COLLABORATIVE AND CORROBORATIVE SEMANTIC WEB SERVICE MONITORING

Mokone Ishmael Makitla

Submitted in accordance with the requirements for the degree of
DOCTOR OF PHILOSOPHY

in the subject

COMPUTER SCIENCE

Supervisor: **DR. J.S. MTSWENI**

University of South Africa

October 2022

Abstract

Service Oriented Computing has emerged as a promising computing paradigm for Internet-scale distributed applications built around services as primary building blocks. Such Internet-scale computing relies heavily on the connectivity infrastructure because existing business functionalities could be made accessible through this infrastructure. Furthermore, the services, as building blocks of these Internet-scale applications, may be developed and managed autonomously by enterprises. Different organizations may provide similar functionalities and it therefore becomes a matter of differentiation to support negotiable quality of service characteristics such as response time, cost, throughput, among others. Service consumers consider these non-functional characteristics of the published services when selecting which service to use; so there is an economic aspect to quality of service (QoS). The negotiated QoS characteristics often involve penalties, making it absolutely critical for the service provider to detect and correct any deviations from the agreed-upon service behaviour.

The need to detect potential failures has resulted in several research activities dedicated to service monitoring, specifically, Semantic Web Service Monitoring. Monitoring entails detecting and signaling whether the participating services behave consistently with the expected functionality and non-functional service properties. Monitoring may further entail enacting corrective mechanisms when there are failures or breaches to agreements by contracted services. Many approaches for both the service-consumer-side and service-provider-side monitoring have been proposed, including the use of dedicated monitoring infrastructure. Contemporary monitoring approaches rely on the service performance data being collected separately by the service provider for service-side monitoring and the consumer for client-side monitoring.

As of this writing and according to our knowledge, no monitoring approach had been developed that facilitates the collaborative and corroborative exchange of monitoring information by both the service consumer and the service provider. The exchange of monitoring information is significant because: (1) the service consumer and the service provider may have different perspectives of the same QoS parameter and collaboration can help develop consensus, (2) enables the support for a flexible quality-based pricing model, which is a potential competitive advantage for service providers, (3) it has a built-in self-checking mechanism so that there is no need for costly incen-

tive schemes to encourage honest reporting, (4) there is no need for costly dedicated infrastructure for monitoring by surveillance because service consumer and service provider exchange their context-dependent monitoring information directly. Having made a case for a collaborative and corroborative monitoring approach, this study hypothesizes that collaborative and corroborative monitoring of semantic web services is a viable alternative approach to client-side, service-side and 3rd-party/dedicated monitoring infrastructure. Therefore, the main objective of this study is to investigate, design, develop, and evaluate a collaborative and corroborative monitoring technique for Semantic Web Services. Design Science Research (DSR) methodology is adopted as it is particularly suitable for research studies that, like this study, aim to design an artefact. The significance of this work is underpinned by the critical importance of effective monitoring for the practical application of Service Oriented Computing technologies, specifically the Semantic Web Services. This study is critical because it purports to introduce a novel and practical approach to Semantic Web Service monitoring that is corroborative and collaborative between a service consumer and a service provider.

In pursuance of the collaborative and corroborative monitoring, the study developed a Generalized Response Time Metric (GRTM), a consensus-based generalized metric for response time QoS parameter. The study further developed a technology-agnostic **M**onitoring **I**nformation e**X**change (MIX) protocol to facilitate the exchange of performance data which is described in terms of the GRTM. The study also found, and demonstrated, that the support for, and implementation of collaborative and corroborative monitoring for Semantic Web Services is technically feasible. This is made possible by implementing a monitoring information exchange mechanism based on the MIX protocol.

The proposed solution has been evaluated in terms of its technical feasibility. There were no other implementations of collaborative or corroborative monitoring of Semantic Web Services at the time of the execution of the research and consequently, there was no need for comparative analysis. The study demonstrated the technical feasibility of the proposed collaborative and corroborative monitoring technique for Semantic Web Service. This was achieved through a reference implementation of the MIX protocol.

Finally, the study makes three recommendations for consideration as future research work and these are based on the identified limitations of the study. Firstly, developing the correct syntax for the semantic description of the mathematical expressions of all the parameters that characterize the per-

formance of a Semantic Web Service such as *latency*, *response time*, *throughput*, and *error-rate*. An expressive ontology model for Semantic Web Service performance needs to include the logical expressions of these performance parameters; the study focused only on *response time*. Secondly, a Semantic Web Service Monitoring tool based on the MIX protocol should be developed further to provide extensible interfaces that software engineers can implement to support MIX functionality. The third recommendation is for software engineers and researchers to explore possible Service Level Agreement (SLA) negotiation strategies and implement these as part of the post-execution processes of the MIX protocol.

Keywords: Semantic Web Services Monitoring, Service Oriented Computing, Ontologies, Quality of Service, Monitoring Information Exchange, Collaborative Monitoring

Dedication

To my children Serote RebaOne and Ntsoe OakangOne, and their mother Moshidi Phora for their enduring love. They are the best of all things I have in Life.
To family and friends...I hope this also brings honor to your names!
In memory of my two loving grandmothers Mrs Mothulwa Makitla and Mrs Maletsebe Ruth Mathebe forever dear to my heart even in death. Your loving grace has sustained me even years after both your passing.

