Pages 401-410. 2006

9. *Conference paper*:
   Elena Gómez-Martínez and José Merseguer. A Software Performance Engineering Tool based on the UML-SPT. Tool demonstration. *In Proc. of the 2nd International Conference on Quantitative Evaluation of Systems (QEST 2005).* IEEE Computer Society. Pages 247-248. 2005.

10. *Short Technical Report*:
    Elena Gómez-Martínez. Performance Analysis of Web Applications. *Technical Report, July 2005.* 15 pages. Universidad de Zaragoza.

Other scientific papers produced during the realization of this dissertation thesis are the following.

1. *Conference paper*:
   Clara Benac Earle, Elena Gómez-Martínez, Stefano Tonetta, Stefano Puri, Silvia Mazzini, Jean-Louis Gilbert, Olivier Hachet, Ramón Serna Oliver, Cecilia Ekelin and Katiusca Zedda. Languages for Safety-Certification Related Properties. *Special Session to present Work in Progress at 39th Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2013).* 2013

2. *Journal paper*:
   Horacio Saggion, Elena Gómez-Martínez, Esteban Etayo, Alberto Anula, Lorena Bourg. Text Simplification in Simplext: Making Text More Accessible. *Revista de la Sociedad Española de Procesamiento de Lenguaje Natural (SEPLN).* Vol. 47. Pages 341-342. 2011

3. *Short Technical Report*:
   Elena Gómez-Martínez. Secure software design. *Technical Report, July 2005*, 15 pages. Universidad de Zaragoza.

In addition to publication, the work carried out in this thesis has been part of the following research projects:

- **nSafeCer** (Safety Certification of Software-Intensive Systems with Reusable Components).
  Reference: ART Call 2011 295373

- **pSafeCer** (pilot Safety Certification of Software-Intensive Systems with Reusable Components).
  Reference: ART Call 2010 269265

- **SIMPLEXT** (Sistema automático de simplificación de textos).
  Reference: TSI-020302-2010-84

- **ATIS4all** (Assistive Technologies and Solutions for All).
  Reference: ICT-PSP 270988

- **INREDIS** (INterfaces de RElación entre el entorno y las personas con DIScapacidad).
  Reference: CDTI CEN-2007-2011

- **Desarrollo de una herramienta para predicción de QoS del software**
  Reference:IBE2005-TEC-10

- **Evaluación de prestaciones de Sistemas de Información haciendo uso de UML y Redes de Petri (Fase II)**
  Reference: TIC2003-05226.

7.3 Future Research

This thesis is not a closed work and many research efforts can still be done. The main issues that we can address in the future to improve our approach are the following.

1. Methodology related improvements. The methodology would improve in the following aspects:

    - Process related aspects. In this thesis, we only focus on performance assessment. Nevertheless, the process can be augmented to include other features relevant to software architectures, such as dependability.
    - Modelling related aspects. We populate the performance scenarios only with closed workloads with discrete values. Therefore, we do not model open or *bursty* workloads, such as web servers. Applying the proposal of Pérez-Palacin et al. (2012), we can model bursty workloads with markovians processes. A similar situation occurs with resources, since we do not model some low level issues, such as random access memory, disk storage or cache memory, among others. Thus, by modeling these features, as well as other contemplated in the GRM (a sub-profile of MARTE), would permit to obtain a more precise performance analysis and, therefore, a better performance assessment.

2. Tool related improvements. As we realized while analyzing performance of the industrial case study with ArgoSPE, this tool should be improved with the following features.

    - Concerning performance annotations. ArgoSPE should support them in MARTE, currently only in SPT. In addition, we should include other parameters related to resources that we manually modify in GSPN.
    - Concerning the GSPN solver. Currently, we use GreatSPN to analyse or simulate GSPNs. Nevertheless, we have demonstrated that, in complex systems, the problem stems from the reachability graph of the GSPN. We can offer to the user tool option to automatically modify the simulation accuracy in order to decrease the computation times.
    - Concerning other new features. ArgoSPE lacks other plugins, such as tools for identifying performance patterns and antipatterns automatically.

    As mentioned in previous chapters, we have detected the need for developing a new framework which integrates all these functionalities: UML modeling, GSPN simulation and analysis, patterns and antipatterns detection in a transparent way to the user and efficient computation times. This framework could also include assessment of other functional and non-functional properties, such as dependability, security or model checking.

Bibliography

Abrams, M., Phanouriou, C., Batongbacal, A. L., Williams, S. M., and Shuster, J. E. (1999). UIML: An Appliance-Independent XML User Interface Language. *Computer Networks*, 31(11-16):1695–1708.

Ajmone Marsan, M., Balbo, G., Conte, G., Donatelli, S., and Franceschinis, G. (1995). *Modelling with Generalized Stochastic Petri Nets*. Wiley Series in Parallel Computing. J. Wiley, 1st edition.

Ajmone Marsan, M., Conte, G., and Balbo, G. (1984). A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. *ACM Trans. Comput. Syst.*, 2(2):93–122.

Alonso, G., Casati, F., Kuno, H., and Machiraju, V. (2004). *Web Services: Concepts, Architectures and Applications*. Data-Centric Systems and Applications. Springer.

Alvargonzález, M., Etayo, E., Gutiérrez, J. A., and Madrid, J. (2010). Arquitectura orientada a servicios para proporcionar accesibilidad. In *Actas del V Jornadas Científico-Ténicas en Servicios Web y SOA (JSWEB'10)*. Editorial Garceta.

Apache Software Foundation (2010a). Xerces Java Parser. Specification available at: `http://xerces.apache.org/xerces-j/`. Version 1.0.

Apache Software Foundation (2010b). Xerces2 Java Parser. Specification available at: `http://xerces.apache.org/xerces2-j`. Version 2.0.

ArgoUML (2013). The ArgoUML project. Tool available at: `http://argouml.tigris.org`. Version 0.26.

Aridor, Y. and Oshima, M. (1999). Infrastructure for Mobile Agents: Requirements and Design. In *Proc. 2nd Int. Workshop on Mobile Agents (MA'98)*, pages 38–49. Springer.

Balsamo, S. and Marzolla, M. (2003a). Simulation Modeling of UML Software Architectures. In *Proc. 17th European Simulation MultiConf. (ESM'03)*, pages 562–567. SCSEuropean Publishing House.

Balsamo, S. and Marzolla, M. (2003b). Towards Performance Evaluation of Mobile Systems in UML. In *Proc. 17th European Simulation MultiConf. (ESM'03)*, pages 61–68. EUROSIS-ETI.

Banks, J., Carson, J. S., Nelson, B. L., and Nicol, D. M. (2009). *Discrete-Event System Simulation*. Prentice Hall, 5th edition.

Bass, L., Clements, P., and Kazman, R. (2005). *Software Architecture in Practice*. SEI Series in Software Engineering. Addison-Wesley.

