Exposing and Aggregating Non-Functional Properties in SOA from the Perspective of the Service Consumer

Hanane Becha

Under the supervision of

Dr. Daniel Amyot

and the co-supervision of

Dr. Azzedine Boukerche

Thesis submitted to the

Faculty of Graduate and Postdoctoral Studies

in partial fulfillment of the requirements for the degree of

PhD in Computer Science

Under the auspices of the Ottawa-Carleton Institute for Computer Science

uOttawa

University of Ottawa

Ottawa, Ontario, Canada

October 2012

Abstract

Non-functional properties (NFPs) represent an important facet of service descriptions, especially when a Service Oriented Architecture (SOA) approach is used. An effective SOA service development approach requires the identification, specification, implementation, aggregation, management and monitoring of service-related NFPs. However, at this point in time, NFPs are either not handled at all or handled partially in proprietary ways. The goal of this thesis is to encourage their availability for use.

In this thesis, the focus is on the NFPs relevant from the perspective of service consumers, in opposition to the perspective of service providers (or developers) and to multi-perspectives. In other words, the scope covers only the NFPs that need to be published to help service consumers determine whether a given service is an appropriate one for their needs (e.g., description of NFPs to be attached to the service along with the functionality description).

This thesis provides the following contributions to the SOA knowledge base:

- Definition of a domain-independent catalogue comprising 17 NFPs relevant to the descriptions of atomic services from the perspective of service consumers. These NFPs have been derived from a literature review and have been validated via a two-step survey;
- Formalization of NFP representation by defining data structures to enable quantifying and codifying them, together with a corresponding XML schema;
- Definition, implementation and validation of algorithms to aggregate the NFPs of the composite service based on the NFPs of its underlying services, with a discussion of the NFP aggregation limitations;
- Definition of a modeling approach for the NFP-aware selection of services, which involves aspect-oriented modeling with the User Requirements Notation, in the context of SOA;

- Integration of NFP descriptions into the Web Services Description Language (WSDL); and
- Definition and use of the discriminator operator in service composition, to enable the creation of fault-tolerant composite services.

Overall, this work contributes to research by providing better insight on the nature, relevance, and composability of NFPs in a service engineering context. As for industrial impact, this work contributes a validated collection of NFPs with a concrete syntax and composition algorithms ready to be used for defining, selecting, and composing NFP-driven services and for evolving current SOA-related standards.

Acknowledgment

The production of this dissertation provided a significant challenge that could not have been met without the assistance of many others.

I would like to recognize the significant expert support provided to me by Dr. Daniel Amyot, who was my supervisor for my Ph.D. thesis, and I thank him for this support. I have benefited tremendously from his invaluable support and patient guidance at every stage of my research. Besides being an excellent supervisor, Daniel treated his students as part of his team and as friends. I am really glad that I had the privilege to work under his supervision.

Also, I thank my co-supervisor Dr. Azzedine Boukerche for our many discussions, his helpful advice and tips that helped me to stay focused and on the right track.I would like to thank the members of my examination committee, namely Dr. Ferhat Khendek (Concordia University), Dr. Abdulmotaleb El Saddik (University of Ottawa), Dr. Liam Peyton (University of Ottawa) and Dr. Michael Weiss (Carleton University), for their helpful comments, criticism and advice.

This work was financially supported by the admission scholarship program of the University of Ottawa, by the NSERC Discovery Grant and the Canada Research Chair of Dr. Boukerche, and by the NSERC Discovery Grant of Dr. Amyot.

I thank Dr. Bilel Jamoussi for his trust in me and the numerous opportunities made available to me so that I was exposed to many difference facets of this exciting topical area as part of my career development.

I am grateful to Dr. Chris Hobbs, who provided me with background information, stimulating discussions, and inspiration for my research area.

Special thanks to Dr. Gunter Mussbacher, who helped tremendously with his explanations on Aspect-oriented URN and his contributions to the BookItWell case study. As well, I really appreciate the support of Joe Zebarth (B.A.Sc., P.Eng.) and Dr. Julio Medina.

I express my deepest gratitude to my parents Ahmad Becha and Mongia Gualdiche; my brothers Firas, Sami and Mohamed; and my husband Samy Jaoua. They have always believed in me, supported and encouraged me.

Finally, I would like to dedicate this dissertation with all my love and gratefulness to my parents, my husband and my sons Adam (first year at school) and Alexandre (first year at kindergarden).

Table of Contents

List of Figures

List of Tables

List of Acronyms

Acronym	Definition
AoGRL	Aspect-oriented Goal-oriented Requirement Language
AoUCM	Aspect-oriented Use Case Map
AoURN	Aspect-oriented User Requirements Notation
AT&T	American Telephone and Telegraph, Inc.
BP	Business Process
BPEL	Business Process Execution Language
CORBA	Common Object Request Broker Architecture
DAML-S	DARPA Agent Markup Language
DCOM	Distributed Component Object Model
E^3	Evolutionary multiobjective sErvice composition optimizEr
GPS	Global Positioning System
GRL	Goal-oriented Requirement Language
ICT	Information and Communication Technologies
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
ITU-T	International Telecommunications Union
JAXB	Java Architecture for XML Binding
MARTE	Modeling and Analysis of Real-Time Embedded Systems
MTBF	Mean Time Between Failures
MTTF	Mean Time to Failure
MTTR	Mean Time to Repair
NFP	Non-Functional Properties
NFR	Non-Functional Requirement
OASIS	Organization for the Advancement of Structured Information Standards
OCL	Object Constraint Language

OGF	Open Grid Forum
OMG	Object Management Group
OWL	Web Ontology Language
OWL-S	Semantic Markup for Web Services
PKI	Public Key Infrastructure
QoS	Quality of Service
REST	Representational State Transfer
RPC	Remote Procedure Call
SHOE	Simple HTML Ontology Extensions
SLA	Service Level Agreement
SLAang	SLA Language
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SQRM	QoS Requirements Matcher
SSL	Secure Sockets Layer
TMF	TeleMangement Form
UCM	Use Case Map
UDDI	Universal Description Discovery and Integration
URI	Uniform Resource Identifier
URN	User Requirements Notation
UX	UDDI eXtension
W3C	World Wide Web Consortium
WCF	Windows Communication Foundation
WS	Web Service
WSCM	Web Service Composition Management
WSDL	Web Services Description Language
WSLA	Web Service Level Agreement
WSM	Web Service Management
WSML	Web Service Management Language
WSMO	Web Service Modeling Ontology
WSOL	Web Service Offerings Language

WS-Policy	Web Services Policy
WS-QDL	WS-Quality Description Language
WS-QF	WS-Quality Factors
WSQM	Web Service Quality Model
WS-QTG	WS-Quality Test Guideline
WS-QUC	WS-Quality Use Case
XML	The Extensible Mark-up Language
XSD	XML Schema Document

Chapter 1. Introduction

This thesis provides a framework to define, describe and aggregate different types of non-functional properties (NFPs) of services from the perspective of service consumers (e.g., what should be included in a service description to help service consumers decide whether a given service suits their needs) when using a Service Oriented Architecture (SOA) approach. These NFPs can be used to differentiate services that have equivalent functionality. Service discovery algorithms can and should take into account these NFPs for service filtering (or matchmaking) and selection.

1.1. Context

An SOA approach promotes sharing and reusing software components. It is the paradigm in which the building of software applications is based on the composition of loosely-coupled services, residing in the network and accessible via standardized protocols, into larger composite services. The use of an SOA approach enables business processes to be implemented quickly from existing services and allows those implementations to be adjusted rapidly to meet changing business processes requirements.

The SOA approach is technology agnostic (i.e., not tied to any specific technology), although it is commonly implemented using Web Service (WS) standards developed primarily by the Organization for the Advancement of Structured Information Standards (OASIS, [61]) and the World Wide Web Consortium (W3C, [85]). A reference model for SOA has been defined by OASIS [60]. Erl [25] states that: “Designers can implement SOA using a wide range of technologies, including REST (Representational State Transfer), RPC (Remote Procedure Call), DCOM (Distributed Component Object Model), CORBA (Common Object Request Broker Architecture), Web Services or WCF (Windows Communication Foundation). As well, SOA can be implemented using one or more of these protocols and, for example, might use a file-system mechanism to communicate data conforming to a defined interface-specification between processes conforming to the

SOA concept. The key is independent services with defined interfaces that can be called to perform their tasks in a standard way, without a service having foreknowledge of the calling application, and without the application having or needing knowledge of how the service actually performs its tasks."

Adoption of the Service Oriented Architecture approach is significantly shifting the traditional software development methodology. The SOA approach brings with it the promise of increased agility and flexibility, often speeding up application development time by an order of magnitude in comparison with the development time in present long-established, tightly-coupled, monolithic environments. As stated by Mohiuddin [53], the SOA approach provides agility and the capability for adjustment of business processes. As the business changes, developers can more easily map business processes changes to applications and then implement the appropriate IT changes.

The typical actions associated with service registration and invocation using an SOA approach are depicted in Figure 1. The SOA concept is predicated on the requirement that service providers publish descriptions of their services in registries (Step 1 of Figure 1). Once registries are available, service consumers can then query (Step 2) these registries to discover (Step 3) the available published services and they can then pick, invoke, and even compose services to meet their business process needs (Step 4). Once invoked, the service will perform its functionality and send a response (Step 5). Figure 2 illustrates an approach to service composition: a service consumer can discover and pick services from the service registry to create a composite service in response to particular business process needs. Using the SOA approach allows consumers to perform service composition without having to develop or even understand the underlying logic and implementation details of services they compose and use. Services that have not been stripped of specific business logic or usage are not generic enough to be reused in different business domains. Such rich services should be reduced so that they are generic in nature and thus may be used in different business domains. The use of distributed services in different contexts by different service consumers who do not have control over them, raises some challenges with respect to the ability to predict the behaviour of the resulting composite service and associated service level agreements (SLAs) without resorting to tight binding to the underlying services.

The focus of this dissertation is on the non-functional properties of services from the perspective of the service consumers (i.e., what should be included in the service description to define its non-functional properties). A relevant NFP refers to an NFP that i) should be exposed as part of the service description to accommodate the service consumer; ii) should be taken into consideration when SLAs are developed; and iii) could be verified by service consumers.

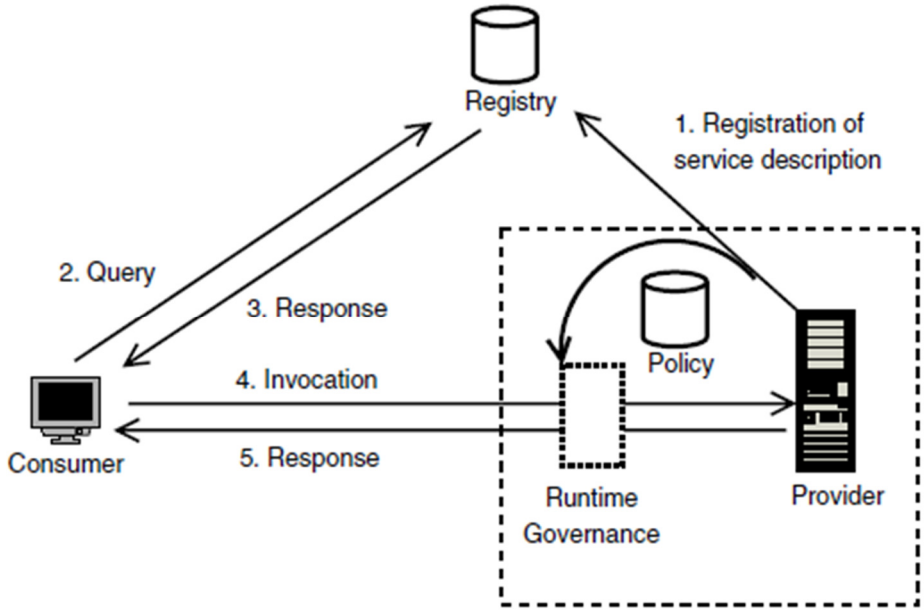


Figure 1 SOA Components and Typical Actions [12]

Attention must be given to handling of non-functional properties of the services, including how to define, represent, aggregate (i.e., compute the NFPs of composite service based on the NFPs of its underlying services) and verify the non-functional properties of service such as cost, availability, reliability, failure, and security.

Currently, there is no standardized model that defines NFPs for atomic services and describes how to aggregate NFPs for services composed using services provided by various service providers. Current NFP description and aggregation methods typically use proprietary implementations (i.e., use a non-standardized form of service descriptions, including the NFPs to be filled by all the service providers), and as a result cannot be aggregated by third parties. There are a number of NFPs for which descriptive mecha-

nisms do not exist. Other NFPs that can be expressed still require significant work to be aligned to enable their aggregation and provide NFPs description of composite services. While these capabilities are important to all services, they are particularly so for time-sensitive and mission-critical ones. In domains such as aircraft control and healthcare systems, improper handling of the NFPs can cause significant economic, human and organizational damage.

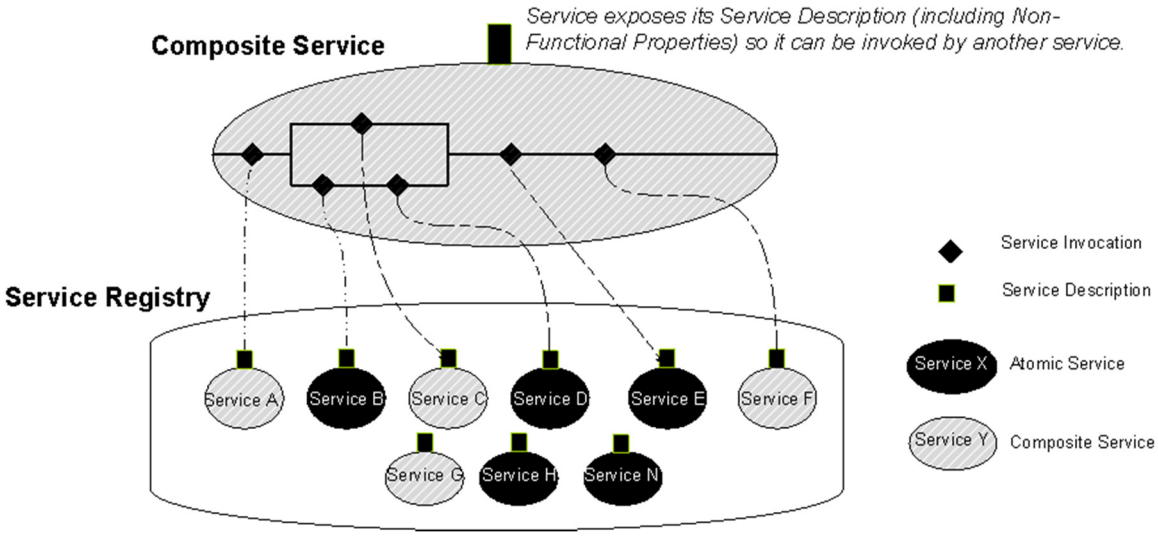


Figure 2 Service Composition

In addition, NFPs play an important role in each stage of the SOA process lifecycle, i.e., in the development, discovery, ranking, selection, substitution and composition of services. This is due to the fact that multiple services can deliver the same functionality and the only differentiators between these available services are their NFPs. Consequently, the ability of the service consumer to choose services based on their NFPs is crucial.

1.2. Illustrative Use Case

A simple but intuitive use case will be used to motivate and later illustrate the proposed approach to handling NFPs in services. People who drive their cars for long distances understand the annoyance associated with trying to find a suitable lodging (e.g., hotel or motel) at the end of a long day of travel if they did not rearrange in advance a place to stay for the evening. Such a search would involve obtaining a map of the local area and

typically driving off the main road to look for hotels with vacancies. The imaginatively named *BookItWell* service provided by *DriveALot Inc.* could be the answer for such people. In this use case, adapted from the example presented in [31], the company DriveALot has composed the underlying services into a business flow known as BookItWell. Each of the underlying services may themselves be composed of lower-level services, but this is of no interest to DriveALot.

Although the BookItWell service is fictitious, all of the necessary components exist and several related services are already available from various service providers. This service BookItWell is built by composing a number of services provided independently by different companies, specifically including those for calendar, weather forecast, location, hotel booking, motel booking, and route calculation. Some of these services may be invoked sequentially, others in parallel. Note that DriveALot is both a service provider (i.e., to the subscribers of BookItWell) and a service consumer (e.g., of another provider's route calculation service).

1.3. Motivation and Challenges

In systems engineering, requirements can be classified into two categories: functional requirements (i.e., requirements that specify something that the delivered system must be capable of doing) and non-functional requirements (i.e., requirements that specify quality attributes the system must possess and how well it must perform its functions). Examples of non-functional requirements include those related to availability, reliability and usability.

Non-functional requirements are of critical importance and, at times, functional requirements might be sacrificed to meet them [27]. Dealing with non-functional requirements is difficult since it is often hard to determine whether they are met, and they often contradict each other and may impact functional requirements when conflicts are solved.

In the SOA context, non-functional requirements are even more difficult to handle than in traditional software engineering due to the highly distributed nature of SOA ap-

plications where services are used in different business domains and under different contexts by a variety of stakeholders that do not have a complete control over them.

In SOA, applications are structured as a collection of services. A composite service is a collection of services, orchestrated to meet a specific design requirement in order to achieve desired real effects with acceptable levels of performance and reliability, while also satisfying other non-functional requirements. A composite service is typically used to implement a particular business process, such as the BookItWell use case discussed in the previous section. As illustrated in Figure 2, a composite service can be composed using published services, intended for use in the composition of services (i.e., composable), whether the published services are composite services or atomic services. The services used in the composition, their NFPs, and the manner in which they are arranged within the composition have a direct effect on the resulting NFPs of the developed composite service. The relationship between NFPs of a composite service and the services from which it is composed can be complex. For example, similar to component based systems, it is *not* true that if all of the services used to compose a composite service (such as BookItWell) have 90% availability, then the availability of the composite service is 90%.

To enable third parties to consume published services and, in turn, to provide composite services and their associated descriptions, the NFPs of the base services must be predictable at design and execution time and be verifiable. This implies, first, that published atomic and composite services should expose some if not all of their NFPs and, second, that these NFPs could be aggregated in some fashion. As already mentioned, NFPs play an important role in each stage of the SOA process lifecycle (i.e., service development, discovery, selection, substitution and composition) because NFPs are the only way to differentiate among services that can deliver the same functionality. For example, in BookItWell, several weather forecast services could be available as candidate subservices, but one may be more precise and less costly to use in a given geographical area, hence affecting the qualities of BookItWell as a whole. Composition techniques should insure a proficient service selection and an adequate NFPs aggregation of the resulting composite service. Without a proper handling of the NFPs, the SOA approach should not be adopted for use in mission critical and time sensitive applications.

Based on what was observed in an extensive literature review, there are a number of ***challenges*** that have to be overcome before composition techniques can properly handle NFPs:

1.3.1 Availability of a Catalogue of NFPs

There is no defined catalogue of the NFPs that should be included in service descriptions. In addition, the current service description languages (i.e., used to describe service interface behaviour and to specify their location) such as Web Services Description Language (WSDL) [86] do not contain all the elements required for generating effective service descriptions. In particular, value estimates (e.g., best-case and worst-case) required for service descriptions are lacking. Generally, advertising mechanisms do not exist for a number of important NFPs. Additionally, the existing composition languages, such as BPEL (Business Process Execution Language [62]), are lacking support for NFPs aggregation.

1.3.2 Ability to Handle Failure Modes

Failure modes of individual services should be predictable. At present, web services cannot advertise their failure mode or specify how best for a service consumers to handle them when something goes wrong or for a composite service to compensate for their failure. In business domains, such as those that involve human safety (e.g., healthcare services or disaster response), throwing an exception is often not an adequate failure handling mechanism. The current service composition paradigm is effectively based on the assumption that web services do not fail. In the real world this assumption is unrealistic as a number of problems may occur while running a service, e.g., unpredictable system failures (e.g., memory errors, resource contention, network errors, site down and processor crashes), invalid data (e.g., faulty user-derived data, out of range values, and incorrect filenames), and programming errors. In a SOA based operating environment, services might be developed by a person other than the consumer; and thus this leaves the consumer in the position of having no control over the faulty behaviour of services being consumed. Moreover, there is no defined ontology of service failure modes. As well,

while it may be possible to express the failure modes of two atomic services X and Y, their resulting failure mode of the composite service may be too difficult to specify in such way that the failure mode of the resulting composite service Z can be computed.

The following illustrates a situation where a composite service comprised of two services can result in a service in which its failure mode is difficult, if not impossible to specify. Consider a first atomic service X. This service fails cleanly and then, because the service is *idempotent* (i.e., the service, when invoked multiple times, has no further effect on its subject after the first time it is performed), may immediately be called again. By itself, this would be acceptable. Consider now the second atomic service Y, which has its own more complex failure mode. The composed service, i.e., composite service Z, contributes its own failure mode, in addition to those of X and Y. It is now apparent that attempting to predict the end result or ultimate outcome of all these failure modes is very difficult. Such composition issues are not unique to failure modes and they will be explored for many other types of NFPs.

1.3.3 Service Provider Issues

Service providers encounter many challenges and lack experience in knowing how to guarantee privacy and data integrity across borders while satisfying differing legal requirements of countries or jurisdictions. In addition, testing SOA-based applications is still in its infancy [8].

1.3.4 Summary of Some Important Challenges

From the above (and from the literature review in Chapter 3), at this time, non-functional properties are either not handled at all or are handled in ad hoc ways. A catalogue of domain-independent NFPs is still to be defined. Descriptive mechanisms are needed for a number of non-functional properties. Aggregating non-functional properties is very challenging at this stage and still requires additional attention. Current methods for handling non-functional properties are inadequate, especially for mission-critical applications. An effective SOA approach requires the identification, specification, implementation, management, aggregation and verification of non-functional properties of services. Means to

evaluate and compare the NFPs of the services, as well as composition techniques based on these NFPs, are needed.

1.4. Research Hypothesis

Our research hypothesis is that a domain-independent framework can be used to handle the description of non-functional properties of services from a consumer perspective. A domain-independent catalogue of NFPs can be defined to better characterize and aggregate NFPs, in order to enable advanced applications such as NFP-aware service selection, service redundancy, service composition and dynamic adaptation.

1.5. Methodology

The methodology used in this thesis is inspired from Hevner's design science research [29], which is about learning by building and about producing viable artifacts. The main artifacts built here are a validated catalogue of NFPs, together with NFP composition algorithms, implemented and integrated to WSDL, a popular service definition language. The detailed methodology is as follows:

1. A literature review was performed to collect approaches and NFPs typically used in SOA, and assess the relevance of a new catalogue of NFPs.
2. An online survey was conducted with experts and practitioners to define and evaluate an initial catalogue of NFPs relevant from a customer's viewpoint. The results of this survey helped refine the initial catalogue of NFPs into a second one, which was validated through a second online survey, and refined again into a final catalogue. Note that this catalogue reflects mainly what the SOA community itself thinks at this period of time.
3. The NFPs were formalized as data structures (i.e., a metamodel) and were given a formal syntax in XML (through an XML schema) in order to enable their precise description and quantification.
4. NFP aggregation algorithms were defined based on these data structures, implemented as programs, and validated through testing.

5. The practical use of the catalogue formalization was validated through its integration with a mainstream WS-enabling technology (i.e., WSDL extension).
6. A goal and scenario modeling language, namely the User Requirements Notation extended with aspect-oriented concepts, was used as a means to model and analyze NFPs, NFP composition, and service selection in SOA. This is again meant to validate the potential of the catalogue in the context of service composition and selection.
7. A use case (BookItWell) was used to illustrate and validate the proposed catalogue, the aggregation algorithms and the service selection approach.
8. The criteria used to evaluate the related work were also used to assess the thesis contributions.

1.6. Thesis Contributions

This thesis provides the following major contributions:

1. Definition of a domain-independent catalogue comprising 17 NFPs relevant to the descriptions of atomic service from the perspective of the service consumers. These NFPs have been derived from a literature review and have been validated via a two-step survey;
2. Formalization of NFP representation by defining data structures to enable quantifying and codifying them, together with a corresponding XML schema;
3. Definition, implementation and validation of algorithms to aggregate the NFPs of the composite service based on the NFPs of its underlying services, with a discussion of the NFP aggregation limitations;

In addition, several minor contributions are provided:

4. Definition of a modeling approach for the NFP-aware selection of services, which involves aspect-oriented modeling with the User Requirements Notation, in the context of SOA;
5. Integration of NFP descriptions into the Web Services Description Language (WSDL), the most widely adopted WS description enabling language; and

6. Definition and use of the discriminator operator in service composition, to enable the creation of fault-tolerant composite services.

Overall, this work contributes to research by providing better insight on the nature, relevance, and composability of NFPs in a service engineering context. As for industrial impact, this work contributes a validated collection of NFPs with a concrete syntax and composition algorithms ready to be used for defining, selecting, and composing NFP-driven services and for evolving current SOA-related standards.

1.7. Publications

The following publications were produced out of this thesis:

- Becha, H., Mussbacher, G., and Amyot, D. (2011) Modeling and Analyzing Non-Functional Requirements in Service Oriented Architecture with the User Requirements Notation. Chapter in: N. Milanovic (Ed.) *Non-functional Properties in Service Oriented Architecture: Requirements, Models and Methods*. IGI Global, 2011, 48–72.
- Becha, H. and Amyot, D. (2012) Non-Functional Properties in Service Oriented Architecture – A Consumer's Perspective. *Journal of Software*, Vol. 7, No. 3, Academy Publisher, March 2012, 575–587.
- Amyot, D., Becha, H., Bræk, R., and Rossebø, J.E.Y. (2008) Next Generation Service Engineering. *ITU-T Innovations in NGN Kaleidoscope Conference*, Geneva, Switzerland, May 2008. IEEE Computer Society, 195-202. (Acceptance rate: 23%)
- Hobbs, C., Becha, H. and Amyot, D. (2008) Failure Semantics in a SOA Environment. *3rd Int. MCeTech Conference on eTechnologies*, Montréal, Canada, January 2008. IEEE Computer Society, 116-121.

1.8. Thesis Outline

The thesis is structured as follows:

- Chapter 2 provides background information such as definitions of the terms and notation elements used in this thesis;
- Chapter 3 provides a literature review on non-functional properties, with an emphasis on their application to service description and composition;
- Chapter 4 defines a domain-independent catalogue comprising 17 NFPs relevant to the descriptions of atomic service from the perspective of the service consumers, together with a formal abstract syntax (metamodel) and a concrete XML-based syntax;
- Chapter 5 investigates the aggregation of these NFPs, which is required to compute the NFPs for a composite service based on composition scenarios. Algorithms are provided for 11 out of the 17 types of NFPs in the catalogue. The sample composite service BookItWell is used to illustrate and validate the composition algorithms;
- Chapter 6 presents how the NFP definitions and aggregation functions are applicable to the NFP-aware selection of services. Again, BookItWell is used to illustrate the approach;
- Chapter 7 presents validation results based on the implementation of the composition algorithms, the integration of the NFP schema to WSDL, and tool support; and
- Chapter 8 summarizes the contributions, provides conclusions and discusses future work items.

Chapter 2. Background

This chapter presents some key terms as well as the scenario/workflow modeling notation and Web Service Description Language (WSDL) essentials used in this thesis. The terminology selected to describe components of a SOA is based in part on the OASIS SOA Reference Model [60]; definitions of terms and concepts are adapted from various sources. Also, a number of specific terms were introduced and defined to facilitate their understanding in the context of the problem statement of this thesis. The User Requirements Notation, defined in [42], will be used to model and analyze non-functional service requirements and will also be used to formalize scenario composition, especially service composition using NFPs. WSDL will be used as proof of concept of the usage of the proposed NFP catalogue.

2.1. Definitions

This section presents basic service-related terminology used throughout the thesis.

Service Oriented Architecture (SOA): "a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations" [60].

SOA is based on the principle of developing applications by using services, which are accessible over a network, and which developers build to be shared and reused inside and even outside a given enterprise. As mentioned above, SOA is technology agnostic and thus it is not tied to any specific technology; however, it is commonly implemented using standard Web Service technologies. SOA could also be implemented using a wide range of technologies, including REST (Representational State Transfer), RPC (Remote

Procedure Call), DCOM (Distributed Component Object Model), CORBA (Common Object Request Broker Architecture), and Web Services.

With SOA, "The key is independent services with defined interfaces that can be called to perform their tasks in a standard way, without a service having foreknowledge of the calling application, and without the application having or needing knowledge of how the service actually performs its tasks." [25]

When the SOA approach is compared with traditional development platforms, it can be seen that SOA has a number of unique features:

- SOA-based applications are highly distributed: requirements such as NFPs are not applied to and assessed against a single application as a whole, but they are to be applied to and assessed against multiple independent distributed services;
- SOA-related published services are developed to be used in different business domains;
- SOA interaction models assume that service providers and service consumers generally are part of different organizations and hence service consumers are often not involved in any stage of the development phase of SOA-based services;
- Many SOA-based applications require that the NFPs of a service be defined and measured prior to services being published;
- Consumers of SOA-based services have no access to the underlying logic of the services that they use and thus they do not have complete control over them. Consequently, consumers of SOA-based services cannot assess the behavioural aspects of services until they are actually invoked.

Service: a mechanism to enable access to one or more capabilities provided by an entity (service provider) for use by others (service consumers).

A service cannot be defined unless at least one consumer of the service can be anticipated. In the SOA approach, services are usually stripped of specific business logic or usage in order to make them generic so that they may be reused in different business domains. Services are intentionally opaque, which means that service providers do not expose the underlying logic or implementation to consumers. Only information required by service consumers is exposed in the service description, i.e., the information needed to

determine whether a given service is appropriate for consumers' needs, and how to interact with it. Services are loosely coupled from consumers; in other words, service providers and consumers usually belong to totally independent groups or even companies. In SOA, services are said to be: discoverable; autonomous; stateless (i.e., services are not required or expected to manage state information); and if the policies permit it, composable into other services.

Service Description: the information required by SOA service consumers to determine whether a given service is appropriate for their needs or not, as well as the information necessary to interact with the service, such as service interfaces, behaviour and location.

The service description should convey what is accomplished when the service is invoked and the conditions for using the service. It also defines the service inputs, outputs, and associated semantics. Based on the OASIS SOA reference model [60], a SOA service description should contain the following four types of information:

1. *Information Model*: A service description should provide an information model that contains the format of the information to be exchanged, the structural relationships existing within the information to be exchanged, and definitions for the terms used. For example: The click-to-call service requires a 10 digit phone number or an identity.
2. *Functional Model*: A service description should provide a functional model that identifies actions on, responses to, and temporal dependencies between actions on the service. For example: In response to the action of a user entering a password, a connection will be established.
3. *Policy and Contract Specifications*: A service description should provide policy and contract specification agreements, whether it be between just two or more than two parties. These policy and contract specification agreements usually specify the conditions of use of a service, i.e., the specified conditions may provide insight as to how the service will work and may identify any constraints associated with the use of the service. For example: While a click-to-call service will provide click-to-call functionality, the contract may specify that it can only be used during

a specified period of the day, e.g., only during office hours (8:00 to 15:00, in a particular time zone or city).

4. *Non-Functional Properties*: A service description should provide information about non-functional properties. This information should describe the service aspects related to availability, reliability, security, scalability, and performance. For example, in a wireless call, more than 90% of the packets should be received without errors in order to have an acceptable communication quality.

Service Policy: a formalized description of conditions of use related to a particular service such as supported encryption algorithms constraints, composability conditions, included security features and constraints regarding use, e.g., confidentiality and privacy rules to be abided by.

Abstract service: a service whose capabilities are known and that is provided in an abstract manner such that the service does not specify an individual implementation or the technology that is used to support it.

Service Level Agreement (SLA): an agreement between the service provider and the service consumer that typically indicates the performance of the service that service consumer can expect and the payment that the service provider will receive. An SLA specifies the negotiated and mutually agreed upon service level commitments including the conditions associated with the service to be provided, including but not limited to: the quality of service to be, how the quality of service will be measured, and what happens if the service quality is not met. Typically, an SLA addresses the functional and the non-functional aspects of the service.

***Note*.** The main difference between an SLA and a service description is that while an SLA is based on a negotiation mechanism between the service provider and the service consumer, the service description is only defined by the service provider. One should not assume that a service consumer will invoke a service on the basis of a Service Level Agreement. For example, a SOA service represented by its Web Service interface could be invoked when WSDL is available. In this situation the service description can be

used by the service consumer to become better informed about the expected service behaviour and conditions.

Atomic Service: an indivisible service.

Note. This is a service that does not invoke another service.

Composite Service: a service that results from the act of service composition. A composite service can be the result of composing atomic services, composite services, or a mixture of both. Note that a composite service cannot by definition be an atomic service.

***Note*.** In the remainder of this thesis, the use of the term "service" covers both "atomic" and "composite" service.

NFPs of a Service: the subset of a service description that describes its non-functional properties.

Note: In the literature, NFPs are referred to as Quality of Service (QoS) properties as well. In the remainder of this thesis, the acronyms "NFP" will be used to refer to a single non-functional property (NFP), and "NFPs" will be used to refer many discrete non-functional properties (NFPs), or an aggregation of non-functional properties that together characterize a service.

Service Composition (or **Service Composability**)**:** the process of combining (or gluing) atomic or composite services into a larger composite service by defining the sequence and conditions in which a higher-level service invokes other services in order to realize a richer function (e.g., the combination of two underlying services, X and Y, to produce a composite service Z). As shown in Figure 2, this concept is at the heart of the SOA approach.

Static Composition: service composition that combines concrete services that are selected at design time.

Dynamic Composition: service composition that combines abstract service descriptions to be instantiated by concrete services at run time.

***Note*.** In the reminder of this thesis, the use of the term "composition" covers both "static" and "dynamic" composition.

Service Substitution: replacing one service with another functionally-equivalent service, without regard to their non-functional properties.

Service Provider: an entity, person, or organization that offers a software capability over the network as a service. Service providers may offer services that they have developed themselves, or services that they have composed based on third party services, or simply resell services developed by third parties.

Service Requestor: an entity, person, or organization that requests a particular service to check and decide if it is suitable for its current needs (one stage before becoming a service consumer).

Service Consumer: an entity that makes use of a particular service. This entity could be a person, a service or any automated mechanism invoking the service. A given service could be invoked as is or be invoked as part of a composite service.

Service Broker: an intermediary between a service provider and a service requestor or consumer (e.g., for service location or for brokered trust arrangements).

Service Participant: a service provider, service requestor, service consumer, or service broker.

***Note*.** The rest of this thesis will use mainly the notions of service providers and service consumers.

2.2. User Requirements Notation

The *User Requirements Notation* (URN) is an International Telecommunication Union (ITU-T) standard for capturing early requirements in the form of scenarios and

goals [42]. URN consists of two complementary visual sub-languages called Goal-oriented Requirement Language (GRL) and Use Case Maps (UCM) for goal modeling and scenario/process modeling, respectively. URN facilitates the elicitation, specification, analysis, and validation of early requirements and business processes/workflows. URN's unique capabilities for modeling both processes with UCM and goals with GRL in a unified way are a significant advantage over other process modeling notations. The integrated view of UCM and GRL not only answers the *where*, *what*, *who*, and *when* questions of process models, but also answers *why* a particular part of a process exists. Furthermore, URN's analysis capabilities can be used to evaluate the goal model for trade-off analysis among stakeholder goals and to establish a test suite for the scenario model. URN allows typed traceability links to be established between modeling elements (e.g., between a GRL goal and a UCM responsibility). UCMs model scenarios and use cases related to GRL elements are used to express relationships between responsibilities (sequencing, alternatives, concurrency, decomposition, etc.). These responsibilities represent activities that need to be performed, and they can be allocated to components describing actors, roles, software modules, network elements, etc. Various links between GRL and UCM elements are supported in URN to specify traceability and refinement. The basic notation elements are summarized in Figure 3 and Figure 4.

GRL and UCMs have successfully been used to model and analyze requirements for various types of systems (telecommunication, wireless, object-oriented, reactive systems, embedded, agent-base, operating, Web-based, e-business, e-health, etc.), and have more recently been used for business processes modeling, patterns formalization, reverse-engineering, performance engineering, test generation, and architecture evaluations. Tool support for the URN notation is provided by *jUCMNav*, an Eclipse plug-in that enables the creation, analysis, and transformation of URN models [45].

As explained by Amyot and Mussbacher [5], URN combines modeling concepts and notations for non-functional requirements and scenarios (mainly for operational requirements, functional requirements, and performance and architectural reasoning). In this thesis, URN will be used as a means to model and analyze functional and non-functional service requirements, especially in NFP composition (Chapter 5) and service discovery (Chapter 6).

2.2.1 Goal-oriented Requirement Language

The Goal-oriented Requirement Language (GRL) is a graphical notation used to model and analyze goals. GRL enables the modeling of stakeholders, business goals, qualities, alternatives, and rationales. Modeling goals of stakeholders with GRL makes it possible to understand stakeholder intentions as well as problems that ought to be solved. GRL enables business analysts to model strategic goals and concerns using various types of intentional elements and relationships, as well as their stakeholders called actors. Core intentional elements include goals themselves, softgoals for qualities, and tasks for activities and alternative solutions. Intentional elements can also be linked through AND/OR decompositions. Elements of a goal model can influence each other through contributions, displayed as arrows. Qualitative positive (make, help, some positive) and negative (break, hurt, some negative) contribution levels exist, as well as quantitative contribution levels on a scale going from –100 to +100. GRL captures business goals of many stakeholders, alternative solutions that are to be considered for a system and how they impact stakeholder goals, decisions that were made, and rationales that helped make these decisions. The notation elements are summarized in Figure 3. GRL strategies enable the analysis of GRL models and the propagation of satisfaction levels from an initial set of intentional elements to the other intentional elements of the model. This mechanism can be used for trade-off analysis.

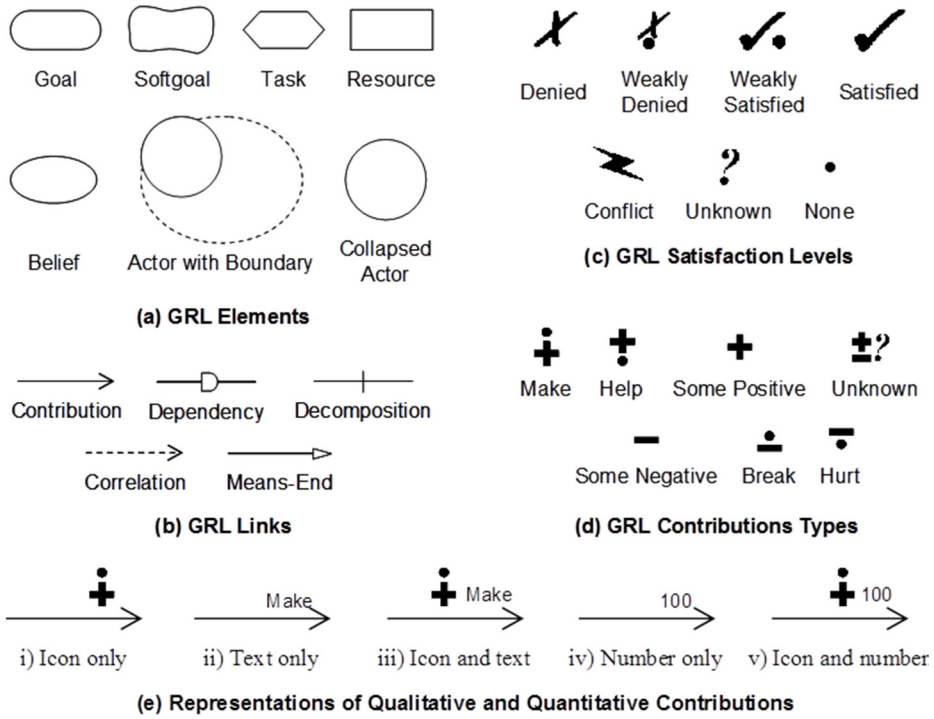


Figure 3 Summary of GRL Basic Notation Elements

2.2.2 Use Case Map

The Use Case Map (UCM) notation is a visual process modeling language for specifying causal scenarios and optionally allocating their activities to an underlying structure of components and actors. UCMs model scenarios, workflows and processes with causal relationships, where responsibilities are sequenced and may be assigned to components. A UCM path begins at a start point (●) and ends with an end point (▌). Stubs (◇) are containers for sub-models. Expanding a stub leads to a sub-map, which provides more details about the step. Responsibilities (✕) are used to illustrate atomic tasks that are not (or cannot be) represented in more detail. UCM components can be decomposed recursively. Composition operators for sequencing, alternatives, and parallelism are included, among others.

A subset of the UCM notation is illustrated in Figure 4.

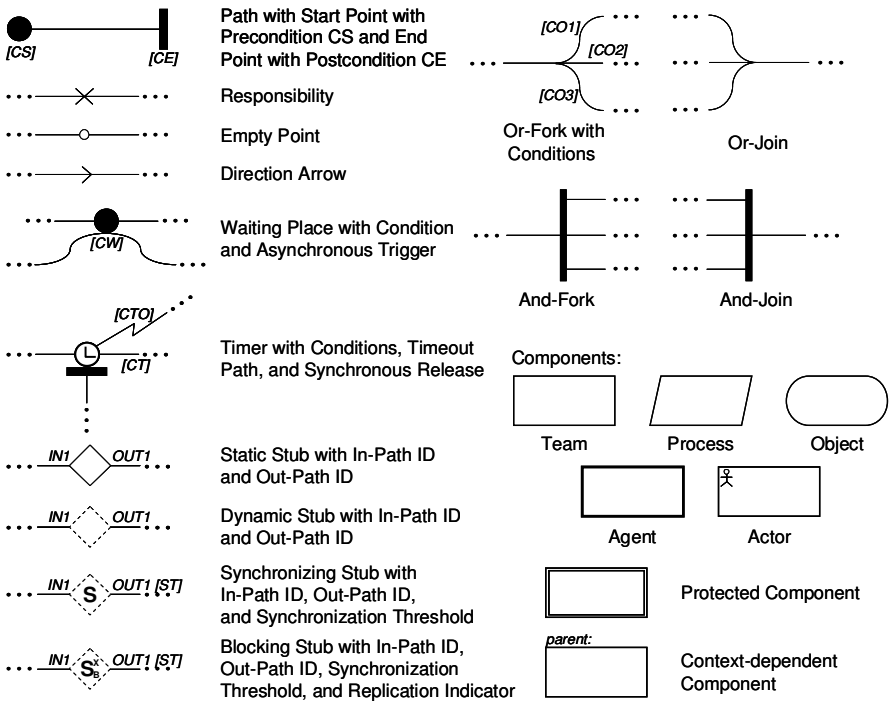


Figure 4 Summary of UCM Basic Notation Elements

2.2.3 Aspect-oriented User Requirements Notation

The *Aspect-oriented User Requirements Notation* (AoURN) is a modeling framework that extends URN with aspect-oriented concepts [55], allowing to better encapsulate crosscutting *concerns* that are hard or impossible to encapsulate with URN models alone.

AoURN treats concerns as first-class modeling elements. AoURN groups all relevant properties of a concern such as goals, behavior, and structure, as well as pointcut expressions needed to apply new goal and scenario elements to a base model or to modify existing elements. A *pointcut expression* is a pattern that must be matched if the aspect is to be applied. AoURN uses GRL and UCM diagrams to describe pointcut expressions. AoURN adds aspect concepts to views, leading to Aspect-oriented GRL (AoGRL) [58] and Aspect-oriented UCMs (AoUCM) [57]. Concerns and aspects are first-class modeling elements in AoURN. AoURN employs an exhaustive aspect composition technique that can fully transform URN models. With aspects, NFRs can easily be encapsulated in their own modules and selectively composed.