Declaration

I **Mokone Ishmael Makitla** declare that the thesis titled **COLLABORATIVE AND CORROBORATIVE SEMANTIC WEB SERVICE MONITORING** is my own work, and that all sources used or quoted in the study have been indicated by way of citations and acknowledged by means of complete references in the bibliographical entries.

Acknowledgements

I want to thank the Council for Scientific and Industrial Research (CSIR) who supported me from the time I started my Masters degree and the first 3 years of my PhD journey. I also want to thank UNISA for granting me a bursary for over 2 years of my PhD journey. Finally I would like to acknowledge and appreciate the support from Rand Merchant Bank (RMB) for financing the last stages of my studies. Without the backing and support of these respected South African institutions, my PhD journey would not have been possible.

A special mention to my advisor and friend Dr. Jabu Mtsweni for guiding me throughout my PhD studies, always providing helpful and timely feedback on my drafts and for the many meaningful engagement we have had both during his own PhD and mine.

Contents

List of Figures

List of Tables

List of Abbreviations

ACK	Acknowledgement
API	Application Programmer Interface
AUP	Acceptable Use Policies
ASC	Autonomous Service Composition
CA	Customer Agreements
CNP	Contract Net Protocol
CSA	Cloud Service Agreements
CSCC	Cloud Standards Customer Council
CSV	Comma Separated Values
DL	Description Logic
DSR	Design Science Research
FLOWS	First Order Logic Ontology for Web Services
FIPA	Foundation for Intelligent Physical Agents
FOL	First Order Logic
GRTM	Generalized Response Time Metric
HTTP	Hyper Text Transfer Protocol
ICNP	Iterated Contract Net Protocol
IEW	Information Entropy Weight
IgS-wBSRM	Information gain and Sliding window-based weighted naive BayeSian Runtime Monitoring
JADE	Java Agent DEvelopment
JAMon	Java-based Monitor
JSON	Javascript Simple Object Notation
MIX	Monitoring Information eXchange
MMEL	Message Encoding Layer
NACK	Negative Acknowledgement
NLP	Natural Language Processing

OWL	Ontology Web Language
OWL-Q	QoS-based Web Service Description based on Web Ontology Language
OWL-S	Web Ontology Language with Semantics
PAYU	Pay-As-You-Use
QoS	Quality of Service
QoSOnt	QoS Ontology
RDF	Resource Description Framework
REST	REpresentation State Transfer Protocol
ROWS	Rules Ontology for Web Services
SaaS	Software as a Service
SARA	Semantic Attribute Reconciliation Architecture
SAWSDL	Semantic Annotation of Web Service Description Language
SEREDASj	SEmantic REstful DAta SErvices using JSON
SIML	Session Initialization and Management Layer
SLA	Service Level Agreement
SMOnt	Service Monitoring Ontology
SMW	Semantic Media Wiki
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SOC	Service Oriented Computing
SOS	Service Oriented System
SPARQL	Spark Query Language
SSE	Service Selection Effort
SWS	Semantic Web Service
SWSF	Semantic Web Service Framework
SWSL	Semantic Web Service Language
SWSL-FOL	Semantic Web Service Language based on First Order Logic
SWSM	Semantic Web Service Monitoring
SWSO	Semantic Web Service Ontology
TCP/IP	Transmission Control Protocol/ Internet Protocol
TDB	Triples Database
TLS	Transport Layer Security
UDDI	Universal Description, Discovery and Integration
URI	Universal Resource Identification
URL	Universal Resource Locator

W3C	World Wide Web Consortium
WoT	Web of Things
WS-Agreement	Web Service Agreement
WS-QoSOnto	Web Service QoS Ontology
WSC	Web Service Composition
WSDL	Web Service Description Language
WSDL-S	Web Service Description Language with Semantics
WSLA	Web Service Level Agreement
WSML	Web Service Modeling Language
WSMO	Web Service Modeling Ontology
WSMO-QoS	QoS extension to the Web Service Modeling Ontology
XHTML	Extensible Hyper Text Transfer Protocol
XML	Extensible Modeling Language

List of Publications

List of Peer-Reviewed Publications

- Makitla, I., & Mtsweni, J. (2014). Towards a collaborative approach to Web Service monitoring: In appreciation of connectivity challenges in Africa. In 2014 IST-Africa Conference Proceedings (pp. 1–9). IEEE. doi:10.1109/ISTAFRICA.2014.6880660

- Makitla, I., & Mtsweni, J. (2015). A generalized web service response time metric to support collaborative and corroborative web service monitoring.In: 2015 10th International Conference for Internet Technology and Secured Transactions (ICITST) (pp. 373–377). IEEE. ISBN: 978-1-9083-2052-0. doi:10.1109/ICITST.2015.
  URL: http://ieeexplore.ieee.org/document/7412124/.

List of Other Publications and Presentations

- Makitla, I., & Mtsweni, J. (2016). Experimental Architecture for Collaborative and Corroborative Monitoring of Semantic Web: Technical Report on PhD Work. CSIR.