Baumeister, H., Koch, N., Kosiuczenko, P., Stevens, P., and Wirsing, M. (2003). UML for Global Computing. In *Proc. IST/FET Int. Workshop on Global Computing. Programming Environments, Languages, Security, and Analysis of Systems (GC'03)*, volume 2874 of *Lecture Notes in Computer Science*, pages 1–24. Springer.

Bäumer, C. and Magedanz, T. (1999). Grasshopper - A Mobile Agent Platform for Active Telecommunication. In *Intelligent Agents for Telecommunication Applications (IATA'99)*, pages 19–32. Springer.

Becker, S., Koziolek, H., and Reussner, R. (2009). The Palladio Component Model for Model-Driven Performance Prediction. *J. Syst. Softw.*, 82(1):3–22.

Bergenti, F. and Poggi, A. (2000). Improving UML Designs Using Automatic Design Pattern Detection. In *In Proc. 12th. Int. Conf. on Software Engineering and Knowledge Engineering (SEKE'00)*, pages 336–343.

Bernardi, S., Donatelli, S., and Merseguer, J. (2002). From UML Sequence Diagrams and Statecharts to analysable Petri Net models. In *Proc. 3rd Int. Workshop on Software and Performance (WOSP'02)*, pages 35–45. ACM.

Bernardi, S. and Merseguer, J. (2006). QoS Assessment via Stochastic Analysis. *IEEE Internet Computing*, 10(3):32–42.

Bernardi, S. and Merseguer, J. (2007). Performance evaluation of UML design with Stochastic Well-formed Nets. *J. Syst. Softw.*, 80(11):1843–1865.

Bernardi, S., Merseguer, J., and Petriu, D. C. (2012). Dependability modeling and analysis of software systems specified with UML. *ACM Comput. Surv.*, 45(1):2.

Berthelot, G. (1987). Transformations and Decompositions of Nets. In *Advances in Petri Nets*, volume 254 of *Lecture Notes in Computer Science*, pages 359–376. Springer.

Bolch, G., Greiner, S., de Meer, H., and Trivedi, K. S. (2001). *Queueing Networks and Markov Chains*. John Wiley & Sons, Inc.

Bracchi, P., Cukic, B., and Cortellessa, V. (2004). Performability Modeling of Mobile Software Systems. In *Proc. 15th Int. Symposium on Software Reliability Engineering (ISSRE'04)*, pages 77–88. IEEE Computer Society.

Brown, W. J., Malveau, R. C., McCormick, H. W., and Mowbray, T. J. (1998). *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis.* John Wiley, 1st edition.

Cabrera-Umpiérrez, M. F., Rodríguez Castro, A., Azpiroz, J., Montalvá Colomer, J. B., Arredondo, M. T., and Cano-Moreno, J. (2011). Developing Accessible Mobile Phone Applications: The case of a Contact Manager and Real Time Text Applications. In *Universal Access in Human-Computer Interaction. Context Diversity*, volume 6767 of *Lecture Notes in Computer Science*, pages 12–18. Springer.

Campos, J. (1998). *Performance Models for Discrete Event Systems with Synchronisations: Formalisms and Analysis Techniques*, chapter Performance Measures and Basic Properties, pages 285–304. Editorial KRONOS.

Campos, J., Colom, J. M., Jungnitz, H., and Silva, M. (1994). Approximate Throughput Computation of Stochastic Marked Graphs. *IEEE Trans. Softw. Eng.*, 20(7):526–535.

Campos, J., Donatelli, S., and Silva, M. (1999). Structured Solution of Asynchronously Communicating Stochastic Modules. *IEEE Trans. Softw. Eng.*, 25(2):147–165.

Campos, J., Sánchez, B., and Silva, M. (1991). Throughput Lower Bounds for Markovian Petri Nets Transformation Techniques. In *Proc. Int. Conf. on Petri Nets and Performance Models (PNPM'91)*, pages 322–331. IEEE Computer Society Press.

Card, S. K., Robertson, G. G., and Mackinlay, J. D. (1991). The information visualizer: An information workspace. In *Proc. SIGCHI Conf. on Human Factors in Computing Systems (CHI'91)*, pages 181–186. ACM.

Catalán, E. and Catalán, M. (2010). Performance Evaluation of the INREDIS framework. Technical report, Departament d'Enginyeria Telemàtica, Universitat Politècnica de Catalunya.

Catley, C., Petriu, D. C., and Frize, M. (2004). Software Performance Engineering of a Web service-based Clinical Decision Support infrastructure. In *Proc. 4th Int. Workshop on Software and Performance (WOSP'04)*, pages 130–138. ACM.

Chandrasekaran, S., Miller, J. A., Silver, G. A., Arpinar, I. B., and Sheth, A. P. (2003). Performance Analysis and Simulation of Composite Web Services. *Electronic Markets*, 13(2).

Chappell, D. A. (2004). *Enterprise Service Bus: Theory in Practice.* O'Reilly Media, Inc.

Chi, C.-F., Tseng, L.-K., and Jang, Y. (2012). Pruning a Decision Tree for Selecting Computer-Related Assistive Devices for People With Disabilities. *IEEE Trans on Neural Systems and Rehabilitation Engineering*, 20(4):564–573.

Chiola, G., Franceschinis, G., Gaeta, R., and Ribaudo, M. (1995). GreatSPN 1.7: GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets. *Perform Eval*, 24:47–68.

Chung, L. and Prado Leite, J. C. (2009). *On Non-Functional Requirements in Software Engineering*, pages 363–379. Springer.

Clark, A., Gilmore, S., Hillston, J., and Tribastone, M. (2007). Stochastic Process Algebras. In *Formal Methods for Performance Evaluation*, volume 4486 of *Lecture Notes in Computer Science*, pages 132–179. Springer.

Clark, G., Courtney, T., Daly, D., Deavours, D., Derisavi, S., Doyle, J. M., Sanders, W. H., and Webster, P. (2001). The Möbius Modeling Tool. In *Proc. 9th Int. Workshop on Petri Nets and Performance Models (PNPM'01)*, pages 241–. IEEE Computer Society. `https://www.mobius.illinois.edu/`.

Clements, P., Kazman, R., and Klein, M. (2002). *Evaluating Software Architectures: Methods and Case Studies.* Addison-Wesley.

Colom, J. M., Teruel, E., and Silva, M. (1998). *Performance Models for Discrete Event Systems with Synchronisations: Formalisms and Analysis Techniques.* Editorial KRONOS.

Cortellessa, V., Di Marco, A., and Inverardi, P. (2011). *Model-Based Software Performance Analysis.* Springer.