Figure 5 shows a subset of the AoURN notation. The *pointcut stub* (P) is used in an aspect to contain the pointcut expression(s) to be matched in the other concerns of the model. The location of each match in the other concerns is indicated by the insertion of various kinds of *aspect markers* (◆), which point to the *advice* parts of the aspect.

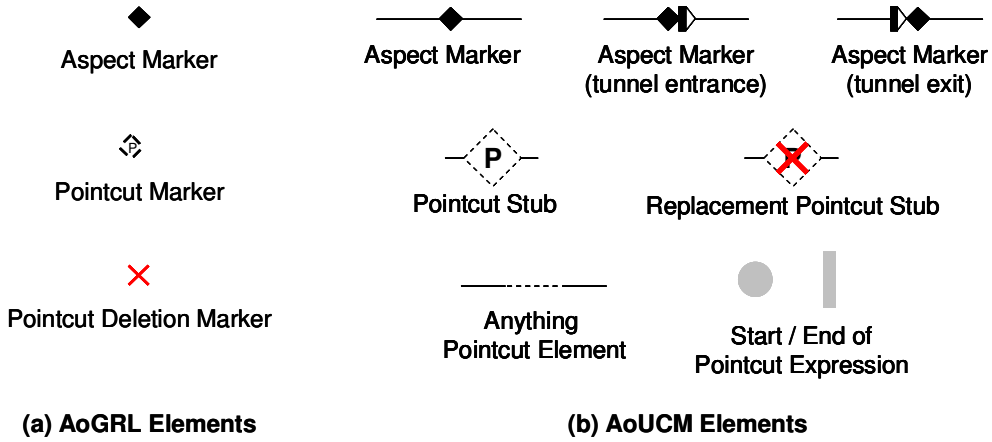


Figure 5 Subset of the AoURN notation

2.3. Web Service Description Language

This section introduces some essential concepts related to the Web Service Description Language (WSDL). WSDL is an XML grammar defining how to describe web services based on four critical pieces of data [86]:

- Interface information that describes all publicly available functions;
- Data type information for all message requests and message responses;
- Binding information about the transport protocol to be used; and
- Address information for locating the specified service.

The WSDL schema document has six major elements [14]:

- **Definitions:** Root element of a WSDL document that contains the name of the web service and its namespace (e.g., a Uniform Resource Identifier – URI);
- **Types:** Data types that will be transmitted (e.g., XSD schema);
- **Messages:** Definition of the names of the messages, which contain data being communicated (e.g., message parameters and message return values);

- **Port type:** Abstract set of operations (e.g., complete one-way or round-trip operation);
- **Binding:** Specific protocol for data transmission for a particular port type; and
- **Service:** The location of the service.

Each WSDL element (not only the six major elements detailed above but also other elements such as definitions/binding/operation, definitions/binding/operation/input, definitions/binding/operation/output and definitions/binding/operation/fault) can be extended and can also have an optional documentation element. Extensibility elements allow WSDL extensions such as support for new protocols, without having to revise the base WSDL specification. According to the standard [86], it should be noted that:

- When the WSDL definitions element is extended, the introduced extensions apply to the whole WSDL document;
- When the WSDL type element is extended, the introduced extensions apply to the format of exchanged messages; and
- When the WSDL port, binding or operations elements are extended, the extensions apply only to those extended elements.

For example, WSDL has only basic data types defined (such as string, float, double, integer, boolean, time, and date). The XML schema specification permits the creation of new data types called *complex data types* that can be attached to any WSDL element. Complex data types could be either defined under the WSDL element or defined in separate documents and referenced (imported) within the WSDL extended element.

In the latter option, the specific XML schema used to validate the data types should be referenced at the WSDL *definitions* element level via the namespace (e.g., www.w3.org/2001/XMLSchema). In this case, the complex types could be reused within multiple services.

It is important to keep in mind that a service could be defined across multiple documents (i.e., in multiple documents or document parts). As well, a WSDL document can define zero or more services [86].

2.4. Chapter Summary

This chapter has defined key terms that will be used throughout the thesis. In addition, the User Requirements Notation was introduced together with its aspect-oriented extensions as they will be used to model some part of NFP aggregation and service selection in later chapters. The WSDL essentials are recalled as they will be needed for the proof of concept of the NFP catalogue usage. The next chapter will be dedicated to the review and evaluation of work related to the current problem statement and proposal.

Chapter 3. Literature Review

The first part of this literature review is dedicated to the identification of documents that outline the existing foundational web services stack technology as one of the SOA enabling technologies. It also identifies documents that define extensions to the SOA framework allowing for the proper handling of non-functional properties of service. Subsequently, existing solutions to address some of the identified gaps will be presented and evaluated against specific criteria. A summary will highlight opportunities for improvement.

3.1. SOA Enabling Technologies

SOA is becoming one of the most popular software architectures for services engineering. It is emerging as the premier integration framework for complex and heterogeneous enterprise systems. According to a recent report by WinterGreen Research Inc. [84], SOA middleware markets grew from $4 billion in 2010 to $5.5 billion in 2011, and are anticipated to reach $8.8 billion by 2018. Several efforts have led to the development of a SOA software platform. Even though SOA is not tied to any specific technology and could be implemented using a wide range of technologies including, REST, RPC, DCOM, CORBA, and WCF, it is commonly implemented using standards Web Service (WS) technologies, with IBM as a market leader [84]. The foundational web services stack technology [34] is founded on multiple XML-based standards that have been developed primarily in W3C [85] and OASIS [61] to specify web services and to support their interactions. The Simple Object Access Protocol (SOAP) [87] defines format rules for exchanging XML-based messages between web services. Interface and implementation details of available web services can be defined using the Web Services Description Language (WSDL) [86]. A distributed online directory has been defined, i.e., the Universal Description Discovery and Integration (UDDI) [63]. The creation of the UDDI directory

enable companies to publish their services, search and discover other companies' services and understand the methods required to invoke a specific service [63]. The Business Process Execution Language (BPEL) [62] was developed to extend the web services interaction model to enable the composition of web services. This extension allows the definition of a complete set of business processes that can be used to support business transactions. Service composition has become a key functionality in SOA service development.

SOA development requires service development support that includes consideration of functional and non-functional properties. However, the currently defined SOA enabling technologies (XML standards and web services) lack support for the development of services where one wishes to consider non-functional properties. Standardized models and ontologies to describe and aggregate NFPs of services across various services providers are still required to be developed.

3.2. NFP-Related Problems in SOA and Evaluation Criteria

In SOA, non-functional properties are of critical importance in the discovery, selection and substitution of services. A proper handling of service-associated NFPs during SOA service development requires further investigation and agreement on a number of aspects. The following is a list of actions, shown as a number of steps, which if undertaken will facilitate the consideration of NFPs in SOA service development:

1. Identify and validate a list of the most relevant domain independent NFPs that need to be considered for service description development (from the service consumer's perspective).
2. For each identified NFP, define and represent the NFP in a formal and measurable way (i.e., provide a data structure with a concrete syntax).
3. For each identified NFP, investigate the ability to aggregate the NFP. It is probable that the aggregation of some NFPs is deemed to be impossible. If the aggregation is possible, an automated aggregation function or algorithm should be defined.
4. For each identified NFP, investigate whether its agreed-on performance was met (deterministically).

5. Propose a dynamic composition technique that allows comparing the NFPs of functionally equivalent services and selecting the service that provides the best trade-off while satisfying the users' constraints and preferences.

The steps described above are rewritten in terms of problems to be solved and are presented with a set of companion evaluation criteria, which can be used to evaluate the related work and, later on, the approach proposed in this thesis.

Problem 1 - NFP Catalogue: Define and validate a formal catalogue of the most relevant domain-independent NFPs for services from the perspective of the service consumers. This catalogue should be capable of being used as a template to be (fully or partially) populated by the service provider.

Relevant NFPs, once exposed, are those parameters that can add value to the interaction between service providers and service consumers as part of the service description. To add value, relevant NFPs have to be measurable, verifiable (i.e., can be tested by the service consumer to assess whether they were met, or certified by external trusted organizations), and ideally can be aggregated to be exposed as part of the resulting composite service characteristics.

Evaluation Criteria: (1) Are the identified NFPs defined in a formal way with a proposed data structure? and (2) Are the identified NFPs validated using an external process?

Problem 2 – Aggregation of the NFPs: When possible, propose an aggregation function for each NFP or explain why one cannot aggregate this NFP.

Evaluation Criteria: Do the aggregation functions take into consideration the different composition operators? Related work with regards to the aggregation function is classified as (1) taking into consideration the different composition operators including conditional branching and parallel branching **or** (2) having a simplistic treatment of composition operators, **or** (3) neglecting composition operators altogether.

Problem 3 – NFP-aware Service Selection: Propose NFP-based service selection algorithms for dynamic composition of services.

Evaluation Criteria: Does the NFP-aware service selection function select the appropriate service for a given task (1) based on *one* NFP as the user's evaluation criterion (i.e., sorting the available services based on one user constraint) **or** (2) based on the user's rating of the different criteria (i.e., once the rating is done, one is back to the previous configuration: sort the available services based on the combination of the weighted criteria) **or** (3) based on the best trade-off between *multiple* NFPs. In addition, whether the selection process (4) investigates the possibility of using multiple services or multiple instances of the same service to best accommodate the user's constraints.

It is impracticable to propose a complete and definitive catalogue of relevant NFPs. Hence, ideally, this catalogue should be extensible; then, the service provider could add other NFPs that he deems useful, including domain-specific ones. The identified NFPs should be defined in a formal way with a proposed data structure. In addition, the degree of formalism should facilitate the aggregation of the NFPs (i.e., specify the aggregation rules and functions). The aggregation functions should take into consideration the different composition operators. Ideally, the NFP-based service selection algorithms for service composition should select the appropriate service for a given task based on the best trade-off between *multiple* NFPs. In addition, ideally, the selection process should investigate the possibility of using multiple services or multiple instances of the same service to best accommodate the user's constraints.

3.3. Evaluation of Existing Solutions

Extensive work has been done in systems engineering on requirements definition. Requirements are generally classified into two categories: functional and non-functional. Whereas functional requirements define what a system is supposed to do, *non-functional requirements* characterize how a system is supposed to be. Although, the acronym NFP is used in this thesis to refer to the non-functional properties of services, it should be understood that in the literature, non-functional requirements are often referred to as *qualities*

of an application as well as "constraints, quality attributes, quality goals, and quality of service (QoS) requirements" [83].

In the following sections, the work related to non-functional requirements (in general, and in the SOA context from the service consumer perspective in particular) is summarized and evaluated using the set of criteria defined in section 3.2. In the following sections, the existing work that can be leveraged is identified as are the limitations that need to be overcome.

3.3.1 NFP Description versus SLA

It is important to define what an NFP description is so that there will be no confusion when NFP discussions occur. While many people will suggest using SLA templates as NFP descriptions, it is important that this not be done. It cannot be stressed often enough that a SLA is a business-level, custom-made, negotiated (and potentially re-negotiated) contract between two specific participants (service provider and service consumer) and hence it is not realistically usable as an NFP description at the technical level. The information contained in a service description is typically a subset of the information that will be contained in an SLA. Using only a service description, the provider can propose several similar services and the consumer may pick the one that seems the most appropriate for the service to be composed, i.e., meets the required needs. While this approach is less powerful than an approach which uses custom-made SLAs, it is sufficient and it may have advantages in certain situations. Using a service description requires less operating expenses, efforts and time during service composition than what would be needed if one had to negotiate an SLA.

In a nutshell, a precise service description (including NFPs) is always needed to enable service comparison and selection. SLAs are typically used by big players who need assurance that they will have access to specific levels of service. Whether service description or SLAs are used, there is still a need for detailed NFP service descriptions since NFPs may have many dimensions of differences [77]. As an example during service selection, how does one compare the availability of different services if their presented NFP formats are different; e.g., one service defines availability as a percentage and the other as a ratio. A similar situation occurs when trying to compare services that express

their prices in different currencies. Having a detailed NFP catalogue enables service comparison and selection. As well, an NFP catalogue complements and should be used by SLAs and service management specifications. The availability of a precise NFP catalogue description would reduce the time it takes to complete the SLA's development process.

The majority of related works define SLAs for XML-based Web Services (WS). These languages make many assumptions about the infrastructures at the service provider and service consumer sides. As a result, these specifications are not easy to reuse in other contexts or infrastructures, or to extend for specific domains.

Among these languages, the Web Service Level Agreement (WSLA) [50] (defined by IBM) and the Web Service Management Language (WSML) [72] (defined by HP) stand out. Both are XML-based SLA specifications. Each has built-in NFPs metrics that make the built-in lists of NFPs hard to extend in some cases and may cause more overhead in other or even the same scenarios.

SLAng [47] is another language that can be used for specification of SLAs for WS. Again, the definitions of NFP metrics are built into the SLAng schema and, as a consequence, SLAs developed using this approach have a predefined format. Thus, the current version of SLAng lacks flexibility.

Web Services Agreement Specification (WS-Agreement) from the Open Grid Forum (OGF) is a formal specification of agreement, and agreement templates using XML for grid services. Grid services are a special case of web services. The agreement (or the SLA) references the service descriptions including the NFPs that will be delivered under the service agreement.

Based on the above discussion it can be seen that SLAs do not provide solutions that could address the three problems identified in section 3.2. SLAs cannot replace the use of NFP descriptions to enable service discovery and comparison.

3.3.2 NFP Description versus Formal Specification of Constraints

NFP descriptions should not be confused with the formal specification of constraints. Specifically, having processes available to provide formal specification of constraints does not benefit either NFP definition or NFP metrics modeling. Several languages are

being developed (e.g., Web Service Offerings Language (WSOL) [77] and Semantic Markup for Web Services (OWL-S) [20]) to support management of web services and their dynamic compositions. In essence these XML-based languages formalize various constraints (a.k.a., policies) for web services, including those related to NFPs. To complement these efforts, NFP taxonomies or ontologies are needed.

While the following identifies the current state of affairs with respect to management, policies and registries with NFPs, it can be observed that just the formal specification of constraints does not address the three problems defined in section 3.2. Specifically, an NFP description is needed to complement the formal specification of constraints languages that support web service management.

Languages: OWL-S, WS-Policy and WSDL

Semantic Markup for Web Services (OWL-S), formerly known as the DARPA Agent Markup Language (DAML-S) [20], is an XML-based language for semantic descriptions of web services. It provides a declarative language for service properties and capabilities advertisement (e.g., placeholders for the description of NFPs). However, OWL-S neither proposes an exhaustive list of NFPs to be exposed as part of the service description nor does it provide the NFP metrics to enable NFP comparison. Hence, NFP descriptions should not be confused with OWL-S. An NFP description is needed to complement the OWL-S view.

As well, Web Services Policy (WS-Policy) [32] is a general framework that may be used to specify policies for web services. The details of the specification of particular categories of policies (including NFP definitions) need to be defined in specialized languages.

The Web Services Description Language (WSDL) was dedicated to describe the functional aspects of the service. When this language was developed, it did not aim to describe the NFPs of the service.

Policies

In Web Service Modeling Ontology (WSMO) [79], the authors sketch some domain specific NFPs from the functional point of view of the web services, such as for Amazon services. The recommended set of NFPs include the following: Accuracy, Contributor,

Coverage, Creator, Date, Description, Financial, Format, Identifier, Language, Network-related QoS, Owner, Performance, Publisher, Relation, Reliability, Rights, Robustness, Scalability, Security, Source, Subject, Title, Transactional, Trust, Type, and Version. These NFPs are just mentioned, without formal definitions.

Registries

In addition, UDDI eXtension (UX), described in [16], empowers the Universal Description, Discovery and Integration (UDDI) registries to collect feedback from the service consumers on their satisfaction about the NFPs of the invoked services, and to compute the average satisfaction. The average satisfaction is used to predict the performance of the service in the future. Their NFP list is limited to response time, cost and reliability.

3.3.3 SOA versus Mashups

Both SOA and mashups are about integration, collaboration, sharing and re-using of components or services. In general, SOA and mashups are two complementary concepts that aim to accelerate and simplify the applications development process. The emergence of SOA is considered, along with the success of social applications, as an enabler for mashups [21]. SOA can be seen as a concept for technologists, while mashups are for business-oriented people. Mashups are defined as relatively simple, component-based development paradigms [21].

One of the main differences between these two concepts is that SOA does not have to be Web-based, while mashups do. In addition, while the focus of SOA is on the interoperability among applications, mashups focus more on data aggregation.

One can argue that enterprise mashups are the new face of SOA or that mashups are changing the nature of SOA. There are actually a number of very practical synergies between mashups and SOA. In general, SOA and enterprise mashups are converging and their differences are mainly cultural. However, we cannot just replace SOA by mashups in this thesis without careful investigation of the possible implications.

In conclusion, the same exercise done in this thesis in the context of SOA can be redone in the context of mashups. The methodology and the lessons learned in the context of SOA will definitely be beneficial for mashups.

3.3.4 ISO/IEC 9126

ISO/IEC 9126 [38] is an international standard for evaluating the quality of conventionally (i.e., non-SOA) developed applications. This standard is divided into four categories: quality model, external metrics, internal metrics and quality-in-use metrics. In conventionally developed applications, NFPs are collected and identified based on the needs stated explicitly or implicitly by the stakeholders and are then implemented via technical solutions and product integration processes. The implementation of these NFPs is monitored and confirmed via verification and validation processes. The NFPs are designed conceptually and then measured and validated operationally. The development process is iterative: the NFPs can be improved using analysis and measurement of the application as well as feedback from the users.

This standard, when applied to traditional applications development environments, addresses the issues associated with Problem 1, defined in section 3.2; but it does not satisfactorily address these issues when applied in a SOA environment. The unique characteristics of SOA, including its distributed nature and its new interaction model between service providers and service consumers, bring up a new set of challenges. Hence, the catalogue defined for traditional software engineering is not entirely relevant for SOA applications, where the NFPs should be defined and measured before publishing the service. The ISO/IEC 9126 standard cannot be used as is to measure the quality of SOA-based applications. The set of quality characteristics depicted in Figure 6 should be reviewed and reassessed to determine whether they could be adapted (e.g., filtered and/or extended) to support SOA, taking into account its unique features.

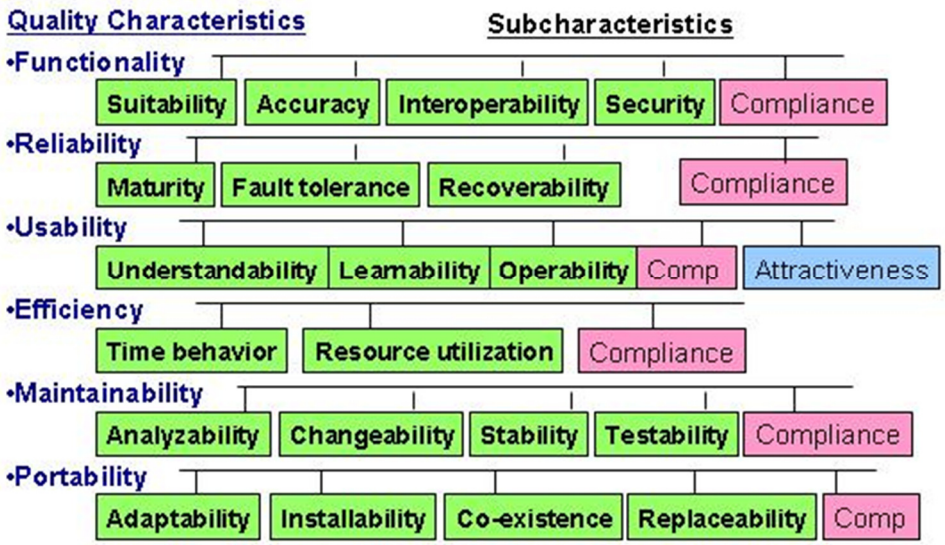


Figure 6 ISO/IEC 9126 Software Engineering: Quality Model [38]

3.3.5 OASIS Quality Model for WS

The Web Service Quality Model (WSQM) [44] was initiated in September 2005 at OASIS with the aim to define NFPs specifically for web services technology. The NFPs of the services are referred to as WS-Quality factors. This standard [44] defines the WS-Quality characteristics and provides guidelines and detailed implementation notations for service providers. The Web Service Quality Model (WSQM) comprises five specifications: WS-Quality Model (WS-QM), WS-Quality Factors (WS-QF), WS-Quality Description Language (WS-QDL), WS-Quality Test Guideline (WS-QTG), and WS-Quality Use Case (WS-QUC).

The WS-Quality Factors specification structure, presented in Figure 7, is the most related part to the focus of this thesis. It aims to define and formalize a set of attributes in the context of contracting for web services. This work addresses only Problem 1. The NFP factors are introduced at the conceptual level and only a few metrics are provided to specify quality attributes (i.e., Problem 1 is only partially addressed).

A method to aggregate these factors (or NFPs) was not proposed and thus Problem 2 and Problem 3 have not been addressed.

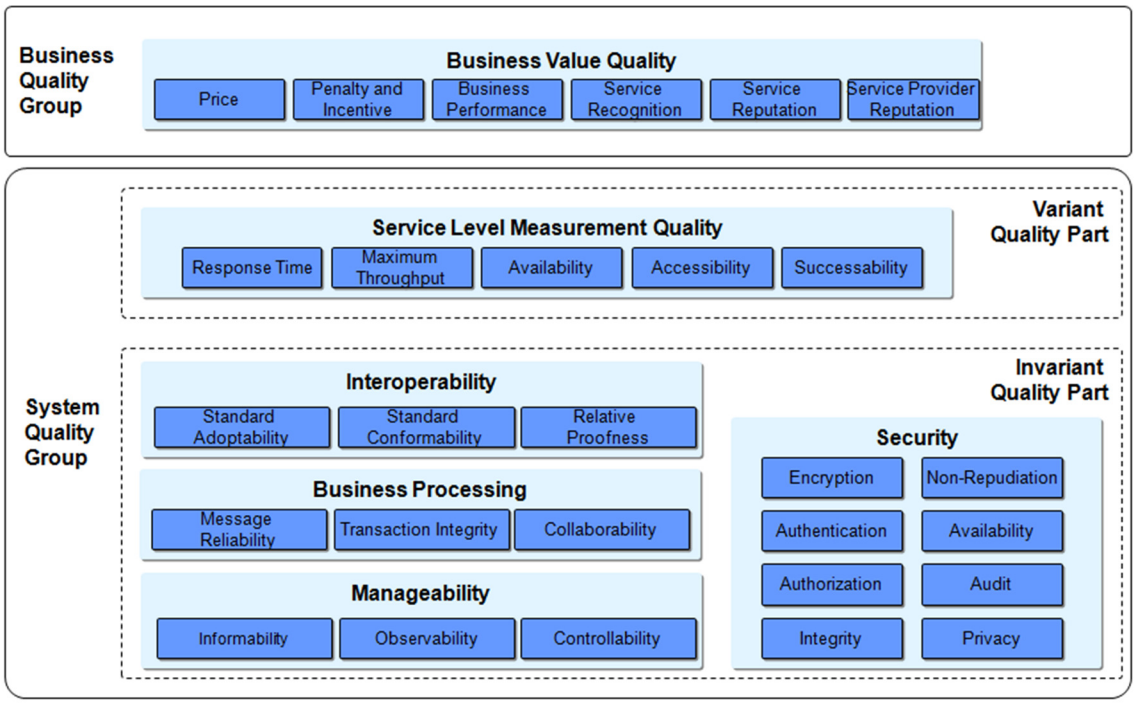


Figure 7 Structure of Web Services Quality Factor [44]

3.3.6 Balfagih's Multi-Stakeholders' Perspective

Balfagih *et al.* examine the QoS of SOA-based services in the literature and classify them from the perspectives of the developer, the provider, and the consumer [7]. While they provide a definition for almost every QoS property, metrics are only defined for a few of them. Their paper indicates that it is believed that if a complete set of metrics QoS is developed, it will enable services monitoring at run time.

The QoS classification according to [7] is as follows:

- Developer's QoS: maintainability, reusability, and conformance.
- Provider's QoS: availability, reputation, security, discoverability, accountability, interoperability, and throughput.
- Consumer's QoS: response time, availability, reliability, security, usability, composability, and robustness.

In regards to security, their work neither provides a definition nor metrics; rather, a mention is made of a few sub-factors including confidentiality, integrity, authentication, and availability.

Similarly, in regards to usability, this paper provides a definition without metrics, with a mention of sub-factors including understandability and configurability.

In regards to composability, the work in [7] again provides a definition but no metrics. Composability is the essence of SOA (e.g., the capability of composing services is a fundamental SOA requirement as a composition capability is required to create more complex services). Hence, one can argue that composability should not be part of the catalogue. The following sub-factors related to composability are identified: conformability, availability, discoverability, and the loosely coupled nature of web services. These sub-factors are the ones proposed in their models for the developer's and provider's perspectives. The paper concludes, without much evidence, that the QoS properties that interest developers and providers are also important to consumers as well.

In summary, this paper provides a good literature review and QoS classification, with the multi-stakeholders perspective as an interesting contribution. This paper does not provide a validation methodology or justification of the chosen or eliminated QoS and thus does not satisfy criterion 2 of Problem 1. However, it provides some metrics to measure specific quality attributes, and thus partially satisfies criterion 1 of Problem 1. Finally, this work does not provide direction with regards to the aggregation and selection of these factors (or NFPs); and thus it does not address Problems 2 and 3.

3.3.7 Choi's QoS Metrics for Service Provider

Si Won Choi *et al.* [17] have identified some of the unique features of SOA and then derived six quality attributes and their respective metrics to measure each attribute, as summarized in Figure 8; their work has hence partially solved Problem 1. The proposed set of attributes is intended to be used by service providers to ensure that a qualified service is published.

This paper is useful as it is possible to add to the proposed set of NFPs, as it currently is not rich enough to address all the concerns of service consumers. For example, it does not include certain NFPs such as cost and the execution time of the services.

This paper does not consider aggregation and selection of these attributes, and thus it does not address Problems 2 and 3.

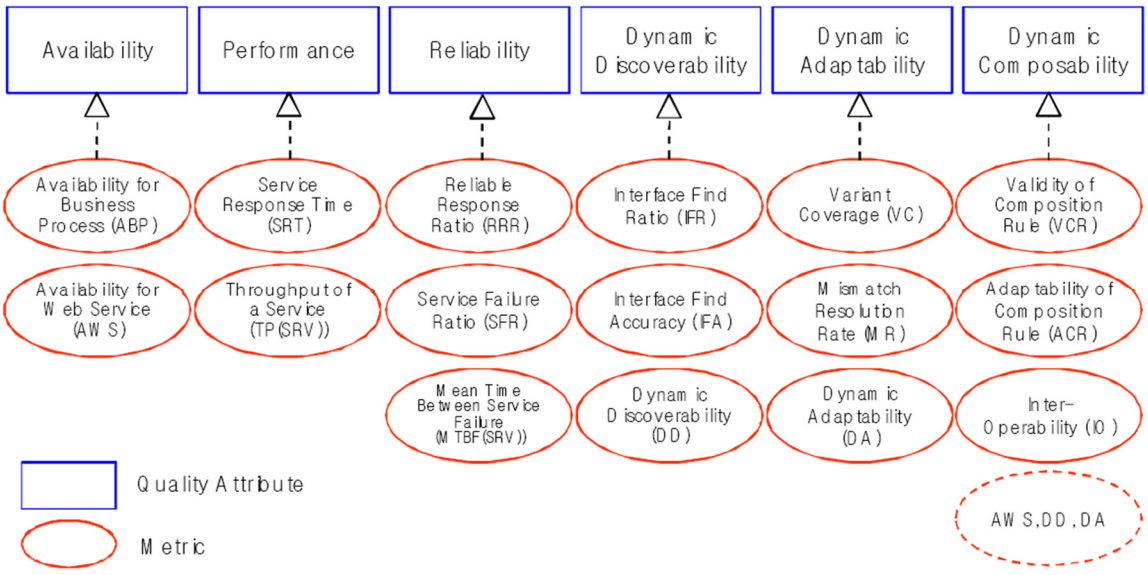


Figure 8 QoS Metrics for Service Provider [17]

3.3.8 Galster's Taxonomy for NFPs

Glaster *et al.* [27] recognized the critical importance and the difficulties associated with handling NFPs in general and the fact that they are even more difficult to handle in the SOA context. They attempted to generate a checklist of NFPs for SOA. Their NFPs are organized into three categories: process requirements, non-functional external requirements, and non-functional service requirements.

The latter category partially addresses Problem 1. The authors provide an extensive and generic list of various NFPs, depicted in Figure 9, to be considered when atomic or composite services are under *development*. However, they provide only informal information on the quantifiability of these NFPs. They do not provide any operational representation or metric to evaluate these NFPs, and as a result fail in fulfilling the required degree of formalism for Problem 1. In addition, this paper does not consider aggregation of these attributes, and thus it does not address Problems 2 and 3.

C3.1	Usability	Measure of the quality of a user's experience in interacting with the service application (e.g., the product must help the user to avoid making mistakes, quantified as the ratio between number of functions understood by the user and the total number of functions).
C3.2	Reliability	Ability to keep operating over time (e.g., mean-time-to failure).
C3.3	Performance	Response time, throughput, and timeliness (e.g., messages exchanged per time-unit).
C3.4	Capacity	Ability to handle different capacities (e.g., max. number of service-users at a time).
C3.5	Portability	Ability to be ported to other environments (e.g., number of supported interface specifications for different target systems, such as WSDL).
C3.6	Safety	Capability of service / system to achieve acceptable levels of risk of harm to people, business or environment in a specified context (e.g., incidence of damage as the ratio between damage occurrences and total number of usage situations).
C3.7	Security	Measures for confidentiality, authenticity and integrity, i.e., the capability of system or service (or the transmission between services) to protect information so that unauthorized services / applications cannot read or modify them and authorized services / applications are not denied access to them (e.g., number of detected types of illegal operations versus the number of types of illegal operations according to service specification).
C3.8	Accessibility	Constraints on interface definitions and the information included in the interface (e.g., number of cases in which service is accessible versus the total number of cases where service was looked up).
C3.9	Visibility	Degree to which a service is visible to and discoverable for potential users (e.g., web service must be visible through the world-wide-web across organizational borders, quantified as ratio between service requests and hits). NOTE: A visible web service is not necessarily accessible.
C3.10	Interoperability	Ability of communication entities to share specific information and operate on it according to an agreed-upon operational semantics (e.g., system must operate with any of the existing web services from the same domain, quantified as ratio between number of data formats which are approved to be exchanged between services and the total number of data formats to be exchanged).
C3.11	Availability	Degree to which a system or component is operational and accessible when required to function (e.g., up-time).
C3.12	Scalability	Ability of service to function well when system is changed in size or volume (e.g., service performance parameters should not change if number of service users doubles).
C3.13	Extensibility	Ease with which service capability can be extended without affecting other services (e.g., credit card service is extended with new security feature, quantified as time at which extension is requested minus time at which service specification is updated and new capability is available).
C3.14	Adaptability	Ease with which a system can be changed to fit changes in requirements (e.g., number of data which are operable but not observed due to incomplete operations caused by adaptation limitations divided by number of data which are expected to be operable in the environment to which the service / system is adapted).
C3.15	Testability	Degree to which a service facilitates the setting of test criteria and the performance of tests to determine if criteria have been met (e.g., if service is to be tested using existing acceptance tests which have been defined during requirements specification: ratio between number of cases in which suitable acceptance tests exist versus the number of test opportunities).
C3.16	Modifiability	Ability to make changes to service quickly and cost-effectively (e.g., impact of change on one service does not have impact on any other service, quantified as the number of impacted services).

Figure 9 Non-functional Service Requirements in SOA [27]

3.3.9 What's in a Service?

The publications [26], [67] and [66] recognize the need for a basic set of domain-independent, non-functional properties that can be used to improve discovery, comparison, and service substitution in SOA.

O'Sullivan *et al.* [26] highlight two concerns with the existing approaches to the NFP description of web services. Their first concern is called "Web service tunnel vision" and results from endorsing the need to utilize lessons learned from conventional service descriptions (e.g., including those from traditional development processes). This thesis shares the view that a broader vision is required than that of just studying conventional service description because SOA brings new challenges that did not exist in the context of conventionally developed services and applications.

Their second concern is called "semantic myopia" and is about not taking advantage of the semantic richness of NFPs. It is recognized that rich descriptions of the exposed services (requiring more effort on the part of the service providers) will increase the complexity of user interfaces and may require ontology alignment (i.e., using currencies conversion for handling prices). This thesis shares the view that NFP descriptions in SOA should take advantage of all the existing work related to non-functional requirements, which should be leveraged to address the new SOA challenges.

The work in [66] is a large technical report that contains 79 models to describe NFPs covering the following: availability, payment, price, discounts, obligations, rights, penalties, trust, security, and quality. O'Sullivan *et al.* have done a comprehensive work on the importance of NFP detailed descriptions as a vehicle to improve discovery, comparison and service substitution. They outlined the importance and the complexities related to NFP handling and highlighted the issues associated with the use of rich NFP descriptions.

It is a fact that NFP descriptions are indispensable for the service life cycle (e.g., service discovery, substitution, composition, and management). The authors highlight some useless service composition scenarios such as when one of the used services is available only on weekends in Europe and another one is available only during weekdays in Asia. The authors deplore the static nature of services catalogues and they indicate that rich service descriptions are needed for service delivery conformance and service man-

agement in general. There is the expectation that services will evolve as a result of participants' interactions, changes to the service environment, and changes to the NFP constraints, including security standards requirements and their implementation mechanisms.

O'Sullivan *et al.* also presented a list of service NFPs. Their methodology is based on the review of existing commercial services, mainly conventional ones. They do recognize that additional NFPs may be needed. O'Sullivan *et al.* focus on justifying the need for clear NFP descriptions by enumerating their usage scenarios. They do not use external validation of their list of NFPs and thus do not satisfy criterion 2 of Problem 1.

The NFPs cited in [67] are the following:

- *Temporal* (i.e., when) and *spatial* (i.e., where, for location-based services) *availability*, including the need to communicate uncertainty;
- *Channels* by which the service is requested and provided (i.e., pushing or broadcasting for unknown number of requestors);
- *Charging and payment styles* (e.g., different business models);
- *Settlement models* (e.g., mutual obligations of the provider and the requestor such as subscription, metered, facilitated, escrow, and swap) *and contracts* (e.g., terms and conditions defined in an offline environment defining who, what, where and what happens if something goes wrong similar to SLA);
- *Payment* (e.g., a list of possible payment instruments including cash, checks, direct funds transfer, credit or charge cards, travelers' checks, wire transfers, postal or money order, securities, bank bills, vouchers, stored value cards, digital cash and anonymous cash);
- *Quality*, with 5 dimensions as follows: (a) reliability (e.g., dependability and accuracy), (b) responsiveness (e.g., promptness and willingness of staff to assist), (c) assurance (e.g., courtesy of the stuff), (d) empathy (attention provided to the requestor), and (e) tangibles (e.g., concrete aspect of the service such as cleanness);
- *Security*, characterized in [67] as being closely related to identity, privacy, alteration and repudiation of information transferred between parties. Security cited approaches include the usage of Secure Sockets Layer (SSL) and the implementation of Public Key Infrastructure (PKI). The authors highlight the in-

creasing complexity of security in the context of composite services. They question the need for authenticating the sub-services of composite services, the need of security certificates of composite services, handling sub-services having different security information policies, etc.;

- *Trust*, which is very subjective. Different models are cited, including mutual trust and exclusive trust. A differentiation between trusting the intentions of a service provider and trusting its competence is made. Trust is referred to as a balance between perceived risk and cost, and benefit. The authors interrogate trust representation, trust perception changes handling, trusting a composition, the usage of previous interactions in determining trust, and implications of being distrustful; and
- *Ownership and rights*, which relate to the degree of control over a service invocation. Rights may include all or some of the following five privileges: (a) the right to comprehend, (b) the right to retract, (c) the right of premature termination, (d) the right of suspension, and (e) the right of resumption. Some of these rights may involve, and make provision for, paying a penalty.

Although each of their 79 proposed models cannot possibly be discussed here, some will be revisited in Chapter 4 to evaluate what can be reused when defining NFP metrics (e.g., defining all the possible pricing models and possible currencies). Indeed, this thesis and paper [67] have several NFPs in common. While some of the NFPs in [67] have the same name as NFPs defined in this thesis, these NFP have very different definitions (e.g., availability). In [66], O'Sullivan *et al.* define availability as temporal or spatial. This appears to be inconsistent with the view taken by this thesis, which contends that the advantage of Internet-based services is being available 24/7 and in seamless way, no matter where one is. This thesis proposes that whenever information about a region or a country is important for service delivery, there should be an NFP in the catalogue (e.g., jurisdiction) that addresses this. Moreover, the work in [67] is neither focused on the service consumer's perspective nor clear about the separation between contracts and SLAs versus service descriptions. The work tends to mix the focus of NFP definitions between the requestor needs (a.k.a. service consumer's perspective) and the provider's perspective. It

can also be noted that the work does not differentiate between mutually negotiated and agreed-on SLAs (e.g., referred to as settlement models) and multiple possible settlement models proposed by providers, allowing consumers to pick the one that match best their needs. Additionally, it appears that NFP quality is defined in a very subjective manner and thus may not be totally relevant in the context of developing SOA services (e.g., assurance and empathy). Furthermore, some of the proposed NFPs are more about information related to SLAs than about service descriptions.

Publications [26], [67] and [66] consider neither the aggregation of factors/NFPs nor NFP-based selection, and thus they do not address Problems 2 and 3.

3.3.10 Tosic's WS Offerings Language

Tosic *et al.* define a Web Service Offerings Language (WSOL) [77] to formally describe different classes (or profiles) of services. The terms "Service offering" and "classes of service" mean that different classes of service do refer to the same WSDL description but have different levels of QoS constraints and management statements. Employing the language used in this thesis, their "classes of service" correspond to "functionally equivalent services" with NFPs being their only differentiator. In WSOL, a service is defined as various formal constraints (functional constraints, QoS, and access rights) and management statements (prices, monetary penalties, and management responsibilities). WSOL along with WSDL could be used to support Web Service Management (WSM) and Web Service Composition Management (WSCM) applications. WSM allows the monitoring, metering, and accounting of web services. WSCM enables dynamic (i.e., run-time) adaptation and management of web service compositions by switching, deactivating, reactivating, and creating different classes of service. Similar to the proposal in this thesis, WSOL confirms the importance of NFPs in service selection and composition and develops mechanisms to monitor and adapt them to enable dynamic service management. However, WSOL does not define QoS constraints to be monitored but uses external ontologies of NFPs. The authors of [77] share the view expressed in this thesis that there is a need for taxonomies and ontologies that contain precise definitions for NFP metrics and how they should be measured and that ontological definitions should be domain independent. Also, ideally, appropriate standardization bodies should develop ontologies to facilitate

interoperability. However, standardization processes are often very time consuming. This work should be driven by other interested parties (as initiated in the past by OMG [65]).

The NFP catalogue developed in this thesis is in line and complementary with the work reported in [77], as it fills in a missing piece of the WSOL puzzle. While the WSOL work does not resolve any of the three identified problems directly, it does emphasize the need for an NFP catalogue as part of the larger WS offering solution.

3.3.11 Chen's QoS Driven WS Composition

Chen *et al.* [15] worked on describing some NFPs to accommodate the requirements of service consumers (e.g., the perspective of the service consumer). They provided an algorithm to compute the aggregation of these NFPs for composite services in support of NFP-based services selection. The authors share the view expressed in this thesis that there is a need for domain-independent NFPs, as well as the fact that NFPs of the services should be quantifiable. They use a seven-dimension vector to describe the NFPs of a web service. Their NFP list includes the following: response time, availability, concurrency (e.g., the equivalent of the scalability NFP in this thesis), expire time (e.g., represents the expiration time of a service, and the reliability of a service can be ensured before the expiration time), price, fine, and security level. However, as this paper does not provide a validation methodology for the proposed catalogue and does not provide the rationale for selecting or excluding particular NFPs, it only addresses criterion 1 of Problem 1.

Interestingly, the authors provide an algorithm that computes the NFPs of a composite service. Chen *et al.*'s work is somewhat simplistic in that it does not provide a case study, proofs, explanation, or details about the complexity of the composition algorithm, and hence it only partially addresses criterion 2 of Problem 2. This work also does not address Problem 3.

3.3.12 Zeng's QoS Middleware for WS Composition

In [89], Zeng *et al.* define QoS-aware middleware for web service composition. In their approach, the NFPs are first defined for elementary services (i.e., atomic services) includ-

ing execution price, completion time, reputation, successful execution rate and availability. Then, an NFP aggregation model for a composite service is expressed.

This paper touches upon the three problems defined in section 3.2: NFP catalogue and formalism, NFP aggregation and NFPs-aware service selection.

The list of NFPs provided in this paper is very restricted and must be extended to be useful; as a result, it only addresses part of Problem 1.

The proposed NFP aggregation solution is based on a limited set of NFPs and on a quite simplistic treatment of the composition operators, and thus it only partially addresses criterion 2 of Problem 2.

Problem 3 was investigated in different settings (i.e., based on local optimization versus using a global planner). The authors propose different solutions that compute the score of each service based on the user's rating of each criterion, which takes us back to the configuration of sorting available equivalent services based on one user constraint. Hence, the service with the highest score and that satisfies all the user constraints will be selected. This addresses criterion 2 of Problem 3. If no service satisfies the user constraints, then no solution will be provided. The selection mechanism neither returns the service with the best trade-off nor does it investigate the possibility of using multiple services or multiple instances of the same service to best accommodate the user's constraints. This addresses Problem 3 per criterion 2.

While this approach is acceptable for some NFPs, it is not accurate for the success rate and availability in particular. For NFP composition, the authors define as "critical" the services that belong to the critical path, which is the path in the composed services that has the longest total sum of weights labelling its nodes. In this paper, the weight is the execution duration of the service. The authors compute the NFPs aggregation taking into account the critical services path only, which is a simplistic treatment of the services that are part of parallel branches and conditional branches in the composite workflow. Also, the authors assume that when the execution of a non-critical service is not successfully done, the task can be re-executed without delaying the whole composite service execution, which is not correct.

3.3.13 Liu's QoS Service Selection

For a fair, open and objective approach on how to handle NFPs of SOA services, Liu *et al.* [49] state that one cannot rely only on NFP-related information advertised by service providers when this information is only subject to manipulation by these (perhaps untrusted) providers. Liu *et al.* propose combining NFP-related information advertised by service providers with NFP active monitoring and user feedback. The registry, where this information would be collected, is responsible for the computation of QoS values for each published service (e.g., collect and compute the users' feedback for a reputation criterion). For example, the following three generic criteria should be captured as follow: execution price is to be provided by service providers, execution duration is to be computed based on active monitoring performed by the service requestors, and reputation is to be collected using service consumer feedback. To prevent false ratings, the authors suggest the use of an identifier and of a password to ensure that only the true user of the service is allowed to provide feedback. The authors introduce the notion of "deterministic" and "non-deterministic" NFPs. The term "deterministic" indicates that the value of the criterion is certain when the service is invoked, as it would be the case for price. The term "non-deterministic" indicates that the value of the criterion is uncertain when the service is invoked (i.e., not until executed), as it would be the case for execution duration. Assuming that there is a set of web services that have the same functional properties, the service selection algorithm performs a matrix normalization to rank the services and to determine which one should be selected based on the user's preferences. For example, if the customer can specify whether the search for a particular type of service should be "price" or "service" sensitive, a price-sensitive search should return a list of the services with the lowest price, while a service-sensitive search should return a list of the service providers with the best-rated services. In this framework, Liu *et al.* consider three generic criteria (price, execution time and reputation) and three business-related criteria (transaction for maintaining data consistency, compensation rate and penalty rate).