Chapter 1

Introduction

A service is a computer program that is self-describing and provides a well-defined set of functionality and is accessible through the connectivity infrastructure such as computer networks and the Internet (Papazoglou, Traverso, Dustdar, and Leymann, 2008). Several such services can be combined loosely to create highly complex and distributed applications. Service Oriented Computing (SOC) or service orientation (Papazoglou, Traverso, Dustdar, and Leymann, 2008) is a paradigm that uses these services as building blocks for developing distributed applications at Internet-scale (Sedayao, 2008). Furthermore, the services, as building blocks of these Internet-scale applications, may be developed and managed autonomously by enterprises. Since different organizations may provide similar functionalities, it becomes a matter of differentiation to support negotiable quality of service characteristics such as response time, cost, throughput, and so forth. Service consumers consider these non-functional characteristics of the published services when selecting which service to use; hence, there is an economic aspect to quality of service (QoS). The negotiated QoS characteristics often involve penalties, making it absolutely critical for the service provider to detect and correct deviations from the agreed-upon service behaviour.

The need for ability to detect potential failures necessitates service monitoring. Service monitoring entails detecting and signaling whether the participating services behave consistently with the expected functionality and non-functional service properties. Monitoring may further entail the enactment of corrective mechanisms when there are failures or breaches to agreements by contracted services. A number of approaches for service-consumer-side and service-provider-side monitoring have been proposed, including the use of

dedicated monitoring infrastructure. Contemporary monitoring approaches rely on the service performance data being collected separately by the service provider for service-side monitoring and the consumer for client-side monitoring.

1.1 Research Problem

This study tackles the challenge of achieving the collaborative and corroborative monitoring of Semantic Web Service Monitoring within the Service Oriented Computing domain. Semantic Web Service Monitoring entails detecting and signalling whether the participating services behave consistently with the expected functionality and that it exhibits the guaranteed non-functional service properties. Semantic Web Service Monitoring may further entail enacting corrective mechanisms when there are failures or breaches of agreements by contracted services.

However, the problem is that current monitoring approaches do not facilitate collaborative and corroborative exchange of monitoring information between the service consumer and the service provider. Collaborative means that both service consumer and service provider are aware of each other's monitoring activities and also that the interaction between them is based on sending to and receiving monitoring information from each other. Corroborative means that each claim pertaining to an individual QoS parameter is checked by the receiving party; as a result, there is generally a comparison of notes by service consumers and service providers; The exchange of monitoring information is vital because of the following reasons:

- Since both the service consumer and the service provider may have different perspectives of the same QoS parameter there is no consensus on the meaning of individual QoS parameters and therefore a great potential for misunderstanding in case of SLA breach.
- Enables the support for a flexible pricing model, which is a potential competitive advantage for service providers; consumers feel empowered to ensure that they receive the appropriate quality of service for the money they pay. This is a desirable alternative to paying penalties; the service provider may negotiate with the consumer to rather pay an

amount appropriate for the quality of service experienced. This Pay-As-You-Use (PAYU) scenario is only possible where both the consumer and the service provider can corroborate their monitoring data and collaboratively renegotiate conditions of service.

- When both the service consumer and service provider exchange and corroborate each other's monitoring information, there is no need for incentive schemes, such as what Artaiam and Senivongse (2008) proposed, aimed at encouraging honest reporting.

- There is no need for dedicated infrastructure for monitoring by surveillance because service consumer and service provider exchange their environment-dependent monitoring information directly.

In view of the research problem as stated in the preceding paragraphs, this study advances the thesis statement as described next in Section 1.2.

1.2 Thesis Statement and Delineation

Collaborative and corroborative monitoring of Semantic Web Services is a viable alternative approach to client-side, service-side and 3rd-party/dedicated monitoring infrastructure.
A thesis statement, by its nature, is an assertion, a hypothetical proposition made without any proof (Hofstee, 2006), in which case the object of the research is to present such proof. Therefore, it is crucial to set out the conditions under which the thesis statement holds. This is necessary as it sets out exactly what the researcher means to investigate (Hofstee, 2006, .p 26). What follows next is the delineation of the thesis statement:

- **Viability** – the assertion that a collaborative and corroborative approach to Semantic Web Service monitoring is a viable alternative is only in terms of (1) transparency of the monitoring process, (2) potential savings on infrastructural costs, (3) support for flexible pricing model, and (4) eliminating the need to incentivise honest reporting by service consumers.

1.3 The Proposed Solution

In order to understand the proposed solution, it is essential first to establish what key requirements are imposed by the need to support the exchange of monitoring information. The requirements are explained below:

- Consensus on the meaning of monitoring values – that is, the consensus on observable QoS parameters;
- Mechanism by which monitoring information can be exchanged – that is, an event-based exchange of monitoring information between parties (service consumer and service provider) during service execution;
- Mechanism by which parties can negotiate resolutions in case of violation or conflict – that is, some technology-independent negotiation protocol;

To satisfactorily support the assertion made in the thesis statement regarding viability, this study sought to investigate, design, develop and evaluate a Collaborative and Corroborative Semantic Web Service Monitoring approach. In order to achieve this objective and address the key requirements imposed by the support for exchanging monitoring information, it is necessary to achieve the following research objectives (RO's):