Cortellessa, V., Di Marco, A., and Trubiani, C. (2010). Performance Antipatterns as Logical Predicates. In *Proc. 15th IEEE Int. Conf. on Engineering of Complex Computer Systems (ICECCS'10)*, pages 146–156. IEEE Computer Society.

Cortellessa, V., Di Marco, A., and Trubiani, C. (2012). An approach for modeling and detecting software performance antipatterns based on first-order logics. *Softw & Syst Modeling*, pages 1–42.

Cortellessa, V., Gentile, M., and Pizzuti, M. (2004). XPRIT: An XML-Based Tool to Translate UML Diagrams into Execution Graphs and Queueing Networks. In *Proc. 1st Int. Conf. on the Quantitative Evaluation of Systems (QEST'04)*, pages 342–343. IEEE Computer Society.

Cortellessa, V. and Mirandola, R. (2002). PRIMA-UML: a performance validation incremental methodology on early UML diagrams. *Sci. Comput. Program.*, 44(1):101–129.

Cortés, U., Annicchiarico, R., Vázquez-Salceda, J., Urdiales, C., Cañamero, L., López, M., Sànchez-Marrè, M., and Caltagirone, C. (2003). Assistive technologies for the disabled and for the new generation of senior citizens: the e-Tools architecture. *AI Commun.*, 16(3):193–207.

Coulouris, G., Dollimore, J., and Kindberg, T. (2005). *Distributed Systems: Concepts and Design (Int. Computer Science)*. Addison-Wesley Longman, 4th rev. ed. edition.

Crimson (2005). The Crimson Java Parser. Specification available at: `http://xml.apache.org/crimson/`. Version 1.1.

CSM2PN (2013). The CSM to GSPN Translator. Tool available at: `http://webdiis.unizar.es/~jmerse/csm2pn.html`.

D'Ambrogio, A. (2005). A model transformation framework for the automated building of performance models from UML models. In *Proc. 5th Int. Workshop on Software and Performance (WOSP'05)*, pages 75–86.

Datla, V. and Goševa-Popstojanova, K. (2005). Measurement-based Performance Analysis of E-commerce Applications with Web Services Components. In *Proc. IEEE Int. Conf. on e-Business Engineering (ICEBE'05)*, pages 305–314.

Davis, D. and Parashar, M. P. (2002). Latency Performance of SOAP Implementations. In *Proc. 2nd IEEE/ACM Int. Symposium on Cluster Computing and the Grid (CCGRID'02)*, pages 407–412.

de Gooijer, T., Jansen, A., Koziolek, H., and Koziolek, A. (2012). An Industrial Case Study of Performance and Cost Design Space Exploration. In *Proc. 3rd ACM/SPEC Int. Conf. on Performance Engineering (ICPE'12)*, pages 205–216. ACM.

de Miguel, M. A. (2003). General framework for the description of QoS in UML. In *Procs. of the 6th IEEE Int. Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'03)*, pages 61–70. IEEE Computer Society.

de Miguel, M. A., Briones, J. F., Silva, J. P., and Alonso, A. (2008). Integration of safety analysis in model-driven software development. *Software, IET*, 2(3):260–280.

Del Rosso, C. (2006). Continuous Evolution through Software Architecture Evaluation: a case study: Practice Articles. *J. Softw. Maint. Evol.*, 18(5):351–383.

Delatour, J. and de Lamotte, F. (2003). ArgoPN: a CASE Tool Merging UML and Petri Nets. In *Proc. 3rd Int. Workshop on New Developments in Digital Libraries and the 1st Int. Workshop on Validation and Verification of Software for Enterprise Information Systems (NDDL/VVEIS'03)*, pages 94–102. ICEIS Press.

Distefano, S., Paci, D., Puliafito, A., and Scarpa, M. (2004). UML Design and Software Performance Modeling. In *Proc. 19th Int. Symposium Computer and Information Sciences (ISCIS'04)*, volume 3280 of *Lecture Notes in Computer Science*, pages 564–573. Springer.

Distefano, S., Scarpa, M., and Puliafito, A. (2011). From UML to Petri Nets: The PCM-Based Methodology. *IEEE Trans. Softw. Eng.*, 37(1):65–79.

Donatelli, S. and Franceschinis, G. (1996). The PSR Methodology: Integrating Hardware and Software Models. In *Proc. 17th Int. Conf. on Application and Theory of Petri Nets (ICATPN'96)*, volume 1091 of *Lecture Notes in Computer Science*, pages 133–152. Springer.

Dugan-Jr., R. F., Glinert, E. P., and Shokoufandeh, A. (2002). The Sisyphus Database Retrieval Software Performance Antipattern. In *Proc. 3rd Int. Workshop on Software and Performance (WOSP'02)*, pages 10–16. ACM.

Elfwing, R., Paulsson, U., and Lundberg, L. (2002). Performance of SOAP in Web Service Environment Compared to CORBA. In *Proc. 9th Asia-Pacific Software Engineering Conf. (APSEC'02)*, pages 84–96. IEEE Computer Society.

Erl, T. (2007). *SOA Principles of Service Design.* Prentice Hall PTR.

Espinoza, H., Dubois, H., Gérard, S., Pasaje, J. M., Petriu, D. C., and Woodside, M. (2005). Annotating UML Models with Non-functional Properties for Quantitative Analysis. In *Procs. of the 2005 Int. Conf. on Satellite Events at the MoDELS*, volume 3844 of *Lecture Notes in Computer Science*, pages 79–90. Springer.

Eurostat (2013). Statistical Office of European Union.
`http://epp.eurostat.ec.europa.eu`.

Florin, G. and Natkin, S. (1989). Necessary and Sufficient Ergodicity Condition for Open Synchronized Queueing Networks. *IEEE Trans. Software Eng.*, 15(4):367–380.

Fowler, M. (2000). *UML distilled-a brief guide to the Standard Object Modeling Language.* Addison-Wesley-Longman.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison–Wesley.

Garlan, D. and Shaw, M. (1993). An Introduction to Software Architecture. In *Advances in Software Engineering and Knowledge Engineering*, pages 1–39. World Scientific Pub.

Gilmore, S., Haenel, V., Kloul, L., and Maidl, M. (2005). Choreographing Security and Performance Analysis for Web Services. In *Proc. European Performance Engineering Workshop and Int. Workshop on Web Services and Formal Methods (EPEW/WS-FM'05)*, pages 200–214. Springer.

Gilmore, S. and Hillston, J. (1994). The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling. In *Proc of the 7th Int. Conf. on Computer Performance Evaluation, Modeling Techniques and Tools*, volume 794 of *Lecture Notes in Computer Science*, pages 353–368. Springer.

Gilmore, S. and Kloul, L. (2003). A Unified Tool for Performance Modelling and Prediction. In *Computer Safety, Reliability, and Security (SAFECOMP'03)*, volume 2788 of *Lecture Notes in Computer Science*, pages 179–192. Springer.