This paper touches upon Problem 1 and Problem 3 with interesting and original contributions, but it does not address the problem of how to come to an agreement on the list of NFPs nor of how to aggregate these NFPs (i.e., not treating Problem 2). The service selection algorithm (i.e., Problem 3) does not return the service with the best trade-

off among the multiple criteria and hence does not return a list of services that are close to the users' real needs (i.e., satisfying criterion 1 and 2 only of the Problem 3). Usually, users neither want to have the cheapest and worst service nor the best service at any cost. They want good value for their money (e.g., quality/price ratio).

3.3.14 Suzuki's SLA Genetic Optimization Framework

In [78], Suzuki *et al.* propose an optimization framework called E^3 (Evolutionary multi-objective sErvice composition optimizEr) to address the QoS-aware service composition problem (i.e., Problem 3). E^3 defines a multiobjective genetic algorithm, called E^3-MOGA, which assumes that services have three QoS attributes: throughput, latency, and cost. While the cost is fixed, the two other criteria can vary at runtime obeying probability distributions known from historical data. E^3 considers running multiple service instances in parallel to improve the availability and performance.

As the E^3 approach determines (1) which concrete services to use and (2) how many instances of each concrete service to use, it satisfies criteria 3 and 4 of Problem 3.

This paper addresses mainly the problem of aggregating the NFPs of the resulting composite service (i.e., Problem 2). However, the composition patterns are quite simple (i.e., no clear treatment for conditional branches, criterion 2 of Problem 2).

3.3.15 Hobbs' Crash Only Services

Hobbs *et al.* recognize in [30] that there are serious gaps in the semantics of the published service descriptions and these hinder the adoption of the SOA approach in mission-critical applications. They identify some of these lacunae and propose a foundation for resolving one of them, namely service failure. The technique of crash-only failure is proposed as a useful first step for building a taxonomy of service failure behaviours in SOA. *Crash-only* failure is a software design approach based on the assumption that it is easier to restart quickly in a known state than to clean up and rebuild to recover from an error condition. Failures of composite services are often due to "Heisenbugs" (i.e., uncertainty principle), which tend to be impervious to conventional debugging and are generally non-reproducible. The crash-only software principle is to forget recovery since it is more

trouble than it is worth. When the server senses a problem, it "crashes" and is programmed to perform a "micro-reboot" to return to some original state (i.e., a well-defined checkpoint state). A server using this process is back working sooner than if it tried to recover via logs and journals, etc. This approach simplifies failure models, testing, and implementations. It fits the SOA paradigm principles nicely: loose coupling of services and little state shared among services. Note that, if the service is not idempotent then all responsibility for determining the state of a recovered server lies with the consumer. Crash-only architectures may reduce availability. This is because they remove layers of sophistication built using fault-tolerant techniques. Crash-only architectures trade Mean Time to Repair (MTTR) for Mean Time to Failure (MTTF). With crash-only architectures, the assumption is made that it is simpler and more effective to reduce MTTR than it is to increase MTTF.

Other than failure modes, this paper does not provide a real list of all the NFPs that a service should expose and that does not treat fully the Problem 1.

This paper does not discuss the problem of aggregating the NFPs for a composite service, and thus does not treat Problem 2.

3.3.16 Milanovic's Availability Modeling

Milanovic *et al.* [52] focus on modeling business process availability. They map dependencies between ICT components, services and business processes (processes are expressed in terms of communication paths, reliability of the services, the routers and the switches).

In this thesis, the focus is on solving the problem of composing the availability of the services. It is assumed that the services are opaque (i.e., owned by other services providers) and thus one cannot know how they are implemented or what technology components are used, as is the case when the SOA approach is used. Thus, it is impossible to translate this work to the availability of the ICT components that they depend on.

While this paper addresses Problems 1, 2 and 3, it does not do it in a manner that is helpful in progressing third-party, SOA-based service development, as it uses a model where services are only shared internally.

3.3.17 Dobson's NFRs Domain-Independent Ontology

Dobson *et al.* recommend in [21] the use of domain-independent ontologies (i.e., define concepts and not taxonomies) to represent, organize and reason about NFPs in SOA, hence unifying the concepts and terms used throughout the service lifecycle. An ontology is an explicit formal specification of how to represent the objects, concepts, and other entities that exist in some area of interest, together with the relationships that hold among them. The authors propose a "NonFunctionalProperty" ontology that consists of two hierarchies – one rooted at the class "QoSAttribute" and one rooted at the class "QoSMetric". To support measurable non-functional requirements, these two hierarchies are joined together by an OWL "ObjectProperty" named "hasMetric". They also differentiate "QoSAttribute" from other non-functional properties that are not considered as QoS (e.g., cost).

To increase the flexibility of comparing the NFPs of different services, this ontology defines explicit semantics that allow the automatic conversion between compatible units and metrics. For example, service availability can be stated as the Mean Time Between Failures (MTBF) and Mean Time to Recover (MTTR). By defining a rule in the ontology that defines availability to be MTBF / (MTTR+MTBF), the discovery component can still correctly compare queries stated in terms of availability. Rules might also be useful to state service consumer's NFP preferences, including levels of trust in providers.

Dobson *et al.* have demonstrated, using the "SeCSE" QoS facet (i.e., allowing the assignment of multiple classifications), that the use of such ontology in the specification and discovery of services provides a greater degree of machine understanding than the current NFP languages. In an example, using a faceted specification, the service discovery techniques sifted through 100 similar specifications and produced a short list of four services.

Although this paper studies the enhancing of service discovery and comparison, it does so without providing any formal specific set of NFPs, and thus does not address Problem 1. This paper does not study the aggregation of the NFPs, and thus does not treat Problem 2. Finally, this work touches on Problem 3 but without providing any specific solution.

3.3.18 Tosic's Requirements for Ontologies in Management of WS

Tosic *et al.* [76] highlight the need for ontologies that define and represent formally QoS metrics (a.k.a. NFP metrics). These ontologies are needed to specify QoS constraints for languages used in the management of web services and their dynamic compositions (e.g., WSOL [77]). This paper depicts requirements for such ontologies, including having QoS metrics, measurement units, currency units, measurement methods, measured properties, and dependencies and relationships between QoS metrics. This paper reports on the authors' study of existing ontologies, including DAML Ontology Library, Cyc Upper Ontology [18] and Simple HTML Ontology Extensions (SHOE) [68], and demonstrates by their review that a satisfying NFP catalogue does not exist.

Tosic *et al.* state that there will be a need for independent, third-party Web Services ontological translations between different QoS metrics, once an appropriate set of ontologies is defined.

This paper does not resolve any of the three problems of interest directly but it emphasizes the need for an NFP catalogue as part of the larger WS offering solution.

3.3.19 Hughes' QoS Explorer

Hughes *et al.* developed QoS Explorer, an interactive tool that predicts the quality of service of a workflow from the QoS characteristics of its constituents [33]. Interestingly, this tool supports the processing of entire statistical distributions and probabilistic states to model NFPs that could be varying, such as completion time. Composition operators include sequence, concurrency, conditional branching, and a simple version of the discriminator, but loops are not covered. QoS Explorer targets optimization at design time and does not really address dynamic service selection.

The authors just mention the QoS (a.k.a. NFPs) that they think are important, and this list includes time to completion, availability, accuracy, peak bandwidth usage, and cost. However, this paper does not formally define a list of NFPs that the service should expose per se, and thus they do not treat Problem 1.

The authors simply mention that they use an "Agrajag" function that they have developed but do not specifically address the problem of aggregating the QoS, and thus they do not treat Problem 2.

The NFP-aware service selection used in this paper assumes that the options are limited to either the premium service or the economy service, and thus the paper has not treated Problem 3.

3.3.20 Dobson's QoS Ontology

Dobson *et al.* propose an ontology (QoSOnt) for Quality of Service [23]. The aim of this ontology is to facilitate interoperability between service providers, to achieve machine reasoning and to assist intercommunication about QoS between service participants. To demonstrate the use of the ontology, a tool called Service QoS Requirements Matcher (SQRM) was been developed. This tool supports service discovery, differentiation and selection based upon QoS requirements. In SQRM, a QoS requirement is represented as a predicate (in XML). In contrast to requirements, the provider's description of the offered service QoS capabilities consists of asserted propositions. Requirement predicates are visualized as a tree, the leaves of which are values or classes of metrics expressed in QoSOnt. The inner nodes are logical and arithmetic operators. The requirements matching approach is a bottom-up evaluation process, which starts from the leaves. That is, for each service being considered, the asserted values of metrics can first be established from the published capabilities.

The authors addressed neither Problem 1 nor Problem 2 in this paper. In addition, the service discovery and differentiation solution is tied to a WS technology (i.e., UDDI), no specific set of required NFPs is defined, and no NFP aggregation functions were developed for composite services. Hence, this paper mainly addresses criteria 3 of Problem 3.

3.3.21 Determining QoS of WS-BPEL Compositions

Mukherjee *et al.* propose in [54] a recursive algorithm for QoS computation for WS-BPEL compositions. A BPEL workflow can be represented as an activity diagram where

activities, scopes and handlers are represented by nodes. The user should provide the reliability, response time and cost of all the involved WS in the composite service; the probabilities of success selecting branches or events in "if" and "pick" activities; the "average number of iterations" in loops and for each "catch" or "catchall" block; and the "average waiting time" for activities, messages and events.

The limitations of this model are as follows: (i) it does not support the detection of mutual exclusiveness of links at a join; (ii) it does not handle isolated scopes that provide control over concurrent access of shared resources; and (iii) forced termination behaviour is not captured by the model and it does not deal with termination handlers.

The quality of QoS is as good as the input provided by the users with respect to the probabilities of execution of different web services, conditions, loops, etc. As well, the termination condition of this QoS computation algorithm is conditional on having a well-formed WS-BPEL process. The QoS of the process does not evolve after a certain number of executions (oblivious mechanism).

The authors focus on three QoS (response time, cost and reliability) without defining a list of other desirable QoS, and thus have addressed only partially Problem 1. This paper does not address Problem 2 and Problem 3.

3.3.22 Weiss's Classification of WS Feature Interactions

Using a fictitious e-commerce case study, Weiss *et al.* [81] demonstrate that independently developed web services, once composed, may interact with each other in unexpected and undesirable ways. The problem is called feature interactions. This problem was first encountered in the telecommunications domain; however, many of the telecom solutions cannot be applied directly to the SOA context due to its unique characteristics, including the lack of central authorities. Weiss *et al.* provide a classification of possible feature interactions to keep in mind while compositing new services.

In this paper, a web service is modeled as one or multiple features (a feature could be either a non-functional property or a functionality). The authors use URN to model a service. The intentional (why) aspects of the service are modeled using GRL. The behavioural (what and when) aspects of the service are modeled as scenarios using UCM. Finite State Processes (FSP) are used to depict the processes and enable the validation anal-

ysis of service interactions and the discovery of feature interactions. These feature interactions can impact the functional behaviour of the composed service as well as its non-functional properties.

Here is an example of feature interaction that impacts the service NFP: when the service is composed to ensure "convenience" for the user via single sign-on, it may be used in a way that violates the user's "privacy". These two NFPs (convenience and privacy) are both important to the user.

One can argue that, ideally, to compute the NFPs of a composite service based on the NFPs of its underlying services, one should start by a validation of the service composition (e.g., to endorse the soundness of the composition — the non-existence of feature interactions in the composition — or to discover the features interactions and either address them or take them into consideration in the composite NFPs computation). In this thesis, the feature interactions problem is treated separately from NFP aggregation. Hence, it is assumed that the composite service is always a sound service. One should be able to count on the service provider to bear in mind the scenarios depicted by Weiss *et al.* [81] and validate the soundness of composite services. Whenever this first assumption is wrong (e.g., the composite service is always a sound service), there is a second assumption that the situation will be self healing and eventually the first assumption will become true as a function of time. This second assumption is based on the view that when there are feature interaction problems, they will be noticed by the service consumers and these consumers will express their opinion by rating very badly the reputation of the invoked service. As a result, the service provider will fix the feature interaction issues. Reputation gives us insights on previous usage scenario satisfaction, including feature interactions problems.

This work does not address any of the three problems of interest directly, but it does give a heads-up on the need to validate the soundness of composite services before performing NFP aggregation and NFP-based service selection.

3.3.23 Lall's Second generation of WS

A contribution from Lall *et al.* [46] evaluates the second generation WS specifications against a set of NFRs. NFRs can be attached (1) to services, (2) to a service interface, and

(3) to a messaging framework. The authors propose three ways to categorize the service NFRs in particular, including (1) generic versus specific, (2) consumers' versus providers' perspectives, and (3) deterministic versus non-deterministic NFRs. The authors define different NFRs for the service interface and the messaging framework. They focus on defining only the non-deterministic NFRs of services. They do not provide data structures for any of the proposed NFRs. Finally, they evaluate and map the large set of defined NFRs against the existing WS stack standards including WS-Transaction, WS-Coordination, Business Process Execution Language, WS-Security Framework (e.g., WS-Trust, WS-Privacy, WS-SecureConversation, WS-Federation, WS-Authorization), WS-ReliableMessaging, WS-Policy Framework, WS-Attachements, WS-Adressing WS-MetadatExchange, and WS-Notification Framework. This work does not address any of the three problems that were identified.

3.3.24 Summary of Evaluation

While the framework for managing the non-functional requirements of conventionally developed applications is well established, the corresponding framework in the context of SOA services is still at an early stage of development. There is still no complete standardized solution specifying what service providers should expose as NFP information in their service descriptions. Parts of the current limitations of the existing solutions include the following:

- Simplified NFP models (e.g., do not support different pricing models and different currencies; very limited NFPs);
- Acknowledgement of need for NFP catalogues, but without delivering it;
- No clear focus of the perspective of the service consumers and/or other stakeholders;
- No methodology for validating catalogues of NFPs;
- No end-to-end solution proposing NFP data structures and aggregation functions;
- Aggregation functions do not take into account the required normalization for NFPs of WS from different service providers;
- Dependencies among NFPs are ignored for NFP-based service selection; and

- Context is not taken into account.

Few approaches addressing NFPs for SOA services, as described in the above sections and as summarized in —Table 1, are proposed for service selection and composition. Generally, these approaches either focus on defining an exhaustive, informal set of non-functional properties for atomic services or focus on the computation of very few properties for composite services. In the related works, several assumptions were made to reduce the complexity of the problem, such as assuming:

- Only limited sets of NFPs for NFP aggregation;
- Only simple composition operators (i.e., no loops and no recursion);
- Simple treatment of the composition operators (i.e., conditional and parallel branches that are ignored); and
- NFPs do not require any normalization. For instance, the mode of payment for the services is the same regardless the service providers (i.e., all the services involved in the composition as based on pay per invocation).

The legend for Table 1 is as follows:

NFP Formalism: + Good; ± Fair; - Poor
NFP Aggregation:
(1) Takes into consideration the different composition operators including conditional branching and parallel branching;
(2) Has a simplistic treatment of composition operators;
(3) Neglects composition operators altogether.
NFP-Aware Service Selection:
(1) Based on *one* NFP as the user's evaluation criterion,
(2) Based on the user's rating of the different criteria,
(3) Based on the best trade-off between *multiple* NFPs,
(4) Investigates the possibility of using multiple services or multiple instances of the same service.
Perspective: Consumer's perspective, Provider's perspective, Multiperspective.
NFP Validation: Standardization; Other
Note. No entry in a cell means that the criterion is not applicable to the reference.

Table 1 Evaluation of Reviewed Solutions

Reference	Formalism	Aggregation	Service Selection	Perspective	NFP Validation	Comments
NFPs versus SLA [77], [50], [72] and [47].	±					SLA is a custom-made, negotiated contract between a service provider and service consumer. A service description is developed without any involvement of service consumer.
NFP Description versus Formal Specification of Constraints [77], [32], [16], [20] and [79].	±				S	These languages formalize various constraints (a.k.a., policies) for WS, including those related to NFPs. To complement these efforts, NFP ontologies are needed.
ISO/IEC 9126 [38]	-				S	An international standard for evaluating the quality of conventionally developed applications and not suitable for SOA.
OASIS Quality Model for WS [44]	±			M	S	The work is still incomplete and the attributes are defined at the conceptual level.
Balfagih's multi-stakeholders' perspective [7]	±			M		Classification of QoS in SOA from the perspective of the developer, provider, and consumer. Authors provide definition for each QoS and metrics only for few of them.
Choi's QoS Metrics for Service Provider [17]	+					Authors derived six quality attributes (NFPs) based on the unique features of SOA. There is no clear perspective and no validation of the list of the NFPs.
Galster's Taxonomy for NFPs [27]	-					Attempts to generate a checklist for NFPs but not formalized and does not address their aggregation. There is no clear perspective and no validation of the list of the NFPs.
O'Sullivan's What is in a service? [26], [67] and [66]	+					A large technical report that contains 79 models to describe NFPs utilizing lessons learned from conventional services description. There is no clear perspective and no validation of the list of the NFPs.
Tosic's WS Offerings Language [77]	-					WSOL does not define QoS constraints to be monitored but uses external ontologies of NFPs.
Chen's QoS Driven WS Composition [15]	+	2		C		Consumer's perspective focused. Seven NFPs: response time, availability, concurrency, expire time, price, fine, and security level. There is no validation of the list of the NFPs.
Zeng's QoS Middleware for WS Composition [89]	+	2	2			NFPs: execution price, completion time, reputation, successful execution rate and availability. There is no clear perspective and no validation of the list of the NFPs. Simplistic treatment of parallel and conditional branches in the NFP aggregation approach.
Liu's QoS Service Selection [49]	+		1			QoS information can be provided by providers, computed based on execution monitoring by the users, or collected via requesters' feedback, depending on the characteristics of each QoS criterion.

Reference	Formalism	Aggregation	Service Selection	Perspective	NFP Validation	Comments
Suzuki's SLA Genetic Optimization Framework [78]	-	2	2			An optimization framework E^3 that provides a multi-objective genetic algorithm, E^3-MOGA, to solve the QoS-aware service composition.
Hobbs' Crash Only Services [30]	-					Crash-only service is proposed as a starting point for building taxonomy of a failure model.
Milanovic's Availability Modeling [52]	-					Map dependencies between ICT components, services and BP availability.
Dobson's NFRs Domain-Independent Ontology [21]	-					Use domain-independent ontologies to represent and reason about NFPs in SOA. Focuses on measurable NFPs. Cost is not considered as an NFP. Proposes an ontology that consists of "QoSAttribute", "hasMetric" and "QoSMetric".
Tosic's Requirements for Ontologies in Management of WS [76], [18] and [68]	-					Highlights the need for QoS ontologies and their requirements: measurement units, currency units, measurement methods, measured properties, and dependencies and relationships between QoS metrics.
Hughes' QoS Explorer [33]	-		1			An interactive tool developed to predict QoS of a workflow from the QoS characteristics of its constituents. It supports the processing of entire statistical distributions and probabilistic states but at design time only.
Dobson's QoS Ontology [23]	-		3			An ontology and a tool called SQRM are developed to facilitate intercommunication about QoS between service participants and to achieve machine reasoning. SQRM supports service discovery and selection based upon QoS requirement.
Determining QoS of WS-BPEL Compositions [54] (Mukherjee)	-	2				A recursive algorithm for QoS computation for WS-BPEL compositions. The user should provide the reliability, response time and cost of all the involved WS in the composite service; the probabilities of success selecting branches or events in "if" and "pick" activities; the "average number of iterations" in loops and for each "catch" or "catchall" block; and "average waiting time" for activities, messages and events.
Weiss's Classification of WS Feature Interactions [81]	-					Independently developed WS, once composed, may interact with each other in unexpected and undesirable ways.
Lall's Second generation of WS [46]	-			M		Definition of non-deterministic NFRs of services and their evaluation against the second generation of WS specifications. Interesting but not totally relevant to this thesis: the definitions of NFRs that apply to service interface, and to messaging framework.

3.4. Chapter Summary

In SOA, NFPs play an important role in service discovery, service selection, SLA establishment and service monitoring. Very little work has been done to address how to handle the NFPs of services from the perspective of the service consumer. Notably, according to the survey, none of the existing work was dedicated to the definition of a domain-independent catalogue of NFPs to accommodate the perspective of the service consumer with the degree of accuracy and formalism required for SOA-based services. Few generic lists of various NFPs that the service should expose were presented in the reviewed literature. However, these lists provide only informal information on the quantifiability of these NFPs. Current NFP specifications are lacking the possibility to add properties and measurements units (i.e., currency units). NFPs cannot, at least at this time, be evaluated and compared via an automated process. The related works do not provide any operational representations or metrics to evaluate these NFPs. As well, some research effort is dedicated to the NFP constraint specifications and they differ only in the expressiveness of those syntactic constraints.

Currently, most of the proposed solutions for aggregating NFPs of composite services address only a few generic criteria such as price, completion time, availability and reliability. In mission-critical applications, these criteria are definitely not sufficient. Hence, there is a need to build on existing approaches, including those from traditional development processes, with a focus on the unique features of SOA (i.e., highly distributed applications, loosely coupled services, and new interactions models between providers and consumers) and develop a domain-independent catalogue of NFPs.

In the next chapter, a catalogue of relevant NFPs targeting the enhancement of the service description from the perspective of the service consumer will be introduced. A superset of NFPs that were proposed in the literature will be studied and then filtered so only those that tackle the concerns of service consumers in the context of SOA and independently from SOA-enabling technology will be retained. An external survey will be conducted to validate the proposed NFP catalogue.

Chapter 4. Domain-Independent Catalogue of Non-Functional Properties

The objective of this chapter is to define a catalogue of domain-independent NFPs that are important for services comparison and selection from the perspective of service consumers. First, the relevance of NFPs for SOA-based development will be discussed, followed by the process for validating the proposed selection of NFPs that should be included in service descriptions. For each selected NFP, a formal definition and a way to quantify it will be provided. The catalogue developed here is intended to be a general NFP catalogue independent of particular SOA application domains. However, this catalogue can be extended with domain-specific NFPs when needed. This chapter concludes with the definition of a metamodel for this extensible domain-independent catalogue of NFPs.

4.1. Relevance of Non-Functional Properties to SOA

Developing applications that satisfy their functional as well as non-functional requirements is vital for their success in any domain. Non-functional requirements are of critical importance and, at times, functional requirements might be sacrificed to meet them [27]. In SOA, non-functional properties of selected services have a direct impact on the quality attributes of the resulting service-based application and the realization of its non-functional requirements.

In traditional software engineering, stakeholders play an important role in expressing explicitly or implicitly the non-functional requirements of their application, and then verifying and validating the implementation of these requirements at various stages of the application's development lifecycle. However, in SOA, service consumers are often not involved in any stage of the development phases of services. Services are used as black boxes (i.e., services are opaque), which means that service consumers do not have

access to their underlying logic or implementation. Hence, service consumers cannot assess a service's behavioural and quality aspects until they actually invoke it. As well, in contrast with traditional application development processes, these NFPs are not applied to and assessed against an application as a whole but against multiple independent distributed services; this brings up additional challenges.

As stated before, NFPs play an important role in each stage of the SOA process lifecycle such as in the discovery, the selection, and the substitution of services. In service-based applications, NFPs of services can drive the design of the composed applications (i.e., composite services) and the selection of the concrete services (e.g., could be atomic or composite services, in opposition to abstract services) from which they are built. The accessibility to multiple services that can deliver the same functionality reinforces the need for NFPs, since NFPs may be the only differentiator between such services. The ability of the service consumer to choose services based on NFPs stated in the service description is crucial, along with the means to evaluate and compare the NFPs of available services. Means to compute the aggregated NFPs of composite services are needed. When a service-based application is being developed and when more than one service can provide the same functionality, the developer should then take into consideration the NFPs of the various functionally equivalent services. The developer should also be able to predict the NFPs of the resulting composite service.

Using services developed externally is of particular interest to service providers in that this allows them to offer new functionalities for which they do not necessarily have the resources or the required development expertise. On the other hand, the ability to predict the behaviour of the resulting composite service without resorting to tight coupling to the underlying services can be difficult and requires the generation of well-defined service descriptions. Clearly defined and well-managed NFPs benefit both service providers and service consumers. Service providers who handle the NFPs of their services well will contribute positively to an increased use of their offered services and, consequently, to a better reputation and larger financial gain. For service consumers, relying on services with well-handled NFPs will impact directly the quality and hence the success of their applications and systems.

Service providers should supply a detailed service description for each service, where they promise the service consumers a certain service quality. The supplied service description has to address both the functional and also the non-functional aspects of the service.

The aim of a domain-independent catalogue is to define a list of various generic (i.e., common to all services) non-functional properties to be considered when service descriptions are developed to accommodate the service consumers. The focus of this effort is to define relevant NFPs that need to be added in the SOA context to the descriptions of atomic services to address the service consumers' needs (i.e., what NFP information should be included, by the service providers, in the service description to help service consumers decide whether a given service suits their needs).

Business rules can drive choosing the relevant NFPs to be considered out of the ones proposed below. Additional domain-specific properties might be needed; therefore this catalogue is designed to be extensible. The focus is on what services should expose as part of their descriptions to guarantee certain NFP levels for the service-based applications. The NFPs at the network infrastructure level is treated as a separate issue; hence it is out of scope of this thesis.

4.2. Survey Process

To extract and validate a useful catalogue of Non-Functional Properties, candidate NFPs (with their definitions) were first identified from the literature, and then proposed in an online survey for validation and comments. The results were analyzed in order to refine the catalogue, and a second survey was conducted to validate the revised version, and make further (and smaller) modifications. This process is detailed in the following subsections.

4.2.1 Overview

Ideally, appropriate standardization bodies would develop a catalogue of NFPs to facilitate interoperability. During the development of this catalogue, it was decided that common standardization practices would be followed as much as possible by involving dif-

ferent academic and industrial players from different geographical regions and different business domains in the survey process.

An initial set of NFPs was extracted from the literature and from work in progress in standardization bodies such as OASIS [61], W3C [85] and ITU-T [35] (see section 3.3). This set was believed to be sufficient as a starting point for the NFP catalogue. Then each NFP was assessed as to whether or not it was relevant and necessary for SOA services description, selection and composition. The NFPs were filtered manually and only those considered relevant from the consumer's perspective were retained.

A first survey [10] entitled "NFPs of services in SOA – Service Consumer's Perspective" was conducted as a research study. The purpose of this survey was to discover requirements for a framework to better describe NFPs of services in SOA purely from the point of view of consumers [11]. The first online survey proposed a catalogue of 19 NFPs to be included in service descriptions to enable advanced applications such as NFP-aware service selection. These 19 initial NFPs were Cost, Completion Time, Trust, Availability, Reliability, Usability, Validity (Accuracy and Precision), Standards Compliance, Dependencies, Failure Mode, Security, Execution Models, Jurisdiction, LifeCycle Updates, Penalty Rate, Compensation Rate, Resource Utilization, Throughput, and Accessibility.

Participants were invited to criticize and enhance the proposed descriptions as well as to assess their relevance from the consumer's perspective in the SOA context. The participants were from industry, academia, and standardization bodies. The results were analyzed and the survey results used to compile an improved catalogue. This process was repeated using the updated catalogue and the survey participants that had expressed their will to collaborate further. Subsequently, the new comments were analyzed and the catalogue was updated again in response to the received comments. At this point the study was concluded and a final set of NFPs and definitions was produced; these are presented in the following sections of this chapter.

Only a basic knowledge of SOA principles was required to understand the possible needs of service consumers. Therefore, participants did not need experience with any specific SOA implementation technology.

The input of the participants was used to evaluate and evolve the proposed catalogue of service NFPs. The outcome of the questionnaire was collected and used anonymously in this research.

The survey questionnaire and summary of answers are available online at [10]. The survey consisted of four major parts. Part one explains the context and the goals of the survey and captures consent information from the participants. The second part targets the characterization of the participants via six demographic questions. The third part proposes 19 NFPs and their meanings and provides text boxes that invite the participants to review, comment, criticize and propose text enhancements to the NFPs. The fourth part targets the prioritization of the proposed NFPs into 4 priority levels (Absolutely essential, Might be needed, Not sure, or Definitely not needed) and collects suggestions about other NFPs that should also be considered.

The invitation to participate was sent mainly to the mailing lists of important standards development organizations including the Object Management Group (OMG), the Organization for the Advancement of Structured Information Standards (OASIS), and the International Telecommunication Union – Telecommunication Standardization Sector (ITU-T), as well as to a number of professors of the University of Ottawa and Carleton University (Canada) known to work on SOA. The survey and the answers are all in English.

Twenty-nine (29) respondents completely answered the survey [10]. 53 people started it but many did not answer the questions beyond the demographic ones, so only those who answered all questions were counted. The outcome shows that this survey is important and timely for industry. The tables in the next section present a visual summary of the participants' demographic profiles.

4.2.2 Demographics Questionnaire

Table 2 to Table 7 present the results of the demographic data related to the first survey, which was completed by 29 participants.

Table 2 Status of Survey Participants

Question 1- Are you from academia, government, or industry?			
Domain	Industry	Academy	Government
	20	5	4

Table 3 Experience of Survey Participants

Question 2- How many years of software development/teaching experience do you have?				
Less than 2 years	2 to 5 years	6 to 10 years	11 to 20 years	More than 20 years
2	2	3	9	13

Table 4 Expertise of Survey Participants

Question 3- What is the level of your expertise in SOA?		
Beginner	Intermediate	Experts
5	14	10

Table 5 Focus of Participants' Organizations

Question 4- What is the primary focus of your organization?	
Consulting	7
Telecom	6
Education	5
Other	4
Security	2
Software Eng.	2
Government	2
Standardization	1

Table 6 SOA-Related Maturity of Participants' Organizations

Question 5- What stage is your organization currently in with respect to SOA?	
Implementing	10
Using	9
Investigating	6
None of the above	3
Planning	1

Table 7 SOA-Related Interest of Participants' Organizations

Question 6- How long has your organization been caring about SOA?	
Less than 2 years	2
Between 2 and 5 years	16
More than 5 years	7
Not interested yet	4

Keeping in mind that the participation in this survey was totally voluntary and unpaid (e.g., done for free), these tables show that a decent level of participation was achieved, with participants from diverse fields, and considering multiple dimensions. The majority of the participants are from the industry (viz., 20 participants out of 29) working in more than 8 different sectors. In addition, more than 2/3 of the participants have over 10 years of experience and almost half of the participants even have more than 20 years of experience. A third of the participants consider themselves experts. The majority of the participants are implementing or using SOA and are facing and dealing with SOA real-life problems. Only 4 participants are involved with organizations that are not yet interested in SOA. This data set is believed to involve an appropriate, decent, and skillful set of participants that provides confidence to the validation results.

4.2.3 First Survey Results and Key Findings

The participants were asked to review and prioritize the set of 19 initial NFPs (the relevancy of the NFP, the label, and the corresponding definition). In addition, they were asked to propose new NFPs when this was deemed essential. All of the comments received were systematically taken into account, and two people (this thesis' author and her supervisor) decided together whether to incorporate or reject them. All the individual

comments and resolutions are documented online [10]. The analysis of the many comments received through this first survey revealed the opportunity to rename many of the NFPs initially proposed, to remove NFPs, and to add new ones. All the proposed definitions needed to be reworked and refined to address the main concerns raised by the participants. A summary of the main comments follows:

Separation of responsibilities: service providers will not commit to things out of their control. The participants are clear about the need to separate the responsibilities of the service providers, service consumers, and network providers. Hence, it is deemed necessary to have NFPs that depict the service requirements from the consumers' side (e.g., Resources Requirements). As well, there is a need to separate the network latency from the providers' promised response time, and in a similar fashion the network availability and reliability have to be separated from the service availability and reliability.

Concise definitions: The participants have expressed the need to keep definitions concise and separate from any supportive text. As a result, a *note* is used to provide supportive text, including information about possible metrics and semantics for the NFPs.

NFP names: Participants are particularly meticulous about NFP names. As a result multiple NFPs are renamed to address the concerns raised and prevent possible ambiguities. For example, in the final list of NFPs, Cost becomes Price, Completion time becomes Response Time, Trust becomes Reputation, Validity becomes Accuracy, Life-Cycle Updates becomes Service Versioning, Resource Utilization becomes Resource Requirements, and Throughput becomes Scalability.

New NFPs: The difference between what the service providers claim to offer and what could be certified by other organizations was not stressed. It was concluded that there is a need for a new NFP, Certification, to be added. There is also a need to add two other NFPs: Transactional Service, and Server Location. The Transactional Service NFP is required when services involve a sequence of independent updates that must be committed as a logical unit. Services may need to participate in transactional two-phase commits and a transactional management function may need to be spread across invoked services for composite services. Server Location is required for some participants who do not want their data to enter a given country.

NFP Prioritization and Deleted NFPs: According to the prioritization results (expressed in terms of percentage of participants for the categories *Absolutely essential / Might be needed / Not sure / Definitely not needed*), four NPFs received overwhelming support: availability (80/20/0/0), security (83/13/3/0), completion time (73/23/3/0), and reliability (83/10/7/0). Although none of the NFPs received more than two "Definitely not needed" verdicts, five NFPs received much hesitant support, both in terms of prioritization ranking and of textual comments. Accordingly, the following five NFPs are deleted from the initial list:

- *Dependencies* contain the list of other required services on which the execution of a given service depends, if any. This applies to a composite service that is built from other services. It is deleted since it is not applicable to atomic services, but it may be considered for composite services in the future.
- *Execution Model*: the service should expose whether it supports a publish/subscribe model (i.e., event-driven SOA) or uses a request/reply model (i.e., conventional SOA). This is deleted since this property is considered as a functional one.
- *Penalty Rate* is the fee that a service consumer has to pay to a service provider whenever he breaches the contract. It is deleted because it should be part of a SLA. Such rate should be negotiated and agreed on after having a clear identification of the causes of the problem at the service level.
- *Compensation Rate* is the fee that a service provider has to pay to a service consumer whenever the service provider was not up to the promise defined in the service agreement. It is removed for the same reason identified for Penalty Rate.
- Accessibility is the possibility of providing access to information to people with disabilities and people with special needs. Accessibility is argued to be a user interface property rather than a SOA property.

Domain independence: The objective pursued here is to define a domain-independent catalogue of NFPs. Having participants from different fields helped tailor the definitions to make them general enough to cover many domains while remaining small enough to avoid too many domain-specific particularities.

4.2.4 Second Survey and Final Catalogue

After having collected the comments of the participants and having addressed them, a revised version of the NFP catalogue was produced and sent by email to a subset of the participants (i.e., eight people who provided an email address to indicate their interest in further collaborating to this validation exercise). Based on this second round of feedback, additional small changes were incorporated to produce the final list of seventeen consumer-oriented NFPs for atomic SOA services: (1) price, (2) response time, (3) reputation, (4) certification, (5) availability, (6) reliability, (7) usability, (8) accuracy, (9) standards compliance, (10) failure modes, (11) transactional service, (12) security, (13) jurisdiction, (14) service versioning, (15) resource requirements, (16) scalability, and (17) server location. These seventeen NFPs will be detailed later in this chapter.

4.3. NFPs Irrelevant from a Consumer Perspective or Already Covered

Even though the following NFPs are important and should be considered by the service providers while developing their services, they are excluded from the catalogue of NFPs that should be exposed as part of the service description to be used by the service consumers.

- **Maintainability** (analyzability, changeability, stability and testability): service consumers invoke the services as they are, without having access to their implementation details (i.e., SOA services are opaque). Hence they are not supposed to do any maintenance or to be even informed about this matter. The expectation is that the service should produce the desired real-world effect once invoked. Consequently, this criterion is not really relevant to consumers and thus should not be exposed in the service description, even though it is an important NFP from the perspective of the service provider.
- **Interoperability**: SOA and its enabling technologies bring the promise of seamless and agile integration and interoperability, as well as cost saving due to the reuse capabilities. Whether interoperability means integrating legacy systems or enabling new

business models with partners and customers, using standards is a must for SOA-based applications and it is the key for interoperability. This NFP is covered to some extent by the Standards Compliance NFP (see subsection 4.4.9) and hence does not really need to be exposed on its own.

- **Portability** (adaptability, instability, and co-existence): In SOA, portability has become less of an issue than for standard software applications. If proper architectural designs are in place, platform-specific applications can be transformed into less technology-dependent and more accommodating to interoperable business needs. Even platform-dependent enterprise applications can communicate through asynchronous messages and SOAP-based web services. It may be said that this large enterprise application exhibits high quality regarding portability, because it has achieved internal platform-independence through SOA. In addition, similar to interoperability, the use of standards to enable the reuse of services is at the heart of SOA principle. This NFP is implicitly covered by the standards compliance NFP (see subsection 4.4.9) and hence does not need to be exposed explicitly either.
- **Suitability**: the service consumer may end-up using a service in a very creative manner, such that even the service provider never thought that its service would be used in that given workflow and context. Hence, this NFP is not really relevant for exposition in the service description.
- **Business Value quality** (service after effect and service brand value): for the same reasons mentioned for the Suitability NFP, this criterion is not really relevant and thus should not be exposed in the service description.
- **Fault Tolerance** and **Recoverability**: these two aspects are covered by the Failure Mode NFP defined in subsection 4.4.10. Hence, there is no need for a separate entry for this NFP.
- **Reliable Messaging** and **Collaboration**: these criteria do not apply to a single service but to a workflow as a whole. Hence, these criteria are not really relevant NFP to be part of the service description.
- **Controllability**: service consumers are not supposed to be controlling the service that they are invoking. The only case where minimum controllability is needed is when something goes wrong with the service. In this case, the failure mode NFP will opti-

mistically dictate what the service consumer should do (see subsection 4.4.10). Hence, this NFP should not be exposed further in the service description.

- **Dynamic Discoverability and Composability**: only the services that can be dynamically discovered and composed are considered in the context of this thesis. Hence, these NFPs are out of scope and will not be added to the catalogue.
- **Safety**: this NFP can be measured by the set of standards to which the service complies (see subsection 4.4.9). Hence, this criterion is already covered in the catalogue.
- **Provenance**: indicates the origin and the history of the service. "Provenance is the documentation about such data generated by a process, providing explanations about who, how, and what resources were used in the process along with the processing steps that occurred to produce that data." [70]. However, as this is getting closer to data content description than to service description, provenance becomes very complex to expose, to measure and to verify from the point of view of the service consumers. As well, provenance composition can be very complex. Thus, it is assumed that the service providers should provide this NFP upon request by the service consumers. Hence, this criterion is not required in the service description.

4.4. Consented NFPs for Atomic Services

This section contains the relevant NFPs resulting from the survey consensus: Price, Response Time, Reputation, Certification, Availability, Reliability, Usability, Accuracy, Standards Compliance, Failure Modes, Transactional Service, Security, Jurisdiction, Service Versioning, Resource Requirements, Scalability, and Server Location. Each of these NFP is presented in its own subsection with a definition, additional notes, a data structure (including a UML-based formalization), and a brief discussion of verifiability. For each of these NFPs, the service provider should indicate its measurement sources (e.g., measurements are collected from a real system or a simulation experiment, or based on theoretical functions such as probabilities). Note that the order in which these NFPs are presented in this section is arbitrary.

In the textual description of the data structures defined below, the term *"real"* is used as one of many data types. The corresponding data type in UML however is *"float"*.

4.4.1 Price

Price is the fee that the service consumer is expected to pay for invoking a given service.

Note. Different pricing models can be supported including: fixed price per invocation, calculated by a function based on the input service request (e.g., when a "back-up storage" service is requested, the price could be calculated based on the total size of the files and on the duration of storage), per subscription (e.g., "all you can use" for a given period), or a mixed model (e.g., having a service invocation fee on top of a subscription fee). Service providers can offer functionally equivalent services using different pricing models. These services are functionally equivalent services which mean they provide the same class of service but they are offered as different services that are to be treated separately.

Relevance: This is an important criterion since organizations and consumers operate according to their financial plans.

Measurable: Two parameters to expose the cost of using a given service: first, how much to pay (and in which currency), and second, which mode of payment (per invocation or for a certain quantity such as a given duration or bytes).

Data Structure: Unit: Currency *enumeration* {AED, AFN, ALL, AMD, ANG, AOA, ARS, AUD, AWG, AZN, BAM, BBD, BDT, BGN, BHD, BIF, BMD, BND, BOB, BRL, BSD, BTN, BWP, BYR, BZD, CAD, CDF, CHF, CLP, CNY, COP, CRC, CUP, CVE, CYP, CZK, DJF, DKK, DOP, DZD, EGP, ERN, ETB, EUR, FJD, FKP, GBP, GEL, GGP, GHS, GIP, GMD, GNF, GTQ, GYD, HKD, HNL, HRK, HTG, HUF, IDR, ILS, IMP, INR, IQD, IRR, ISK, JEP, JMD, JOD, JPY, KES, KGS, KHR, KMF, KPW, KRW, KWD, KYD, KZT, LAK, LBP, LKR, LRD, LSL, LTL, LVL, LYD, MAD, MDL, MGA, MKD, MMK, MNT, MOP, MRO, MTL, MUR, MVR, MWK, MXN, MYR, MZN, NAD, NGN, NIO, NOK, NPR, NZD, OMR, PAB, PEN, PGK, PHP, PKR, PLN, PYG, QAR, RON, RSD, RUB, RWF, SAR, SBD, SCR, SDG, SEK, SGD, SHP, SLL, SOS, SPL, SRD, STD, SVC, SYP, SZL, THB, TJS, TMM, TND, TOP, TRY, TTD, TVD, TWD, TZS, UAH, UGX, USD, UYU, UZS, VEB, VEF, VND, VUV, WST, XAF, XAG, XAU, XCD, XDR, XOF, XPD, XPF, XPT, YER, ZAR, ZMK, ZWD}.

The enumeration of currencies codes, based on the ISO 4217 Currency Code List [37], is implemented in the UML model but due to its size, it was decided not to include it in this thesis document.

Price is the aggregation of different Pricing Elements, as defined in Figure 10. PricingElement has 2 attributes: value: real, and type:Pricing *enumeration* {Invocation, SubscribeMonth, SubscribeDay, SubscribeHour, QuantitySecond, QuantityBytes, Other}. Value is the price per type (i.e., an invocation costs 2 USD):

- If the type is Invocation, then the value is the number of invocations.
- If the type is SubscribeMonth, then the value is the price per month.
- If the type is SubscribeDay, then the value is the price per day.
- If the type is SubscribeHour, then the value is the price per hour.
- If the type is QuantitySecond, then the value is the price per second.
- If the type is QuantityBytes, then the value is the price per byte.

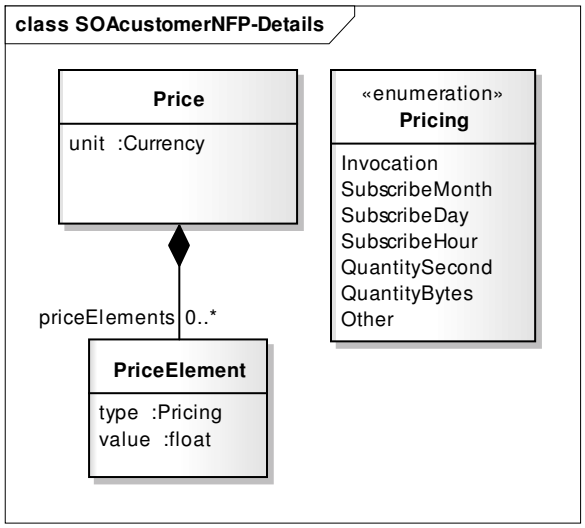


Figure 10 Price NFP

Verifiable: Keep track and compare the payments made and the price of the consumed services.