- Develop generalized metrics for specific QoS parameters so that service consumers and service providers can reach a consensus on the meaning of each QoS parameter **(RO-1)**;
- Develop a Monitoring Information eXchange (MIX) protocol - a technology-independent protocol for service consumers and service providers to exchange monitoring information **(RO-2)**;

The above research objectives directly and completely attend to the key requirements discussed earlier in this section. In addition, however, there are other objectives that pertain specifically to demonstrating the technical feasibility of the proposed Collaborative and Corroborative Semantic Web Service Monitoring approach. The emphasis of this study is not on the technology solution and as such the technology demonstrations are not expected to be exhaustive and are meant only to demonstrate the possibility of implementing the proposed solution using existing technological tools. The following objectives pertain to the technical feasibility of Collaborative and Corroborative Semantic Web Service Monitoring:

- Develop a Semantic Web Service Monitoring tool as a prototype to demonstrate the value of collaborative and corroborative utility in Semantic Web Service Monitoring **(RO-3)**;
- Evaluate the technical feasibility of the Semantic Web Service Monitoring prototype **(RO-4)**;

By completing the above tasks, the study will be making conceptual as well as technical contributions.The conceptual and technical contributions made by the study are outlined in Section 1.4 next.

1.4 Research Contributions

Based on the proposed solution, the study makes the following conceptual and technical contributions:

- **Conceptual Contribution 1**: Generalized Response Time Metric (GRTM) - collaborative measurement of response time values measured at both client and service sides.
- **Conceptual Contribution 2**: Monitoring Information eXchange (MIX) Protocol - a mechanism for exchanging monitoring information between participating monitoring agents.
- **Technical Contribution 1**: A technology-agnostic architecture for a Collaborative and Corroborative Semantic Web Service Monitoring.
- **Technical Contribution 2**: Semantic Web Service Monitoring prototype which illustrates how to implement aspects of the Monitoring Information eXchange (MIX) protocol.

1.5 Research Methodology

This section presents the research methodology adopted for this study.

According to Welman and Kruger (2004, p. 2), research involves applying various techniques and methods to create scientifically obtained knowledge. Research methodology specifies how the research is carried out in terms of research design, how research effort is measured and what constitutes success.

Research design ensures that the evidence obtained through the research process enables the researcher to present compelling evidence supporting the hypothesis advanced by his or her research.

Olivier (2004, p. 12) advises that research studies with technical objectives, such as this study, ought to apply creative or constructive research methods intended to devise new mechanisms for use in computing. The quality of such creative or constructive research methods is measured in terms of the utility of what is created (March and Smith, 1995; Olivier, 2004).

Given the above, the most suitable research methodology for this genre of study is the Design Science Research method (Hevner et al., 2004; March and Storey, 2008). The Design Science Research (DSR) methodology, according to Peffers et al. (2007), comprises three key elements:

- Conceptual principles that define what constitutes DSR,
- Practice rules of how to carry out acceptable DSR, and
- Process for carrying out and presenting DSR.

The above elements informed the manner in which this study was conducted.

In terms of the conceptual principles, DSR is understood by March and Smith (1995, p. 253) as being about developing artefacts to attain specific goals such as, in the case of this study, the monitoring of Semantic Web Services. In keeping with this view, and through literature synthesis, as well as Carlsson et al. (2010, p. 126) note that the definition of DSR encapsulates any designed object that embeds a solution to an understood research problem. According to Reeves (2006, p. 52) DSR seeks to investigate the development of solutions that target practical problems, culminating in the identification of design patterns or re-usable design principles. In the case of this study, the solution should culminate in the design principles for a collaborative and corroborative monitoring mechanism for Semantic Web Services.

For the practice rules, Hevner et al. (2004) offer seven guidelines for conducting DSR. Venable (2010) provides a detailed review of the level of consensus within the research community on these guidelines and which indicates a broader acceptance by the community.

Literature on DSR (Hevner et al., 2004; Ellis and Levy, 2010; OATES, 2006) warns specifically regarding research contributions that there must be a clear differentiation between a routine system development and DSR. According to Hevner et al. (2004, p. 81), the key difference between routine

system development and design research is the clear articulation of a contribution to the body of knowledge. In the case of this study, the contributions are listed in Section 1.4 and are discussed further in the subsequent chapters of this thesis.

Finally, the process of conducting DSR is presented by Peffers et al. (2007) by way of a DSR research process model. As will be shown later in this section, this study followed the research process model with adaptions where appropriate. In view of this adaptation of the Design Science Research paradigm, the subsequent sections describe how Design Science Research was executed in this study in terms of practice rules of how to carry out acceptable Design Science Research, and the acceptable process for carrying out and presenting Design Science Research.

Figure 1.1 depicts how the study aligned its research activities according to the seven guidelines for conducting Design Science Research outlined by Hevner et al. (2004, p. 81).