Giménez, R., Pous, M., and Rico-Novella, F. (2012). Securing an Interoperability Architecture for Home and Urban Networking: Implementation of the Security Aspects in the INREDIS Interoperability Architecture. In *Proc. 26th Int. Conf. on Advanced Information Networking and Applications Workshops (WAINA'12)*, pages 714–719. IEEE Computer Society.

Glinz, M. (2007). On Non-Functional Requirements. In *Proc. 15th IEEE Int. Conf. on Requirements Engineering (RE'07)*, pages 21–26. IEEE.

Gokhale, S. S. and Trivedi, K. S. (2002). Reliability prediction and sensitivity analysis based on software architecture. In *Procs. of the 13th Int. Symposium on Software Reliability Engineering (ISSRE'03)*, pages 64–75.

Goldratt, E. M. and Cox, J. (1992). *The Goal: A process of Ongoing Improvement.* North River Press.

Gomaa, H. (2000). *Designing Concurrent, Distributed, and Real-Time Applications with UML.* Addison-Wesley Longman Publishing Co., Inc., 1st edition.

Gómez-Martínez, E., González-Cabero, R., and Merseguer, J. (2013a). Performance Assessment of an Architecture with Adaptative Interfaces for People with Special Needs. *Empir Softw Eng.* Accepted for publication.

Gómez-Martínez, E., Ilarri, S., and Merseguer, J. (2007). Performance Analysis of Mobile Agents Tracking. In *Proc. 6th Int. Workshop on Software and Performance (WOSP'07)*, pages 181–188. ACM.

Gómez-Martínez, E., Linaje, M., Iglesias-Pérez, A., Sánchez-Figueroa, F., Preciado, J. C., González-Cabero, R., and Merseguer, J. (2013b). Interacting with Inaccessible Smart Environments: Conceptualization and evaluated recommendation of Assistive Software. Submitted to publication.

Gómez-Martínez, E. and Merseguer, J. (2005). A Software Performance Engineering Tool based on the UML-SPT. In *Proc. 2nd Int. Conf. on the Quantitative Evaluation of Systems (QEST'05)*, pages 247–. IEEE Computer Society.

Gómez-Martínez, E. and Merseguer, J. (2006a). ArgoSPE: Model-based Software Performance Engineering. In *Proc. 27th Int. Conf. on Applications and Theory of Petri Nets and Other Models of Concurrency (ICATPN'06)*, volume 4024 of *Lecture Notes in Computer Science*, pages 401–410. Springer. Tool available at: `http://argospe.tigris.org`.

Gómez-Martínez, E. and Merseguer, J. (2006b). Impact of SOAP Implementations in the Performance of a Web Service-Based Application. In *Proc. Workshop on Middleware Performance (WOMP'06) at the Int. Conf. on Frontiers of High Performance Computing and Networking (ISPA'06)*, volume 4331 of *Lecture Notes in Computer Science*, pages 884–896. Springer.

Gómez-Martínez, E. and Merseguer, J. (2010). Performance Modeling and Analysis of the Universal Control Hub. In *Proc. the 7th European Performance Engineering Workshop (EPEW'10)*, volume 6342 of *Lecture Notes in Computer Science*, pages 160–174. Springer.

González-Cabero, R. (2010). A Semantic Matching Process for Detecting and Reducing Accessibility Gaps in an Ambient Intelligence Scenario. In *Proc. 4th Int. Symposium of Ubiquitous Computing and Ambient Intelligence (UCAmI'10)*, pages 315–324. IBERGACETA Publicaciones.

Grand, M. (1998). *Patterns in Java, volume 1: a catalog of reusable design patterns illustrated with UML*. John Wiley & Sons, Inc.

Grand, M. (2001). *Java Enterprise Design Patterns: Patterns in Java Volume 3*. John Wiley & Sons, Inc.

Grassi, V., Mirandola, R., and Sabetta, A. (2004). UML based modeling and performance analysis of mobile systems. In *Proc. 7th Int. Symposium on Modeling Analysis and Simulation of Wireless and Mobile Systems (MSWiM'04)*, pages 95–104. ACM.

Head, M. R., Govindaraju, M., Slominski, A., Liu, P., Abu-Ghazaleh, N., van Engelen, R., Chiu, K., and Lewis, M. J. (2005). A Benchmark Suite for SOAP-based Communication in Grid Web Services. In *Proc. ACM/IEEE Conf. Supercomputing (SC'05)*, page 19.

Hermanns, H., Herzog, U., and Katoen, J. P. (2002). Process Algebra for Performance Evaluation. *Theoretical Computer Science*, 274(1-2):43–87.

Hillston, J. and Ribaudo, M. (2004). Modelling Mobility with PEPA Nets. In *Proc. 19th Int. Symposium Computer and Information Sciences (ISCIS'04)*, volume 3280 of *Lecture Notes in Computer Science*, pages 513–522. Springer.

Huber, N., Becker, S., Rathfelder, C., Schweflinghaus, J., and Reussner, R. H. (2010). Performance Modeling in Industry: a Case Study on Storage Virtualization. In *Procs. of the 32nd ACM/IEEE Int. Conf. on Software Engineering (ICSE'10)*, pages 1–10. ACM.

IEEE (2010). Std 1901-2010 for Broadband over Power Line Networks: Medium Access Control and Physical Layer Specifications.

Iglesias-Pérez, A., Linaje, M., Preciado, J. C., Sánchez-Figueroa, F., Gómez-Martínez, E., González-Cabero, R., and Ángel Martínez-Usero, J. (2010). A Context-Aware Semantic Approach for the Effective Selection of an Assistive Software. In *Proc. 4th Int. Symposium of Ubiquitous Computing and Ambient Intelligence (UCAmI'10)*, pages 51–60. IBERGACETA Publicaciones.

Ilarri, S., Trillo, R., and Mena, E. (2006). SPRINGS: A Scalable Platform for Highly Mobile Agents in Distributed Computing Environments. In *Proc of the 4th Int. Workshop on Mobile Distributed Computing (MDC'06)*. IEEE Computer Society.

INREDIS Consortium (2010a). Deliverable-78.2.1. Final Guide to a Generic Platform Deployment.

INREDIS Consortium (2010b). INterfaces for RElations between Environment and people with DISabilities. Project website. `http://www.inredis.es/`.

Isa, M. A. and Jawawi, D. N. A. (2011). Comparative Evaluation of Performance Assessment and Modeling Method for Software Architecture. In *Software Engineering and Computer Systems*, volume 181 of *Communications in Computer and Information Science*, pages 764–776. Springer.

ISO (2011). 9999:2011-Assistive products for persons with disability–Classification and terminology.

ISO/IEC (2008). 24752-1:2008-Information technology–User interfaces–Universal remote console–Part 1: Framework.