Note about the aggregation: for Price, services need to be able to normalize the currencies and the payment modes of the underlying services to be able to calculate the aggre-

gation of this NFP, which is to be exposed as part of the composite service characteristics. While this normalization is quite simple to do for currencies (assuming that currencies are converted on a daily basis), this can become very tricky if the providers do not offer the same options for the payment mode.

4.4.2 Response Time

Response Time is the duration of the execution of a given service.

Note. Response time can be expressed as the probability of executing the service during a certain number of time units. Many duration-probability pairs can be used. Service request processing under a certain workload is considered, but not network delays (the latter being out of the control of the service provider). For example, process queries in less than 30 seconds 95% of the time. Response time could be affected as well by the input of the invoked service (e.g., the size of files that the "back-up storage" service is invoked against). If the response time is longer than the acceptable bound, then the service is considered as being unavailable.

Relevance: Time is a common measure of performance. Organizations are enhancing continuously their processes to be more responsive to the needs of their customers. Shorter service execution time is a competitive advantage allowing delivering rapidly to be assigned more value.

Measurable: This can be expressed in terms of time units that are observable by the service consumer excluding probabilistic time units for the network delay (i.e., network delay is not included in the response time).

Data Structure: Since the response time is often a non-deterministic value (e.g., it depends on the load of the system), response time can be expressed in terms of intervals such as [best, worst] or [best, average, worst] cases. Three numeric fields (best, average, and worst) can cover these cases, and their value will be "0" when unused. As well, time units are needed (DurationUnitKind *enumeration*) {microseconds, seconds, minutes, hours} and a confidence level (0% to 100%). This is described in the UML clad diagram of Figure 11. The worst, best and average values are expressed as statistical qualifiers

max, min and mean respectively. In fact, these are not intervals; they are limits on the expected values that can be measured, estimated or calculated.

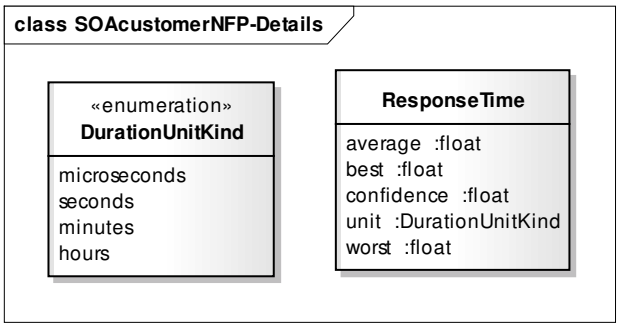


Figure 11 Response Time NFP

Verifiable: Feasible by setting a timer once the service is invoked. The real complexity lies in separating the responsibilities between the service providers and the network providers. In this thesis, it is assumed that the service provider-consumer contract is dependent on the networking service contracts and enforceability is conditional on the network service contracts.

4.4.3 Reputation

Reputation is the opinion of service consumers toward a service.

Note. Reputation can be expressed as the average ranking (feedback) of the service by the consumers of the service. Level 0 to level 5 (i.e., [0..5]) can be used, the latter being the best score (e.g., the highest satisfaction). Reputation is populated by previous service consumers and exposed by the service providers. Security mechanisms can be put in place by the service registry to ensure that only consumers can enter that ranking of the service reputation.

Relevance: Reputation is a subjective perception of service consumers toward a service. Reputation says nothing about how the service was tested or how secure it is. Still, reputation is a key criterion in establishing a functioning relationship between service provid-

ers and service consumers; hence the transactions can take place. Service consumer will not do business unless they understand what they will get for their money, the terms and conditions associated with the service provisioning, and the reputation of the provider. Service providers need to be assured of the consumer's ability and willingness to pay for the service in a timely fashion. The focus of this thesis is on the perspective of the service consumer; hence the reputation is collected only for service providers.

Measurable: This is based on the perception of the service consumers.

Data Structure: reputationLevel: *real* (Figure 12)

Constraints: formalized in OCL as follows:

```
context Reputation
inv: reputationLevel >= 0 and reputationLevel <= 5
```

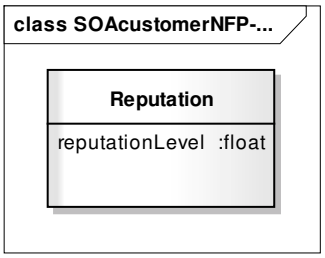


Figure 12 Reputation NFP

Verifiable: A mechanism should be in place to ensure that the service providers cannot tamper with this information. The service registry or a trusted service broker should handle this field of the service description.

4.4.4 Certification

Certification is the confirmation of certain characteristics (i.e., affiliation, signature, specification, policy, standard, law, regulation, etc.) of a tested service by a certification body. Generally certification is provided by some form of external review, audit, or assessment performed by an independent entity.

Note. Certification gives great guarantees of compliance. Certification can be expressed as many Characteristics-Level pairs. For example, Common Criteria certification

(ISO/IEC 15408) specifies to what Evaluation Assurance Level (EAL1 through EAL7) the service has been tested. Certification is ensured and populated by third-party organizations and exposed by the service providers.

Relevance: It is deemed necessary to have independent third party testers for certain aspects of services. For example, under the Common Criteria, vendors must submit their products for a third party to test and grant some level of assurance from level one to level seven (where level 7 is a very high level of assurance).

Measurable: By definition, certification is ensured by an authorized certification body.

Data Structure: Certification is an aggregation of Certificates (certifiedBy*: string,* level*: string*, standard*: string*, version*: string*), as formalized in UML in Figure 13.

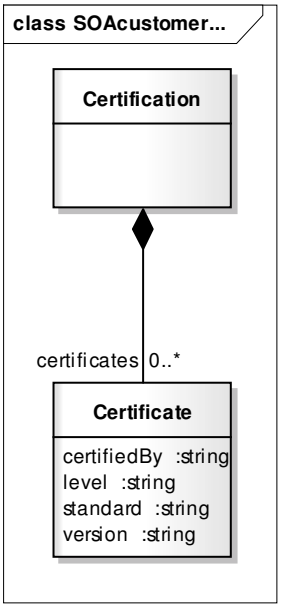


Figure 13 Certification NFP

Verifiable: By definition, certification is ensured by an authorized certification body.

4.4.5 Availability

Availability is readiness for correct service.

Note. Availability can be expressed as the probability that a service is accessible and operational when required *for use at a given moment*. Even if the service is available, if its

response time is greater than the acceptable bound, the service is considered as being unavailable. Availability computations should be based on unscheduled downtime. In an environment with a dynamic infrastructure, there can be variable probabilities of availability. Availability may vary, in a known way, as a function of the time of day, day of the week, day of the month, etc. Hence, probabilities may need to have associated time boundaries. Many percentage-duration pairs can be used. The availability of the network is out of scope.

Relevance: Availability is a well-established criterion in the literature. Downtime per year is a very intuitive way of understanding the availability. Poor availability can cause a loss of business opportunities and slow down productivity. Service providers that publish their predicted availability numbers will help them gain an advantage over their competitors who either do not publish their numbers or who have lower numbers.

Measurable: Availability = 1- (Down Time/Measured Time).

Data Structure: availabilityLevel: *real* (Figure 14)

Constraints: formalized in OCL as follows:

```
context Availability
inv: availabilityLevel > 0 and availabilityLevel <= 1
```

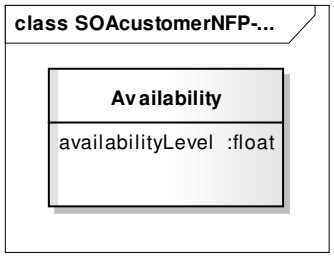


Figure 14 Availability NFP

Verifiable: The service consumer can monitor the service availability from her/his end.

4.4.6 Reliability

Reliability is the continuity of correct service.

Note. Reliability can be expressed as the probability that a service, after being invoked, operates correctly (i.e., without failure or unacceptable degradation of performance) under given conditions *for a given interval of time*. Many percentage-duration pairs can be used. The reliability of the network is out of the scope.

Relevance: Reliability is a well-established criterion in the literature. A service provider's reputation is very closely related to the reliability of their services. Reliability is very important for service consumers, since it is a mandatory requirement for their satisfaction. For service providers, if a service fails to perform its function, this will negatively affect their profits. As well, a high reliability is an important competitive advantage for service providers since their services will require less debugging and/or less maintenance. Service providers that publish their predicted reliability numbers gain an advantage over their competitors who either do not publish their numbers or who have lower numbers.

Measurable: Reliability = Number of Successful Executions / Total Number of Invocations (for a given interval of time).

Data Structure: reliabilityLevel: *real* (Figure 15)

Constraints: formalized in OCL as follows:

```
context Reliability
inv: ReliabilityLevel > 0 and ReliabilityLevel <= 1
```

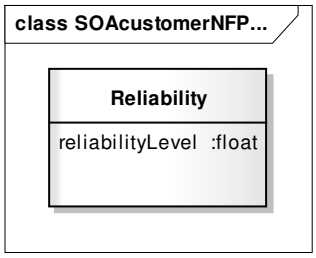


Figure 15 Reliability NFP

4.4.7 Usability

Usability is the capability of the service (when used under specified conditions) to be understood, learned, used, and attractive to the user.

Note. Usability can be calculated as the average ranking of the feedback provided by the service consumers regarding time to become proficient, frequency of resorting to help, and frequency of rework. Usability Level 0 to Usability Level 5 (i.e., [0..5]) can be used, the latter being the best score (e.g., best experience of use). Note that usability may be inapplicable to some services, for instance when a service does not interact directly with a human being. Hence, the usability property must also state whether it applies to the service or not. Usability is populated by previous service consumers and exposed by the service providers.

Relevance: Usability began to obtain recognition back in the 1980s when the importance of usability was acknowledged[1]. If a service is not easy to use, then the service consumer will not feel confident about invoking it, in particular when other functionally-equivalent services are available and easy to access. No matter how good the service will perform its functionality, the service consumer has to feel comfortable about invoking it first. High usability results in increased revenues for service providers and service consumers. For service consumers, usability will improve their productivity, and result in fewer errors. For a service provider, usability can return significant benefits on investment with increased service consumers' attraction and retention.

Measurable: This is based on the perception of the service consumers. A mechanism to ensure that only service consumers can enter this information should be in place.

Data Structure: usabilityLevel, *real* in [0..5] (Figure 16)

Constraints: formalized in OCL as follows:

```
context Usability
inv: UsabilityLevel >= 0 and UsabilityLevel <= 5
```

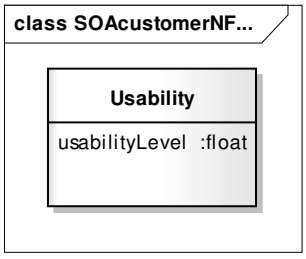


Figure 16 Usability NFP

[1] http://www.pureinterface.com/importance.htm, last accessed November 2008.

Verifiable: A mechanism should be in place to ensure that the service providers cannot tamper with this information. The service registry or a trusted service broker should handle this field of service description.

4.4.8 Accuracy

Accuracy is the degree of proximity of a measured or calculated result to its true result.

Note. Accuracy applies only to services that provide information (or data). Accuracy can be expressed as a percentage.

Relevance: The level of accuracy required for particular applications varies greatly. It is recognized that inaccuracy can "make or break" many types of services. While inaccuracy can make the results of some services almost worthless, excessive accuracy is not only costly but can cause considerable overhead and delays. For example for a GPS location service, the order of magnitude will be meters and the time will be hours. Accuracy could be specified in terms of percentage.

Measurable: Invoking the service using an input for which the output is known and comparing the results.

Data Structure: accuracyLevel: *Real*, unit: *enumeration* UnitKind = {meter, degree, meter2, meter3, liter, hour, kilogram, newton, pascal, bar, joule, watt, degreeCelsius, decibel, dotPERmeter, hertz, coulomb, ampere, volt, ohm, byte, gigabyte, bytePerSecond, other} [1], and other: *string* (to be used when Unit="other", to define any other unit that is not part of predefined ones in the UnitKind enumeration). This is specified in UML in Figure 17.

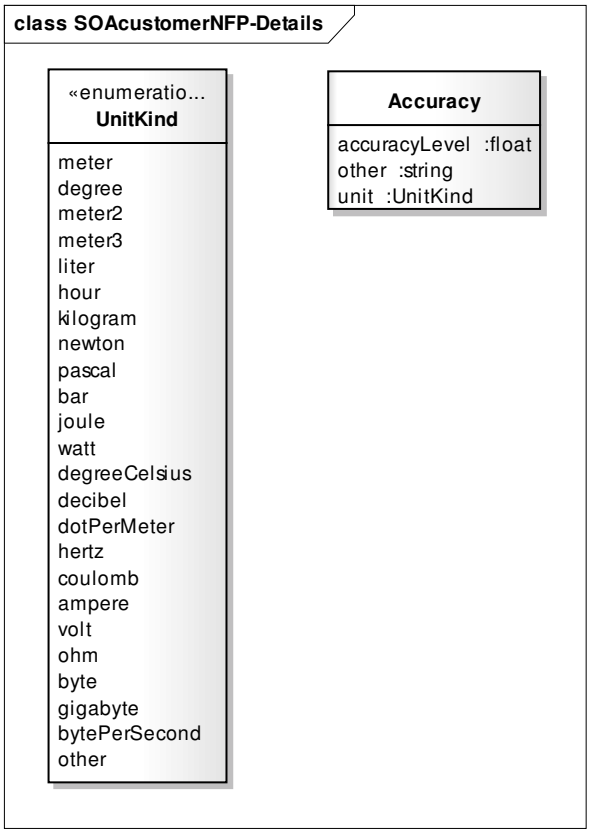


Figure 17 Accuracy NFP

Verifiable*:* Accuracy can be verified by invoking the service using an input for which the output is known or and by comparing the output to a value that is computed using a different method (often more precise but also slower).

4.4.9 Standards Compliance

Standards compliance is the list of standards, including de facto standards or technologies, with which the service claims to comply or to interoperate. Whenever the compliance is only partial (compliance to a specific part of a standard such as an annex or an option), the level of compliance to each standard may also be specified.

Note. Whenever a standard defines levels of compliance, this can be used in the definition to further specify the compliance degree. Compliance is self-declared. There are no certification mechanisms or authorities to verify or guarantee standards (or de facto

standards and technologies) compliance or interoperability. Certification is the topic of a complementary NFP. Standards compliance can be expressed as a list of tuples with three components, one to specify the name of the standard, a second to specify the version, and third to specify the level of compliance.

Relevance: "*The great thing about standards is there are so many to choose from*". Many SOA-enabling standards (e.g., standards for messaging; service description and discovery; and implementation) are available for developers to build interoperable services and composite applications. Standards not only reduce efforts to use new services but also provide certain aspects of user requirements. Therefore, there is a need for service providers to specify the (de facto) standards that they have chosen to implement, the version, and exactly to which part they comply or interoperate. The service consumers should not, and should not have to, go through the burden of testing the standards compliance or interoperability of the services.

Measurable: Can be done through certification or observation.

Data Structure: StandardsCompliance, an aggregation of Standard, the latter being defined as follows: standard: *string,* version: *string,* level: *string* (see Figure 18). This may be enhanced in the future to allow for more open and yet comparable ways of defining lists of standards.

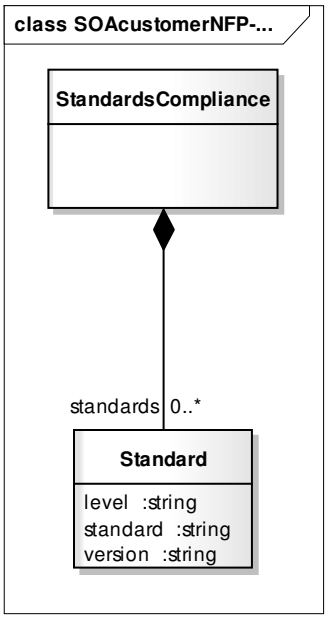


Figure 18 Standards Compliance NFP

Verifiable: Tests can be done from both sides to check whether the service complies or not to the stated list of standards. The certification mechanism should be used to alleviate the burden of testing on the service consumer.

Note about aggregation: Aggregation is not a straightforward composition here. For simple cases, it appears that the aggregation of the standards compliance can be the result of the intersections of the lists of standards of the underlying services. However, this is will not always be the case; some restrictions/conditions may apply, and at times the union of the lists might also be considered. This will be investigated in depth in Chapter 5.

4.4.10 Failure Modes

Failure Modes are the possible ways in which service failures may occur and should be handled.

Note. Services have to enumerate the failures that can happen and the required consumer behaviour for each. Examples include: the database is locked by another process, or try again after a random back off period (or re-invoke the service, or roll back, or contact the customer service, etc.).

The following properties need to be captured: 1) the failure (explanation of the exception that caused a failure such as "Bad Input Format", "Required Resource is Unavailable", "Right Access Denied", etc.) and 2) the recovery type. Such types include a) whether the service is idempotent (e.g., a service when invoked multiple times, has no further effect on its subject after the first time it is performed), and b) the capability to rollback (i.e., to maintain data integrity, it is important to detect faults before they become failures and to allow roll back to a sane state). This can help guarantee that the service returns to a sane state but may require allocating the responsibility for resubmission of inputs to the service consumer.

Relevance: Service consumers must anticipate that services from their providers will, from time to time, crash (preferably cleanly) without the opportunity for sophisticated failure handling (e.g., perhaps because of a loss of power to the provider's computer running the service). Consumers must therefore have the capability of handling such a crash.

Moreover, knowing the failure mode enables the service consumer to determine which uses of the service are suitable and which are not.

Measurable: This could be tested by causing the failure and trying to handle it as per the defined associated required consumer behaviour.

Data Structure: failureResponse: *enumeration* ResponseKind = {CrashOnly, Other}; idempotency: *boolean;* rollbackCapacity: *boolean* (Figure 19).

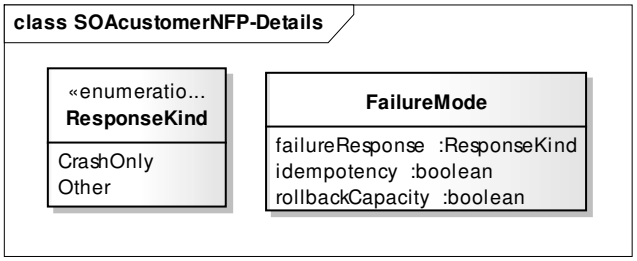


Figure 19 Failure Modes NFP

Verifiable: This NFP can be tested by emulating the failure and trying to handle it as per the defined associated required consumer behaviour.

Note on aggregation: aggregation is not obvious at this stage. The failure models of services X and Y may be very different: perhaps X fails cleanly and may, because of its idempotency, immediately be called again whereas Y has more complex failure modes. Service Z, composed of X and Y, will add its own failure modes to those of X and Y and accurately predicting the outcome could be very difficult.

4.4.11 Transactional Service

Transactional Service indicates whether the service can participate in transactional commits to maintain the integrity of data/solution. If a transaction is not committed, all intermediate effects of the transaction shall be rolled back.

Note. Transactional services require extra memory since they need to preserve the original states until they are notified to either rollback or commit their changes.

Relevance: an NFP that specifies whether a service can participate in transactional two-phase commits might be needed. This involves a sequence of independent updates that must be committed as a logical unit.

Here is an example taken from[25]: X is a composite service built out of services A, B, and C. When the services A and B complete their tasks successfully, they initiate a local transaction to temporarily save the current state of the database prior to making their changes. If service C fails, the transaction management system restores the databases back to the original states (e.g., undoing the effects of services A and B). The business task is effectively reset or rolled back across services within the pre-defined transaction boundary.

Measurable: Observable.

Data Structure: isTransactional: *boolean* (Figure 20)

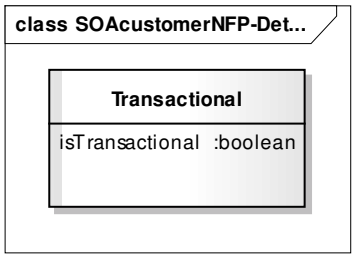


Figure 20 Transactional Service NFP

Verifiable: tests for a given input allow the verification of the service provider's claim.

4.4.12 Security

Security encompasses the protective measures that ensure a state of inviolability from hostile acts or influences.

Note. In service descriptions, service providers should state their guarantees (for example, the famous Domino's Guarantee - Your Pizza in 30 Minutes Or It's Free) and their conditions (e.g., the responsibilities of the service consumers such as for banking services - signing out and closing the Internet browser at the end of each service invocation).

The service description should indicate whenever it does not increase threats to the service security whether it supports the following functionalities:

- Data Privacy and Confidentiality: whether it is stored or in-transit, data should be kept private so that it cannot be viewed by unauthorized individuals. The service provider should state the mechanisms that ensure data privacy and confidentiality (i.e., the use of encryption technology).
- Authentication of External Service Providers: for services offered by third parties, the service should support the authentication of the service providers.
- Auditing: the availability of the auditing capability of service invocations that can be traced to specific users for logging and repudiation.
- Global Signoff: there should not be any token left in the cache memory (if any) after users exit the system.
- Password Policies: access management policies (i.e., password aging, password history, reminder hint or reset, password complexity, password lockout, and authentication to external authorities).
- Non-Repudiation: mechanisms to be able to check who did what (i.e., ability to prove that a transaction originated from a particular party, so that this party cannot deny she/he performed a certain transaction).
- The ability to delegate security to specialized components/services.
- Perimeter Security Requirements: a set of physical security and programmatic security policies that provide levels of protection against remote malicious activity.
- Protection of Resources involved in service delivery.

Relevance: Service consumers need to know the security mechanisms of each service to reason about their holistic security approach to their composite services (e.g., application).

Measurable: Could be based on certifications or reputation.

Data Structure: (1) encryption: *boolean*, dataPrivacyConfidentiality: *string*, (2) authenticationCertification (e.g., External Service Providers): *boolean,* authenticationCertifica-

tionExtra: *string*, (3) auditing: *boolean*, (4) globalSignoff: *boolean*; (5) passwordPolicies: *boolean;* passwordPoliciesExtra: *string,* (6) nonRepudiation: *boolean*, (7) securityDelegation: *boolean*, (8) securityPerimeterReqts: *string*, (9) resourcesProtection: *string*, and othersExtra: *string*. This is captured with UML in Figure 21.

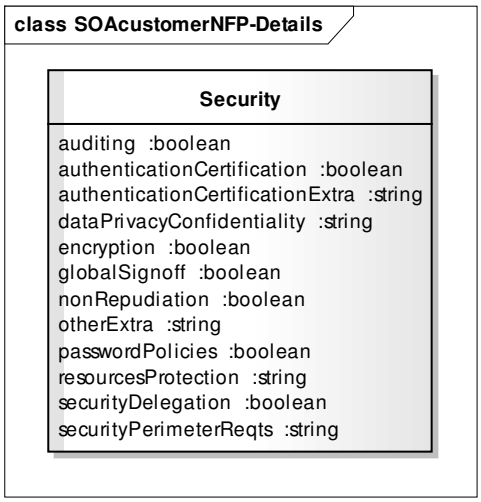


Figure 21 Security NFP

Verifiable: Using certification or testing.

4.4.13 Jurisdiction

Jurisdiction is a list of countries or territories for which a service complies with national or territorial regulations.

Note. With the ever increasing need for systems to comply with regulatory legislation, services should expose their countries/territories of jurisdiction to reflect the presumably respected legal obligations (e.g., privacy laws). This NFP is complementary to standards compliance.

Relevance: For ethics, law, and tax purposes.

Measurable: Could be based on certification or reputation.

Data Structure: CountryCode, which is an *enumeration* of all existing country codes according to the ISO 3166 standard [36] (defined below), and an aggregation of Territory, which includes name: *string*. If there is no specified Territory for a Country, then this

means that the jurisdiction applies to the whole country (i.e., Territory provides explicit restrictions).

CountryCode enumeration { AD, AE, AF, AG, AI, AL, AM, AN, AO, AQ, AR, AS, AT, AU, AW, AZ, BA, BB, BD, BE, BF, BG, BH, BI, BJ, BM, BN, BO, BR, BS, BT, BU, BV, BW, BY, BZ, CA, CC, CF, CG, CH, CI, CK, CL, CM, CN, CO, CR, CS, CU, CV, CX, CY, CZ, DD, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, EH, ER, ES, ET, FI, FJ, FK, FM, FO, FR, FX, GA, GB, GD, GE, GF, GH, GI, GL, GM, GN, GP, GQ, GR, GS, GT, GU, GW, GY, HK, HM, HN, HR, HT, HU, ID, IE, IL, IN, IO, IQ, IR, IS, IT, JM, JO, JP, KE, KG, KH, KI, KM, KN, KP, KR, KW, KY, KZ, LA, LB, LC, LI, LK, LR, LS, LT, LU, LV, LY, MA, MC, MD, MG, MH, ML, MN, MM, MO, MP, MQ, MR, MS, MT, MU, MV, MW, MX, MY, MZ, NA, NC, NE, NF, NG, NI, NL, NO, NP, NR, NT, NU, NZ, OM, PA, PE, PF, PG, PH, PK, PL, PM, PN, PR, PT, PW, PY, QA, RE, RO, RU, RW, SA, SB, SC, SD, SE, SG, SH, SI, SJ, SK, SL, SM, SN, SO, SR, ST, SU, SV, SY, SZ, TC, TD, TF, TG, TH, TJ, TK, TM, TN, TO, TP, TR, TT, TV, TW, TZ, UA, UG, UM, US, UY, UZ, VA, VC, VE, VG, VI, VN, VU, WF, WS, YD, YE, YT, YU, ZA, ZM, ZR, ZW, ZZ}.

The enumeration of country codes is implemented in the class diagram model but due to its length, it has not been included in Figure 22.

Verifiable: Could be based on certifications or testing.

Note on aggregation: The level of a third party's obligation to protect the privacy and accuracy of personal and other information with which it has been entrusted, to which services X and Y conform, may differ; perhaps X is located in a country where privacy rules are much tighter than those of the country where Y is located. Service Z may be located in yet a third country. What are the legal responsibilities of the service provider offering Z and how can that service provider determine and combine the privacy models of X and Y? This is more of a legal issue than a technical one (and therefore probably much more complex to solve). Perhaps requiring X and Y to publish their country of jurisdiction would suffice.

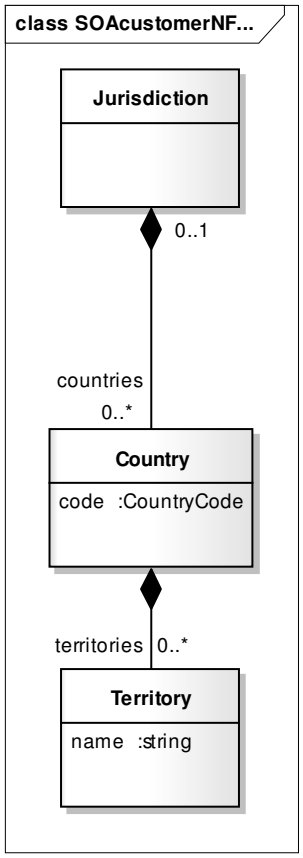


Figure 22 Jurisdiction NFP

4.4.14 Service Versioning

Service Versioning is the process of documenting the nature of changes for a service (i.e., interface update, backward compatibility, added, or removed functionalities, or fixed bugs).

Note. Service versioning allows service consumers to compare two consecutive versions of a service and decide which version they should be using. The nature of changes should be specified (1) whether to change or preserve the same interface when the service is updated; (2) whether there is a possibility of backward compatibility; (3) a list of the removed functionalities; (4) a list of the extensions or modification in service functionalities; and/or (5) bug fixes.

Relevance: To know how to handle future updates, i.e., simply invoke the service or look for new updates/versions.

Measurable: It is not measurable; one can simply invoke the service and observe its behaviour.

Data Structure: sameInterface: *boolean*, backwardCompatibility: *boolean*, removed-Functionalities: *string*, addedFunctionalities: *string*, fixedBugs: *string*. Whenever sameInterface and backwardCompatibility are both true and when all of the other fields are empty strings, one can assume that there is only one version of that given service. See Figure 23 for the UML-based representation.

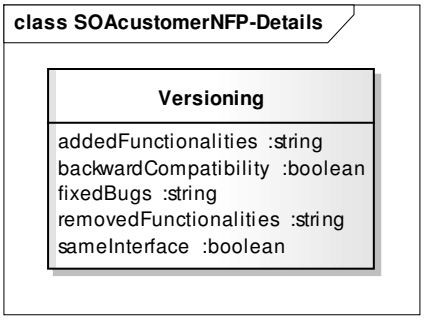


Figure 23 Service Versioning NFP

Verifiable: by testing it.

4.4.15 Resource Requirements

Resource Requirements are the minimum and/or the recommended sets of requirements needed at the consumers' end to support successful execution of a service.

Note. Resources requirements may include but are not restricted to hardware (including computers), systems and applications software, telecommunications (voice and data) and critical files.

Resource requirements are specified as a 4-tuple: <category, description, quantity, unit>.

Categories include the following: random access memory (RAM), storage, computer graphics display, peripherals, APIs and drivers, Web browser, Internet connection (type and speed), and other requirements.

It is possible to use the different kinds of resource categories defined in OMG's MARTE standard [64], including:

- Storage Resource (e.g., memory on the hard disk drive and its size in bits);
- Timing Resource (e.g., a chronometric clock);
- Sync Resource (e.g., to arbitrate concurrent execution flows, and in particular the mutual exclusive access to shared resources);
- Computing Resource (e.g., a virtual or physical processing device capable of storing and executing program code);
- Concurrency Resource (e.g., to perform its associated flow of execution concurrently with others);
- Device Resource (e.g., an external device that may require specific services in the platform for its usage and/or management);
- Communication Resources (e.g., packet size and the communication media); and
- Other (the resource category is documented in the Description of the resource)

These resource kinds are captured in Figure 24.

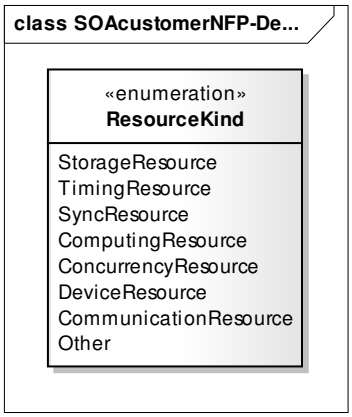


Figure 24 Kinds of Resources

Relevance: Service consumers should be in a position to ensure that they have the required resources before invoking any given service.

Measurable: By monitoring resources usage with a set of counters of each required resource.

Data Structure: category: *ResourceKind* (e.g., the enumeration is defined above), description: *string*, quantity: *real*, unit: *string* (Figure 25).

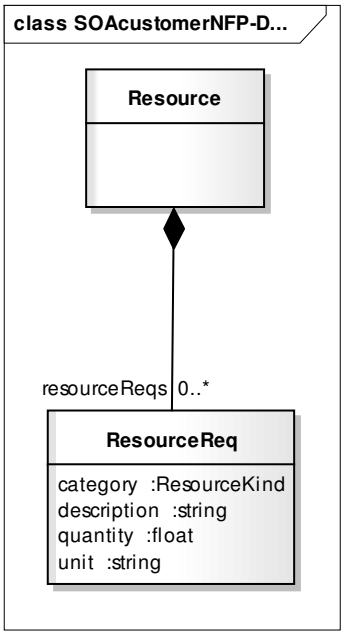


Figure 25 Resource Requirements NFP

Verifiable: by testing it, i.e., by checking that the resources available are equal or superior to the required resources for invoking the service.

4.4.16 Scalability

Scalability is the property of handling increasing service invocations attempts and could be measured as the maximum number of service requests that can be executed at the same time.

Note. Scalability of a given service could be significantly impacted by utilization of other services on the same set of resources. Scalability is tightly coupled with response time. Scalability can be expressed as a natural number.

Relevance: Service consumers may need to know whether a given service can scale to increasing demand.

Measurable: By monitoring how well increasing demand is handled

Data Structure: scaleTo: *integer* (Figure 26), which represents the maximum number of service requests that can be executed at the same time.

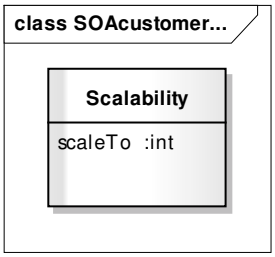


Figure 26 Scalability NFP

Verifiable: by testing it.

4.4.17 Server Location

Server Location is a particular place in the physical space.

Note. For distributed and mirrored servers, a list of countries with territories will be provided. However, there is no guarantee that the data does not transit through different countries. The network topology and routing are out of the control of the service provider.

Relevance: This is useful to consumers who do not want their data to enter a given country.

Measurable: Not measurable.

Data Structure: CountryCode, which is an *enumeration* of all the codes (defined above as part of the Jurisdiction NFP), and an aggregation of Territory defined as name: *string*. See the UML representation in Figure 27. If there is no specified Territory for a Country, then this means that the server is located somewhere in that country.

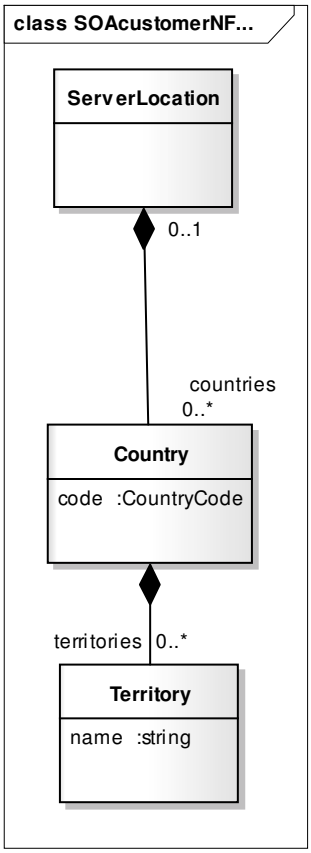


Figure 27 Server Location NFP

Verifiable: Not verifiable. Reputation of the service provider is the only guarantee.

Figure 28 represents the combination of ServerLocation and Jurisdiction since they do refer to the same data structure, that is, a list of countries and territories.

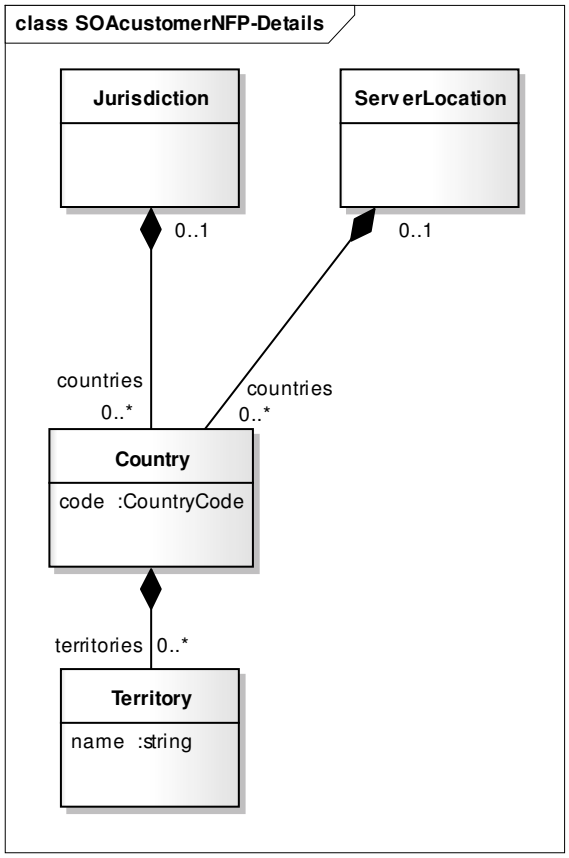


Figure 28 Server Location and Jurisdiction

4.5. Catalogue Metamodel

The UML class diagrams defined above provide the definitions of NFPs of services in SOA and their specific data types.

These UML classes (summarized as one diagram in Figure 29), combined with relevant aggregated classes and enumeration classes, are used to generate an XML schema (found in Appendix A) that provides a concrete syntax enabling the description of the NFP specification of a service. Such specification, being based on XML, will enable the integration of NFP descriptions with other technologies, such as WSDL.

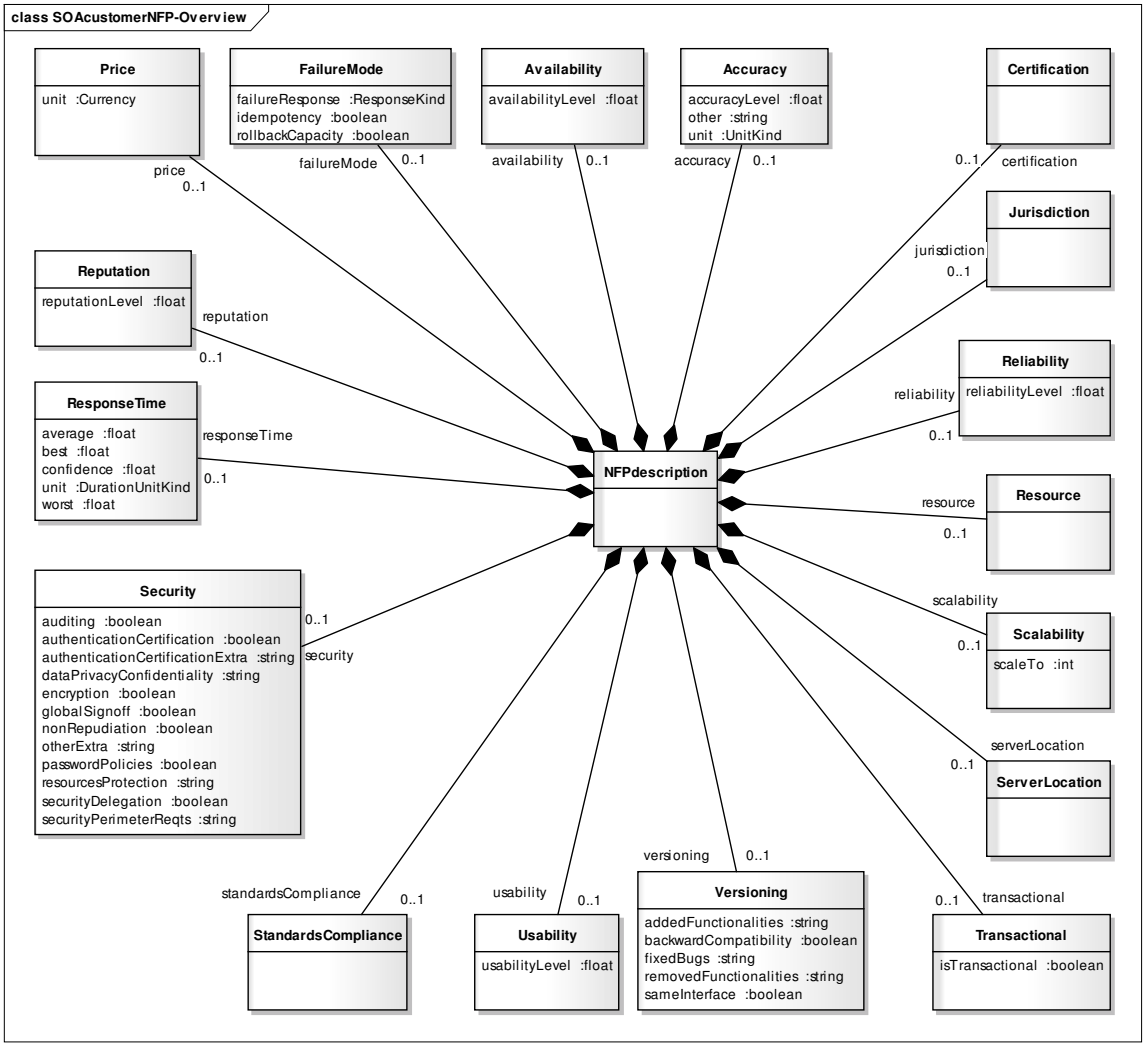


Figure 29 Overview of the NFP Catalogue

4.6. Chapter Summary

The aim of this chapter was to assess the relevance of NFPs for SOA and investigate, in particular, the properties that should be exposed as part of the service description of atomic services in order to empower service consumers (as opposed to service providers). To meet this objective, a literature review was performed and an initial catalogue of 19 NFPs is defined, which was then distributed as part of an online survey. The answers and comments received from 29 knowledgeable participants from many domains helped refine this list by removing 5 of the initial NFPs, adding 3 more, and revisiting the scope,

terminology, and definitions of the remaining ones. The new catalogue was further sent for inspection and comments to a subset of eight of the initial survey participants.

For each of the resulting 17 NFPs, this chapter provided a concise definition, provided additional notes, justified its relevance, and provided a data structure that formalizes precisely how it can be described and measured. A UML model comprising all the classes, attributes and associations discussed in this chapter was used to generate a concrete XML schema, providing users with a concrete syntax to describe precisely the NFPs of their services.

This new catalogue is important because, unlike what can be found in the literature, it focuses on the customer's perspective, it was validated with real users and experts, and it was defined in a domain-independent and technology-independent way so that it can now be used by multiple developers and standardization bodies. One threat however is that many of the survey participants were technologists working with service providers, and not necessarily plain service consumers (although often service providers are service consumers themselves).

Such NFP catalogue could obviously be debated indefinitely. To mitigate this threat, participation to the survey was encouraged as much as possible, and the details of the resolutions were documented precisely in a separate document [10]. This catalogue needed to get frozen at some point to be able to progress and enable advanced applications such as service monitoring, comparison and substitution. However, this catalogue is implemented in an extensible way, so that other NFPs, perhaps more domain-dependent, can eventually be incorporated.

When compared to the state of the art based on the selected criteria in Chapter 3, and especially on Problem 1 related to NFP taxonomies, it can be seen that this work has ended up with overall better results. The first criterion is the degree of formalism of the data structure of the NFPs. This criterion is satisfied, as all the NFPS are represented by a formal data structure (in UML), in a measurable way whenever possible, and with a concrete XML-based syntax. The second criterion is the external validation, and again this is satisfied via the results obtained through an external survey.

The following chapter will explore how NFPs of composite services can be computed from the NFPs of atomic services in a context of service selection/composition and dynamic adaption.

Chapter 5. Aggregation of Non-Functional Properties

This chapter explores the aggregation of the seventeen types of NFPs defined for atomic services in the previous chapter, and then defines and illustrates their aggregation functions whenever possible for various composition operators (defined in section 5.1).

The *BookItWell* use case is used to illustrate and validate NFP aggregation procedures.

5.1. Composition Operators

Commonly, five composition operators (viz. Sequence, Concurrency, Conditional Branching, Loop and Discriminator) are used in the description of workflows. Only four of these operators (Sequence, Concurrency, Conditional Branching, and Loop) are frequently used for service composition. The use of the Discriminator operator in service composition is an original contribution that results from this work. The Discriminator operator is particularly important for mission critical applications since it enables the creation, for instance, of fault-tolerant composite services. Figure 30 shows the UCM notational elements that represent these operators, where services are captured with *stubs* visualized with diamond symbols; note that all of these operators can be combined.

Sequence: All services placed in series are invoked and the result of each service is needed (i.e., the failure of any service results in the failure of the composite service).

Concurrency: Two or more services placed in parallel are invoked concurrently and the results of all services are needed. Usually these services have different but complementary tasks – hence there is a need for synchronization.

Conditional branching: Only one service out of several services is invoked depending on the result of branching conditions.