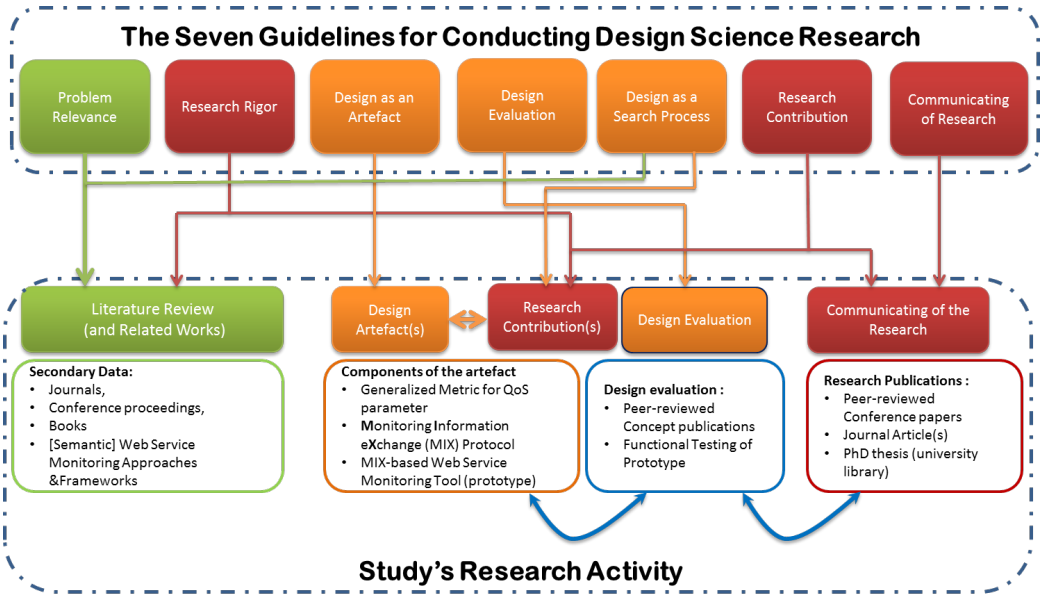


Figure 1.1: Alignment with Seven Guidelines of Design Science Research

The research activities that satisfactorily addressed the problem relevance guideline included reading the literature and critical analysis of related work. The related work explicitly focused on those research projects and studies

that sought to address the problem of Semantic Web Service Monitoring, including the review of frameworks and approaches that other researchers proposed.
The critical analysis of the existing body of research is necessary to satisfy the research rigor guideline. Another important consideration for research rigor is for the study to contribute to the scientific body of knowledge. In view of this, the contributions of the study in terms of conference papers and journals as well as the design artefact(s) mentioned in Section 1.4 do satisfy the research rigor requirement.

According to the guideline regarding design as an artefact, it is necessary that a Design Science Research study produces an artefact, which according to Hevner et al. (2004), may take the form of a construct, model, a process or an instantiation. The nature of the artefacts developed in this study take the following forms:

- **Construct** – this nature of contribution takes a more conceptual form and is about introducing key concepts that are critical for solving an identified problem. In the case of this study, the artefact takes the form of a generalized metric for computing QoS parameter values collaboratively and corroboratively between service consumer and service provider.

- **Process** - a collaborative and corroborative monitoring approach for Semantic Web Services. The process of collaborative and corroborative monitoring is captured in a form of the Monitoring Information eXchange (MIX) protocol, which outlines the structure and sequence of messages to be exchanges between service consumer and service provider.

- **Instantiation** – this contribution will take the form of a proof-of-concept Semantic Web Service monitoring software tool that implements the Monitoring Information eXchange (MIX) protocol.

Since this study adopted a thesis-proving approach to research, it is necessary to spell out how the outcomes of this research can be evaluated. March and Smith (1995) warn that the lack of evaluation metrics that define precisely what the research is trying to accomplish makes it impossible to judge the research efforts effectively. A key evaluation criterion for this study is enabling the exchange of monitoring information between the service consumer

and the service provider. This proves, for instance, that the Monitoring Information eXchange (MIX) protocol is sound and technically feasible.

In order to satisfy the *design evaluation* guideline, the study applies two main criteria:

- **Peer-reviews** – the blind review, by international experts, of conference papers and journals covering key concepts of the study including the design and its rationale.
- **Proof of concept implementation** – to demonstrate the technical feasibility and the utility of the proposed artefact, a prototype implementation is developed. The black-box testing technique (Krichen and Tripakis, 2004) is adopted for the functional evaluation of the prototype.

The fifth guideline is to consider design as a search process. This is somewhat an instrumentalist idea of using whatever means available within the confines of the problem domain in the best possible way to solve the identified problem. In this study, such a search process included a critical analysis of existing literature and critiquing related works to identify important characteristics of a desirable Semantic Web Service Monitoring approach. The results of this search process culminated in what is ultimately the research contributions of this study.

The penultimate guideline pertains to research contributions, this study makes three key contributions, which are necessary to address the research problem:

- Generalized Metric for QoS parameters
- Monitoring Information eXchange (MIX) Protocol that supports dynamic QoS and SLA re-negotiation
- Monitoring Information eXchange (MIX) based Semantic Web Service Monitoring prototype

Finally, the research process, findings and lessons learned during the iterative design cycle must be communicated effectively to various interested audiences; including the scientific, technical as well as non-technical stakeholders. In the case of this study, the research was communicated in the form of a written thesis supported by several peer-reviewed conference papers and journal articles. These are listed in an Appedix ("List of Contributions").