ISO/IEC (2009). 24756:2009-Information technology–Framework for specifying a common access profile (CAP) of needs and capabilities of users, systems, and their environments.

ISO/IEC (2011a). 15909-2:2011-Systems and software engineering–High-level Petri nets–Part 2: Transfer format.

ISO/IEC (2011b). 25010:2011-Systems and software engineering–Systems and software Quality Requirements and Evaluation (SQuaRE)–System and software quality models.

ISO/IEC (2012). 19505-1:2012-Information technology–Object Management Group Unified Modeling Language (OMG UML)–Part 1: Infrastructure.

ISO/IEC/IEEE (2010). 24765:2010-Systems and software engineering–Vocabulary.

ISO/IEC/IEEE (2011). 29148:2011-Systems and software engineering–Life cycle processes –Requirements engineering.

ISO/IEC/IEEE (2011). 42010:2011-Systems and software engineering–Architecture description.

Jain, R. K. (1991). *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.* Wiley professional computing. John Wiley.

Jin, Y., Tang, A., Han, J., and Liu, Y. (2007). Performance Evaluation and Prediction for Legacy Information Systems. In *Proc. 29th Int. Conf. on Software Engineering (ICSE'07)*, pages 540–549. IEEE Computer Society.

Kadouche, R., Abdulrazak, B., Giroux, S., and Mokhtari, M. (2009). Disability Centered Approach in Smart Space Management. *Int. Journal of Smart Home*, 3(3):13–26.

Kastidou, G., Pitoura, E., and Samaras, G. (2003). A Scalable Hash-Based Mobile Agent Location Management Mechanism. In *Proc. 1st Int. Workshop on Mobile Distributed Computing (MDC'03)*, pages 472–478.

Kauppi, T. (2003). Performance Analysis at the Software Architectural Level. Technical Report 512, VTT Technical Research Centre of Finland.

Kerth, N. L. (1995). *Pattern languages of program design.* ACM Press/Addison-Wesley Publishing Co.

Kounev, S. (2006). Performance Modeling and Evaluation of Distributed Component-Based Systems Using Queueing Petri Nets. *IEEE Trans on Softw Eng*, 32(7):486–502.

Koziolek, A., Koziolek, H., and Reussner, R. (2011). PerOpteryx: Automated Application of Tactics in Multi-Objective Software Architecture Optimization. In *Proc. 7th Int. Conf. on the Quality of Software Architectures (QoSA'11)*, pages 33–42. ACM.

Kozoliek, H., Schlich, B., Becker, S., and Hauck, M. (2012). Performance and reliability prediction for evolving service-oriented software systems. *Empir Softw Eng*, pages 1–45.

Lange, D. B. and Oshima, M. (1999). Seven Good Reasons for Mobile Agents. *Commun. ACM*, 42(3):88–89.

Lange, D. B., Oshima, M., Karjoth, G., and Kosaka, K. (1997). Aglets: Programming Mobile Agents in Java. In *Proc. Int. Conf. on Worldwide Computing and Its Applications (WWCA'97)*, volume 1274 of *Lecture Notes in Computer Science*, pages 253–266. Springer.

Lazowska, E. D., Zahorjan, J., Graham, G. S., and Sevcik, K. C. (1984). *Quantitative System Performance: Computer System Analysis Using Queueing Network Models.* Prentice-Hall.

Lea, D. (1999). *Concurrent Programming in Java. Second Edition: Design Principles and Patterns.* Addison-Wesley Longman Publishing Co., Inc., 2nd edition.

Levandoski, J. J., Ekstrand, M. D., Ludwig, M., Eldawy, A., Mokbel, M. F., and Riedl, J. (2011). RecBench: Benchmarks for Evaluating Performance of Recommender System Architectures. *PVLDB*, 4(11):911–920.

Liang, S., Fodor, P., Wan, H., and Kifer, M. (2009). OpenRuleBench: an Analysis of the Performance of Rule Engines. In *Proc. 18th Int. Conf. on World Wide Web (WWW'09)*, pages 601–610. ACM.

Lindemann, C. (1995). DSPNexpress: A Software Package for the Efficient Solution of Deterministic and Stochastic Petri Nets. *Perform. Eval.*, 22:15–29.

Liu, Y., Fekete, A., and Gorton, I. (2004). Predicting the Performance of Middleware-based Applications At The Design Level. In *Proc. 4th Int. Workshop on Software and Performance (WOSP'04)*, pages 166–170. ACM.

Liu, Y., Fekete, A., and Gorton, I. (2005). Design-Level Performance Prediction of Component-Based Applications. *IEEE Trans. Softw. Eng.*, 31(11):928–941.

Liu, Y., Gorton, I., and Zhu, L. (2007). Performance Prediction of Service-Oriented Applications based on an Enterprise Service Bus. In *Proc. 31st Annual Int. Computer Software and Applications Conf. (COMPSAC'07)*, pages 327–334. IEEE Computer Society.

Llinás, P., Montoro, G., García-Herranz, M., Haya, P., and Alamán, X. (2009). Adaptive Interfaces for People with Special Needs. In *Proc. 10th Int. Work-Conf. on Artificial Neural Networks: Part II: Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living (IWANN'09)*, pages 772–779. Springer.

López-Grao, J. P., Merseguer, J., and Campos, J. (2004). From UML Activity Diagrams to Stochastic Petri Nets: Application to Software Performance Engineering. In *Proc. 4th Int. Workshop on Software and Performance (WOSP'04)*, pages 25–36. ACM.

Mani, N., Petriu, D. C., and Woodside, M. (2011). Towards Studying the Performance Effects of Design Patterns for Service Oriented Architecture. In *Proc. 2nd joint WOSP/SIPEW Int. Conf. on Performance Engineering (ICPE'11)*, pages 499–504. ACM.

Margetis, G., Antona, M., Ntoa, S., and Stephanidis, C. (2012). Towards Accessibility in Ambient Intelligence Environments. *Ambient Intelligence*, 7683:328–337.

Marzolla, M. and Balsamo, S. (2004). UML-PSI: The UML Performance Simulator. In *Proc. 1st Int. Conf. on the Quantitative Evaluation of Systems (QEST'04))*, pages 340–341. IEEE Computer Society.

Mena, E., Royo, J. A., Illarramendi, A., and Goñi, A. (2002). Adaptable Software Retrieval Service for Wireless Environments Based on Mobile Agents. In *Proc. Int. Conf. on Wireless Networks (ICWN'02)*, pages 116–124. CSREA Press.

Menascé, D. A. (2004). Composing Web Services: A QoS View. *IEEE Internet Computing*, 8(6):88–90.

Menascé, D. A. and Almeida, V. A. F. (2001). *Capacity Planning for Web Services: metrics, models, and methods.* Prentice Hall PTR.