Discriminator: Two or more equivalent services are invoked in parallel to achieve a given task but only one is required to finish before proceeding with the invocation of the next composed services of the composite service. It is presumed that these services are equivalent in terms of functionalities but different from the perspective of their NFPs. The results of the first service to finish are used while the results of the remaining invoked services are ignored. At least one service of the invoked set of services must succeed for the composite service to succeed.

Loop: One or more services can be part of a "cycle" in the composite service. These services will be invoked as many times as needed until a given loop condition is satisfied. All invoked services that are part of the loop must succeed as many times as they are invoked for the composite service to succeed. For a meaningful aggregation of NFPs, the maximum number of required invocations must be known.

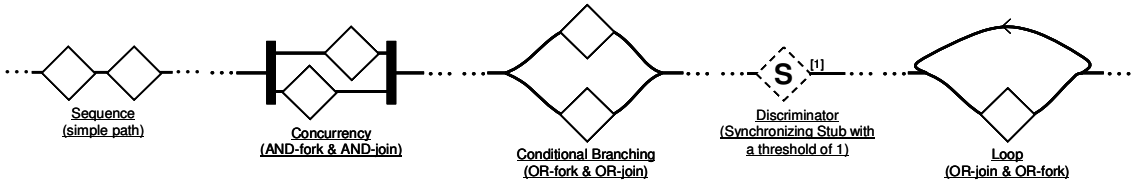


Figure 30 Visual Representation of Composition Operators in UCM

5.2. Aggregation Algorithms and Illustrative Examples

The final list of NFPs is composed the following seventeen properties: (1) price, (2) response time, (3) reputation, (4) certification, (5) availability, (6) reliability, (7) usability, (8) accuracy, (9) standards compliance, (10) failure modes, (11) transactional service, (12) security, (13) jurisdiction, (14) service versioning, (15) resource requirements, (16) scalability, and (17) server location

For each of these seventeen NFPs, the aggregation functions for each of the five previously defined composition operators are defined. The proposed aggregation approach used in this thesis is based on a conservative or *pessimistic* view of the composite service. For example, in theory, the Conditional Branching operator has to take into account the chosen/invoked service only. However, because it is not known which services out of the branch will be eventually executed, it is assumed that the worst case occurs.

This means that it is assumed that the invoked service will be the one with the highest price, the poorest response time, and the lowest reliability and the loosest availability.

In the following subsections, the resulting aggregated NFP must also be described using the same data structure as what is used as input, i.e., the data structure defined in Chapter 4. Also, *n* is defined as the number of services composed with the composition operator.

5.2.1 Price

This subsection proposes aggregation algorithms for the Price NFP. For these algorithms, the aggregated price will have one currency and as many "PriceElement" as different price elements that the invoked services have. The resulting Price will have the same data structure as what has been used as input: the price of atomic service described in subsection 4.4.1.

Algorithm I, *AggregatePrices*, covers the aggregation of Prices for all the operators. The aggregation function Price.value = $\sum_{i=1}^{n} Pi$ is used for all operators except Conditional Branching. The Loop is actually a special case of the Sequence operator, and its function $\sum_{i=1}^{n} Pi$ could be simplified to $n * P$. For Conditional Branching, the aggregation function Price.value = $max\ (P_1 \ldots P_n)$ is used. Algorithm I makes use of *ConvertCurrency*, discussed briefly in Algorithm II.

Algorithms for Price Aggregation

Algorithm I: AggregatePrices

Inputs: aggregateUnit: Currency
prices: collection(Price), *// for Loops, the collection contains n Prices, where n is the number of iterations.*
composition: CompositionType
Output: aggregatedPrice: Price

```
begin
// the aggregate price is of a given unit
aggregatedPrice = new Price
aggregatedPrice.unit = aggregateUnit
// handle the collection of prices
for each p:Price in prices do
 // handle each price element for the next price p
    for each pe:PriceElement in p.priceElements do
       // find the output priceElement (named ape) in the collection aggregated
       // Price.priceElements with the same type of pe, if any
          aggregatePE:PriceElement = find ape:PriceElement in aggregatedPrice.priceElements
                                     where ape.type == pe.type
       // create one new output price element if none of a matching type is found
```

```
        if aggregatePE == null
           then
                aggregatePE = new PriceElement
                aggregatePE.type = pe.type
                aggregatePE.value = 0
                aggregatePrice.priceElements.add(aggregatePE)
          end if
        if composition == ConditionalBranching
           // find the output price element with the higher price for each type of PriceElement
           then aggregatePE.value = max (aggregatePE.value, ConvertCurrency (pe.value, pe.unit, aggregateUnit))
           // add current value to the price element, using the appropriate currency
           // always convert currency (even if the currency is the expected one)
           else aggregatePE.value = aggregatePE.value + ConvertCurrency (pe.value, pe.unit, aggregateUnit)
        end if
     end for
  end for
return aggregatedPrice
end
```

The ConvertCurrency algorithm converts a value from a source currency to a value in another currency. It is assumed that such service already exists and thus it is not explained further here.

Algorithm II: ConvertCurrency

Inputs: srcValue: Real
srcUnit: Currency
convertedUnit: Currency
Output: convertedValue: Real

Example of Price Aggregation

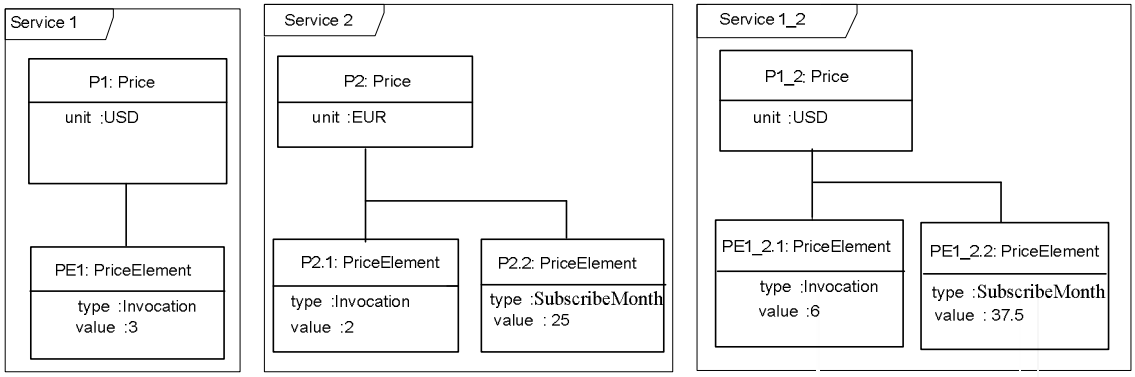


Figure 31 Example of Price Aggregation

As shown in Figure 31, the Price descriptions of two services, Service 1 and Service 2, have been defined. Then, the Price aggregation for the resulting composed service Ser-

vice 1_2 is computed. In this example, the composition could be any operator except the Conditional Branching.

The input unit for the aggregation algorithm is the aggregateUnit = USD and the conversion is based on the following rate: 1 EUR = 1.5 USD.

First, the algorithm creates the Price object P1_2, with unit=USD and only one PriceElement (PE1_2.1) for which type=Invocation and value=0. As the composition is not conditional branching, the aggregate value of that first PriceElement is then set to 0 USD + 3 USD converted to USD (i.e., 3 USD in the end). The second PriceElement considered is P2.1. As this is also an Invocation, its value is converted and added to that of PE1_2.1: 3 USD + 2 EUR *1.5 USD/EUR = 6 USD. For P2.2, an additional PriceElement of type=SubscribeMonth needs to be created. The computation of value is a simple conversion of 25 EUR to USD (i.e., 37.5 USD). Thus, P1_2.2 = 37.5.

Note that despite the proposed aggregation functions, which can compute a default composite price that is reasonable, it is suggested that the service provider could and should have her/his own logic to decide about the price of the composed service that is to be offered. The price should take into account the business domain, the business model and the strategy of the company. Probably, some services compositions (price wise in particular) do not make sense from the business strategy perspective. A service provider may decide to compose a service based on services for which she/he pays a monthly subscription and offers the composed service for a price per invocation. It is most likely that the services that are for a subscription (e.g., monthly) do not have the same starting (and ending) dates.

5.2.2 Response Time

Algorithm III covers the aggregation of Response Time NFPs for all the operators.

The part of this Response Time algorithm handling Sequence and Loop consists of computing the sum of Best, Average and Worst times after having normalized them to the same unit (e.g., $\sum_{i=1}^{n} RTi$). Loop is a special case of Sequence here; the function used for Loop could be simplified to $m * RT$. Since the proposed approach followed here is pessimistic, the confidence is computed as the minimum one, out of the confidences of the collection of invoked services.

The part of the algorithm handling the operators Concurrency and Discriminator computes the maximum time of the Worst, Average, Best times of the invoked services, i.e., max$(RT_1..RT_n)$. The confidence is computed here as the minimum value.

Finally, the handling of Conditional Branching is computed as follows: the maximum of the Worst times, the average of the Average times, and the minimum of the Best times of the invoked services. The confidence is yet again computed as the minimum value. Algorithm III makes use of *ConvertTime*, discussed briefly in Algorithm IV.

Algorithms for Response Time Aggregation

Algorithm III: AggregateResponseTime

```
Inputs:  aggregateUnit: DurationUnitKind
         ResponseTimes: collection(ResponseTime),     // for Loops, the collection contains n ResponseTimes,
                                                      // where n is the number of iterations.
         composition: CompositionType
Output:  aggregatedResponseTime: ResponseTime

begin
// create and initialize the default attributes of the aggregateResponseTime
aggregatedResponseTime = new ResponseTime
aggregatedResponseTime.unit = aggregateUnit
aggregatedResponseTime.worst = 0
aggregatedResponseTime.average = 0
aggregatedResponseTime.best = 0
aggregatedResponseTime.confidence = 100

if composition in {Sequence, Loop}
   // handle the collection of ResponseTimes
  for each RT: ResponseTime in ResponseTimes do
     // add Worst, Average, and Best to the AggregatedResponseTime, using the appropriate Unit.
     // always convert units.
     aggregatedResponseTime.worst =   aggregatedResponseTime.worst
                                      + ConvertTime (RT.worst, RT.unit, aggregateUnit)
     aggregatedResponseTime.average = aggregatedResponseTime.average
                                      + ConvertTime (RT.average, RT.unit, aggregateUnit)
     aggregatedResponseTime.best =    aggregatedResponseTime.best
                                      + ConvertTime (RT.best, RT.unit, aggregateUnit)
     aggregatedResponseTime.confidence = min (aggregatedResponseTime.confidence, RT.confidence)
   end for
end if

if composition in {Concurrency, Discriminator}
   for each RT: ResponseTime in ResponseTimes do
      // compute the maxs and mins for the operators: Concurrency and Discriminator.
      aggregatedResponseTime.worst =
                    max (aggregatedResponseTime.worst, ConvertTime (RT.worst, RT.unit, aggregateUnit))
      aggregatedResponseTime.average =
                    max (aggregatedResponseTime.average, ConvertTime (RT.average, RT.unit, aggregateUnit))
      aggregatedResponseTime.best =
                    max (aggregatedResponseTime.best, ConvertTime (RT.best, RT.unit, aggregateUnit))
      aggregatedResponseTime.confidence = min (aggregatedResponseTime.confidence, RT.confidence)
  end for
end if
```

```
if composition in {ConditionalBranching}
   partialAverageSum:real = 0
   for each RT: ResponseTime in ResponseTimes do
      // compute the sums, maxs and mins for the operator Conditional Branching.
      aggregatedResponseTime.worst =
                  max (aggregatedResponseTime.worst, ConvertTime (RT.worst, RT.unit, aggregateUnit))
      partialAverageSum = partialAverageSum + ConvertTime (RT.average, RT.unit, aggregateUnit))
      aggregatedResponseTime.best =
                  min (aggregatedResponseTime.best, ConvertTime (RT.best, RT.unit, aggregateUnit))
      aggregatedResponseTime.confidence = min (aggregatedResponseTime.confidence, RT.confidence)
   end for
   // compute average for the Average based on the sum
   aggregatedResponseTime.average = partialAverageSum / ResponseTimes.size()
end if
return aggregatedResponseTime
end
```

Algorithm IV converts a time unit from a source time unit to a requested time unit. It is assumed to exist and is not explained further here.

Algorithm IV: *ConvertTime*

Inputs: srcValue: Real
srcUnit: DurationUnitKind
convertedUnit: DurationUnitKind
Output: convertedValue: Real

Example of Response Time Aggregation

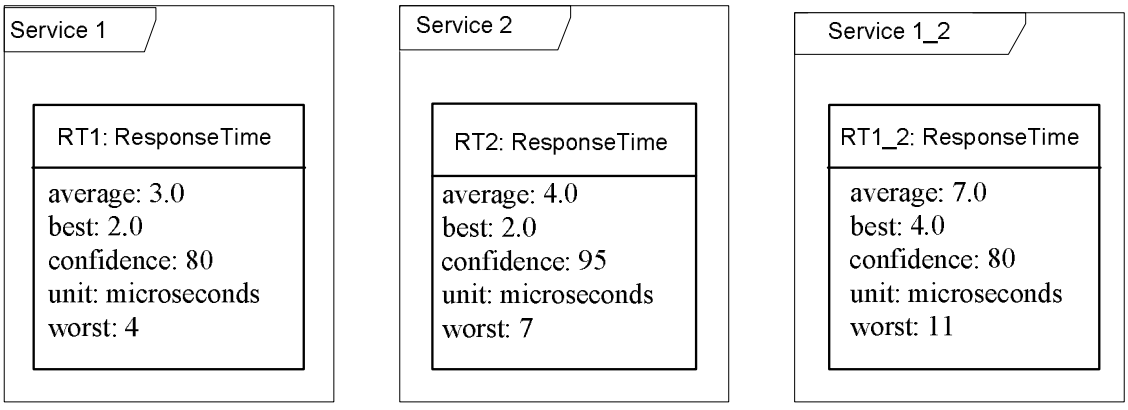


Figure 32 Example of Response Time Aggregation

As shown in Figure 32, the Response Time descriptions of Service 1 and Service 2 are defined. Then, the Response Time aggregation for the resulting composed service, Service 1_2, is computed. In this example, the composition could be Sequence or Loop.

The input unit for the aggregation algorithm is the aggregateUnit = microseconds. First, the algorithm creates the ResponseTime object RT1_2, with unit = microseconds, worst = 0, average = 0, best = 0, and confidence = 100. As the composition is Sequence or Loop, the aggregate values of worst, average and best are equal to the sum of these similar attributes of Service 1 and Service 2, after their conversion to the aggregateUnit, which is the same as the input unit in this example. The confidence of Service1_2 is equal to the minimum confidence among those of Service 1 and Service 2, which is 80.

5.2.3 Reputation

This subsection proposes an aggregation algorithm for the Reputation NFP. Reputation of a composed service is equal to the reputation of the service that has the worst reputation (e.g., the minimum reputation), and this is valid for all the operators.

Algorithm for Reputation Aggregation

Algorithm V: AggregateReputation

Inputs: reputations: collection (Reputation)
Output: aggregatedReputation: Reputation

```
begin
// create and initialize the default attributes of the aggregateReputation
aggregatedReputation = new Reputation
aggregatedReputation.reputationLevel = 5 // the best reputation, by default

// handle the collection of reputations
for each rep: Reputation in reputations do
   aggregatedReputation.reputationLevel = min (aggregatedReputation.reputationLevel, rep. reputationLevel)
end for
return aggregatedReputation
end
```

Example of Reputation Aggregation

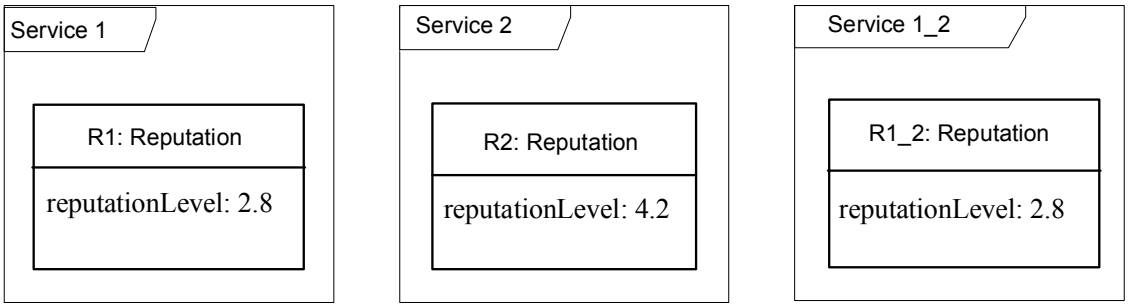


Figure 33 Example of Reputation Aggregation

As shown in Figure 33, the Reputation descriptions of Service 1 and Service 2 have been defined. Then, the Reputation aggregation for the resulting composed service Service 1_2 is computed. In this example, the composition could be any operator.

First, the algorithm creates the Reputation object R1_2 with reputationLevel = 5. Then, the aggregate value of reputationLevel is computed as the minimum of the reputationLevel values of Service 1 and Service 2, which is 2.8 here.

5.2.4 Certification

Automated aggregation of the Certification NFP is not possible, not even for one service that is invoked *m* times as part of a loop. The whole is *different* from the sum of its parts. The fact that basic services are certified does not guarantee in any way that the composite service is certified. The composite service should be assessed and certified on its own, and this requires external intervention.

5.2.5 Availability

Algorithm VI covers the aggregation of the Availability NFP for all the operators. For Sequence, Concurrency and Loop, the product per the function $\prod_{i=1}^{n} Ai$, where Ai is the availability of service i, is computed. This means that the availability of the aggregated

service will be lower than that of any of its sub-services. For the Loop, this function can be simplified as A^n.

The aggregation of Availability for the operator Conditional branching is the minimum between the availabilities of the involved services (because a pessimistic view is taken).

The aggregation of Availability for the operator Discriminator is computed as follows: $1 - \prod_{i=1}^{n}(1 - Ai)$. This means that the availability of the aggregated service will be higher than that of any of its sub-services.

Algorithm for Availability Aggregation

Algorithm VI: AggregateAvailability

```
Inputs:   availabilities: collection (Availability)
          composition: CompositionType
Output:   aggregatedAvailability: availabilityLevel

begin
// create and initialize the default attributes of the aggregatedAvailability
aggregatedAvailability = new Availability
aggregatedAvailability.availabilityLevel = 1 // full availability by default

// handle Availability for Sequence, Concurrency, and Loop - compute the product
if composition in {Sequence, Concurrency, Loop}
    for each av: Availability in availabilities do
        aggregatedAvailability.availabilityLevel = aggregatedAvailability.availabilityLevel * av.availabilityLevel
    end for
end if

// handle Availability for Conditional Branching -  compute min (A1…An)
if composition in {ConditionalBranching}
    for each av: Availability in availabilities do
        aggregatedAvailability.availabilityLevel = min (aggregatedAvailability.availabilityLevel, av.availabilityLevel)
    end for
end if

// handle Availability for Discriminator - compute 1 − ∏_{i=1}^{n}(1 − Ai)
if composition in {Discriminator}
     partialProduct = 1
    for each av: Availability in availabilities do
        partialProduct =  partialProduct * (1 – av.availabilityLevel)
    end for
    aggregatedAvailability.availabilityLevel = 1 – partialProduct
end if
return aggregatedAvailability
end
```

Example of Availability Aggregation

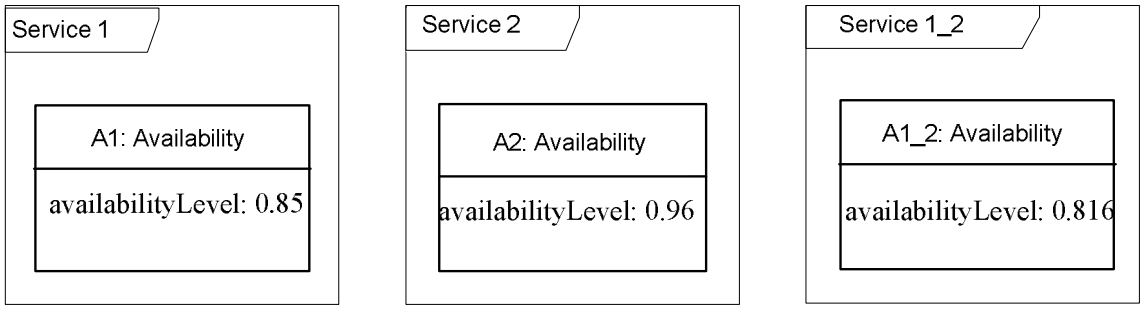


Figure 34 Example of Availability Aggregation

As shown in Figure 34, the Availability descriptions of Service 1 and Service 2 have been defined. Then, the Availability aggregation for the resulting composed service, Service 1_2, is computed. In this example, the composition could be Sequence, Concurrency or Loop.

First, the algorithm creates the Availability object A1_2, with availabilityLevel initially equal to 1. Then, the aggregate value of availabilityLevel is computed as the product of availabilityLevel of Service 1 and Service 2, which is equal to 0.816.

5.2.6 Reliability

This subsection proposes Algorithm VII for the handling the Reliability NFP for all the operators. For Sequence, Concurrency and Loop, the product per the function $\prod_{i=1}^{n} Ri$, where Ri is the reliability of service i, is computed. This means that the reliability of the aggregated service will be lower than that of any of its sub-services. For the Loop, this function could be simplified as R^n.

The aggregation of Reliability for the operator Conditional branching is the minimum value of reliabilities of all the involved services.

The aggregation of Reliability for the operator Discriminator is computed as follow: $1 - \prod_{i=1}^{n}(1 - Ri)$. This means that the reliability of the aggregated service will be higher than that of any of its sub-services.

Algorithm for Reliability Aggregation

Algorithm VII: AggregateReliability

Inputs: reliabilities: collection (Reliability)
composition: CompositionType
Output: aggregatedReliability: Reliability

begin
// create and initialize the default attributes of the aggregatedReliability
aggregatedReliability = **new** Reliability
aggregatedReliability.reliabilityLevel = 1

// handle Reliability for Sequence, Concurrency, and Loop - compute the product
if composition **in** {Sequence, Concurrency, Loop}
  for each re: reliability **in** reliabilities **do**
    aggregatedReliability. reliabilityLevel= aggregatedReliability. reliabilityLevel * re. reliabilityLevel
  end for
end if

// handle Reliability for Conditional Branching - compute min ($R_1 \ldots R_n$)
if composition **in** {ConditionalBranching}
  for each re: Reliability **in** reliabilities **do**
    aggregatedReliability.reliabilityLevel= **min** (aggregatedReliability.reliabilityLevel, re.reliabilityLevel)
  end for
end if

// handle Reliability for Conditional Branching - compute $1 - \prod_{i=1}^{n}(1 - Ri)$
if composition **in** {Discriminator}
  partialProduct = 1
  for each re: Reliability **in** reliabilities **do**
    partialProduct = partialProduct * (1 – re.reliabilityLevel)
  end for
  aggregatedReliability.reliabilityLevel = 1 – partialProduct
end if
return aggregatedReliability
end

Example of Reliability Aggregation

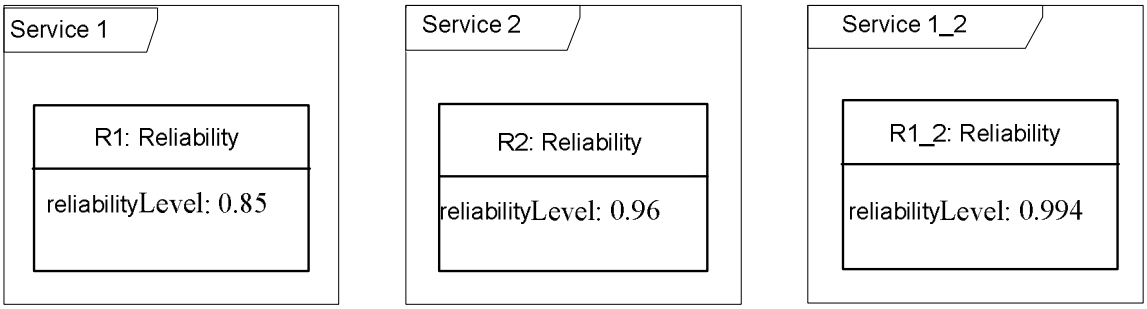


Figure 35 Example of Availability Aggregation

As shown in Figure 35, the Reliability descriptions of Service 1 and Service 2 have been defined. Then, the Reliability aggregation for the resulting composed service, Service 1_2, is computed. In this example, if the CompositionType is Sequence, Concurrency or Loop, then the aggregated Reliability will be computed in a way similar the previous example for Availability aggregation. Hence, this example computes the Reliability aggregation for another operator: the Discriminator.

First, the algorithm creates the Reliability object R1_2 with reliabilityLevel = 1. Then, the aggregate value of reliabilityLevel is computed per the following function, $1 - \prod_{i=1}^{n}(1 - Ri)$, which results in R1_2 = 1 – [(1 – 0.85) * (1 – 0.96)] = 0.994.

5.2.7 Usability

For the computation of the aggregation of usability NFP, it was decided to use $min(U_1 \ldots U_n)$ instead of the average of the usability levels. This approach means that the usability of the service is as "good" as the worst service involved in the aggregation.

Algorithm for Usability Aggregation

Algorithm VIII: *AggregateUsability*

Inputs: usabilities: collection(Usability)
Output: aggregateUsability: Usability

```
begin
// create and initialize the default attributes of the aggregateUsability
aggregatedUsability = new Usability
aggregatedUsability. usabilityLevel = 5

// handle the collection of Usabilities
for each us: Usability in usabilities do
    aggregatedUsability.usabilityLevel = min (aggregatedUsability.usabilityLevel, us.usabilityLevel)
end for
return aggregatedUsability
end
```

Example of Usability Aggregation

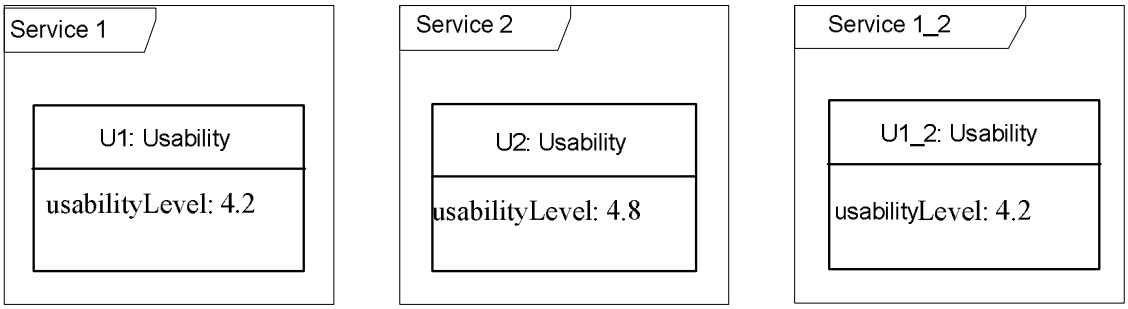


Figure 36 Example of Usability Aggregation

As shown in Figure 36, the Usability descriptions of two services, Service 1 and Service 2, have been defined. Then, the Usability aggregation for the resulting composed service Service 1_2 is computed. In this example, the composition could be any operator.

First, the algorithm creates the Usability object U1_2, with reputationLevel = 5. Then, the aggregate value of usabilityLevel is computed as the minimum UsabilityLevel value of Service 1 and Service 2, which is equal here to 4.2.

5.2.8 Accuracy

Due to the propagation of loss of accuracy, also known as error/uncertainty propagation, it is not possible to compute the aggregation of Accuracy without accessing and understanding the logic of the underlying services, which is in opposition with the SOA principles. Hence, manual intervention is required to compute the aggregation of accuracy. As explained by Taylor in [75], there is no absolute accuracy propagation. Even for simple composition scenarios, Accuracy propagates differently when performing different arithmetic operations. When the service calculates addition and subtractions, the accuracy propagates differently than when it calculates multiplications and divisions. Consequently, the aggregation should be computed differently as well.

5.2.9 Standards Compliance

Similar to the Certification NFP and for the same rationale (e.g., "the whole is something different from the sum of its parts"), the Standards Compliance NFP of the aggregated service should be assessed manually to be able to establish the new list of standards to which it does comply.

5.2.10 Failure Modes

Failure Modes cannot be aggregated automatically either. The failure models of services X and Y may be very different: perhaps X fails cleanly and may, because of its idempotency, immediately be called again whereas Y has more complex failure modes. Service Z, which aggregates X and Y, will add its own failure modes to those of X and Y.

Based on the principles of SOA, the services should be opaque. Hence, one cannot give details about the behaviour of each underlying service and how it should be handled in situations where it fails. Due to this interaction problem, the composed service could have a totally different failure mode due to conflicts (e.g., bad interaction) between two or more of its underlying services. Even when all the underlying services have the same failure response, the failure mode of the composed service should be assessed on its own.

5.2.11 Transactional Service

This subsection proposes Algorithm IX for aggregating Transactional Service NFPs. Whenever the composite service is not successfully completed, its underlying services that are performed may have made changes that compromise the integrity of the solution. Hence, the underlying performed services need to perform a rollback feature (e.g., reset all actions and changes) via the centralized transaction management system.

For a composed service to be able to become transactional, it is required that all its underlying services be transactional. Thus, the algorithm computes the Boolean function $\bigwedge_i^n Trans_i$ for all the operators, where $Trans_i$ indicates whether sub-service i is transactional. This function can be simplified to $Trans_i$ for the Loop operator.

Algorithm for Transactional Service Aggregation

Algorithm IX: AggregateTransactional

Inputs: transactionalPlus: collection (Transactional)
Output: aggregatedTransactional: Transactional

begin
// create and initialize the default attributes of the aggregatedTransactional
aggregatedTransactional = **new** isTransactional
aggregatedTransactional.isTransactional = **true**

for each trans:Transactional **in** transactionalPlus **do**
    // compute the and operation
    aggregatedTransactional.isTransactional = aggregatedTransactional.isTransactional **and** trans.isTransactional
end for
return aggregatedTransactional
end

Example of Transactional Service Aggregation

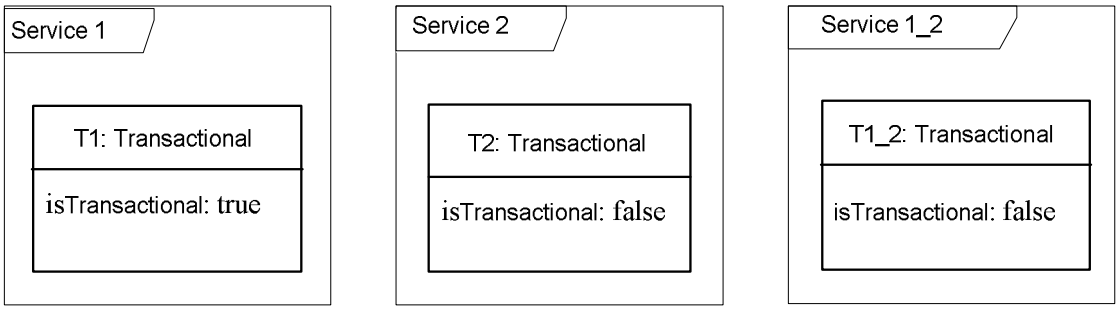


Figure 37 Example of Transactional Service Aggregation

As shown in Figure 37, the Transactional Service descriptions of Service 1 and Service 2 have been defined. Then, the Transactional Service aggregation for the resulting composed service, Service 1_2, is computed. In this example, the composition could any operator.

First, the algorithm creates the Transactional Service object T1_2, with the attribute isTransactional = true. Then, the aggregate value of this attribute is computed per the following function: $\bigwedge_i^n Trans_i$. In this example, based on the attributes of Service 1 and Service 2, aggregatedTransactional = (true ^ false) = false.

5.2.12 Security

Using the pessimistic aggregation logic, the aggregation of Security requirements could be equal to the weakest aspects of each involved service. This requires having an ontology of an ordered set of security requirements. However, some security aspects are not comparable.

In addition, comparing different security aspects cannot be done without a classification of dependencies and effects between them. "Interactions between these requirements must be considered. A requirement interaction is the effect that two requirements have on each other" [13]. Two requirements could be independent or equivalent, or prevent, restrict, complement, require, conflict or exclude each other. The possibility of interacting security requirements has to be considered even for atomic services. Understanding requirements interactions is more challenging in the context of composed services.

"Each service has its own security requirements that might cause complex dependencies between the requirements of different services… Due to possible interactions between security requirements, it is likely that the security requirements of the composite service are unequal to the union of all requirements [of involved services]" [13]. Hence, even if the basic services are secure, this does not guarantee in any way that the composite service is secure. Human intervention is required, at least for now, to check and resolve security requirements conflicts in composite services (e.g., conflict resolution strategy to decide which requirement should be preserved). For example, confidentiality requirements could be preferred over monitoring requirements. The security aspects of the composite service should hence be assessed on their own.

5.2.13 Jurisdiction

This subsection proposes an Algorithm X for aggregating Jurisdiction NFPs. The *intersection* of all the lists of the countries and their territories is computed in order to return the aggregation of all common jurisdictions in all the services (independently of the composition operator): $\bigcap_{i=1}^{n} Jurisdiction_i$. The same algorithm applies for Sequence, Con-

currency, Conditional Branching, Discriminator and even Loop (the latter could also be simplified as the jurisdiction remains the same for all $Jurisdiction_i$).

Algorithm for Jurisdiction Aggregation

Algorithm X: *AggregateJurisdiction*

Inputs: jurisdictions: collection(Jurisdiction)
Output: aggregatedJurisdiction: Jurisdiction

```
begin
// create and initialize the default attributes of the aggregatedJurisdiction
aggregatedJurisdiction = new Jurisdiction
aggregatedJurisdiction = Jurisdictions.first().copy() //initialises result with first Jurisdiction

// handle the collection of Jurisdictions
for each jur: Jurisdiction in jurisdictions do
    // compute the intersection of the lists of the countries
    aggregatedJurisdiction.countries.intersection(jur.countries)

    for each aggregatedCountry:Country in aggregatedJurisdiction.countries do
        // find the country in the input collection of jurisdictions with the same code as that of aggregatedCountry
        c:Country = find ac:Country in jur.countries where c.code == ac.code

        if (aggregatedCountry.territories.notEmpty() and c.territories.notEmpty())
        then
            // there are territories in both jurisdictions. Take the intersection.
            aggregatedCountry.territories.intersection(c.territories)
            if (aggregatedCountry.territories.isEmpty())
            then
                // no territories resulting from intersection. Remove the country.
                aggregatedJurisdiction.countries.remove(aggregatedCountry)
            end if
        else
            // if one or both countries has no territory, then the jurisdiction is the most local one
            aggregatedCountry.territories.union(c.territories)
        end if
    end for
end for
return aggregatedJurisdiction
end
```

Example of Jurisdiction Aggregation

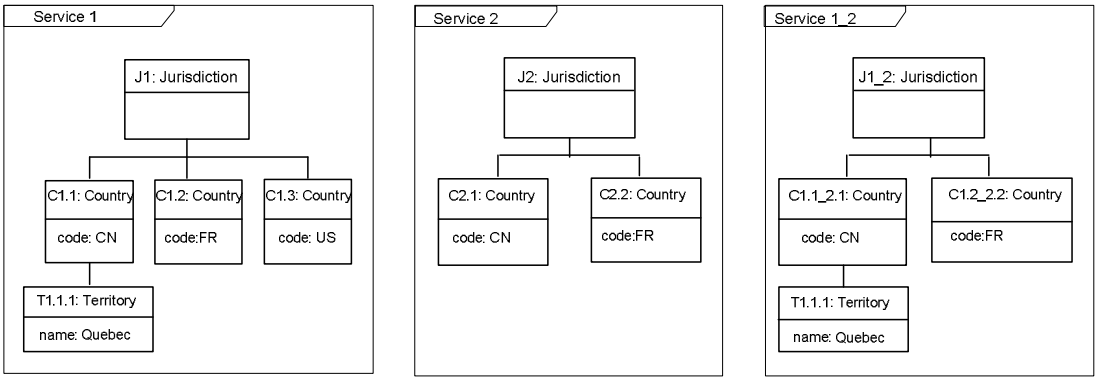


Figure 38 Example of Jurisdiction Aggregation

As shown in Figure 38, the Jurisdiction descriptions of Service 1 and Service 2 have been defined. Then, the Jurisdiction aggregation for the resulting composed service, Service 1_2, is computed. In this example, the composition could any operator.

First, the algorithm creates the Jurisdiction object Service 1_2, which is initialized with the attributes of Service 1. The initial description of Service 1_2 is the same as Service 1, which is as follow: {(code = CN, Territory = [Quebec]); (code = FR); (code = US)}. Then, the algorithm computes the intersection of the countries' codes between Service 1_2 and Service 2 by checking whether each code in the country list of Service 1_2 is part of Service 2's country codes or not. If one country of Service 1_2 does not exist in Service 2's countries, then it is removed from the list Service 1_2. If the country exists, then the algorithm runs through the sub-country-code to compute the intersection of the lists of territories, taking in this way the most local (and common) locations.

In this example, based on the attributes of Service 1 and Service 2, the aggregated Jurisdiction of Service1_2 is as follow: {(code = CN, Territory = [Quebec]); (code = FR)}. The rest of the CN country and the entire US country are no longer included.

5.2.14 Service Versioning

Service versioning of the composite service should be assessed on its own. The service provider should decide if it is appropriate to keep the same interface and backward compatibility regardless of the characteristics of the underlying services of the composite ser-

vice. The removed functionalities, added functionalities and fixed bugs of the underlying services should be summarized explicitly to the service consumers when relevant. This type of aggregation cannot be computed automatically.

5.2.15 Resource Requirements

This subsection proposes an aggregation algorithm (Algorithm XI) for the Resource Requirements NFPs. Ideally, all categories of resources would have well-defined units, which would enable comparisons and summations during aggregation. If precise, restricted units and perfect comparison operators were given, then the following could have been done (assuming that different units for a same type of resource can be normalized):

- Sequence and Loop: compute the sum of storage and timing resources (which are consumed for good at each invocation), as well as the maximum values for the remaining categories of resources (synch, computing, concurrency, device, and communication resources, which are assumed to be released after each underlying service invocation);
- Conditional Branching: compute the maximum values for each category of resources, using a pessimistic approach; and
- Concurrency and Discriminator: compute the sum for each category of resources, except for timing resources, for which the maximum is computed.

However, flexibility and extensibility for resource descriptions are targeted here, as well as alignment with OMG's MARTE profile, where unit definitions are so flexible that they compromise the ability to compare resources. It is believed that storage and timing resources can each have comparable units in general, but the other categories do not at this point. Therefore, the proposed trade-off is an algorithm that covers the aggregation of Resource Requirements for all the composition operators, where:

- Sequence and Loop: compute the sum of storage and timing resources, but compute the simple collection (in a bag sense, i.e., where duplicates are allowed) of values for each of the remaining categories of resources. Further comparisons and simplifications are left to the service consumer;

- Conditional Branching: compute the maximum values for storage and timing resources, but compute the simple collection (in a bag sense) of values for each of the remaining categories of resources; and
- Concurrency and Discriminator: compute the maximum for timing resources, the sum for the storage resources, and the simple collection (in a bag sense) of values for each of the remaining categories of resources. Algorithm XI makes use of *ConvertResUnits*, discussed briefly in Algorithm XII.

Algorithms for Resources Requirements Aggregation

Algorithm XI: AggregateResources

Inputs: resourcesPlus: collection (Resource)
composition: CompositionType
Output: aggregatedResource: Resource

```
begin
// create and initialize the default attributes of the aggregatedResources
aggregatedResource = new Resource

// handle the collection of Resources
for each res: Resource in resourcesPlus do
   // handle resource requirements for the next Resource
   for each req:ResourceReq in res.resourceReq do
      if req.category in {StorageResource, TimingResource}
      then
            // find the output resourceReq with a type matching resource category, if any
            aggregateReq:ResourceReq = find aReq:ResourceReq in aggregatedResource.resourcesReq
                                       where aReq.category == req.category
            if (aggregatedReq == null)
            then
                // first instance of timing/storage requirement encountered. Add as is, whatever the operator.
                aggregatedResource.add(req)
            else
                // take the composition operator into consideration for timing/storage requirement
                // note: only the description of the original requirement is kept, the other is dropped.
                if composition in {Sequence, Loop}
                then
                     aggregateReq.quantity = aggregateReq.quantity
                                             + ConvertResUnits(req.quantity, req.unit, aggregateReq.unit)
                elseif composition in {ConditionalBranching}
                then
                     aggregateReq.quantity =
                          max (aggregateReq.quantity, ConvertResUnits(req.quantity, req.unit, aggregateReq.unit))
```

```
                else // Concurrency or Discriminator
                    if aggregateReq.category == TimingResource
                    then
                        aggregateReq.quantity =
                            max (aggregateReq.quantity, ConvertResUnits(req.quantity, req.unit, aggregateReq.unit))
                    else
                        aggregateReq.quantity = aggregateReq.quantity
                                                + ConvertResUnits(req.quantity, req.unit, aggregateReq.unit)
                    end if
                end if
            end if
        else
            // simply add this requirement to the collection, as is. Applies to all categories of resources except
            // timing/storage, for any type of composition.
            aggregatedResource.add(req)
        end if
    end for
end for
return aggregatedResource
end
```

Algorithm XII converts a value of a given resource timing/storage unit to a value in a requested timing/storage unit. This algorithm is assumed to exist and to be tolerant of various unit description styles. It is not explained further here.

Algorithm XII: *ConvertResUnits*

Inputs: srcQuantity: Real
srcUnit:String
convertedUnit: String
Output: convertedQuantity: Real

Example of Resources Requirements Aggregation

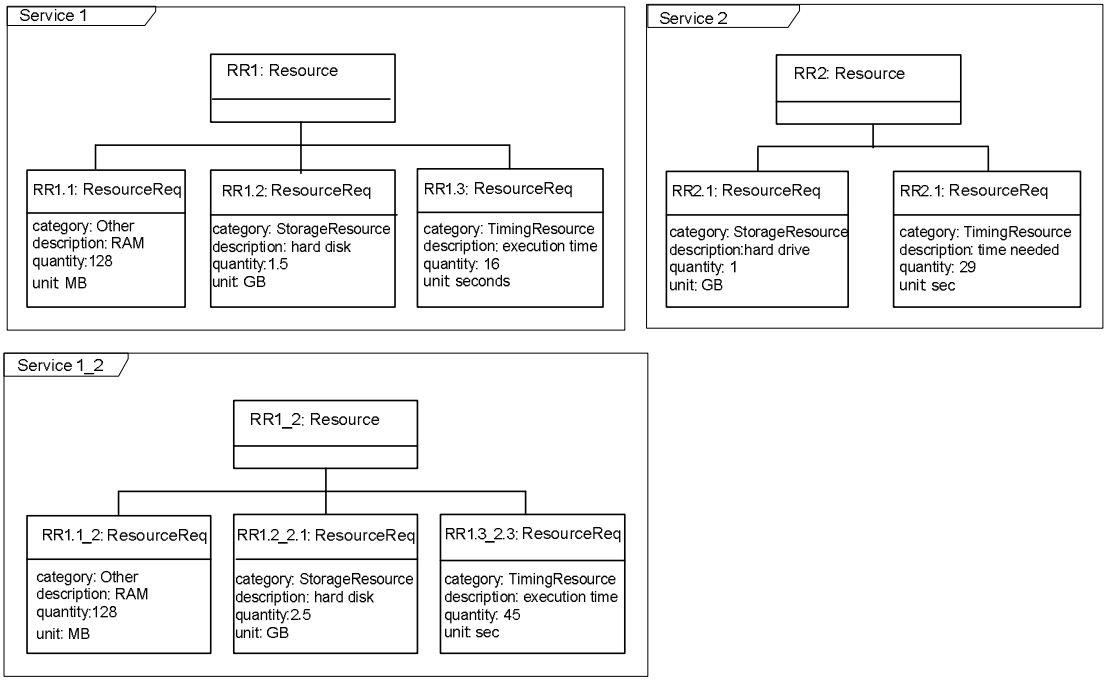


Figure 39 Example of Resources Requirements Aggregation

As shown in Figure 39, the Resources Requirements descriptions of two services, Service 1 and Service 2, have been defined. Then, the Resources Requirements aggregation for the resulting composed service, Service 1_2, is computed. In this example, the composition can be either Sequence or Loop.