Whereas the seven guidelines provide the necessary checklist that allows the researcher to determine, as objectively as possible, if what was done is "good" design science research, there is still a need to follow a proper research process that adheres to the above guidelines. Peffers et al. (2007, p. 54) provide a process model for conducting design science research. Figure 1.2 demonstrates this study's adaptation of the design science research process model.

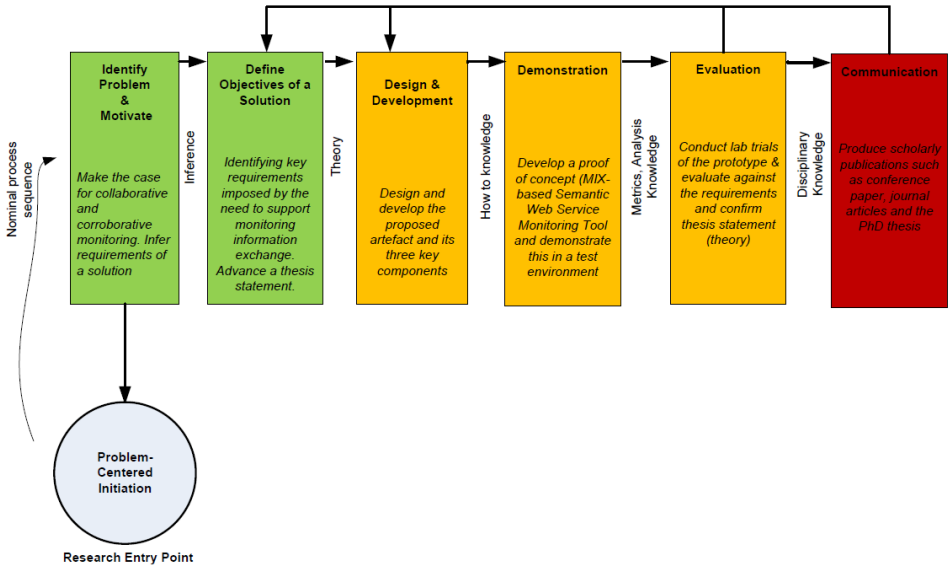


Figure 1.2: Design Science Research Process Model (adapted from Peffers et al. (2007, p. 54))

The following paragraphs outline how the study implemented each step in the nominal process sequence which makes up the Design Science Research Process Model.

In the *Identify and Motivate* activity, the researcher identifies a problem within the domain and motivates the significance of the problem and thus the value of its solution. This study identified a problem and made a case for a collaborative and corroborative monitoring mechanism for Semantic Web Services and inferred the requirements of a viable solution to the identified problem. The identification of the problem is informed by the literature review conducted as part of this study - including the realization that no

existing research has satisfactorily addressed the identified problem.

In terms of *defining the objectives of the solution* the researcher articulates a clear and finite set of objectives or requirements that a desirable artifact to be developed must satisfy. For this study, this meant defining the requirements imposed by the collaborative and corroborative monitoring of Semantic Web Services. The clear articulation of the requirements of a solution is important in that it already provides the researcher with the evaluation metrics to use in evaluating the design artifact in terms of whether it meets these requirements and refine it iteratively during design and development activities of the design cycle.

During the *Design and Development* activities the researcher creates the artifact according to the requirements of the solution to purposefully address the identified problem. Even in the case of theoretical or conceptual contributions, this is the process of continuous refinement of the ideas and concepts. For the purpose of this study, this activity entails defining the Generalized Response Time Metric (GRTM), conceptualizing the Monitoring Information eXchange (MIX) protocol and finally instantiating the MIX protocol. See Chapter 4 and Chapter 5 respectively.

In the *Demonstration* phase the technical viability of the proposed solution is demonstrated by doing a proof of concept implementation. In this study, this activity entails demonstrating the technical viability of the MIX protocol (see Chapter 5).

During the *Evaluation* phase the solution is evaluated against a finite set of objectives or requirements. This phase further entails conducting lab trials of the prototype solution (proof of concept) as well as evaluating the thesis statement based on the experimental results.

Finally, in the *Communication* phase, the outcomes of the research are communicated. This can take the form of presentations at research symposia, conference papers, journal articles and the actual PhD thesis document.

The preceding section presented the description of the methodological aspects of this study. The following section outlines the limitations of the research.

1.6 Limitations of the Research

The scope of the proposed research study is focused on the Enactment Sub-cycle of the Service Delivery Lifecycle framework (Kuropka et al., 2008): The framework itself will be discussed in greater detail as part of literature review in Chapter 2. The Service Delivery Lifecycle framework provides a simplified view of the key steps involved in processing a service request in a service-oriented computing scenario. According to this view, the processing of a service request is done in three key phases or sub-cycles of the service delivery lifecycle namely (1) *Planning*, (2) *Binding* and (3) *Enactment*.