Merlin, P. M. and Farber, D. J. (1976). Recoverability of Communication Protocols: Implications of a Theoretical Study. *IEEE Trans on Communications*, 24(9):1036–1043.

Merseguer, J. (2003). *Software Performance Engineering based on UML and Petri nets*. PhD thesis, University of Zaragoza, Spain.

Merseguer, J., Bernardi, S., Campos, J., and Donatelli, S. (2002). A Compositional Semantics for UML State Machines Aimed at Performance Evaluation. In *Proc. 6th Int. Workshop on Discrete Event Systems (WODES'02)*, pages 295–302. IEEE Computer Society.

Merseguer, J., Campos, J., and Mena, E. (2003). Analysing Internet Software Retrieval Systems: Modeling and Performance Comparison. *Wirel. Netw.*, 9(3):223–238.

Miller, R. B. (1968). Response time in Man-Computer Conversational Trans. In *Proc. AFIPS Fall Joint Computer Conf. (AFIPS'68)*, volume 33, pages 267–277.

Milojičić, D., Douglis, F., and Wheeler, R. (1999). *Mobility: Processes, Computers, and Agents*. ACM Press/Addison-Wesley Publishing Co.

Milojičić, D., LaForge, W., and Chauhan, D. (1998). Mobile Objects and Agents (MOA). In *Proc. 4th USENIX Conf. on Object-Oriented Technologies and Systems (COOTS'98)*. USENIX.

Molloy, M. K. (1982). Performance Analysis Using Stochastic Petri Nets. *IEEE Trans on Computers*, 31(9):913–917.

Monroe, R. T., Kompanek, A., Melton, R., and Garlan, D. (1997). Architectural Styles, Design Patterns, and Objects. *IEEE Softw.*, 14(1):43–52.

Murata, T. (1989). Petri Nets: Properties, Analysis and Applications. *Proc. IEEE*, 77(4):541–580.

Murphy, A. L. and Picco, G. P. (2002). Reliable Communication for Highly Mobile Agents. *Autonomous Agents and Multi-Agent Systems*, 5(1):81–100.

Murua, A., González, I., and Gómez-Martínez, E. (2011). Cloud-based Assistive Technology Services. In *Proc. 3rd Workshop on Software Services: Semantic-based Software Services (WoSS) at the Federated Conf. on Computer Science and Information Systems (FedCSIS'11)*, pages 985–989.

Natkin, S. O. (1980). *Les réseaux de Petri stochastiques et leur application à l'évaluation des systèmes informatiques*. PhD thesis, Conservatoire National des Arts et Métiers (CNAM), Paris, France.

Newell, A. F. (2008). Accessible Computing–Past Trends and Future Suggestions: Commentary on "Computers and People with Disabilities". *ACM Trans. Access. Comput.*, 1(2):9:1–9:7.

Ng, A., Chen, S., and Greenfield, P. (2004). An Evaluation of Contemporary Commercial SOAP Implementations. In *Proc. 5th Australasian Workshop on Software and System Architectures (AWSA'04)*, pages 64–71.

Nielsen, J. (1993). *Usability Engineering.* Morgan Kaufmann.

OASIS (2005). Universal Description Discovery and Integration (UDDI). Specification available at: `http://uddi.xml.org/`. Version 3.0.2.

OMG (2005). UML Profile for Schedulabibity, Performance and Time Specification (UML-SPT). Specification available at: `http://www.uml.org`. Version 1.1.

OMG (2008). UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms (QoS & FT). Specification available at: `http://www.omg.org/spec/QFTP/`. Version 1.1.

OMG (2011a). A UML profile for Modeling and Analysis of Real Time Embedded Systems (MARTE). Specification available at: `http://www.omgmarte.org/`. Version 1.1.

OMG (2011b). Unified Modeling Language (UML). Specification available at: `http://www.omg.org/spec/UML/2.4.1/`. Version 2.4.1.

OMG (2011c). XML Metadata Interchange (XMI). Specification available at: `http://www.omg.org/spec/XMI/`. Version 2.4.1.

Pérez-Jiménez, C. J. (2002). *Técnicas de aproximación de throughput en redes de Petri estocásticas.* PhD thesis, Universidad de Zaragoza, Spain.

Pérez-Palacin, D., Merseguer, J., and Mirandola, R. (2012). Analysis of Bursty Workload-Aware Self-Adaptive Systems. In *Proc. 3rd ACM/SPEC Int. Conf. on Performance Engineering (ICPE'12)*, pages 75–84. ACM.

Peterson, J. L. (1981). *Petri Net Theory and the Modeling of Systems.* Prentice Hall PTR.

Petri, C. A. (1962). *Kommunikation mit Automaten.* PhD thesis, Universität Hamburg, Germany.

Petriu, D. B. and Woodside, M. (2007). An Intermediate Metamodel with Scenarios and Eesources for Generating Performance Models from UML designs. *Softw & Syst Modeling*, 6(2):163–184.

Petriu, D. C., Alhaj, M., and Tawhid, R. (2012). Software Performance Modeling. In *Formal Methods for Model-Driven Engineering-12th Int. School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM'12)*, volume 7320 of *Lecture Notes in Computer Science*, pages 219–262. Springer.

Petriu, D. C. and Shen, H. (2002). Applying the UML Performance Profile: Graph Grammar-Based Derivation of LQN Models from UML Specifications. In *Proc. 12th Int. Conf. on Computer Performance Evaluation, Modelling Techniques and Tools (TOOLS'02)*, volume 2324 of *Lecture Notes in Computer Science*, pages 159–177. Springer.

Petriu, D. C. and Woodside, M. (2002). Software Performance Models from System Scenarios in Use Case Maps. In *Proc. 12th Int. Conf. on Computer Performance Evaluation, Modelling Techniques and Tools (TOOLS'02)*, volume 2324 of *Lecture Notes in Computer Science*, pages 141–158. Springer.

Phanouriou, C. (2000). UIML: A Device-Independent User Interface Markup Language. Technical report, Virginia Polytechnic Institute and State University.

Pooley, R. J. and Abdullatif, A. A. L. (2010). CPASA: Continuous Performance Assessment of Software Architecture. In *Proc. 17th IEEE Int. Conf. and Workshops on the Eng of Computer-Based Systems (ECBS'10)*, pages 79–87. IEEE Computer Society.

Pous, M., Serra-Vallmitjana, C., Giménez, R., Torrent-Moreno, M., and Boix, D. (2012). Enhancing accessibility: Mobile to ATM case study. In *Proc. IEEE Consumer Communications and Networking Conf. (CCNC'12)*, pages 404–408. IEEE Computer Society.

Prud'hommeaux, E. and Seaborne, A. (2006). SPARQL Query Language for RDF. Specification available at: `http://www.w3.org/TR/rdf-sparql-query/`.