The ResourceReq objects RR1.2_2.1 and RR1.3_2.3 are computed as the sum of the storage and timing resources respectively. The description attribute is not taken into consideration here (and the first one encountered us used for the aggregate). Also, “sec” and “seconds” are assumed to be understood by the *ConvertResUnits* algorithm (which is tolerant to such different unit styles). The ResourceReq object RR1.1_2 is added as a simple collection of the remaining categories of resources.

5.2.16 Scalability

This subsection proposes an Algorithm XIII for aggregating Scalability NFPs. The aggregated scalability is computed as the minimum value of all the scalability values of the

invoked services (each of which represent the maximum number of concurrent service invocations possible). This algorithm covers the aggregation of Scalability for all composition operators.

Algorithms for Scalability Aggregation

Algorithm XIII: AggregateScalability

Inputs: scalabilities: collection(Scalability)
Output: aggregatedScalability: Scalability

```
begin
// create and initialize the default attributes of the aggregateScalability
aggregatedScalability = new Scalability
// initialize "aggregatedScalability.scaleTo" to the first value that is part of the collection of Scalabilities
aggregatedScalability.scaleTo = Scalabilities.first().scaleTo

// handle the collection of scalabilities
for each sca: Scalability in scalabilities do
    // compute the min
    aggregatedScalability.scaleTo = min (aggregatedScalability.scaleTo, sca.scaleTo)
end for
return aggregatedScalability
end
```

Example of Scalability Aggregation

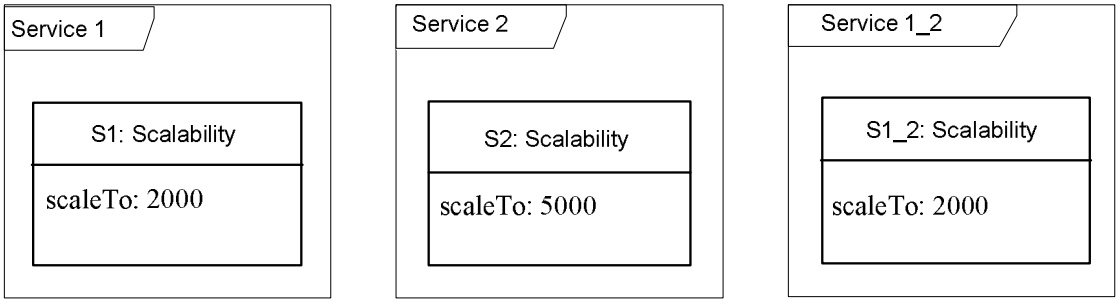


Figure 40 Example of Scalability Aggregation

As shown in Figure 40, the Scalability descriptions of Service 1 and Service 2 have been defined. Then, the Scalability aggregation for the resulting composed service, Service 1_2, is computed. In this example, the composition could any operator.

First, the algorithm creates the Scalability object S1_2, with the attribute scaleTo equal to the attribute scaleTo of Service 1, which means 2000 in this example. Then, all scaleTo attributes of all involved services are compared one after the other. The aggre-

gate value of scaleTo is computed as the minimum of scaleTo attributes of Service 1 and Service 2, which is equal here to 2000.

5.2.17 Server Location

This subsection proposes an aggregation algorithm (Algorithm XIV) for Server Location NFPs. The *union* of all the lists of countries and their territories is computed in order to return the aggregation of all server locations in all the services (independently of the composition operator): $\bigcup_{i=1}^{n} ServersLocation_i$, where $ServerLocation_i$ is the location of server i. The same algorithm applies for Sequence, Concurrency, Conditional branching, Discriminator and even Loop (the latter could also be simplified as $ServerLocation_i$ because the server's location remains the same for all $ServersLocation_i$).

Algorithm for Server Location Aggregation

Algorithm XIV: ***AggregateServersLocation***

Inputs: locations: collection(ServersLocation)
Output: aggregatedLocation: ServersLocation

```
begin
// create and initialize the default attributes of the aggregatedLocation
aggregatedLocation = new ServersLocation

// handle the collection of server locations
for each loc: ServersLocation in locations do
    // compute the union of the lists
    for each c:Country in loc.countries do
       // find the country in the aggregated list of locations with the same code as that of c, if any
       aggregateCountry:Country = find ac:Country in aggregatedLocation.countries where c.code == ac.code

       if aggregateCountry == null
           then
               // add the new server location country and its target territories
               aggregatedLocation.countries.union(c)
           else
               // add the new server location territories to the matched country
               if (aggregatedCountry.territories.notEmpty() and c.territories.notEmpty())
                  then
                      aggregatedCountry.territories.union(c.territories)
                  else
                      // if one or both countries have no territory, then the servers can be anywhere in the country
                      aggregatedCountry.territories = { } // remove all territory restrictions, if any
               end if
       end if
    end for
end for
return aggregatedLocation
end
```

Example of Server Location Aggregation

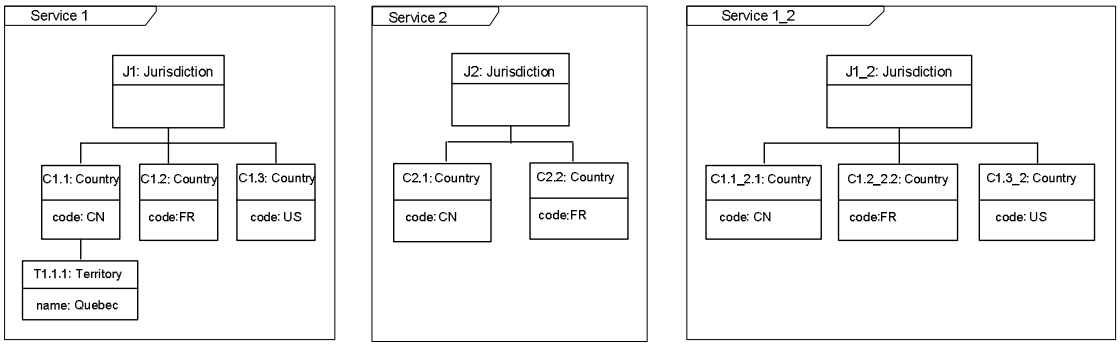


Figure 41 Example of Server Location Aggregation

As shown in Figure 41, the Server Location descriptions of Service 1 and Service 2 have been defined. Then, the Server Location aggregation for the resulting composed service, Service 1_2, is computed. In this example, the composition could any be any operator.

First, the algorithm creates the Service 1_2 object (of type ServersLocation). Then, the algorithm computes the union of the country codes and territories between Service 1 and Service 2. If one or both countries have no territory, then the servers can be anywhere in the country and the algorithm removes all territories (as they are restrictions).

In this example, based on the attributes of Service 1 and Service 2, the aggregated Server Location of Service 1_2 is as follows: {(code = CN); (code = FR); (code = US)}. The Quebec territory restriction for country CN found in Service 1 has disappeared because the CN country in Service 2 has no restriction.

5.3. Case Study: BookItWell Composite Service

In this case study, based on the composite service BookItWell introduced in section 1.2 and defined in this section by Figure 42, first a service description for each underlying services of the BookItWell composite service is provided. Then, the NFP description of the composed service BookItWell are computed by aggregating the NFPs of its underlying services using the aggregation algorithms described in the previous section. Note that

to illustrate the proposed aggregation methodology, the focus will only be on the 11 NFPs for which the aggregation algorithms have been provided.

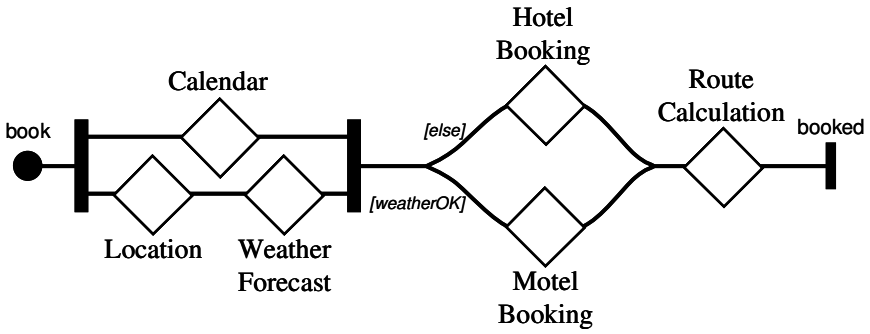


Figure 42 BookItWell Composite Service

5.3.1 BookItwell Underlying Services' Descriptions

The BookItWell service is built out of six sub-services, composed as modeled by the UCM diagram of Figure 42. The *Calendar* is invoked in parallel with the *Location* followed by the *Weather Forecast*. Then, the *Motel Booking* is performed if the weather is fine, else the *Hotel Booking* is invoked. Finally, in both cases, the *Route Calculation* service is invoked. The NFPs of each of these six services are provided here, using the data structures formalized in Chapter 4.

Service 1: Calendar Service Description

Price = {unit = USD; priceElement (type = Invocation, value = 0.0)}.
ResponseTime = {unit = microseconds; average = 2; best = 1; worst = 3; confidence = 90}
Reputation = {reputationLevel = 2.3}
Availability = {availabilityLevel = 0.90}
Reliability = {reliabilityLevel = 0.95}
Usability = {usabilityLevel = 2.1}
Transactional service = {isTransactional = False}
Jurisdiction = {(code = CN, Territory = [Quebec]); (code = FR)}
Resource Requirements = {(category = StorageResource; description = hard disk; quantity = 0.5; unit = GB)}
Scalability = {scaleTo = 500}
Server Location = {(code = CN, Territory = [Quebec])}

Service 2: Location Service Description

Price = {unit = USD; priceElement (type = SubscribeMonth, value = 5.0)}.
ResponseTime = {unit = microseconds; average = 2; best = 1; worst = 4; confidence = 95}
Reputation = {reputationLevel = 3.2}

Availability = {availabilityLevel = 0.98}
Reliability = {reliabilityLevel = 0.96}
Usability = {usabilityLevel = 3.5}
Transactional service = {isTransactional = False}
Jurisdiction = {(code = US); (code = CN) ; (code = FR), (code = ES)}
Resource Requirements = {(category = CommunicationResource; description = minimum speed for the connection; quantity = 1; unit = mbps)}
Scalability = {scaleTo = 10000}
Server Location = {(code = US); (code = CN) ; (code = ES)}

Service 3: Weather Forecast Service Description

Price = {unit = USD ; priceElement (type = Invocation,value = 0.0)}.
ResponseTime = {unit = microseconds; average = 5; best = 2; worst = 9; confidence = 80}
Reputation = {reputationLevel = 2.4}
Availability = {availabilityLevel = 0.93}
Reliability = {reliabilityLevel = 0.97}
Usability = {usabilityLevel = 2.8}
Transactional service = {isTransactional = False}
Jurisdiction = {(code = FR); (code = CN, Territory = [Quebec, Ontario]) ; (code = ES)}
Resource Requirements = {}
Scalability = {scaleTo = 10000}
Server Location = {(code = CN); (code = FR) ; (code = US)}

Service 4: Hotel Booking Service Description

Price = {unit = EUR ; priceElement (type = Invocation, value = 5.0) }.
ResponseTime = {unit = microseconds; average = 4; best = 2; worst = 7; confidence = 98}
Reputation = {reputationLevel = 4.2}
Availability = {availabilityLevel = 0.97}
Reliability = {reliabilityLevel = 0.98}
Usability = {usabilityLevel = 4.6}
Transactional service = {isTransactional = True}
Jurisdiction = {(code = FR); (code = US); (code = CN)}
Resource Requirements = {(category = Other; description = international payement card; quantity = ; unit =)}
Scalability = {scaleTo = 5000}
Server Location = {(code = FR); (code = US) ; (code = CN)}

Service 5: Motel Booking Service Description

Price = {unit = EUR; priceElement (type = Invocation, value = 3.0)}.
ResponseTime = {unit = microseconds; average = 3; best = 2; worst = 5; confidence = 95}
Reputation = {reputationLevel = 3.8}
Availability = {availabilityLevel = 0.92}
Reliability = {reliabilityLevel = 0.89}
Usability = {usabilityLevel = 4.1}
Transactional service = {isTransactional = True}
Jurisdiction = {(code = FR); (code = US); (code = CN)}
Resource Requirements = {}

Scalability = {scaleTo = 2500}
Server Location = {(code = FR); (code = CN)}

Service 6: Route Calculation Service Description

Price = {unit = USD; priceElement (type = SubscribeMonth, value = 2.0)}.
ResponseTime = {unit = microseconds; average = 3; best = 1; worst = 6; confidence = 95}
Reputation = {reputationLevel = 4.3}
Availability = {availabilityLevel = 0.87}
Reliability = {reliabilityLevel = 0.85}
Usability = {usabilityLevel = 4.7}
Transactional service = {isTransactional = True}
Jurisdiction = {(code = US); (code = CN); (code =ES)}
Resource Requirements = {(category = StorageResource; description = hard disk; quantity = 0.2;
unit = MB)}
Scalability = {scaleTo = 1200}
Server Location = {(code = US); (code = CN); (code =ES)}

5.3.2 BookItwell NFPs Aggregation

Similar to the reliability of electronic systems represented by block diagrams, it is not necessary to understand the functional details of a block or a service to manipulate a block diagram. In the current context, blocks represent underlying services, whereas lines describe the composition of the services. Following a set of transformations, NFPs of composite service can be derived by combining (i.e., reducing) services. The reduction is done by replacing several services by one combined service. With the aggregation algorithms provided in this chapter, one can recursively reduce any set of services composed using *one* out of the five composition operators, until getting the composite service reduced to one service (e.g., the termination condition of this recursive process). Although one specific reduction is presented in this example, with a specific ordering, reductions could be done using different order to get the final to a single service. One can arrive at different service reduction ordering scenarios depending on which services are chosen first to be combined. For example, in the following scenario, STEP 1 and STEP 2 could be done at the same time or in any order since they are independent.

STEP 1: Aggregation of Location and Weather Forecast NFPs

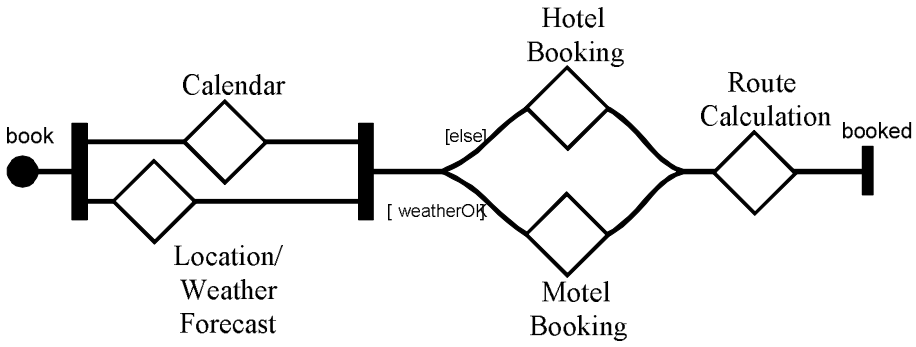


Figure 43 Combining Location and Weather Forecast Services

These two services (service 2 and service 3) are composed in sequence, leading to one new aggregate service, as shown in Figure 43.

Price = {unit = USD; priceElement (type = SubscribeMonth, value = 5.0)}.
ResponseTime = {unit = microseconds; average = 7; best = 3; worst = 13; confidence = 80}
Reputation = {reputationLevel = 2.4}
Availability = {availabilityLevel = 0.9114}
Reliability = {reliabilityLevel = 0.9312}
Usability = {usabilityLevel = 2.8}
Transactional service = {isTransactional = False}
Jurisdiction = {(code = CN, Territory = [Quebec, Ontario]) ; (code = FR), (code = ES)}
Resource Requirements = {(category = CommunicationResource; description = minimum speed for the connection; quantity = 1; unit = mbps)}
Scalability = {scaleTo = 10000}
Server Location = {(code = US); (code = CN) ; (code = ES); (code = FR)}

STEP 2: Aggregation of Hotel Booking and Motel Booking NFPs

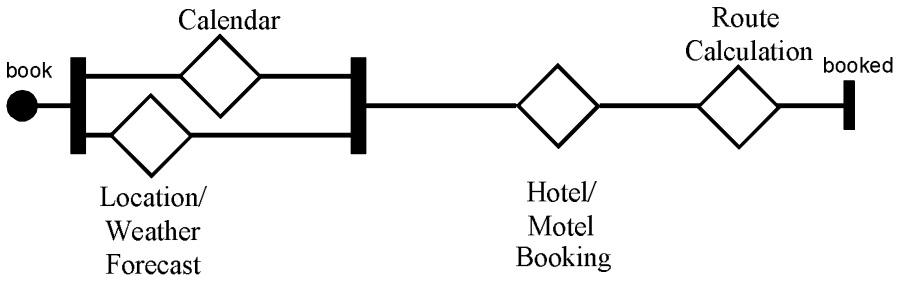


Figure 44 Combining Hotel Booking and Motel Booking Services

These two services (service 4 and service 5) are composed into one service (Figure 44) using Conditional Branching.

Price = {unit = EUR; priceElement (type = Invocation, value = 5.0)}.
ResponseTime = {unit = microseconds; average = 3.5; best = 2; worst = 7; confidence = 95}
Reputation = {reputationLevel = 3.8}
Availability = {availabilityLevel = 0.92}
Reliability = {reliabilityLevel = 0.89}
Usability = {usabilityLevel = 4.1}
Transactional service = {isTransactional = True}
Jurisdiction = {(code = FR); (code = US); (code = CN)}
Resource Requirements = {(category = Other; description = international payement card; quantity = ; unit =)}
Scalability = {scaleTo = 2500}
Server Location = {(code = FR); (code = US) ; (code = CN)}

Step 3: Aggregation of Location/Weather Forecast and Calendar NFPs

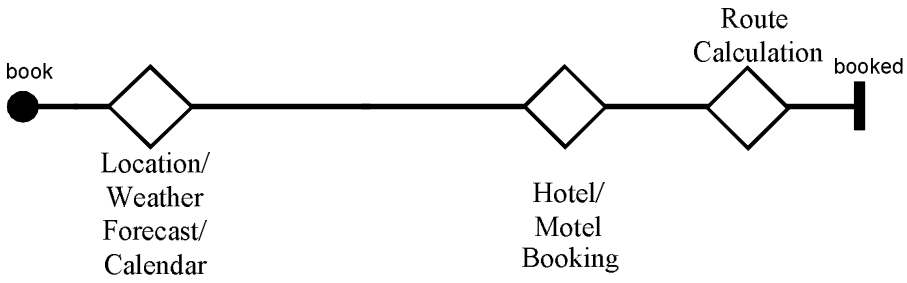


Figure 45 Combining Location/Weather Forecast and Calendar Services

These two services (Location/Weather Forecast service and service 1) are composed using the concurrency operator, see Figure 45.

Price = {unit = USD; priceElement (type = SubscribeMonth, value = 5.0)}.
ResponseTime = {unit = microseconds; average = 7; best = 3; worst = 13; confidence = 80}
Reputation = {reputationLevel = 2.3}
Availability = {availabilityLevel = 0.82026}
Reliability = {reliabilityLevel = 0.88464}
Usability = {usabilityLevel = 2.1}
Transactional service = {isTransactional = False}
Jurisdiction = {(code = CN, Territory = [Quebec]) ; (code = FR)}
Resource Requirements = {(category = StorageResource; description = hard disk; quantity = 0.5; unit = GB), (category = CommunicationResource; description = minimum speed for the connection; quantity = 1; unit = mbps)}
Scalability = {scaleTo = 500}
Server Location = {(code = US); (code = CN) ; (code = ES); (code = FR)}

STEP 4: Aggregation of all the remaining services

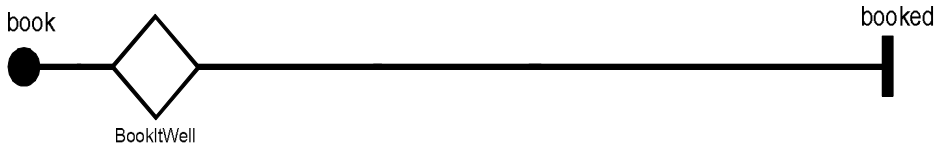


Figure 46 Combining All the Resulting Services

The three services left in Figure 45 can now be composed into one, hence providing the resulting NFPs for BookItWell as a whole (Figure 46).

Note. Unit = USD, and a conversion rate of 1 EUR = 1.5 USD is assumed.

The aggregation algorithms are capable of combining more than two services using a given CompositionType. In this scenario, where the CompositionType is sequential composition, the aggregation algorithms are used to reduce the three remaining service descriptions directly into a single service description (provided at the end of this section).

This is also equivalent to reducing the services two by two. For instance, one can first aggregate Location/Weather Forecast/Calendar and Hotel/Motel Booking:

Price = {unit = USD; priceElement (type = SubscribeMonth, value = 5.0),
priceElement (type = Invocation, value = 7.5)}.
ResponseTime = {unit = microseconds; average = 10.5; best = 5; worst = 20; confidence = 80}
Reputation = {reputationLevel = 2.3}
Availability = {availabilityLevel = 0.7546392}
Reliability = {reliabilityLevel = 0.7873296}
Usability = {usabilityLevel = 2.1}
Transactional service = {isTransactional = False}
Jurisdiction = {(code = CN, Territory = [Quebec]) ; (code = FR)}
Resource Requirements = {(category = Other; description = international payement card; quantity = ;
unit =), (category = StorageResource; description = hard disk; quantity = 0.5;
unit = GB), (category = CommunicationResource; description = minimum
speed for the connection; quantity = 1; unit = mbps)}
Scalability = {scaleTo = 500}
Server Location = {(code = US); (code = CN) ; (code = ES); (code = FR)}

Then, the two remaining services, i.e., Location/Weather Forecast/Calendar/Hotel/Motel Booking service and Route Calculation service, can be aggregated, with the same final result.

Price = {unit = USD; priceElement (type = SubscribeMonth, value = 7.0),
priceElement (type = Invocation, value = 7.5)}.
ResponseTime = {unit = microseconds; average = 13.5; best = 6; worst = 26; confidence = 80}
Reputation = {reputationLevel = 2.3}
Availability = {availabilityLevel = 0.656536104}
Reliability = {reliabilityLevel = 0.66923016}
Usability = {usabilityLevel = 2.1}
Transactional service = {isTransactional = False}
Jurisdiction = {(code = CN, Territory = [Quebec])}
Resource Requirements = {(category = Other; description = international payement card; quantity = ;
unit =), (category = StorageResource; description = hard disk; quantity = 0.7;
unit = GB), (category = CommunicationResource; description = minimum
speed for the connection; quantity = 1; unit = mbps)}
Scalability = {scaleTo = 500}
Server Location = {(code = US); (code = CN) ; (code = ES); (code = FR)}

5.4. Chapter Summary

This chapter has presented five important service composition operators, including a discriminator operator that enables redundancy, which is an original contribution of this work. Aggregation functions were investigated for all seventeen NFPs defined in the catalogue in Chapter 4. Aggregation algorithms for eleven out of the seventeen NFPs were defined for all five composition operators, and they were demonstrated in the context of the ongoing BookItWell case study introduced in Chapter 1. For the remaining six NFPs (certification, accuracy, standards compliance, failure modes, security, and service versioning), aggregation is not possible without further restrictions of the data structures or human intervention, and the rationale for requiring this was provided for each of these cases.

The aggregation algorithms developed here take into consideration the different composition operators including conditional branching and parallel branching. These algorithms provide proper treatment of all composition operators. In addition, the discriminator operator is taken into account in the developed aggregation functions. This goes well beyond what was surveyed in the literature review.

The next chapter will utilize NFP definitions for enabling service comparison and selection. The BookItWell example will be used as a case study.

Chapter 6. Non-Functional Properties-based Service Selection

In this chapter, a modeling approach for NFP-based service selection and composition is presented. The modeling and analysis of NFPs for SOA are undertaken from three points of view, which can be used as standalone or preferably in combination:

a. The use of standard URN for modeling service goals and functions in an integrated way, hence enabling the analysis of NFPs in context.
b. The use of aspect-oriented extensions to URN (AoURN) to better modularize in individual units called *concerns*, different NFPs that can crosscut services or service components.
c. The use of new NFP-oriented annotations for service composition. These scenario annotations, based on a subset of the catalogue introduced in Chapter 4, enable the specification of quantitative NFP values for services and the computation of the NFP characteristics (e.g., the quality of service) of their composition.

This chapter introduces step-by-step the proposed approach to service selection and service composition based on the Aspect-oriented User Requirements Notation (AoURN, see section 2.2.3) and the framework for measuring and composing various non-functional properties (NFPs) as introduced in Chapter 5. This approach consists of nine steps, each described in a different section. One particularity here is that multiple services or multiple instances of the same service can be selected to best accommodate the user's constraints. This approach is illustrated using a variant of the composite service BookIt-Well introduced in Chapter 1 and used in section 5.3.

Note that the use of AoURN is not mandatory to enable NFP-based service selection. A system could simply compare the NFP description required by a service consumer

against the NFP descriptions of service providers, and select the most appropriate option, if any meets the NFP requirements [12]. However, as is shown in this chapter, the use of AoURN permits to go one step beyond by reasoning about NFP tradeoffs dynamically. In addition, AoURN could be used as a basis for a workflow engine that supports dynamic selection and adaptation [59].

6.1. Model Composite Service with UCM (Step 1)

A composite service is modeled based on its required abstract services with the Use Case Map (UCM) notation. Abstract services are represented as static stubs (diamonds) that are composed with each other using the UCM-equivalents of the composition operators defined earlier. As shown in Figure 42, the BookItWell composite service is built by combining six services provided independently by different companies: Location, Weather Forecasting, Calendar, Hotel Booking, Motel Booking, and Route Calculation.

First, the location of the user is determined with the help of a Location service. Once the location is known, a Weather Forecast service is invoked to obtain today's forecast. In parallel, the Calendar service of the user is queried to retrieve location and time information of planned activities for the current date and the following day. Based on the weather forecast and the schedules, BookItWell will either book a more expensive hotel or a cheap motel. The Hotel Booking service is used if the weather does not allow scheduling of planned outdoor activities. The Motel Booking service is used if the weather permits scheduling of planned outdoor activities. In both cases, the locations of the planned activities are taken into account when booking accommodation. Finally, the Route Calculation service provides directions to the planned activities and the chosen accommodation.

At this point, the modeler may consider adding additional functionality to the composite service, e.g., for integrity checks before and after the booking services, for billing functionality, for after-the-fact auditing features, and for redundancy in case of potentially unavailable services. These services could be added explicitly to the URN model in the same way as abstract services in Figure 42, but their crosscutting nature

makes them perfect candidates for aspect-oriented modeling with the help of Aspect-oriented UCM (AoUCM) as described in the next step.

6.2. Model Crosscutting Concerns with AoUCM (Step 2)

In this step, each identified crosscutting concern is modeled using the Aspect-oriented UCM (AoUCM) notation. In the case of the BookItWell example, the required crosscutting concerns are Billing, Integrity, Audit, and Redundancy, as shown in Figure 47. These crosscutting concerns are modeled in a generic way and are therefore applicable to many different composite services. The *pointcut stubs* (the diamonds with the **P** inside) represent in a generic way abstract services in a composite service that meet certain matching criteria. In this step, the matching criteria are customized to BookItWell, thus applying the crosscutting concerns to the selected example.

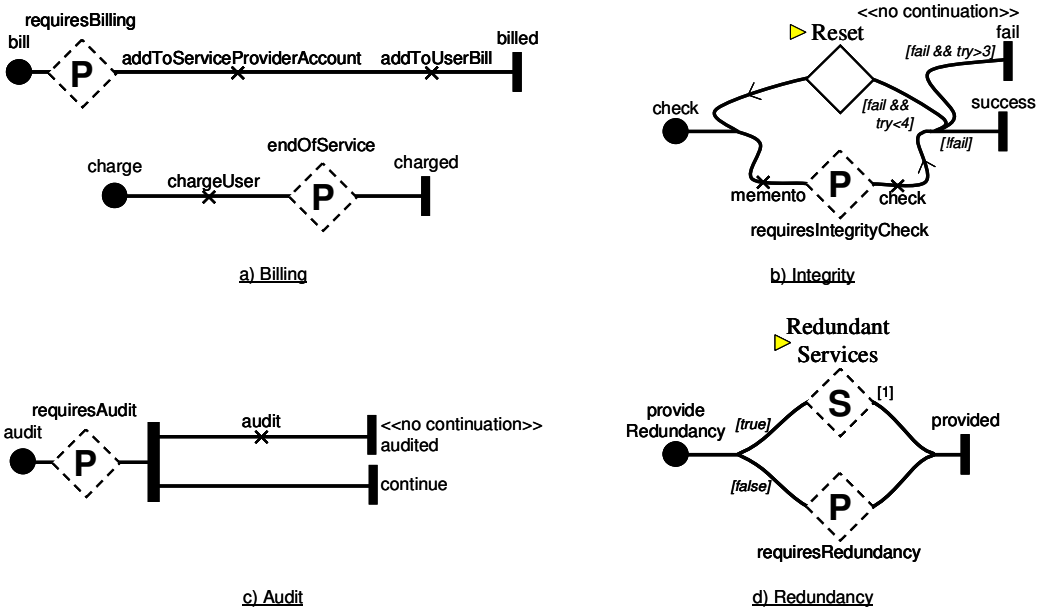


Figure 47 Crosscutting Concerns Relevant for the BookItWell Composite Service

The *Billing* aspect (Figure 47a) adds functionality after a service requiring billing. The fee for the service charged by the service provider is added to the service provider's account and the same amount is added to the bill of the service user. In addition, the Billing

aspect adds further functionality by, before the end of the composite service, charging the user the amount added to the bill (plus possibly a margin).

The *Integrity* aspect (Figure 47b) takes a snapshot of the system state (memento) before a service requiring an integrity check and then compares the snapshot against the system state (check) after the service completed. If the check fails, the matched service is reset before retrying at the most three times (note how the matched service is reused by the Integrity aspect with the help of a URN link indicated by the small shaded triangle). The loop counter is initially zero and is incremented by one by the memento responsibility for each attempt. After the third unsuccessful attempt, the service fails and the composite service stops at the end point as indicated by the «no continuation» stereotype.

The *Audit* aspect (Figure 47c) starts the Audit service in parallel to the composite service and after the service requiring an audit. The Audit service is an internal process of the DriveALot company that aims to investigate selections of high-cost services after-the-fact to avoid them in the future. The end point of the branch with the Audit service is tagged as «no continuation», since the composite service does not continue at this end point but at the other end point. Note that an improved Audit aspect could verify upon completion of an individual service if the service provider violated its SLA.

Finally, the *Redundancy* aspect (Figure 47d) starts a number of redundant services as indicated by the discriminator stub (implemented as a standard UCM synchronization stub, shown with a **S** inside the diamond symbol) on the true branch instead of the service requiring redundancy (as it is located on the false branch). The true and false branches are AoUCM's way of modeling a replacement (i.e., an aspect replaces existing behavior with its own behavior). When the Redundancy aspect is applied, the false branch is often not shown as it will never be traversed. Similar to the Integrity aspect, the Redundancy aspect reuses the matched service with the help of the URN link to indicate that the matched abstract service needs to be composed in a redundant manner.

6.3. Model Pointcut Expressions for Concerns with AoUCM (Step 3)

The third step of the proposed approach requires pointcut expressions to be defined for each crosscutting concern applicable to the composite service under design. A pointcut expression makes a generic aspect specific to the composite service under design by describing the matching criteria as shown below in Figure 48.

For the BookItWell example, the pointcut expression for the requiresBilling pointcut stub of the Billing aspect matches against any abstract service except those named Reset (since * matches against any name). Therefore, after all services except one, the billing functionality defined by the Billing aspect is performed. The pointcut expression for the endOfService pointcut stub matches against the end point booked. Therefore, the user is charged just before the end of the composite service. The pointcut expression of the Integrity aspect matches against any abstract service with a name ending in Booking. Therefore, the preconditions and postconditions of the Hotel Booking and the Motel Booking services can be verified to ensure that correct amounts have been charged for the booking. The URN link ensures that the matched booking service is reset if the integrity check fails.

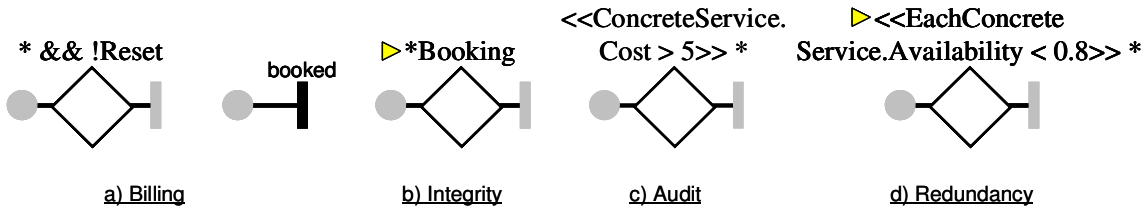


Figure 48 Pointcut Expressions for the Crosscutting Concerns

The pointcut expressions of the Audit and Redundancy aspects illustrate another dimension of the matching capabilities of AoUCM by using the NFPs of services for the match. The Audit aspect matches against any abstract service if the NFP Price of at least one of its concrete services is greater than five units. The Redundancy aspect, on the other hand, matches against any abstract service if the NFP availability for each of its concrete services is less than 0.8 (80%). The URN link ensures that the matched services are used in

the discriminator in the Redundancy aspect. These pointcut expressions are formulated with concrete services and their NFPs in mind, but at this point neither the actual concrete services nor their NFPs are captured by the UCM model. The next step takes care of this.

6.4. Populate Abstract Services with Concrete Services (Step 4)

In the proposed approach, the data structure of the concrete services has to comply with the one introduced in Chapter 4 (e.g., «Price= {unit = CAD; priceElement (type = Invocation, value = 3.0)}». However, the focus of this chapter is the approaches to be used for modeling, comparison and selection of concrete services. For simplicity it has been assumed that all the services in the registry have the same Price.unit = CAD and the same priceElement.type = Invocation. As well, all the ResponseTime.unit = milliseconds and the service providers have only populated the ResponseTime.worst value. It has also been assumed that the service consumer is interested only by 4 NFPs out of the 17 that have been proposed (e.g., Price, ResponseTime, Availability and Reliability). In addition, it has been assumed that the service consumer has made a business decision to invoke only services that will be paid for on a per Invocation basis (e.g., avoiding services that requires long term commitment or may reserve billing surprises when the pricing is based on some quantities). Stripping out some details related to data structures at this point makes it easier for readers to follow the proposed modeling approach. These assumptions will not change the logic of the service selection approach. Only some extra steps would be needed to normalize disparate data, but this is out of the scope of this chapter and has already been dealt with in previous chapters. Such normalization steps would include invoking web services for currency and time unit conversions.

Based on the description of services in registries, the abstract services in the UCM model can be augmented with the concrete offerings for each abstract service. As abstract services are modeled with stubs in the UCM model, the concrete services for an abstract service are modeled as plug-in maps for the stub. The example in Figure 49 shows that three concrete Location services were found in the registry. A stub with more than one plug-in map is called dynamic and is shown with a dashed outline. In addition, each plug-

in map (i.e., each concrete service) is annotated with the metrics of its NFPs as retrieved from the registry.

Each plug-in map has a condition that must evaluate to true for the plug-in map to be selected when its stub is reached in the composite service. By default, the condition of a plug-in map of a dynamic stub is set to false for all concrete services. The condition of a plug-in map for a static stub is set to true because only one plug-in map exists for a static stub (i.e., there is only one option for the service to be selected).

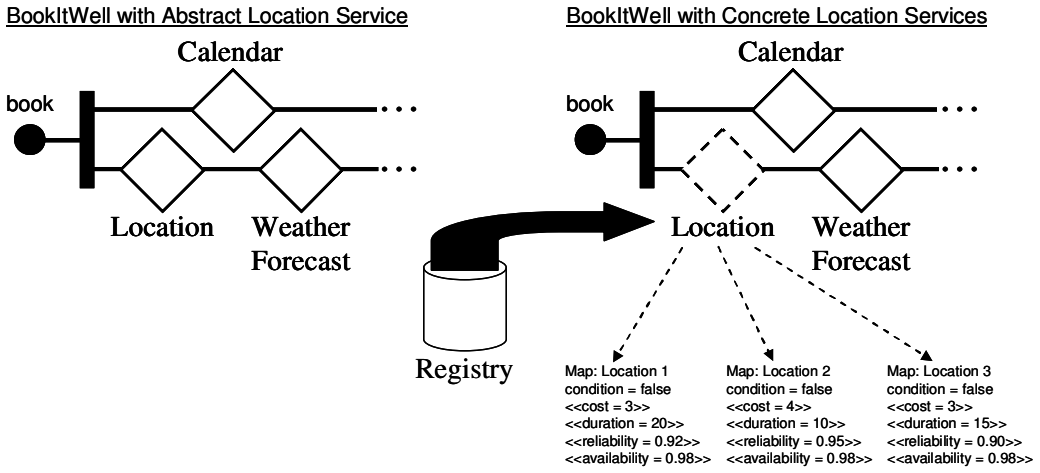


Figure 49 Concrete Services for the Abstract Location Service

Table 8 Summary of Concrete Services

Concrete Service	Price*	Response Time**	Reliability	Availability
Location L1	3	20	0.92	0.98
Location L2	4	10	0.95	0.98
Location L3	3	15	0.90	0.98
Calendar C1	3	10	0.90	0.98
Weather Forecast WF1	3	20	0.80	0.70
Weather Forecast WF2	3	20	0.90	0.60
Weather Forecast WF3	5	33	0.95	0.40
Weather Forecast WF4	6	18	0.95	0.20
Hotel Booking HB1	3	15	0.90	0.92
Hotel Booking HB2	2	10	0.90	0.90
Motel Booking MB1	2	13	0.95	0.90
Route Calculation RC1	3	10	0.90	0.95

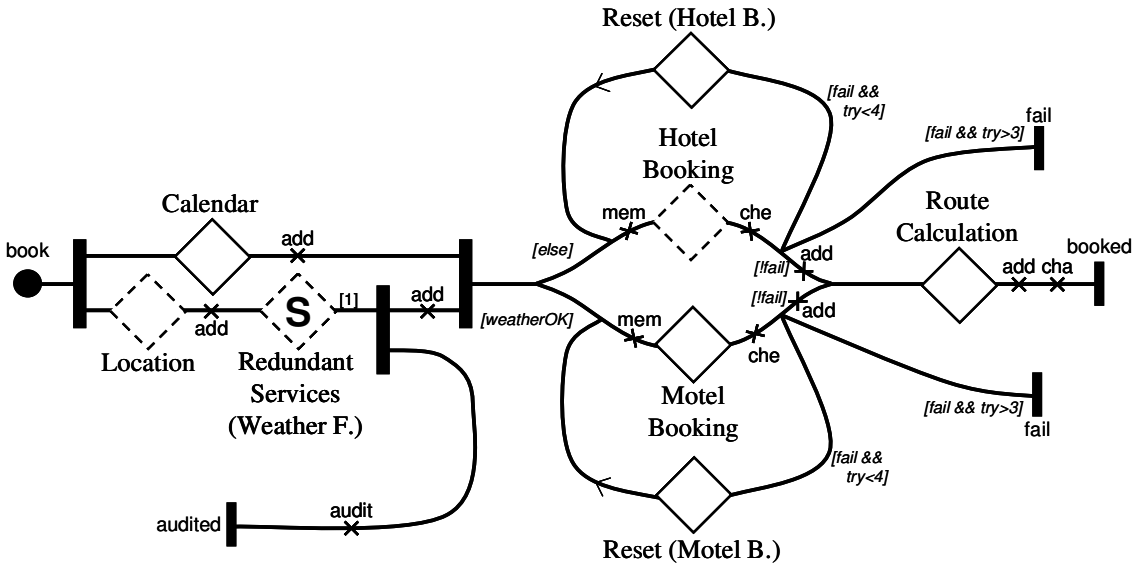
Note. * Price currency is CAD (or more realistically in cents) and the priceElement type is per Invocation

** Response Time is in the worst case, and the confidence is 100%

The concrete services of all abstract services of the BookItWell example are listed with their NFPs in Table 8. The question that remains at this point is: Which one of the concrete services to choose if there is a choice (e.g., it is not clear which one of the three location services is the best choice)? For this purpose, the NFPs of the service candidates need to be evaluated because the functionalities of the service candidates are equivalent. Before this evaluation is made, the aspects should be applied to the composite service as this may add further services and functionalities to the composite service or even cause restructuring of the composite service.

6.5. Apply Aspects to the Composite Service (Step 5)

The fifth step uses the aspect composition algorithm of AoURN to apply the aspects to the composite service given the pointcut expressions. As aspects may interact with each other in undesirable ways, an ordering of aspects defines which aspect is to be applied when it is to be applied. For the BookItWell example, the Redundancy aspect must be applied first and the Integrity aspect applied last. Therefore, the Weather Forecast stub was replaced by the Redundant Services stub in the resulting composite service (see Figure 50) because all concrete Weather Forecast services have an NFP availability of less than 0.8. Note that the Redundant Services stub now contains the same concrete services as the Weather Forecast stub. Billing functionality is included after each stub except when resetting a service (i.e., after Location, Calendar, Redundant Services, Hotel Booking, Motel Booking, and Route Calculation), and additional billing functionality is added just before the booked end point. The parallel Audit service is added after the Redundant Services stub because it contains the fourth concrete Weather Forecast service which has an NFP Price greater than five. Finally, the loop of the Integrity service is added around the Hotel Booking and Motel Booking stubs. Note that the Calendar, Motel Booking, and Route Calculation stubs are not dynamic because only one concrete service exists for each one of them (see Table 8).



Abbreviations: add ... both addTo... responsibilities from Billing; cha ... chargeUser; che ... check; mem ... memento

Figure 50 The BookItWell Composite Service with Applied Aspects

The composed system showcases the powerful aspect composition mechanism of AoUCM. For example, instead of adding the billing functionality explicitly after each service, a simple pointcut expression achieves the same effect but allows individual maps to remain simple. Furthermore, the UCM model can be visualized in an aspect-oriented way that does not require the complicated composed model in Figure 50. The composed model is now ready to be evaluated based on NFPs of concrete services.

6.6. Model NFPs with GRL (Step 6)

In the sixth step, a goal model is created using the Goal-oriented Requirement Language (GRL). The goal model describes the impact of NFP metrics on the high-level goals of the stakeholder of the composite service (see Figure 51), thereby facilitating trade-offs between high-level goals. Such models enable the service consumer to specify how best to choose among services that meet the minimal functional and NFP requirements.