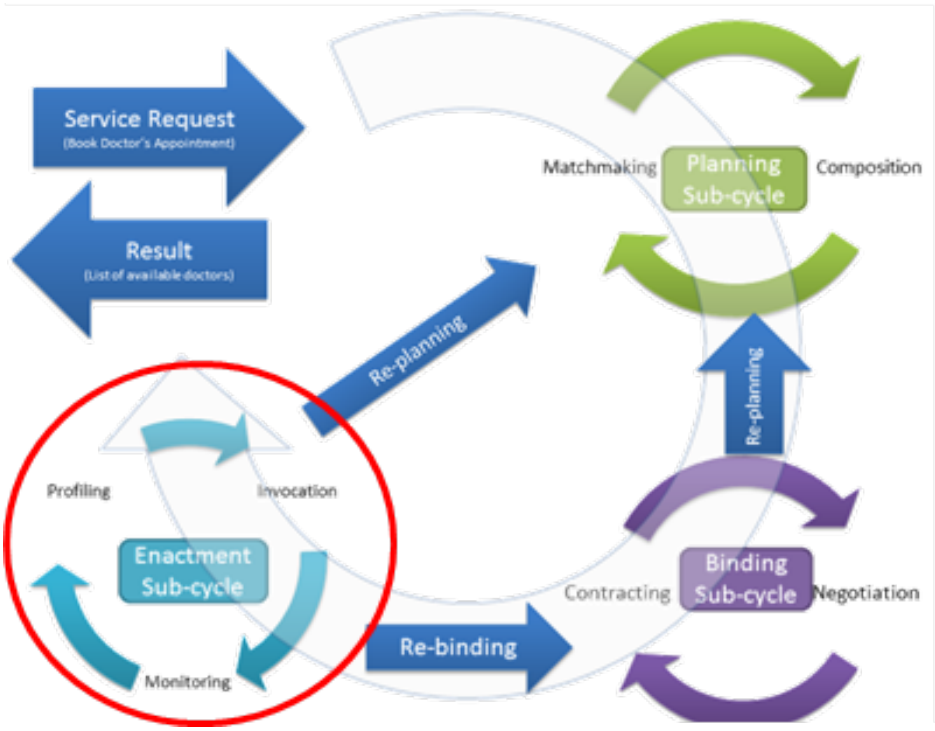


Figure 1.3: Service Delivery Lifecycle (adapted from Kuropka et al. (2008))

As depicted in the Service Delivery Lifecycle framework (Figure 1.3), this study assumes that all the preceding sub-cycles have already been completed. Most importantly, the study takes it for granted that the target services as published by service providers are already semantically enriched. In the proof-of-concept implementation, SAWSDL was used as a standard for se-

mantic annotation of Web services, the evaluation of possible approaches to semantic annotation is out of scope of this research.

In terms of QoS ontology development, this study does not claim to have included an exhaustive list of QoS parameters – the main focus as been on measurable QoS parameters as these make a good case for collaborative monitoring.

1.7 Thesis Outline

Chapter 1: Introduction - This chapter will be a revised version of the proposal and will cover only key aspects of the research process – the problem statement, research objectives and the hypothesis. The main objective of this chapter will be to present the rationale for this study and to support its case.

Chapter 2: Service Oriented Computing - This chapter follows the nominal structure of a literature survey on Semantic Web Service and introduces an illustrative framework to capture the overview of Service Oriented Computing (SOC).

Chapter 3: Semantic Web Service Monitoring - This chapter discusses the conceptual principles and the monitoring mechanisms for Semantic Web Services. The chapter also discusses the fundamentals of Quality of Service (QoS) and the approaches to semantically describing QoS of Web Services. The chapter further reviews the related works and highlights their shortcomings to make a case for collaborative and corroborative monitoring of Semantic Web Services.

Chapter 4: Collaborative and Corroborative Semantic Web Service Monitoring - This chapter presents the conceptual aspects of the model for Collaborative and Corroborative Monitoring of Semantic Web Services. In this chapter the proposed solution is outlined, along with its conceptual underpinnings and design specification for the proposed Monitoring Information eXchange (MIX) protocol.

Chapter 5: Proof of Concept Implementation of MIX - This chapter draws from the conceptual design presented in Chapter 4 and develops a

technology-specific demonstrator. It will present results from experimental prototype of a Semantic Web Service monitoring tool developed based on the findings of this study. This will be a proof of concept and not necessarily a production-ready Semantic Web Service monitoring tool. This chapter focuses on the technical infrastructure and the reference implementation of the Monitoring Information eXchange (MIX) protocol.

Chapter 6: Evaluation and Results - In this chapter, the proof of concept tool developed in Chapter 5 is systematically evaluated and its performance is analysed. This is meant to ascertain that it achieves the expected functionality of a collaborative and corroborative monitoring of a Semantic Web Service. The evaluation process will also seek to establish whether the monitoring approach has any noticeable performance implications. The chapter thus discusses the results of the evaluation of the collaborative and corroborative monitoring technique for Semantic Web Services as proposed by this study.

Chapter 7: Summary, Conclusion and Recommendations - This chapter draws conclusion of the study and hints at any future research work in the context of the current study. This chapter also serves to reflect on the research process in its entirety and also identifying key contributions of the study and their implications. This chapter presents the conclusions that the study deduces from the work accomplished by the entire research project.