Q-ImPrESS (2009). Q-ImPrESS Consortium: Project website. `http://www.q-impress.eu`.

QoSA (2005-2013). *Int. ACM Sigsoft Conf. on the Quality of Software Architectures.* SIGSOFT.

Ramchandani, C. (1974). *Analysis of Asynchronous Concurrent Systems by Timed Petri Nets.* PhD thesis, Massachusetts Institute of Technology (MIT).

Reisig, W. and Rozenberg, G., editors (1998). *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets*, volume 1491 of *Lecture Notes in Computer Science*. Springer.

Ross, S. M. (1983). *Stochastic Processes.* John Wiley.

Rumbaugh, J. E., Jacobson, I., and Booch, G. (1999). *The Unified Modeling Language Reference Manual.* Addison-Wesley-Longman.

Runeson, P. and Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empir Softw Eng*, 14(2):131–164.

Sainz, F., Casacuberta, J., Díaz, M., and Madrid, J. (2011). Evaluation of an Accessible Home Control and Telecare System. In *Proc. 13rd Human-Computer Interaction (INTERACT'11)*, volume 6949 of *Lecture Notes in Computer Science*, pages 527–530. Springer.

SAX (2004). Simple API for XML (SAX). Specification available at: `http://www.saxproject.org/`. Version 2.0.2.

Scarpa, M., Villari, M., Zaia, A., and Puliafito, A. (2002). From client/server to mobile agents: an in-depth analysis of the related performance aspects. In *Proc. 7th IEEE Symposium on Computers and Communications (ISCC'02)*, pages 768–773. IEEE Computer Society.

Schmidt, D. C., Stal, M., Rohnert, H., and Buschmann, F. (2000). *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*. John Wiley & Sons, Inc., 2nd edition.

Schmietendorf, A. and Scholz, A. (2001). Aspects of performance engineering-An overview. In *Performance Engineering. State of the Art and Current Trends*, volume 2047 of *Lecture Notes in Computer Science*, pages IX–XII. Springer.

Sereno, M. and Balbo, G. (1997). Mean Value Analysis of Stochastic Petri Nets. *Perform. Eval.*, 29(1):35–62.

Shaw, M. (1990). Toward higher-level abstractions for software systems. *Data & Knowledge Engineering*, 5(2):119–128.

Shaw, M. and Garlan, D. (1996). *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall.

Sifakis, J. (1979). Use of Petri nets for Performance Evaluation. *Acta Cybernetica*, 4(2):185–202.

Silva, A. R., Romão, A., Deugo, D., and Silva, M. M. D. (2001). Towards a Reference Model for Surveying Mobile Agent Systems. *Autonomous Agents and Multi-Agent Systems*, 4(3):187–231.

Silva, M. (1985). *Las redes de Petri en la Automática y la Informática*. AC.

Slominski, A. (2004). XML Pull Parser (XPP). Tool available at: `http://www.extreme.indiana.edu/xgws/xsoap/xpp/`.

Smith, C. U. (1981). Increasing Information Systems Productivity by Software Performance Engineering. In *Proc. 7th Int. Conf. Computer Measurement Group (CMG'81)*, pages 5–14.

Smith, C. U. (1990). *Performance Engineering of Software Systems*. Addison–Wesley.

Smith, C. U. and Williams, L. G. (2000). Software Performance Antipatterns. In *Proc. 2nd Int. Workshop on Software and Performance (WOSP'00)*, pages 127–136. ACM.

Smith, C. U. and Williams, L. G. (2001). Software Performance AntiPatterns; Common Performance Problems and their Solutions. In *Proc. 27th Int. Conf. Computer Measurement Group (CMG'01)*, pages 797–806.

Smith, C. U. and Williams, L. G. (2002a). New Software Performance AntiPatterns: More Ways to Shoot Yourself in the Foot. In *Proc. 28th Int. Conf. Computer Measurement Group (CMG'02)*, pages 667–674.

Smith, C. U. and Williams, L. G. (2002b). *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software.* Addison–Wesley.

Smith, C. U. and Williams, L. G. (2003). More New Software Antipatterns: Even More Ways to Shoot Yourself in the Foot. In *29th Int. Conf. Computer Measurement Group (CMG'03)*, pages 717–725.

Sosnoski, D. (2001). XML and JAVA technologies: Document models, Part 1: Performance.
`http://www-128.ibm.com/developerworks/xml/library/x-injava/`.

Sosnoski, D. (2002). XMLBench Document Model Benchmark.
`http://www.sosnoski.com/opensrc/xmlbench`.

Spyrou, C., Samaras, G., Pitoura, E., and Evripidou, P. (2004). Mobile agents for wireless computing: the convergence of wireless computational models with mobile-agent technologies. *Mobile Networks and Applications*, 9(5):517–528.

Stephanidis, C. (2001). Adaptive Techniques for Universal Access. *User Modeling and User-Adapted Interaction*, 11:159–179.

Tribastone, M. and Gilmore, S. (2008). Automatic Translation of UML Sequence Diagrams into PEPA Models. In *Proc. 5th Int. Conf. on the Quantitative Evaluation of Systems (QEST'08)*, pages 205–214. IEEE Computer Society.

Trubiani, C. and Koziolek, A. (2011). Detection and solution of Software Performance Antipatterns in Palladio Architectural Models. In *Proc. 2nd joint WOSP/SIPEW Int. Conf. on Performance Engineering (ICPE'11)*, pages 19–30.

URC Consortium (2005). Universal Remote Console. Specification available at:
`http://myurc.org`.

URC Consortium (2010a). iPhone client for UCH (iUCH). Specification available at:
`http://myurc.org/tools/iPhone/`. Version 1.1.

URC Consortium (2010b). Universal Control Hub for C++ (UCHe). Specification available at: `http://myurc.org/tools/UCHe/`.

URC Consortium (2010c). Universal Control Hub for Java (UCHj). Specification available at: `http://myurc.org/tools/UCHj/`.

Verdickt, T., Dhoedt, B., Gielen, F., and Demeester, P. (2005). Automatic Inclusion of Middleware Performance Attributes into Architectural UML Software Models. *IEEE Trans. Softw. Eng.*, 31(8):695–711.

Vernadat, F., Dicesare, F., Harhalakis, G., Proth, J. M., and Silva, M. (1993). *Practice of Petri-nets in manufacturing*. Chapman and Hall.

W3C (2001). Web Services Description Language (WSDL). Specification available at: `http://www.w3.org/TR/wsdl`. Version 1.1.

W3C (2004). Web Services Architecture. Specification available at: `http://www.w3.org/TR/ws-arch/`.

W3C (2007). Simple Object Access Protocol. Specification available at: `http://www.w3.org/TR/soap`. Version 1.1.

W3C (2009). Document Object Model (DOM). Specification available at: `http://www.w3.org/DOM/`.