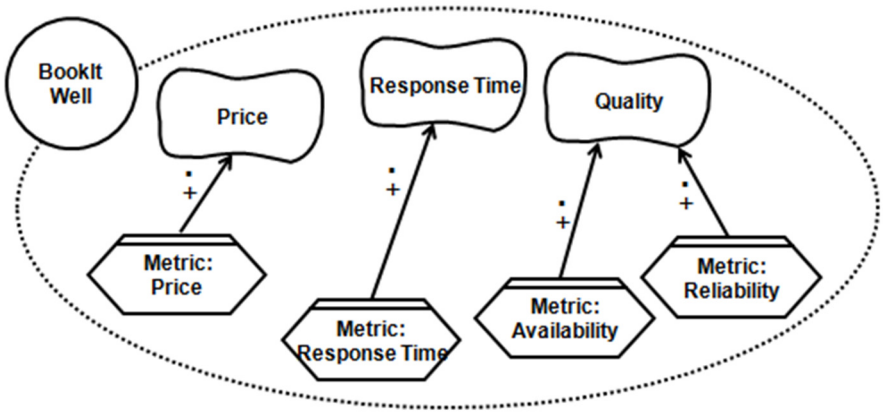


Figure 51 The GRL Model of NFPs

The BookItWell example is rather simplistic in this regard as it only shows the Price metric influencing the high-level Price goal, the response time metric influencing the high level response time goal, and both, reliability and availability metrics, influencing the high-level quality goal. Arguably, GRL graphs are not required for a simple model like this, and a textual syntax could also be used to express such models. However, besides the metrics based on the proposed framework for measuring and composing various NFPs, other factors that impact high-level goals may be modeled with GRL. To decide which particular service to choose, the BookItWell actor needs to be evaluated given the metrics for the particular service. This is discussed in the next step.

Each metric is modeled as a Key Performance Indicator (KPI) in the GRL model. KPIs allow metrics to be normalized to the scale ranging from -100 to 100 that is used by GRL. For each metric, a target value (the best possible) and a worst value must be defined. A threshold value, which lies somewhere between the target value and the worst value, must be selected and it is used as the 0 value in the satisfaction scale used by GRL. These values have to be established based on the requirements of the company providing the composite service. For the BookItWell example, the values in Table 9 were defined for the four chosen NFPs.

Table 9 KPI Target, Threshold, and Worst Case Values

NFP	Target	Threshold	Worst
Price	0	5	15
Response Time	1	20	35
Reliability	0.99999	0.9	0.8
Availability	0.99999	0.9	0.8

6.7. Evaluate NFPs of Concrete Services with GRL (Step 7)

In the seventh step, the goal model is evaluated with the GRL evaluation mechanism that propagates initial satisfaction values of the KPIs defined in a GRL strategy to satisfaction values of goals and eventually actors [5]. GRL strategies are created for each abstract service with multiple concrete service options. This enables the evaluation of each concrete service individually and then allows the comparison of concrete services. One GRL strategy is created per concrete service of a dynamic stub, while one GRL strategy is created per combination of concrete services of a synchronizing stub. Therefore in the BookItWell example: three GRL strategies are created, one for each of the three location services; two GRL strategies are created, one for each of the two hotel booking services; and 15 GRL strategies are created to cover all possible combinations of the four weather forecasting services. This is feasible since the number of services is not expected to be in the dozens. Another alternative would be to explore top-down propagation algorithms for GRL, which can search for an optimal strategy automatically based on constraint-oriented programming concepts [51], but this approach is still immature and left to future work.

For example, the GRL strategy for the first location service in the BookItWell example sets the satisfaction value for the KPI Price to 40 (given 3 for the NFP Price), for the KPI completion time to 0 (given 20 for the NFP completion time), for the KPI Reliability to 20 (given 0.92 for the NFP reliability), and for the KPI Availability to 80 (given 0.98 for the NFP availability) (see KPIs marked with * in Figure 52). For the strategies based on a combination of services as is the case for the weather forecast services, the NFPs of the individual services first have to be combined using the aggregation functions defined earlier – in this case, the functions for the discriminator. Given the satisfaction values for the KPIs, the GRL evaluation mechanism then calculates the values of the

goals and the actor (22 in the example). GRL also allows for an importance attribute to be specified for the high-level goals of an actor which influences the final evaluation of the actor. In the BookItWell example, however, all goals have the same importance.

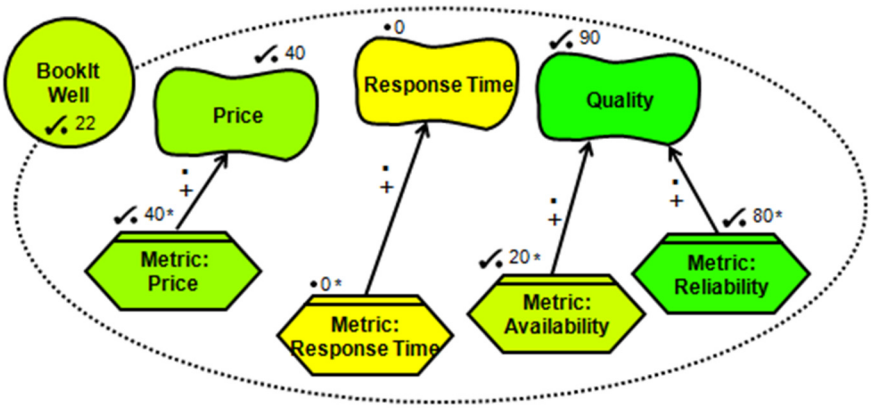


Figure 52 Evaluated GRL Model for the First Location Service

The results of the evaluation of all location services, all hotel booking services, and all combinations of weather forecasting services are shown in Table 10, with the best choice for each service highlighted. The services or combination of services with an X as the result are below the worst case for one of the NFPs and are therefore not even considered (e.g., the first weather forecasting service has an availability of 0.70 which is beyond the required worst value of 0.80; similarly, the combination of all four weather forecasting services has a Price of 3+3+5+6=17, which is beyond the required worst value of 15).

Table 10 Evaluation Results for Service Selection

Location 1	22	Weather F. 1	X	Weather F. 1/3	-18	Weather F. 1/2/3	**-9**
Location 2	27	Weather F. 2	X	Weather F. ¼	X	Weather F. 1/2/4	**3**
Location 3	24	Weather F. 3	X	Weather F. 2/3	X	Weather F. 1/3/4	**-20**
Hotel B. 1	14	Weather F. 4	X	Weather F. 2/4	X	Weather F. 2/3/4	**-28**
Hotel B. 2	**19**	**Weather F. ½**	**8**	**Weather F. 3/4**	**X**	**Weather F. 1/2/3/4**	**X**

6.8. Select Concrete Services (Step 8)

Step 8 uses a shortest path approach to select concrete services (i.e., each stub in the composite service is addressed individually one after the other). A static stub does not require any decision because only one concrete service exists in this case. The conditions of plug-in maps of dynamic and synchronizing stubs, however, need to be defined based on the results of the evaluation from the previous step (i.e. Step 7). Setting the condition of a plug-in map to true essentially selects the concrete service the plug-in map represents (e.g., Location 2, Hotel Booking 2, and Weather Forecast 1 and 2).

Note. Computing the "shortest path" to evaluate and select concrete services is proposed. It is recognized that using such an approach is only practical when the number of services is limited. When the number of services is very large, one should find other alternatives to reduce the size of the concrete equivalent services to be considered.

6.9. Calculate NFPs for the Composite Service (Step 9)

The final step of the proposed approach calculates the NFP metrics for the composite service from the selected individual services. Heuristics have to be employed to be able to parse the UCM model and obtain the aggregation function. Based on UCM slicing techniques presented in [28], the proposed approach suggested here is to start exploring from the expected end of the composite service (the booked end point) and traverse the UCM model in a backward direction until the start point book is reached. The resulting function derived by parsing the sliced UCM in this manner is:

```
Result = Sequence (Part1, Part2, Part3);
Part1 = Concurrent {C1, Sequence [L2, add, Discriminator (WF1, WF2), add]};
Part2 = Conditional Branching (Branch1, Branch2);
        Branch1 = Sequence [mem, HB2, che, Loop2 (Reset-HB2, mem, HB2, che), add];
        Branch2 = Sequence [mem, MB1, che, Loop2 (Reset-MB1, mem, MB1, che), add];
Part3 = Sequence (RC1, add, cha);
```

Finally, the NFP metrics of the selected concrete services are plugged into this function and the overall NFP metrics for the composite service are established based on the aggre-

gation algorithms defined in section 5.2. The following is the simplified function for reliability and availability:

```
Result = ∏ (Part1, Part2, Part3);
Part1 = ∏ {C1, ∏ [L2, add, 1 - ∏ (1-WF1, 1-WF2), add]};
Part2 = min (Branch1, Branch2);
        Branch1 = ∏ [mem, HB2, che, (∏ (Reset-HB2, mem, HB2, che))2, add];
        Branch2 = ∏ [mem, MB1, che, (∏ (Reset-MB1, mem, MB1, che))2, add];
Part3 = ∏ (RC1, add, cha);
```

The calculation of the NFPs for the composed service requires NFPs to be defined for responsibilities introduced by aspects similar to the NFPs of services, by annotating the responsibilities with metadata. As the company in this example has total control over the aspects, the values of NFPs are more easily established. In this example, each responsibility is deemed to have a cost of 0, a response time of 1, 0.99999 reliability, and 0.99999 availability. Resetting a service is also assumed to have the same NFP values.

Therefore, the NFP cost for the overall composite service is 22, response time is 72, reliability is 0.54967, and availability is 0.58523.

6.10. Chapter Summary

This chapter proposed composition and selection techniques that compare the NFPs of functionally equivalent services and selects the services that best meet the users' constraints and preferences, captured both with NFP descriptions and with a goal model for handling trade-off analysis.

The UCM notation was used to model composite services based on their required abstract services. Aspect-oriented UCMs enable the modeling and the weaving of additional functionality (or internally developed services) to the composite service such as integrity checks, billing, after-the-fact auditing, and redundancy in case of potentially unavailable services. Plug-in maps could also specify the behavior of the concrete service, allowing for more fine-grained estimates of composite NFPs. Aspects may also be used to describe other services and add them to the composite service.

GRL models were used to assess the impact of the NFPs of different functionally-equivalent services on the NFPs of the composite service, hence facilitating the selection of a concrete service with the best NFPs or tradeoff.

Our approach were defined and illustrated with the BookItWell service. Several modeling steps are obviously manual (albeit supported by tools), but the others are amenable to automation. The main URN tool (jUCMNav) does not yet support these steps entirely, but required prototype functionalities are being developed.

This chapter treated the Problem 3 presented in Chapter 3: NFP-aware service selection. The proposed evaluation criterion for the selection of the appropriate service for a given task is based on the best trade-off between multiple NFPs. In addition, the selection process investigates the possibility of using multiple services or multiple instances of the same service to best accommodate the user's constraints.

It is recognized that computing the "shortest path" to evaluate and select concrete services in step 7 is only practical when the number of services is limited. When the number of services is very large, one should find other alternatives to reduce the size of the concrete equivalent services to be considered.

In addition, some NFPs such as reliability and availability could be improved by taking the overall composite service into account instead of a shortest path approach. This is feasible for static composition; however this may not be possible if highly dynamic composition is required.

The next chapter will discuss the evaluation of the work proposed so far, together with the integration of the NFP schema with web service technologies such as WSDL.

Chapter 7. Development and Evaluation

This chapter is dedicated to the following development and evaluation aspects of the thesis:

1. The validation strategy of the NFP data structures and schema defined in Chapter 4, which involves the implementation and testing of the aggregation algorithms introduced in Chapter 5;
2. The comparison of this work against the problems/criteria and related work reviewed in Chapter 3;
3. The integration of NFPs into WSDL service descriptions;
4. The extensibility of the proposed NFP catalogue; and
5. The description of the required extensions to jUCMNav for tool automation, based on the service selection process discussed in Chapter 6.

7.1. Validation of NFP Schema and Automated Aggregation

This section presents the details of the NFP schema generation and of the proof-of-concept implementation of the aggregation algorithms that use NFP descriptions described in XML.

First, the proposed generic domain-independent catalogue of NFPs is shown to be sufficiently formal and precise to enable the automation of the NFP aggregation during the creation of the composite services. Then, the results of the automated aggregation of the NFPs of test composite services are compared to the results obtained manually.

Figure 53 gives an overview of the steps performed in order to define and validate the formality of the NFP definitions, the preciseness of the NFP data structures, and the correctness of the NFP aggregation functions. Each of these nine steps (plus a tenth one for testing) is described below.

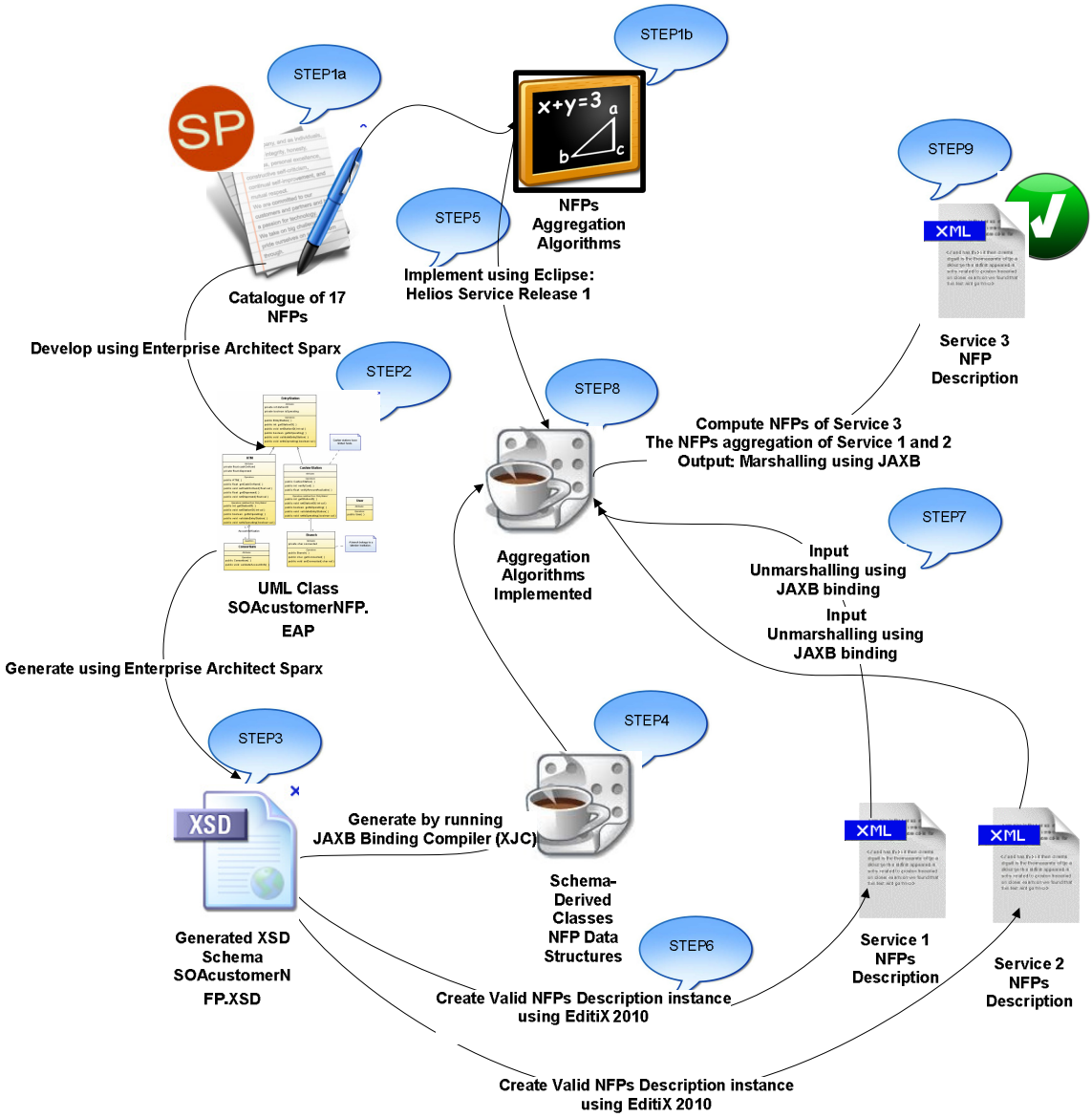


Figure 53 Process and Experiment Validation

Step 1: NFP Catalogue Consensus and Definition of Aggregation Algorithms

Step 1a: Descriptions of NFPs

A catalogue of seventeen NFPs was defined based on a literature review and validated via an external online survey, see Chapter 4. For each NFP, a textual definition and a precise data structure are provided. This step did not require any particular software (except for the use of an online survey tool, namely SurveyMonkey.com in this case).

Step1b: Definition of Aggregation Algorithms

An aggregation is defined for each NFP for which the aggregation was deemed to be possible (11 out of 17 NFPs, see Chapter 5). Note that this step could have been done later (i.e., after steps 2, 3, 4 and/or 6). Yet, it was decided to do the fundamentals first, and then the implementation later.

Step 2: **Creation of the UML Class Diagrams**

The UML class diagrams formalizing the data structures of all the seventeen NFPs defined in the catalogue were developed using the Enterprise Architect Sparx Systems tool (Professional Edition, Version 9.0). For such NFP language metamodel, one could have used any other UML tool. This step helped validate the abstract syntax of the data structures for the proposed NFPs and provided an appropriate degree of formality for the catalogue.

Step 3: Creation of the XML Schema

The XSD schema (see Appendix A) corresponding to the overall UML class model was generated using the Enterprise Architect Sparx Systems tool. "An XSD schema uses XML syntax to describe the relationships among elements, attributes and entities in an XML document" [74]. In other words, an XSD schema is one of the several XML schema languages that can be used to define and validate what can be included inside the XML instance document (elements and structures). This schema specifies the concrete textual syntax used to define NFPs.

As shown in Figure 53, this step enables the syntactic validation of any specific NFPs for service description. XML validation tools can be used to determine whether an instance of NFP description complies with the NFP data structure / XSD schema.

Enterprise Architect Sparx Systems tool was chosen in part because it supports the automated generation of XSD schema from UML class diagrams.

Step 4: Creation of Schema-Derived Java Classes

An XSD schema can also be used to derive an object model, for example as a set of Java classes (one per type of element, including enumerations), for parsing and manipulating

XML descriptions. The Java Architecture for XML Binding (JAXB, [48]) is a convenient framework for processing XML documents (see Figure 54 for JAXB architectural overview).

By running the JAXB *xjc* compiler, a binding to the source XML schema (e.g., the XSD file in Appendix A) is created to generate the Java classes that map to constraints in the source XML schema (e.g., Java classes corresponding to the data structures). As shown in Figure 53, this step enables the access to and the manipulation of the NFP data structures. With the xjc compiler, all Java classes (32 files) are generated automatically into one package (see Appendix E). These classes preserve the hierarchy relationships defined in the NFP catalogue metamodel. One should keep in mind that the code for these classes should never be modified. Following any future changes in the NFP catalogue metamodel, one should re-compile the schema to generate new Java classes.

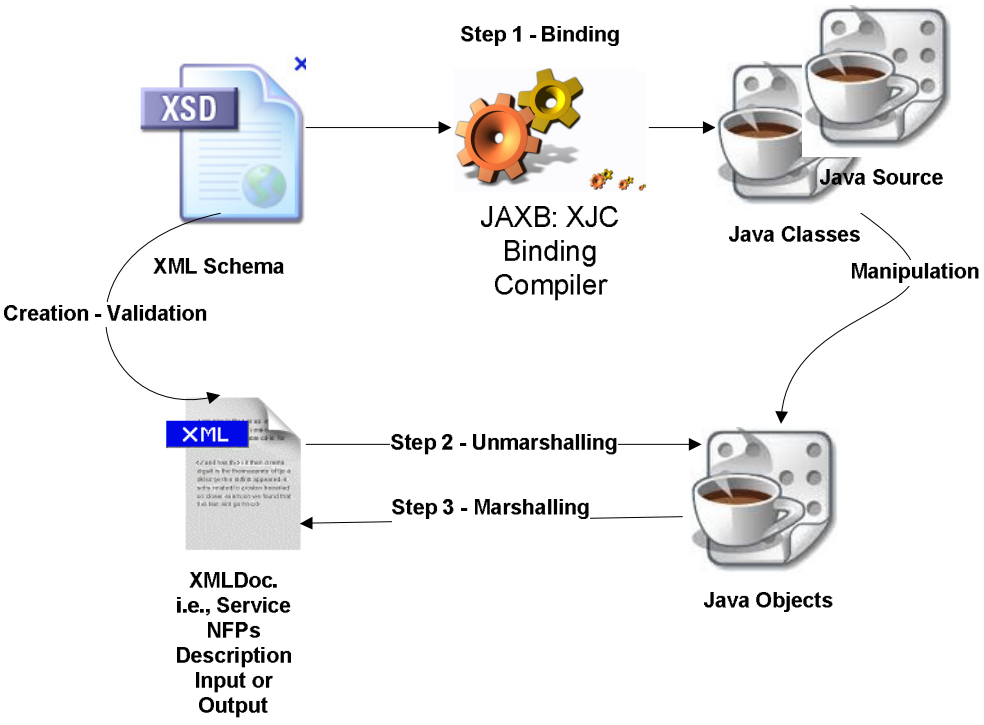


Figure 54 JAXB Architectural Overview

Step 5: **Implementation of the NFP Aggregation Algorithms as Java Programs**

The Eclipse development platform [24] (version Helios Service Release 1) was used to implement the aggregation algorithms defined in Chapter 5.

The Java language was chosen since it is one of the most popular and portable programming languages in use for web services creation. In addition, Java allows taking advantage of platform-neutral XML data without a need for a deep knowledge of XML programming techniques. It is also possible to bind Java programs and XML schemas using JAXB. The latter provides methods for unmarshalling XML instance documents into Java content trees, and also allows for marshalling Java content trees into XML instance documents (see Figure 54 for a JAXB overview).

All the Java implementations are available online (see Appendix E). This step (see Figure 53) enables the automation of NFP aggregation and allows correctness testing of the proposed aggregation algorithms.

Step 6: Creation of Valid XML NFP Descriptions

Two XML instance documents (Service1.xml in Appendix B, and Service2.xml in Appendix C) describe two of the services introduced in section 5.2. These XML instance documents, created using EditiX 2010, are written according to the constraints defined in the source schema. The XSD schema (Appendix A) is used to create a baseline template and then values of different NFPs are entered manually. These documents are used as input to the aggregation algorithms (see Figure 53). To ensure that the XML documents are valid, a pointer to the source schema is added at the top of the files. This enables validating tools such as EditiX 2010 to check the correctness of the source XML documents.

Step 7: Unmarshalling of XML (NFP description instance) files to Java Objects.

The XML instance documents (Service1.xml and Service2.xml) are unmarshalled as input to the JAXB binding framework (see Figure 53 and Figure 54). In other words, a Java representation in the form of a content tree is generated to allow the extraction of values stored in the XML documents.

Step 8: Java Objects Manipulation

The NFP values that are extracted from the XML instance documents are manipulated and used as inputs to the aggregation algorithms. The aggregated NFP values are then computed (see Figure 53 and Figure 54).

Step 9: Marshalling of Java objects to XML files

The aggregated NFP values computed as Java objects in the previous step were marshalled out to an XML document (Service3.xml, see Appendix D). In JAXB, marshalling involves parsing an XML content object tree (in memory) and writing out an XML document that captures this information in a structure that is valid with respect to the source schema (see Figure 53 and Figure 54).

Step 10: Testing

Two XML instances Service1.xml and Service2.xml were created and validated against the SOAcustomerNFP.xsd schema. The values of NFPs of these 2 services, entered manually using a template, are fictitious. These values and the manually computed NFP aggregation results are shown in section 5.2.

Once the main program in Appendix E is invoked, a Service3.xml instance is created. This instance is valid and includes the automated computation of the aggregated values of NFPs of Service 1 and Service 2. These NFP values of Service 3 were compared to the expected values obtained by doing the aggregation exercise manually. The NFP values obtained were the same. The same exercise was done with other pairs of services extracted from chapter 5 and 6.

Note. As part of the aggregation, a web service for currencies conversions is invoked along the way, in order to check the feasibility of the proposed solution in that context. No problem was encountered.

7.2. Comparison against Reviewed Solutions

This section examines whether the proposed approach scores better than other NFP-related work in SOA, summarized in Table 1, against the evaluation criteria defined in section 3.2.

7.2.1 Problem 1: NFP Catalogue

To address this problem, the first criterion was the definition and validation of a formal catalogue of domain-independent NFPs for services from the perspective of the service consumers.

The approach proposed in this thesis succeeded in defining a formal NFP catalogue focused on the consumers' perspective. For each NFP of this catalogue, Chapter 4 provided a concise definition and justified its relevance.

The degree of formality of the proposed NFP catalogue, although still limited semantic-wise, is better than what is usually seen in the literature. Each NFP has a data structure that formalizes precisely how it can be described and measured. In addition, a concrete XML schema was generated to give service providers and consumers a concrete syntax for precisely describing the NFPs of their offered and required services.

The following reviewed papers achieved a good degree of formality, but without any clear focus on the consumers' perspective and without any validation process: Choi's QoS Metrics for Service Provider [17], O'Sullivan's What is in a service? [26], [67] and [66], Zeng's QoS Middleware for WS Composition [89] and Liu's QoS Service Selection [49]. It should be noted that O'Sullivan's work provided the most extensive list of models describing NFPs (79 different types). This can be helpful for providers who wish to extend the catalogue to include domain-specific NFPs.

The only reviewed paper that was focused on the service consumers' perspective with a good degree of formality is Chen's QoS Driven WS Composition [15]. It provides seven NFPs but with simplistic data structures that have not been validated.

With respect to the NFP validation process, unlike what can be found in the literature, the NFP catalogue offered in this thesis was validated externally with real users and experts.

The current catalogue is the most complete domain-independent one that focuses on the consumers' perspective and achieves a very good degree of formality. As well, this catalogue was defined technology-independent way so that it can now be used by multiple developers and standardization bodies.

7.2.2 Problem 2: Aggregation of the NFPs

To address this problem, the first criterion was the definition of NFP aggregation algorithms that take into consideration the different composition operators.

Such aggregation algorithms were proposed for eleven out of the seventeen NFPs, and for all five composition operators in the context of the case study BookItWell formality introduced in Chapter 1. For the remaining six NFPs (certification, accuracy, standards compliance, failure modes, security, and service versioning), aggregation is not possible without human intervention or further restrictions of the data structures. For each NFP where aggregation deemed to be impossible, a rationale was provided.

Four reviewed papers (including Chen's QoS Driven WS Composition [15], Zeng's QoS Middleware for WS Composition [89], Suzuki's SLA Genetic Optimization Framework [78] and Mukherjee's Determining QoS of WS-BPEL Compositions [54]) have worked on the NFP aggregation problem and proposed solutions following a simplistic treatment of composition operators (criterion 2).

The aggregation algorithms developed here represent the most complete solution that takes into account precise NFP data structures (i.e., price is not only one value, it has different business models and different currencies) and providing proper treatment of all the composition operators. In addition, the discriminator operator is introduced in service composition and taken into account in the developed aggregation functions.

7.2.3 Problem 3: NFP-aware Service Selection

To address this problem, an approach for NFP-aware service selection that preferably selects the appropriate service for a given task based on the best trade-off between *multiple* NFPs (criterion 3) and that investigates the possibility of using multiple services or

multiple instances of the same service to best accommodate the user's constraints (criterion 4) is needed.

In the reviewed papers, Liu's QoS Service Selection [49] and Hughes' QoS Explorer [33] have proposed solutions per criterion 1 (sorting the available services based on one user constraint). Zeng's QoS Middleware for WS Composition [89] and Suzuki's SLA Genetic Optimization Framework [78] have proposed solutions per criterion 2 (sorting the available services based on the user's rating of the different criteria). Finally, Dobson's QoS Ontology [21] has proposed solutions per criterion 3 (based on the best trade-off between *multiple* NFPs).

Our proposed solution is the only one that achieves NFP-aware service selection per criteria 3 and 4 (based on the best trade-off between *multiple* NFPs and using multiple services or multiple instances of the same service). In addition, the catalogue empowers service consumers as what to expect as NFPs and what they should look for and compare.

7.2.4 Conclusion of the Comparison

While, ISO/IEC 9126 [38] and the OASIS Quality Model for WS [44] are the closest research efforts to this thesis in terms of *defining* NFPs, the *publishing* context of Chen's QoS Driven WS Composition [15] is the closest one to the focus of the research effort of this thesis, Zeng's QoS Middleware for WS Composition [89] is the closest one in terms of treated problems. Table 11 summarizes the comparison between these four papers and the NFP-related work contributed here (and hence completes Table 1). The solutions provided in this thesis are based on a richer and more precise set of NFPs that were externally validated. As well, the proposed solutions go well beyond what was surveyed in the literature review and score better than the reviewed research efforts.

Table 11 Thesis Contributions versus Reviewed Solutions

Reference	Formalism	Aggregation	Service Selection	Perspective	NFP Validation	Comments
This thesis	++	1	3, 4	C	E	An end-to-end approach for handling the NFPs of the services in SOA covering (i) the NFP description, and (ii) NFP usage in NFP-aware service comparison and selection, and (iii) in NFP aggregation for composite services. Seventeen NFPs are specified, including eleven that are composable.
ISO/IEC 9126 [38]	-				S	An international standard for evaluating the quality of conventionally developed applications, and not tailored for SOA.
OASIS Quality Model for WS [44]	±			M	S	The work is still incomplete and the attributes are defined at the conceptual level.
Chen's QoS Driven WS Composition [15]	+	2		C		Seven NFPs: response time, availability, concurrency, expire time, price, fine, and security level. There is no validation of the list of the NFPs. No external validation of the list of the NFPs. Simplistic treatment of parallel and conditional branches in the NFP aggregation approach. Does not treat the NFP-based Service Selection Problem.
Zeng's QoS Middleware for WS Composition [89]	+	2	2			Five NFPs: execution price, completion time, reputation, successful execution rate and availability. There is no clear perspective and no validation of the list of the NFPs. Simplistic treatment of parallel and conditional branches in the NFP aggregation approach.

The legend for Table 11 is as follows:

NFP Formalism: ++ Very Good , + Good; ± Fair; - Poor
NFP Aggregation:
(1) Takes into consideration the different composition operators including conditional branching and parallel branching;
(2) Has a simplistic treatment of composition operators;
NFP-Aware Service Selection:
(2) Based on the user's rating of the different criteria,
(3) Based on the best trade-off between *multiple* NFPs,
(4) Investigates the possibility of using multiple services or multiple instances of the same service.
Perspective: Consumer's perspective
NFP Validation: External

7.3. Integration of NFPs into WSDL

As stated in previous chapters, current service description languages do not contain all elements required for effective service descriptions that include the NFP aspects. Even

the Web Services Description Language (WSDL) [86], the most widely adopted description language for creating services, out of a multitude of WS enabling technologies, does not allow for the description of service NFPs. In order to enable NFP descriptions, several research efforts proposed NFPs integration into the WSDL schema, including [69], [2] and [90].

As proof of concept of the usage of the proposed NFP catalogue, this section proposes new service NFP description integration into WSDL. The integration is performed using the WSDL extensible elements, without introducing any change to the base WSDL specification itself.

Note that the essential concepts related to WSDL, including the various parts of the schema and the potential for extensibility, were covered in section 2.3.

7.3.1 Requirements for Integration of NFPs into WSDL

A number of requirements for NFPs integration into WSDL framework are enumerated below. The integrated framework should:

- Allow the creation of an association between the proposed NFP description with the WS description, without changing the used service description language (i.e., WSDL here);
- Be extensible to support other NFPs (i.e., domain-specific NFPs) in addition to the proposed generic NFPs, without changing the algorithms and tools for NFP-aware service filtering and selection;
- Work with the existing WS stack, without causing WS tools incompatibility; and
- Take advantage of the WSDL extensibility elements.

7.3.2 Related Efforts on the Integration of NFPs into WSDL

Semantic Integration

Some research efforts are focused on extending WSDL functional aspect by associating semantic annotations (e.g., the meaning of objects or information) with Web services such as Web Service Semantics (WSDL-S) [90]. The aim of these extensions is to augment the functional description (e.g., adding semantic to the syntactic) of services in such

a manner as to avoid possible confusion between two services that can have the same syntactic definition but perform significantly different functions, and to enable the detection of syntactically dissimilar services that have equivalent functionalities. This work should not be confused with extending WSDL to enable the description of non-functional properties.

Parimala and Saini's Integration of NFPs into WSDL

Other initiatives are focused on extending WSDL to attach particular non-functional properties such as security requirements, performance and testing properties to web services. Parimala and Saini [69] suggest extending the WSDL schema to support "generic" NFPs, by adding new keynames to the definitions and service elements.

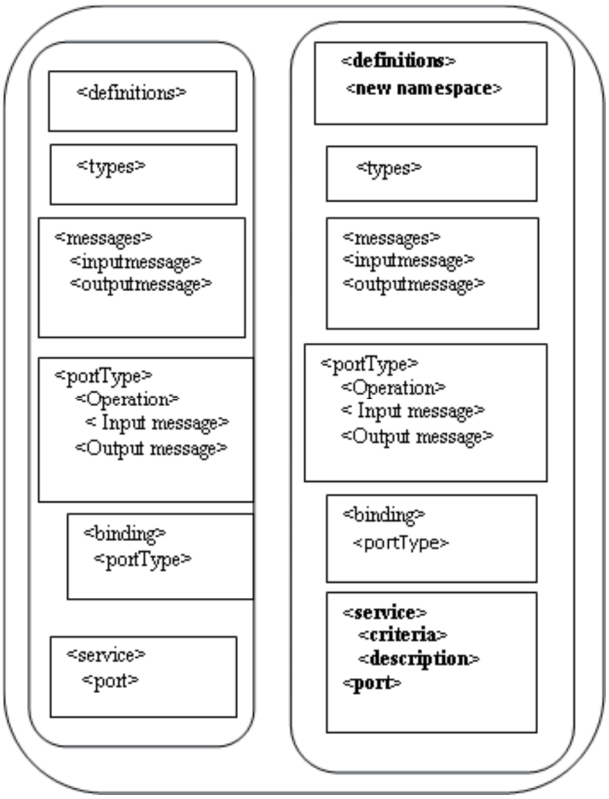


Figure 55 Elements of WSDL Versus Extended WSDL [69]

As depicted in Figure 55, Parimala and Saini extend WSDL by using the extensibility of definitions and service WSDL elements. These authors also define a complex type called *criteriaService*, which contains three attributes. The first two attributes are defined under

the service element (e.g., the service element is extended) (1) the criteria name (e.g., the NFP name) and (2) the description (e.g., detailed description of the NFP in text format). The third attribute is the definition of the NFP and it is specified along with other namespaces under the definitions element. One can have as many criteria names as NFPs to be described for a given service.

```
<definition name=" "
   targetNamespace="http://localhost:8080/X-UDDI/wsdl1"
     xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
     xmlns:cr=" http://localhost:8080/X-UDDI/wsdl1"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <message name=" ">
          <part name=" " type=" "/>
      </message>
      <portType name=" ">
         <operation name=" ">
            <input message=" "/>
            <output message=" "/>
         </operation>
      </portType>
     <binding name=" " type=" ">
   <operation name=" ">
        <soap:operation soapAction=" "/>
        <input>
          <soap:body  encodingStyle=" "/>
        </input>
        <output>
          <soap:body encodingStyle=" "/>
        </output>
    </operation>
   </binding>
<cr:service name=" ">
       <criteria name=" "
       <description=" "
<port binding=" " name=" ">
       <soap:address  location="">
</port>
</service>
   </definition>
```

Figure 56 WSDL Document with Extended Elements [69]

Similarities with the work undertaken in this thesis: The research undertaken during the development of this thesis is in line with the above WSDL service and definitions elements extensions in support of the description of NFPs. As explained in theory in Figure 56 and showed by example in Figure 57, WSDL descriptions should be extended by referring to the NFP schema under the ***definitions element*** and by adding the NFP concrete specification under the ***service element***. Introducing additional information that applies to the whole service should be done under the service element. The relationship between the service element and the WSDL service is a 1-to-1 relationship. Each WSDL service has one service element and each element service defines a single service. The remaining WSDL service elements could be shared between many services. Their extensions apply only to a few parts of the service and not to the service as a whole. In addition, the information or definitions included in the service element are the ones to which service consumers can have access.

Differences with the work undertaken in this thesis: this thesis does not share the same vision as that of Parimala and Saini [69] of the exact schema definitions to be attached to a service. It is believed that their schema (Figure 56) for NFP description, which is restricted to the name of the NFP and some textual description, is a simplistic view of the NFP definitions. First, the textual description is not formal enough to enable NFP comparison and service selection. As well, it is not sufficient to enable NFP aggregation for composed services since there is no predetermined format for the NFP values. Based on the research work undertaken to support this thesis, it was concluded that one cannot reduce all NFP data structures to these three attributes. Actually, the future work stated in this paper is about providing composable criteria for services, which is actually one of the contributions of this thesis. Consequently, the work done in this thesis is few steps ahead in terms of NFP schema definition and service aggregation.

It is very important to note that the meaning of *generic NFPs* in Parimala and Saini's work is different than the one used in this thesis. In [69], the NFPs are more about *adding details/criteria about the service functionality and not about how the service is rendered.* Specifying "Booking a hotel room" service with an "Internet connection" or/and a "balcony" is more information about the functionality (real world effect taking place after invoking a given service – reserving the connected hotel room, or a room with

balcony) rather than really generic, domain-independent NFPs common to all services and not only to hotel reservation service.

Agarwal and Jalote's Integration of NFPs into WSDL

Other approaches to extending WSDL to support generic NFPs like, and in particular Agarwal and Jalote's approach [2], define a complex type structure that arguably can fit any NFP. The NFP is defined as an extension of the "string" data type with the following four attributes: name (URI), dataType (parsing info of the NFP), unit (string), and validUntil (date of expiry of dynamic NFPs; static NFPs are ValidUntil forever). As one can see in Figure 57, only the name is mandatory, the 3 remaining attributes are optional.

```
<complexType name="PropertyType">
    <simpleContent>
        <extension base="string">
            <attribute name="name" type="anyURI"
                use="required">
            </attribute>
            <attribute name="dataType" type="QName"
                use="optional">
            </attribute>
            <attribute name="unit" type="string"
                use="optional">
            </attribute>
            <attribute name="validUntil" type="dateTime"
                use="optional">
            </attribute>
        </extension>
    </simpleContent>
</complexType>
```

Figure 57 Schema for Specifying Properties [2]

Similarities with the work undertaken in this thesis: This thesis shares the same perception of NFPs to be defined and added to service description as Agarwal and Jalote. For example, they specify Reputation for the "calculatorService", Cost for "CalculatorPort", ResponseTime for the two operations "add" and "substract" in Figure 58. The NFPs are defined and attached to various service elements. Both research efforts have the same opinion about the need to define NFPs such as Reputation, Cost and Response Time.

Differences with the work undertaken in this thesis: the thesis NFPs apply to the whole service and not to parts of the service. The focus of this thesis is on the NFPs

from the perspective of the service consumer's for the service as one unit, unlike the research work in [2].

It is believed that, from the perspective of the service consumers, NFPs of a service should be defined as a whole and not for each element of a service.

```
<description>

    ....
    ....

   <wsdl:service name="CalculatorService">
     <wsdl:port name="CalculatorPort"
               binding="impl:CalculatorSoapBinding">
        <wsdlsoap:address location="http://localhost:8080/Calc/
               services/Calculator" />
     </wsdl:port>
   </wsdl:service>

   <nf:properties name="CalculatorService">
     <nf:property name="http://www.ibm.com/wsdl/NFProperties/
               Reputation">
        good
     </nf:property>
   </nf:properties>

   <nf:endpointProperties name="CalculatorPort">
    <nf:property name="http://www.ibm.com/wsdl/NFProperties/Cost"'
             dataType="decimal">
         2.5
     </nf:property>

    <nf:operation name="add">
         <nf:property name="http://www.ibm.com/wsdl/NFProperties/
                  ResponseTime" dataType="decimal" unit="seconds"
                  validUntil="2009-12-31T12:00:00">
              4
         </nf:property>
    </nf:operation>

    <nf:operation name="subtract">
         <nf:property name="http://www.ibm.com/wsdl/NFProperties/
                  ResponseTime" dataType="decimal" unit="seconds"
                  validUntil="2009-12-31T12:00:00">
              3
         </nf:property>
    </nf:operation>
   </nf:endpointProperties>

</description>
```

Figure 58 Specifying Properties at the Top Level [2]

Taking a closer look at Agarwal and Jalote schema definition process, four requirements for specifying NFPs can be observed: (1) generic NFPs, (2) capacity to describe static and dynamic NFPs, (3) flexibility to add NFPs to any element of the service, and (4) the ease of integration of the NFPs with services description languages (e.g., WSDL). This

thesis is not totally aligned with these requirements, although it does have some commonality, as described below.

(1) In the context of this thesis, the definition of generic NFPs is different from the one used in [2]. In this thesis, *generic NFPs* means that the NFPs of services are not related to a particular service functionality/operation or service domain. This means that "generic" does not contradict with detailing and proposing an NFP set that applies to most services independently from the service domain.

(2) In [2], an attribute to the labeling of each NFP as dynamic or static. This thesis goes beyond this labelling and supports the addition of different values for dynamic NFPs (best, average, worst values instead of having one value) and only one deterministic value for static NFPs.

Regarding (3) and (4), both Agarwal and Jalote's work and this thesis agree on the importance of an extensible NFP catalogue and the necessity of simple integration of NFP descriptions into the service description language.

7.3.3 Proposed Integration of NFPs into WSDL

This thesis provides a library of complex types as an XML schema that can be fully or partially populated without having to do it from scratch and that apply to the whole service as one unit. This library of NFPs is extensible and some data structures can be reused as they are (or at least may inspire service developers) to include other domain-specific NFPs. Having defined precise but generic NFPs enable applications such as service comparison, selection and substitution, as well as automated NFP aggregation.

The catalogue can be included within a WSDL instance document or can be specified separately and paired with WSDL. The latter approach is done by referring to the XSD schema in Appendix A, which is kept in a separate document, using a suitable namespace in the definitions element and including NFP description in the WSDL instance of the service description element).

It was decided to use the extensibility of *service* WSDL elements for two reasons. First, all defined complex types are visible only within their extended WSDL element except for service and definition WSDL elements. Second, the definition of a WSDL

element cannot be extended because the definition of a WSDL element could be used to define more than one WS.

The NFP catalogue is WS technology agnostic (i.e., it can be used in conjunction with any service enabling technology). WSDL is widely adopted and it is XML based, which makes it a good candidate for demonstrating the feasibility of such integration.

Just like for Agarwal and Jalote's approach [2], the required NFP extensions are maintained outside of the WSDL standard specification and are referenced from the WSDL document via WSDL service and definitions extensibility elements.

7.3.4 Validation of the Integration of NFPs into WSDL

The following are the steps performed to test and validate the WSDL extension to support the proposed NFP integration. This example contains a simplistic web service, but it is sufficient as a proof of concept and can easily be extended to more realistic services.

First Step: Tools

The Eclipse Java EE IDE for Web Developers was installed (Indigo Service Release 2). This particular version supports WS development and deployment.

Second Step: Creation of a Simple WS

A Dynamic WebProject was created and a web server selected. Then, a simple Java class (`HelloImpl.java`, see Figure 59) was created. The Eclipse environment was used to generate a Java Bean web service and a Web service client from that Java class. This led to the automatic generation of a default WSDL document (`HelloImpl.wsdl`).

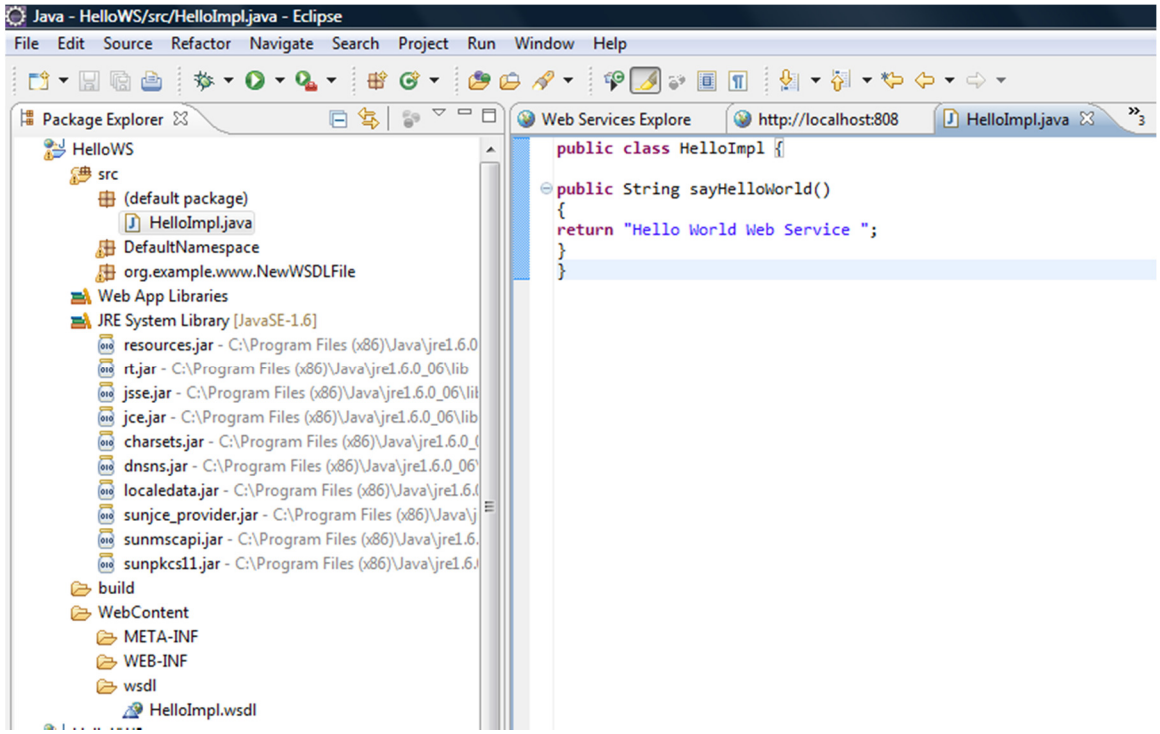


Figure 59 HelloImpl.java Class

Third Step: Integration of NFPs into WSDL

The WSDL document was modified to include the description of NFPs by adding a reference to the proposed schema and by adding the NFPs under the WSDL *service* element. In Figure 60, a reference to the NFP schema of Appendix A is added under the definition WSDL element. In this example, 11 different types of NFPs relative to the service were specified. The NFP tags are all prefixed by the specified namespace (`xsd2` here).

```
<?xml version="1.0" encoding="UTF-8"?>
- <wsdl:definitions xmlns:xsd2="http://dl.dropbox.com/u/2920864/SOAcustomerNFP.xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:intf="http://DefaultNamespace"
xmlns:impl="http://DefaultNamespace" xmlns:apachesoap="http://xml.apache.org/xml-soap"
targetNamespace="http://DefaultNamespace">
        <!--WSDL created by Apache Axis version: 1.4 Built on Apr 22, 2006 (06:55:48 PDT)-->
  + <wsdl:types>
  + <wsdl:message name="sayHelloWorldRequest">
  + <wsdl:message name="sayHelloWorldResponse">
  + <wsdl:portType name="HelloImpl">
  + <wsdl:binding name="HelloImplSoapBinding" type="impl:HelloImpl">
  - <wsdl:service name="HelloImplService">
      - <xsd2:NFPdescription>
          + <xsd2:price>
          + <xsd2:responseTime>
          + <xsd2:reputation>
          + <xsd2:availability>
          + <xsd2:reliability>
          + <xsd2:usability>
          + <xsd2:transactional>
          + <xsd2:jurisdiction>
          + <xsd2:resource>
          + <xsd2:scalability>
          + <xsd2:serverLocation>
        </xsd2:NFPdescription>
      + <wsdl:port name="HelloImpl" binding="impl:HelloImplSoapBinding">
    </wsdl:service>
</wsdl:definitions>
```

Figure 60 HelloImpl.wsdl

Fourth Step: Testing

For the client that uses the HelloImpl service, client stubs were generated through the Dynamic Web project Axis2WSTestClient. Then, Eclipse's Web Service Explorer was used to test the Web service via native WSDL and SOAP invocations.

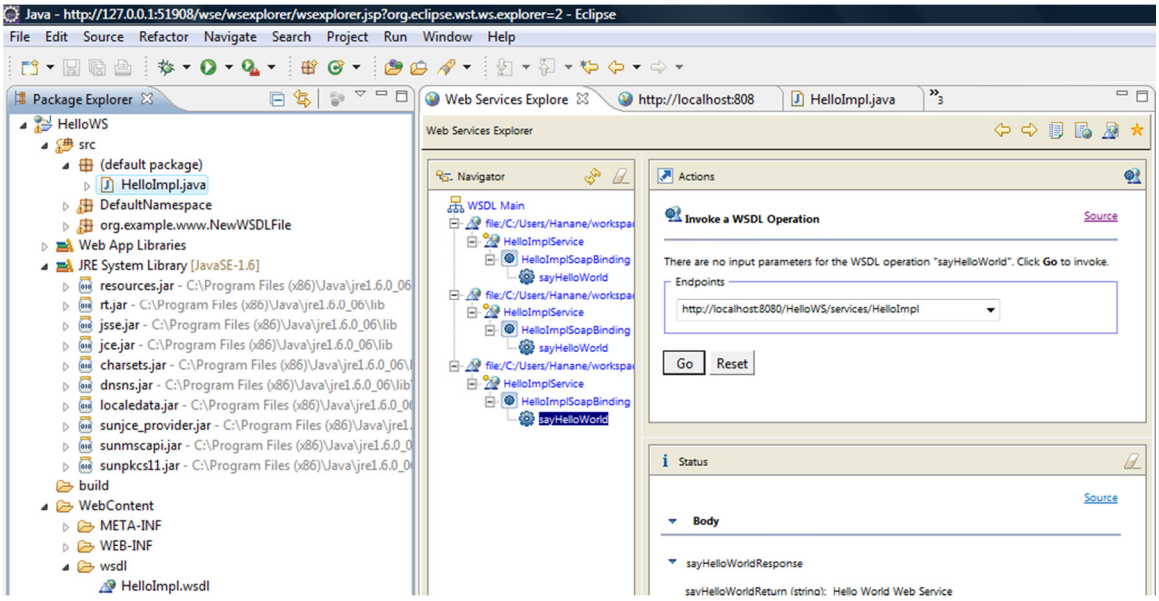


Figure 61 NFPs Web Service Invocation

Using the extensibility elements of WSDL, this thesis succeeded in augmenting the expressivity of WSDL by adding non-functional property descriptions to service descriptions.

7.4. NFP Catalogue Extensibility

There are two dimensions to the extensibility of the NFP catalogue. The first dimension is adding new NFPs and the second one is adding extra details for the existing NFPs (e.g., expanding the NFP attributes and/or adding an alternative domain-specific NFP structure). **Note.** In many places, if extra details for the proposed NFPs are needed, the catalogue already supports a few variables with "others" as a value in the data structures and enumerations.

In order to extend the NFP catalogue by adding new NFPs, one can simply create the UML class diagrams of the new NFPs and follow the given approach (described in 7.1, Steps 2, 3 and 4) to regenerate the XSD schema and the class diagrams.

Alternatively, one can extend the NFP catalogue either (i) by defining new complex types within the service WSDL element and add the values for that NFP underneath it; or (ii) by creating a new schema of the new NFPs in a separate document, refer to it

under the definition WSDL element and added the values of that NFPs under the service WSDL element.

While following the first approach requires only the knowledge of basic concepts of UML, the two other alternatives require a good knowledge of XML and WSDL. The first approach is highly recommended since it is less prone to errors.

7.5. Tool Support for jUCMNav-based Automation

In Chapter 6, a modeling approach for service selection and service composition based on the User Requirements Notation (URN, and its aspect-oriented extensions in AoURN) and the framework for describing and composing various non-functional properties was introduced. Current tool support by jUCMNav, however, does not allow full automation of the introduced approach at this point. The following extensions to jUCMNav are required:

1. Support for NFP descriptions and annotations. URN and jUCMNav support user-defined metadata, which can be used to support NFP to some extent;
2. Support for the retrieval of concrete services from the registry and to allow for the creation of plug-in maps for them in the UCM model (step 4);
3. Establishing GRL strategies and initial satisfaction values based on the annotations of plug-in maps (step 7);
4. Setting the condition of plug-in maps to true based on the evaluation of GRL strategies (step 8); and
5. Parsing the UCM model to get the function for the NFPs of the composite service and calculating the NFPs for the composite service (step 9).

These extensions require a substantial software development effort, but as their absence at this point does not invalidate the research results in Chapter 6, these additions are left to future work.

7.6. Chapter Summary

This chapter described the various angles through which the thesis contributions were validated. First, the NFP schema (Appendix A) and the NFP aggregation algorithms were implemented and tested. A comparison of the NFP catalogue, aggregation algorithms and service selection process with some of the closest research contributions from the literature was performed, against the same problems/criteria used to assess the related work in the literature review. As well, this chapter described and demonstrated the feasibility of integrating the NFP catalogue with a mainstream WS technology, namely WSDL. The extensibility of the catalogue was also briefly discussed. Finally, the tool extensions required for automating the proposed AoURN-based service selection approach were identified.

The next chapter presents the conclusions of this thesis, several limitations and threats to its validity, and several future work items.

Chapter 8. Conclusions

"*No model is correct, but some are useful*" – George E. P. Box

SOA cannot achieve agility via service composition in different contexts by different independent service consumers if service providers do not publish (expose) the NFPs of their services as part of the main service descriptions, in a standardized format. Many researchers have been focused on handling the NFPs of the messaging framework and the services interfaces, and on developing web services management frameworks, semantics, constraints languages, ontologies and/or SLAs. Only a few of them worked on defining the NFPs of the services from the perspective of service consumers, in opposition to the perspective of the service providers (developers) and multi-perspectives.

Before this thesis, a formal and structured catalogue of domain-independent NFPs, described from the perspective of the service consumer, was yet to be defined. Such a catalogue is needed to better characterize NFPs, in order to enable advanced applications such as NFP-aware service selection, NFP aggregation, service redundancy, service composition, and dynamic adaptation.

Having built and reached consensus, via an external validation process, on a formal domain-independent catalogue of 17 NFPs as shown in Chapter 4, service providers are now empowered to better handle the description of generic NFPs and to extend the proposed catalogue with domain-specific NFPs (and perhaps even consumer-specific ones) to accommodate consumers who have become more knowledgeable of what to expect in NFPs and what to look for, compare, and select. The catalogue of NFPs is used to better characterize and aggregate NFPs, as described in Chapter 5. Chapter 6 demonstrated that this NFP catalogue could be used to compare and select services based on their NFPs, while enabling advanced applications such as NFP-aware service selection. A modeling approach for the NFP-aware selection of services, which involves Aspect-oriented URN as a means to formally model and analyze service NFPs, and to perform

service selection and NFP aggregation in SOA, was also defined. Finally, Chapter 7 demonstrated an integration of the NFP catalogue with the Web Services Description Language (WSDL).

8.1. Contributions Impact

The contributions of this thesis represent significant advances over the existing approaches. As for industrial impact, this work contributes a validated collection of NFPs with a concrete syntax, composition algorithms, and proof-of-concept implementations ready to be used for defining, selecting, and composing NFP-driven services and for evolving current SOA-related standards. This thesis defined a catalogue of 17 domain-independent NFPs from the perspective of service consumers to be published by the service providers as part of service descriptions. This catalogue contributes to knowledge by providing more detailed insight on the nature, relevance, and composability of NFPs in a service engineering context, with a particular attention to SOA and web services. This work complements related efforts including the NFPs of the messaging framework and the services interfaces, semantics, specification of constraints languages, ontologies and/or SLAs. The NFP catalogue can now be used to serve different purposes, including:

- Providing concrete guidelines on the description of the NFPs of released atomic services by the service providers;
- Providing a flexible extensible framework for domain-specific NFPs description;
- Describing the desired NFRs of services requested by the service consumers; Enabling search, discovery, filtering and selection of services based on their NFPs;
- Alleviating the ambiguity and redundancy problems associated with proprietary disparate NFP descriptions;
- Enabling cost-effective service redundancy and/or substitution (adaptation);
- Providing means to compare NFPs to enable NFP-aware service selection;
- Facilitating the NFP aggregation for composite services;
- Automating some of the NFP management activities;
- Supporting NFP measurement and monitoring approaches;
- Enabling WSDL to describe the NFPs of services; and

- Permitting BPEL extension for automated NFP aggregation.

In addition, the catalogue could potentially be used to complement the following research activities:

- Reducing the time required for developing SLAs;
- Replacing SLAs, whenever the situation does not require custom-made, mutually negotiated, lengthy and expensive SLAs;
- Serving as a flexible and extensible framework for domain-specific NFPs specification;
- Providing taxonomies and/or ontologies to complement the formal specification of constraints languages that support web service management such as the Web Service Offerings Language (WSOL) [77] and Semantic Markup for Web Services (OWL-S) [20];
- Complementing XML-based languages for semantic descriptions of web services such as Semantic Markup for Web Services (OWL-S), which provides a declarative language for service properties and capabilities advertisement, but only placeholders for the description of NFPs;
- Supporting NFP monitoring approaches in driving their requirements;
- Enabling the Web Services Description Language (WSDL) standard to describe the NFPs of services; and
- Enabling the automated aggregation of the NFPs of composite services by extending the Business Process Execution Language (BPEL).

8.2. Related Work Limitations

This thesis helped to overcome many limitations of existing solutions, such as:

- Defining an exhaustive but informal set of NFPs for atomic services;
- Proposing simplistic NFP models, such as NFPs that do not require any normalization and that do not support different pricing models or different currencies;
- Acknowledging the need for NFP catalogues, but without delivering one;
- Having no clear focus on the service consumers' (or publishing) perspective;

- Proposing no methodology for validating NFP catalogues;
- Lacking end-to-end solutions that propose NFP data structures and aggregation functions;
- Delivering NFP aggregation based on a simplistic treatment of the composition operators; and
- Ignoring context and dependencies among NFPs for NFP-based service selection.

8.3. Limitations of the Proposed Solution

8.3.1 NFP Description

One can argue that some NFPs are very simplified (e.g., usability, limited to a single score) while other NFPs are so complex that they cannot be aggregated (e.g., security). However, this catalogue was developed based on a literature review and validated by experts and does reflect what the SOA community thinks at this point in time.

8.3.2 Survey Validity Limitations

Possible threats to the validity of the results of the survey are recognized. First, the participants are technologists and mostly from the service provider's side (i.e., solutions developers, mainly familiar with B2B and SLAs); hence they are not necessarily the best people to directly represent the service consumers' perspective (although they often act on their behalf).

The survey tool (SurveyMonkey) did not have a feature that allows a randomization of the order of the questions. In addition, due the nature of the questionnaire (lengthy and requiring high concentration), participants may get tired or simply rush through the last couple of NFPs/questions. Indeed, in addition to the 29 participants who completed the questionnaire, another 24 participants started the questionnaire and gave up quickly after a few NFP-related questions. This suggests that the survey was indeed not easy to answer. Several people might also simply have been curious about the content of the survey. This last threat was somewhat mitigated by the survey tool, which provided feedback on the progress of the participant as he/she answered the survey questions.

8.3.3 Adoption Limitations

A survey of domain experts to get their insight and determine their perceptions regarding the proposed NFP catalogue was conducted. However, there is still much effort required to achieve adoption by practitioners. This work, and the NFP data structures in particular, should also be standardized via contributions to standardization organizations such as the Object Management Group (OMG).

As well, there is still no real large deployment of the NFP framework involving different service providers and consumers, participants from the industry, and real-life utilization.

8.3.4 Tooling Limitations

The jUCMNav tool is not yet extended to fully support AoURN-based automation as described in modeling approach, or to map in the background URN models to WSDL and BPEL. Other issues might be discovered while automating the service selection approach proposed here.

8.4. Future Work

Many future work items were identified throughout the thesis. Some of the main ones are recalled here, with a few additional ones:

- Explore further potential usages of the NFP catalogue, in various domains, and evolve is content when necessary;
- Review the NFPs for which no aggregation functions are proposed to determine if their aggregation is possible for specific domains and/or for subsets of their attributes;
- Provide (of transform) NFP models in an OWL-S representation;
- Implement the required extensions to the jUCMNav tool to support automated computation of service composition, NFPs aggregation, and service selections;

- Submit contributions to standardization bodies, more likely at the Object Management Group (OMG), which showed interest in such topic [65] , to further validate and facilitate the adoption of the catalogue of domain-independent NFPs;
- Extend the most widely used service composition languages, such as BPEL [62], to integrate NFP aggregation;
- Define protocols on how to populate and update the NFPs from measurements and monitoring, in order to keep up with the changing/dynamic nature of NFPs;
- Investigate the NFP catalogue usage and integration with mashups [21] and REST;
- Reassess the definition of availability in a dependability context, by revisiting conventional notions of MTTF and MTTR;
- Define a more precise and complete ontology of service failure modes; and
- Extend the GRL model of consumer concerns (for service selection) to consider non-functional requirements beyond the NFPs of the services, taking non-technical issues into account such as the financial status of the company, Internet virus threat levels, overloaded networks, and customer satisfaction.

References

[1] Accelware: *Unit Conversion Tool 5.1*, online at http://www.sciencemadesimple.net/units.html (last accessed September 2012).

[2] Agarwal, V., and Jalote, P.: Enabling end-to-end support for non-functional properties in web services. *IEEE SOCA*, 2009. IEEE CS, 1-8.

[3] Amyot, D.: Introduction to the User Requirements Notation: Learning by Example. *Computer Networks*, Vol. 42(3), 2003, 285-301.

[4] Amyot, D., Becha, H., Bræk, R., and Rossebø, J.E.Y.: Next Generation Service Engineering. *ITU-T Innovations in NGN Kaleidoscope Conference,* Geneva, Switzerland, May 2008. IEEE Computer Society, 195-202.

[5] Amyot, D. and Mussbacher, G.: User Requirements Notation: The First Ten Years, The Next Ten Years. Invited paper, *Journal of Software (JSW)*, Vol. 6, No. 5, Academy Publisher, May 2011, 747-768.

[6] Baida, Z.: *Software-aided Service Bundling - Intelligent Methods & Tools for Graphical Service Modeling*, PhD thesis, Vrije Universiteit Amsterdam, May, 2006

[7] Balfagih, Z., and Hassan, M.F.: Quality Model for Web Services from Multi-Stakeholders' Perspective. *Information Management and Engineering (ICIME'09)*. Kuala, Lumpur, 2009. IEEE CS, 287-291.

[8] Barbir, A., Hobbs, C., Bertino, E., Hirsch, F., and Martino, L.: Chapter 14: Challenges of Testing Web Services and Security in SOA Implementations. Baresi, L.; Di Nitto E. *Test and Analysis of Web Services*, Springer, 2007.

[9] Bass, L. and Bonnie J.: Linking usability to software architecture patterns through general scenarios. *Journal of Systems and Software*, 66 (3), June 2003, 187-197.

[10] Becha, H. and Amyot, D.: Non-Functional Properties of Services in SOA – Consumer's Perspective. Survey questionnaire and summary of answers, 2010. http://www.eecs.uottawa.ca/~damyot/pub/NFP4SOA/

[11] Becha, H. and Amyot, D.: Non-Functional Properties in Service Oriented Architecture – A Consumer's Perspective. *Journal of Software (JSW)*, Vol. 7, No. 3, Academy Publisher, March 2012, 575-587.

[12] Becha, H., Mussbacher, G., and Amyot, D.: Modeling and Analyzing Non-Functional Requirements in Service Oriented Architecture with the User Requirements Notation. Chapter in: N. Milanovic (Ed.) *Non-functional Properties in Service Oriented Architecture: Requirements, Models and Methods*. IGI Global, 2011, 48-72.

[13] Brüning, A., Hölker, F., and Wolter, C.: Towards the Aggregation of Security Requirements in Cross-Organisational Service Compositions. *Aquatic Sciences - Research Across Boundaries*, 2011, Vol. 73, Number 1, 143-152

[14] Cerami, E.: *WSDL essentials: Distributed Applications with XML-RPC, SOAP, UDDI & WSDL*, O'reilly chapter available on-line at http://oreilly.com/catalog/webservess/chapter/ch06.html (Last accessed September 2012)

[15] Chen, Y.-p., Li Z., Jin Q., and Wang C.: Study on QoS Driven Web Services Composition. *APWeb 2006*, LNCS 3841, 2006. Springer, 702-707.

[16] Chen, Z., Lianf-Tien, C., Silverajan, B., and Bu-Sung, L.: UX— an architecture providing QoS-aware and federated support for UDDI, *Proceedings of the 2003 International Conference on Web Services* (ICWS'03), Las Vegas, USA, June 2003. CSREA Press, 171-176.

[17] Choi, S.W., Her, J.S., and Kim, S.D.: QoS Metrics for Evaluating Services from the Perspective of Service Providers. *IEEE International Conference on e-Business Engineering (ICEBE 2007)*, Hong Kong, China, Oct. 2007. IEEE Computer Society, 622-625.

[18] Cycorp. *Quantity Vocabulary*. On-line at: http://www.cyc.com/cycdoc/upperont-diagram.html (last accessed: September 2012)

[19] Czajkowski, K., Dan, A., Rofrano, J., Tuecke, S., and Xu, M.: *Agreement-based grid service management (OGSI-Agreement)*, Version 0, Global Grid Forum, June 2003.

[20] DAML, *OWL-S: Semantic Markup for Web Services*, on-line at: http://www.daml.org/services/, last accessed October, 2010.

[21] Daniel, F., Matera, M., Weiss, M.: Next in Mashup Development: User-Created Apps on the Web. *IT Professional*, Volume 13 (5), IEEE, September 2011

[22] Dobson, G., Hall, S., and Kotonya, G.: A Domain-Independent Ontology for Non-Functional Requirements. *IEEE International Conference on e-Business Engineering (ICEBE)*, 2007. IEEE CS, 563-566.

[23] Dobson, G., Lock, R., and Sommerville, I.: QoSOnt: a QoS ontology for service-centric systems. *Software Engineering and Advanced Applications (31st EUROMICRO)*, 2005, 80-87.

[24] *Eclipse Modeling Tools*, online at http://eclipse.org/ (Last accessed September 2012)

[25] Erl, T.: *SOA Design Patterns*, Prentice Hall/PearsonPTR. Available on-line at http://www.soapatterns.org/atomic_service_transaction.php (last accessed September 2012)

[26] Edmond, D., and O'Sullivan, J., and Hofstede, A.H.t.: Two Main Challenges in Service Description: Web Service Tunnel Vision and Semantic Myopia. *W3C Workshop on Frameworks for Semantics in Web Services*, 9-10 June 2005, Austria.

[27] Galster, M., Bucherer, E.: A Taxonomy for Identifying and Specifying Non-functional Requirements in Service-oriented Development. *Proc: 2008 IEEE Congress on Services (SERVICES 2008: Part 1)*. Honolulu, USA, 2008, 345-352.

[28] Hassine, J., Dssouli, R., and Rilling, J.: Applying Reduction Techniques to Software Functional Requirement Specifications. *System Analysis and Modeling - Fourth International SDL and MSC Workshop*, SAM 2004, Ottawa, Canada. LNCS 3319, Springer, 2005, 138-153.

[29] Hevner, A.R., and Chatterjee, S.: *Design Research in Information Systems*. Integrated Series in Information Systems, Volume 22, Springer, 2010.

[30] Hobbs, C., Becha, H., and Amyot, D.: Failure Semantics in a SOA Environment; *3rd Int. MCeTech Conference on eTechnologies,* Montréal, Canada. IEEE Computer Society, January 2008, 116-121.

[31] Hobbs, C. and Storrie, J.: Time-sensitive Service-Oriented Architectures. *Nortel Technical Journal*, 5(3), 2006, 20-43.

[32] Hondo, M., and Kaler (Eds.) C.: *Web services policy framework (WS-Policy)*, version 1.0, BEA/IBM/Microsoft/SAP, ftp://www6.software.ibm.com/software/developer/library/ws-policy.pdf, December 2002.

[33] Hughes C., and Hillman, J.: QoS Explorer: A Tool for Exploring QoS in Composed Services. *International Conference on Web Services (ICWS '06)*, 2006, 797-806.

[34] innoQ Deutschland GmbH: *An overview of the Web services standards landscape as of Q1 2007*. http://www.innoq.com/soa/ws-standards/poster/ (last accessed September 2012).

[35] International Telecommcunication Union, Telecommunication Standardization Sector (ITU-T). http://www.itu.int/ITU-T/ (last accessed September 2012).

[36] ISO: *Country Code List: ISO 3166-1993*, on-line at http://xml.coverpages.org/country3166.html (last accessed September 2012).

[37] ISO: *ISO 4217 Currency Code List*, available on-line at http://www.xe.com/iso4217.php - (Last accessed September 2012).

[38] ISO: *Software Engineering - Product Quality - Part 1: Quality Model*. ISO/IEC 9126-1, June, 2001.

[39] ITU-T: *Recommendation E.802, Framework and methodologies for the determination and application of QoS parameters*, Geneva, Switzerland, 2007.

[40] ITU-T: *Recommendations, series E: Overall Network Operation, Telephone Service, Service Operation and Human Factors*, Geneva, Switzerland.

[41] ITU-T: *Recommendation Z.150 (02/11), User Requirements Notation (URN) – Language Requirements and Framework*. Geneva, Switzerland, 2003.

[42] ITU-T: *Recommendation Z.151 (11/08), User Requirements Notation (URN) – Language definition*. Geneva, Switzerland, November 2008.

[43] Kealey, J. and Amyot, D.: Enhanced Use Case Map Traversal Semantics. *13th SDL Forum (SDL'07)*, Paris, France. LNCS 4745, Springer, 2007, 133-149.

[44] Kim, E. and Lee, Y.: *Quality Model for Web Services*, version 1.0, OASIS, October 2011.

[45] *jUCMNav*, Version 5.1, http:// softwareengineering.ca/jucmnav/, University of Ottawa (last accessed September 2012).

[46] Lall, M., Venter, L.M. and van der Poll, J.A.: Evaluating the Second Generation Web Services Specifications for Satisfying Non-Functional Requirements. *Proc. of World Conf. on E-Learning in Corporate, Government, Healthcare, and Higher Education*, Chesapeake, VA: AACE, 2010, 1919-1929.

[47] Lamanna, D.D., Skene, J., and Emmerich, W.: SLAng: a language for defining service level agreements, *Proceedings of the Ninth IEEE Workshop on Future Trends in Distributed Computing Systems* (FTDCS), Puerto Rico, May 2003. IEEE-CS Press, 100-106.

[48] Laun, W.: *A Java Architecture for XML Binding (JAXB) Tutorial*, http://jaxb.java.net/tutorial/ (Last accessed September 2012).

[49] Liu, Y., Ngu, A.H.H., and Zeng, L.: QoS Computation and Policing in Dynamic Web Service Selection, *Proceedings 13th Int'l Conf. World Wide Web (WWW)*, May 2004. ACM Press, 66-73.

[50] Ludwig, H., Keller, A., Dan, A., King, R.P., and Franck, R.: *Web service level agreement (WSLA) language specification*, version 1.0, Revision wsla-2003/01/28, International Business Machines Corporation (IBM), available online at: http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf, 2003.

[51] Luo, H., and Amyot, D.: Towards a Declarative, Constraint-Oriented Semantics with a Generic Evaluation Algorithm for GRL. *5th International i* Workshop (iStar 2011)*, Trento, Italy, August 2011. CEUR-WS, Vol-766, 26-31.

[52] Milanovic, N., Milic, B., and Malek, M.: Modeling Business Process Availability. *Proc. of the IEEE Int. Workshop on Methodologies for Non-functional Properties in Services Computing*, at SCC 2008, Honolulu, USA, 2008, 315-321.

[53] Mohiuddin, S.: The Business Technology Drivers and Benefits of SOA, *BPTrends*, November, 2007.

[54] Mukherjee, D., Jalote, P., and Nanda, M. G.: Determining QoS of WS-BPEL Compositions. *Proc. 6th Int. Conf. on Service-Oriented Computing*. LNCS 5364, 2008. Springer, 378-393.

[55] Mussbacher, G.: *Aspect-oriented User Requirements Notation*. PhD thesis, School of Information Technology and Engineering, University of Ottawa, Canada, 2010.

[56] Mussbacher, G. and Amyot, D.: Assessing the Applicability of Use Case Maps for Business Process and Workflow Description. *3rd Int. MCeTech Conference on eTechnologies*, Montréal, Canada, 2008. IEEE Computer Society, 219-222.

[57] Mussbacher, G., Amyot, D., and Weiss, M.: Visualizing Early Aspects with Use Case Maps. In Rashid, A. & Aksit, M. (Eds.), *Transactions on Aspect-Oriented Software Development III*, Springer, 2007, 105-143.

[58] Mussbacher, G., Amyot, D., Araújo, J., Moreira, A., and Weiss, M.: Visualizing Aspect-Oriented Goal Models with AoGRL. *Second International Workshop on Requirements Engineering Visualization (REV'07)*, New Delhi, India, 2007.

[59] Mussbacher, G., Kienzle, J., and Amyot, D.: Transformation of Aspect-oriented Requirements Specifications for Reactive Systems into Aspect-oriented Design Specifications. *1st Model-Driven Requirements Engineering Workshop (MoDRE 2011)*, Trento, Italy, 2011. IEEE CS, 39-47. DOI:10.1109/MoDRE.2011.6045365

[60] OASIS: *Reference Model for Service Oriented Architecture 1.0 OASIS Standard*, 12 October 2006. http://www.oasis-open.org/specs/index.php#soa-rmv1.0 (Last accessed September 2012).

[61] Organization for the Advancement of Structured Information Standards (OASIS). http://www.oasis-open.org/home/index.php (last accessed September 2012).

[62] OASIS: *Web Services Business Process Execution Language (WSBPEL)*. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel (last accessed September 2012).

[63] OASIS: *Universal Description, Discovery and Integration (UDDI)*. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=uddi-spec. (last accessed September 2012).

[64] OMG: *A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems*, version 1.0. OMG Adopted Specification. OMG Document Number ptc/2009-11-02, 2009. http://www.omg.org/spec/MARTE/1.0/PDF/

[65] OMG: *Handling of Non-Functional Properties in SOA*. Draft Request for Information, Telecommunication Platform Special Interest Group, October 2008. mars/08-12-26

[66] O'Sullivan, J., Edmond, D., and Hofstede, A.H.t.: *Formal description of non-functional service properties*. Technical FIT-TR-2005-01, Queensland University of Technology, Brisbane (2005). Available on-line at http://www.citi.qut.edu.au/about/research_pubs/technical/non-functional.jsp, (Last accessed September 2012).

[67] O'Sullivan, J., Edmond, D., and Hofstede, A.H.t.: What's in a Service? Towards Accurate Description of Non-Functional Service Properties. *Distributed and Parallel Databases*, 12, 2002, 117-133.

[68] Parallel Understanding Systems Group (Department of Computer Science, University of Maryland at College Park): *Measurement Ontology 1.0 (draft)*. Available on-line at: http://www.cs.umd.edu/projects/plus/SHOE/onts/measure1.0.html (last accessed September 2012)

[69] Parimala, N., and Saini, A.: Web service with criteria: Extending WSDL. *Digital Information Management (ICDIM)*, Melbourne, Australia, 2011. IEEE CS, 205-210.

[70] Rajbhandari, S., and Walker W. D.: Incorporating Provenance in Service Oriented Architecture. *Proc. Int. Conf. on Next Generation Web Services Practices*, IEEE Computer Society, Washington, DC, USA, 2006. IEEE CS, 33-40.

[71] Roy, J.-F., Kealey, J., and Amyot, D.: Towards Integrated Tool Support for the User Requirements Notation. *SAM 2006: Language Profiles - Fifth Workshop on System Analysis and Modelling*, Kaiserslautern, Germany. LNCS 4320, Springer, 2006, 183-197.

[72] Sahai, A., Durante, A., and Machiraju, V.: *Towards automated SLA management for web services*, Research Report HPL-2001-310 (R.1), Hewlett-Packard (HP) Laboratories, Palo Alto, USA. Available on-line at: http://www.hpl.hp.com/techreports/2001/HPL-2001-310R1.pdf, July 26, 2002.

[73] Sanders, R.T., Bræk, R., Bochmann, G.v., and Amyot, D.: Service Discovery and Component Reuse with Semantic Interfaces. A. Prinz, R. Reed, and J. Reed (Eds.) *12th SDL Forum (SDL 2005)*, Grimstad, Norway, June 2005. LNCS 3530, Springer, 85-102.

[74] Sun Microsystems: *The Java Web Services Tutorial*, 2005 http://download.oracle.com/docs/cd/E17802_01/webservices/webservices/docs/1.6/tutorial/doc/index.html (Last accessed September 2012)

[75] Taylor J. R.: *Introduction to uncertainty and propagation of uncertainty*. Available on-line at: http://www.phy.olemiss.edu/~herice/files/error_analysis.pdf (Last accessed September 2012)

[76] Tosic, V., Esfandiari, B., Pagurek, B, and Patel, K.: On requirements for ontologies in management of web services, *CAiSE '02/ WES '02 Revised Papers from the International Workshop on Web Services, E-Business, and the Semantic Web*, LNCS 2512, Springer, 237-247.

[77] Tosic, V., Pagurek, B., Patel, K., Esfandiari, B., and Ma, W.: Management applications of the web service offerings language (WSOL). *Inf. Syst.* 30, 7 (Nov. 2005), 564-586.

[78] Wada, H., Champrasert, P., Suzuki, J. and Oba, K.: Multiobjective Optimization of SLA-aware Service Composition. *Proc. of the IEEE Int. Workshop on Methodologies for Non-functional Properties in Services Computing*, at SCC 2008, Honolulu, USA, 2008. IEEE CS, 368-375.

[79] *Web Service Modeling Ontology (WSMO)*, WSMO Final Draft 13 April 2005, online at: http://www.wsmo.org/TR/d2/v1.2/D2v1-2_20050414.pdf, (Last accessed September 2012).

[80] Weibull, W.: *On-line Reliability Engineering Resources for the Reliability*. Available on-line at: http://www.weibull.com/SystemRelWeb/blocksimtheory.htm (Last accessed September 2012).

[81] Weiss, M., Esfandiari, B., and Luo, Y.: Towards a classification of web service feature interactions. *Computer Networks* 51, 2 (Feb. 2007), 359-381.

[82] Wikipedia: http://en.wikipedia.org/wiki/Accuracy (Last accessed September 2012)

[83] Wikipedia: http://en.wikipedia.org/wiki/Non-functional_requirement (Last accessed September 2012)

[84] WinterGreen Research Inc.: *Services Oriented Architecture (SOA) Middleware Market Shares, Strategies, and Forecasts, Worldwide, 2012 to 2018*, April 2012, 671 pages. Index available online at http://wintergreenresearch.com/reports/SOA%20Engines.html

[85] The World Wide Web Consortium (W3C). http://www.w3.org/ (Last accessed September 2012).

[86] W3C: *Web Services Description Language (WSDL)*, version 1.1, 2001 http://www.w3.org/TR/wsdl (Last accessed September 2012).

[87] W3C: *Simple Object Access Protocol (SOAP)*. http://www.w3.org/TR/soap/ (Last accessed September 2012).

[88] Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., and Sheng, Q.Z.: Quality Driven Web Services Composition. *Proc. of the 12th international conference on World Wide Web (WWW)*, Budapest, Hungary, May 2003. ACM Press. 411-421.

[89] Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J. and Chang, H.: QoS-aware Middleware for Web Services Composition. *IEEE Transactions on Software Engineering*. Vol. 30, No. 5, 2004, 311-327.

[90] Web Service Semantics - WSDL-S, W3C Member Submission, 2005, Version 1.0 avaialble at : http://www.w3.org/Submission/WSDL-S/ (Last accessed September 2012)

Appendix A: SOAcustomerNFP XSD Schema

This appendix contains the complete schema (`SOAcustomerNFP.xsd`) defined in this thesis, together with interactive documentation. It is available online at: http://www.eecs.uottawa.ca/~damyot/pub/SOAcustomerNFP/

Appendix B: NFP Description of Service 1

This appendix (`Service1.xml`) is also available online at:

http://www.eecs.uottawa.ca/~damyot/pub/SOAcustomerNFP/

```
<?xml version="1.0" encoding="utf-8"?>
<NFPdescription xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="SOAcustomerNFP.xsd">

        <price>
                <unit>USD</unit>
                <priceElements>
                    <type>Invocation</type>
                    <value>3</value>
                </priceElements>
        </price>

        <responseTime>
                <average>3.0</average>
                <best>2.0</best>
                <confidence>80</confidence>
                <unit>microseconds</unit>
                <worst>4.0</worst>
        </responseTime>

        <reputation>
                <reputationLevel>2.8</reputationLevel>
        </reputation>

        <availability>
                <availabilityLevel>0.85</availabilityLevel>
        </availability>

        <reliability>
                <reliabilityLevel>0.85</reliabilityLevel>
        </reliability>

        <usability>
                <usabilityLevel>4.2</usabilityLevel>
        </usability>

        <transactional>
                <isTransactional>true</isTransactional>
        </transactional>
```

```
        <jurisdiction>
                <countries>
                        <code>CN</code>
                        <territories><name>Quebec</name></territories>
                </countries>
                <countries>
                        <code>FR</code>
                </countries>

                <countries>
                        <code>US</code>
                </countries>
        </jurisdiction>

        <resource>
                <resourceReqs>
                        <category>Other</category>
                        <description> RAM</description>
                        <quantity>128</quantity>
                        <unit>MB</unit>
                </resourceReqs>

                <resourceReqs>
                        <category>StorageResource</category>
                        <description>hard disk</description>
                        <quantity>1.5</quantity>
                        <unit>GB</unit>
                </resourceReqs>

                <resourceReqs>
                        <category>TimingResource</category>
                        <description>execution time</description>
                        <quantity>16</quantity>
                        <unit>seconds</unit>
                </resourceReqs>
        </resource>

        <scalability>
                <scaleTo>2000</scaleTo>
        </scalability>

        <serverLocation>
                <countries>
                        <code>CN</code>
                        <territories><name>Quebec</name></territories>
                </countries>

                <countries>
                        <code>FR</code>
                </countries>

                <countries>
                        <code>US</code>
                </countries>
        </serverLocation>
</NFPdescription>
```

Appendix C: NFP Description of Service 2

This appendix (`Service2.xml`) is also available online at:

http://www.eecs.uottawa.ca/~damyot/pub/SOAcustomerNFP/

```
<?xml version="1.0" encoding="utf-8"?>

<NFPdescription xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="SOAcustomerNFP.xsd">

        <price>
                <unit>EUR</unit>
                <priceElements>
                    <type>Invocation</type>
                    <value>2</value>
                </priceElements>
                <priceElements>
                    <type>SubscribeMonth</type>
                    <value>25</value>
                </priceElements>
        </price>

        <responseTime>
                <average>4.0</average>
                <best>2.0</best>
                <confidence>95</confidence>
                <unit>microseconds</unit>
                <worst>7.0</worst>
        </responseTime>

        <reputation>
                <reputationLevel>4.2</reputationLevel>
        </reputation>

        <availability>
                <availabilityLevel>0.96</availabilityLevel>
        </availability>

        <reliability>
                <reliabilityLevel>0.96</reliabilityLevel>
        </reliability>

        <usability>
                <usabilityLevel>4.8</usabilityLevel>
        </usability>
```

```
        <transactional>
                <isTransactional>false</isTransactional>
        </transactional>

        <jurisdiction>
                <countries>
                        <code>CN</code>
                </countries>

                <countries>
                        <code>FR</code>
                </countries>
        </jurisdiction>

        <resource>
                <resourceReqs>
                        <category>StorageResource</category>
                        <description>hard drive</description>
                        <quantity>1</quantity>
                        <unit>GB</unit>
                </resourceReqs>


                <resourceReqs>
                        <category>TimingResource</category>
                        <description>the time needed</description>
                        <quantity>29</quantity>
                        <unit>sec</unit>
                </resourceReqs>
        </resource>

        <scalability>
                <scaleTo>5000</scaleTo>
        </scalability>

        <serverLocation>
                <countries>
                        <code>CN</code>
                </countries>

                <countries>
                        <code>FR</code>
                </countries>
        </serverLocation>

</NFPdescription>
```

Appendix D: NFP Description of Service 3

This appendix (`Service3.xml`) is also available online at:

http://www.eecs.uottawa.ca/~damyot/pub/SOAcustomerNFP/

```
<?xml version="1.0" encoding="utf-8"?>
<NFPdescription xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="SOAcustomerNFP.xsd">

    <price>
        <unit>USD</unit>
        <priceElements>
            <type>Invocation</type>
            <value>5.775599956512451</value>
        </priceElements>
        <priceElements>
            <type>SubscribeMonth</type>
            <value>34.69499945640564</value>
        </priceElements>
    </price>

    <responseTime>
        <average>0.0</average>
        <best>0.0</best>
        <confidence>80.0</confidence>
        <unit>microseconds</unit>
        <worst>0.0</worst>
    </responseTime>

    <reputation>
        <reputationLevel>2.8</reputationLevel>
    </reputation>

    <availability>
        <availabilityLevel>0.816</availabilityLevel>
    </availability>

    <reliability>
        <reliabilityLevel>0.816</reliabilityLevel>
    </reliability>

    <usability>
        <usabilityLevel>4.2</usabilityLevel>
    </usability>
```

```
    <transactional>
        <isTransactional>false</isTransactional>
    </transactional>

    <jurisdiction>
        <countries>
            <code>CN</code>
            <territories>
                <name>Quebec</name>
            </territories>
        </countries>
        <countries>
            <code>FR</code>
        </countries>
    </jurisdiction>

    <resource>
        <resourceReqs>
            <category>Other</category>
            <description> RAM</description>
            <quantity>128.0</quantity>
            <unit>MB</unit>
        </resourceReqs>
        <resourceReqs>
            <category>StorageResource</category>
            <description>hard disk</description>
            <quantity>2.5</quantity>
            <unit>GB</unit>
        </resourceReqs>
        <resourceReqs>
            <category>TimingResource</category>
            <description>execution time</description>
            <quantity>45.0</quantity>
            <unit>seconds</unit>
        </resourceReqs>
    </resource>

    <scalability>
        <scaleTo>2000</scaleTo>
    </scalability>

    <serverLocation>
        <countries>
            <code>CN</code>
        </countries>
        <countries>
            <code>FR</code>
        </countries>
        <countries>
            <code>US</code>
        </countries>
    </serverLocation>

</NFPdescription>
```

Appendix E: Java Source Code

This appendix contains the complete source code for:

- Java classes generated from the XSD schema via JAXB (in package `seg.jUCMNav.nfp.generated`)
- Java implementation of NFP aggregation algorithms (in package `NFP.Algorithms`)
- All source code zipped (`src.zip`)

It is available online at: http://www.eecs.uottawa.ca/~damyot/pub/SOAcustomerNFP/