W3C (2010). eXtensible HyperText Markup Language. Specification available at: `http://www.xhtml.org/`. Version 1.0.

W3C (2012). OWL 2 Web Ontology Language. Specification available at: `http://www.w3.org/TR/owl2-overview/`.

Williams, L. G. and Smith, C. U. (2002). PASA^{SM}: A Method for the Performance Assessment of Software Architectures. In *Proc. 3rd Int. Workshop on Software and Performance (WOSP'02)*, pages 179–188. ACM.

Woodcock, A., Fielden, S., and Bartlett, R. (2012). The user testing toolset: a decision support system to aid the evaluation of assistive technology products. *Work: A Journal of Prevention, Assessment and Rehabilitation*, 41:1381–1386.

Woodside, M., Franks, G., and Petriu, D. C. (2007). The Future of Software Performance Engineering. In *Future of Software Engineering (FOSE'07)*, pages 171–187. IEEE Computer Society.

Woodside, M. and Menascé, D. A. (2006). Application-Level QoS. *IEEE Internet Computing*, 10(3):13–15.

Woodside, M., Neilson, J. E., Petriu, D. C., and Majumdar, S. (1995). The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software. *IEEE Trans. Computers*, 44(1):20–34.

Woodside, M., Petriu, D. C., Merseguer, J., Petriu, D. B., and Alhaj, M. (2013). Transformation challenges: from software models to performance models. *Software and Systems Modeling*. In Press.

Woodside, M., Petriu, D. C., Petriu, D. B., Shen, H., Israr, T., and Merseguer, J. (2005). Performance by Unified Model Analysis (PUMA). In *Proc. 5th Int. Workshop on Software and Performance (WOSP'05)*, pages 1–12. ACM.

Zimmermann, A., Freiheit, J., German, R., and Hommel, G. (2000). Petri Net Modelling and Performability Evaluation with TimeNET 3.0. In *Proc. 11th Int. Conf. Computer Performance Evaluation: Modelling Techniques and Tools (TOOLS'00)*, volume 1786 of *Lecture Notes in Computer Science*, pages 188–202. Springer. Tool available at: `http://www.tu-ilmenau.de/sse/timenet/`.

Zimmermann, G. and Vanderheiden, G. (2007). The Universal Control Hub: An Open Platform for Remote User Interfaces in the Digital Home. In *Proc. 12th Int. Conf. Human-Computer Interaction. Interaction Platforms and Techniques (HCI'07)*, volume 4551 of *Lecture Notes in Computer Science*, pages 1040–1049. Springer.

Zimmermann, G. and Vanderheiden, G. (2008). Accessible design and testing in the application development process: considerations for an integrated approach. *Universal Access in the Information Society*, 7(1-2):117–128.

Index

Activity (diagram), 11
- performance annotations, 85
- performance role, 76

Agents
- mobile, 9, 42
- SPRINGS platform, 42

Antipatterns, 100, 104, 143

ArgoSPE (tool), 88, 132, 157
- architecture, 160
- features, 157
- limitations, 151
- model analyser, 163
- model configurer, 161
- model convertor, 163
- model editor, 160
- model processor, 162
- performance annotations, 159
- results convertor, 163

CDSS, 26
- performance analysis, 29
- performance improvements, 33
- service, 28

Class (diagram)
- performance annotations, 85

CSM, 2, 18, 29, 48, 105

Deployment (diagram), 10
- performance annotations, 85
- performance role, 76
- queries in ArgoSPE, 159

Design patterns, 96

DOM, 161

fix-it later (approach), 13, 16

GSPN, 21

Host demand
- performance annotation, 83

Interaction Overview (diagram), 11
- performance annotations, 85
- performance role, 76

Interoperable Architecture, 109
- performance analysis, 132
- performance assessment, 139
- performance improvements, 146
- performance model, 126
- performance results, 134

MARTE, 11, 12, 126
- NFP, 12, 80
- VSL, 12, 80, 126

Mobile agents, 9, 42

NFP, 9, 12, 82

Performance
- analysis, 22, 89
- antipatterns, 104
- assessment, 9
- evaluation tools, 84, 157
- models, 14, 84
- patterns, 97

Performance antipatterns, 100, 143

Performance Methodology
- assessment, 94
- performance analysis, 89
- performance annotations, 85, 86
- performance design, 76
- performance model, 84
- performance objectives, 89
- resource replication, 95

responsiveness, 91
scalability, 92
utilization, 93
Performance patterns, 96, 140
Petri net, 19
analysis, 22
GSPN, 21
place/transition net, 19
reachability, 20
steady state, 23, 24, 89
stochastic, 20
token game, 20
transient state, 24, 89
PUMA, 16, 29, 48, 61, 74, 105
CSM, 18

Queueing Network, 14
extended (EQN), 14
layered (LQN), 15

Resources
performance annotation, 83
replication, 95, 139
utilization, 14, 93, 139
Response time, 13, *see* Responsiveness
computation, 92
performance annotation, 83
Responsiveness, 14, 91

Scalability, 14, 92
computation, 92
Sequence (diagram), 11
performance annotations, 85
performance role, 76
Simulation models, 15
SOA, 7
SOAP, 8, 26, 36
Software
architectures, 5
performance engineering (SPE), 12
Software architecture
assessment, 9
mobile agents, 9
Service-Oriented (SOA), 7
styles, 6
SPE, 13
antipatterns, *see* Performance antipatterns
assessment methodology, 73, 74, 125
performance measures, 22
performance models, 14
performance objectives, 13
performance patterns, *see* Performance patterns
performance scenarios, 13
process, 16
PUMA, 16
workload, 13
SPRINGS, 42
architecture, 43
performance analysis, 46, 48
performance improvements, 55
performance results, 54
scalability, 55
tracking, 44
State Machine (diagram), 11, 158
performance annotations, 85
performance role, 77
queries in ArgoSPE, 158
Steady state, 23, 24, 89
Stereotype, 80
Stochastic
Petri net (SPN), 20
process algebra (SPA), 15

Tag value, 80
Throughput, 13
Transient state, 24

UCH, 60, 109
architecture, 60
performance analysis, 66
performance improvements, 71
performance models, 61
scalability, 68
UDDI, 8, 26
UML, 10
diagrams, 10
MARTE, 12
profiles, 11

SPT, 12, 18, 29, 46, 159
URC, 59, 60
Usability principles, 91, 133
Use Case (diagram), 11
performance role, 76
Utilization, 14, 93, 134
computation, 94

VSL, 12, 80, 126

Web Services, 7
performance issues, 26
SOAP, 8, 36
UDDI, 8
WSDL, 8
XML, 34
Workload, 13
performance annotation, 83
WSDL, 8, 26

XMI, 160, 161
XML, 8, 60
deserialization, 27
parsers, 34
serialization, 27