1.8 Summary

This chapter essentially made a case for the study and discussed the key aspects of the scientific research process; it discussed the research problem, the thesis statement, as well as the objectives of the proposed solution. The chapter outlined both the conceptual and the technical contributions that the study sought to make. The chapter also discussed the methodological aspects of the study as well as acknowledged the limitations of the study. Finally, the chapter presented the outline of the thesis document in terms of the synopsis of each chapter and the sequence of chapters.

The next chapter discusses the Service Oriented Computing (SOC) literature.

Chapter 2

Service Oriented Computing

This chapter provides the theoretical grounding for the study by giving a focused overview of Service Oriented Computing (SOC).

A nominal structure of a literature survey on Semantic Web Service starts off with the discussion of the classical Web services accomplished mainly through Web Service Description Language (WSDL) and Simple Object Access Protocol (SOAP) technologies. The discussion then moves on to highlight the limitations of the classical Web services such as requiring significant human intervention and not being readily machine-understandable. In so doing, a case is made for Semantic Web Services. Discussion on Semantic Web Service in turn necessitates discussing ontologies – basically, it is these ontologies which augment the existing Web Service standards to provide the semantic layer necessary to enable machine processing (locating, selecting, composing and executing) of the Web services. Once this background is laid, the topical issues of Semantic Web Services which include standards, approaches and challenges to combining the Semantic Web and Web services are then discussed for instance by Nacer and Aissani (2014), Mohebbi, Ibrahim, and Idris (2012); as well as Bruijn et al. (2008). Therefore without being too exhaustive, this nominal structure covers a big-enough research area of Semantic Web Services (Fensel et al., 2011).

This chapter adopts the nominal structure above and introduces an illustrative framework to capture the overview of Service Oriented Computing (SOC). This allows the reader to appreciate the practical implications of some of the outstanding research challenges in SOC without tussling with the more abstract theoretical aspects of SOC research. Using an illustrative framework is helpful in presenting literature review because it already

provides the reader with a perspective from which to look at the referenced materials such as Web Service standards. In this chapter the Service Delivery Lifecycle by Kuropka et al. (2008, p. 16) is used as an illustrative framework for presenting the literature review, which is relevant to position the solution proposed by this study. Additionally, the sub-cycles of the Service Delivery Lifecycle are used to aid in the understanding and identification of key areas of research focus within both the classical Web Service as well as Semantic Web Service. The chapter ends by positioning this study within the Service Delivery Lifecycle and hinting at the subsequent discussion in the next chapter (Chapter 3) which focuses on Semantic Web Service Monitoring.

2.1 Introduction

Service Oriented Computing (SOC) is a computing paradigm for addressing the complexities of distributed applications and uses a Service as the building block for heterogeneous, distributed applications (Huhns, 2005; Michlmayr et al., 2009; Papazoglou, Traverso, Dustdar, Leymann, and Krämer, 2006). As a paradigm, SOC is an approach for building distributed, loosely-coupled applications. The 2017 Service Computing Manifesto (Bouguettaya et al., 2017) describes SOC or Service Computing as a paradigm that enables the support of business services by offering cross-disciplinary computational abstractions, architectures and technologies.

However, and critically, SOC may also be viewed as an approach to exposing computational resources as services. In this sense, Cloud Computing may be considered a variant of SOC. The computational resources offered by Cloud Computing are dynamically scalable,low cost and location-independent (Bokhari and Azam, 2015). According to Shaukat Dar et al. (2016), what is core to the service-orientation in Cloud Computing is its support for developing rapid, low-cost, interoperable, and flexible applications. Baraldo, Zuccato, and Vardanega (2015) stated that, arising from this service orientation model, are the concepts of *Quality of Service* (discussed later in Chapter 3) and *Software as a Service* paradigm. Software as a Service (SaaS) is a delivery model used to provide software-based solutions over the Internet by making them accessible on rental or subscription basis (Baraldo, Zuccato, and Vardanega, 2015). One of the benefits of the SOC paradigm is that the on-demand basis on which computational resources are provided can actually contribute towards addressing the hardware and software acqui-

sition challenges faced by the resource-constrained enterprises such as small businesses.

Service-orientation as an approach is shared in common by both Cloud Computing as discussed above, and Service Oriented Architecture (SOA). SOC views whole business functions (booking a doctor's appointment, for example) as modular, standards-based software services (Luthria and Rabhi, 2009). Given that, from the perspective of SOC, services are building blocks for distributed applications (Stollberg and Fensel, 2010), the expectation is that such services may be autonomous and span multiple organizations. SOC does not make any mention of the architectural style for building these service oriented applications. SOA is the architectural style that promotes interoperability and reusability of software components and is best suited for building complex and distributed service-oriented computing applications (Fensel et al., 2011). The fundamental idea of SOA is about loose coupling and encapsulation, whereby each software component provides individual service (Bokhari and Azam, 2015). SOA is an architectural approach that provides a flexible set of design principles used during model-driven engineering of distributed enterprise software systems (Boukaye Boubacar et al., 2018). According to Rojas, Arias, and Renteria (2021), SOA enables interoperability and integration of cross-platform applications in support of distributed information systems.

The three most basic components of Service Oriented Architecture (SOA) are *service consumer*, *service broker* or *registry* and *service provider* (Alshinina and Elleithy, 2017). Figure 2.1 depicts the three basic components of SOA:

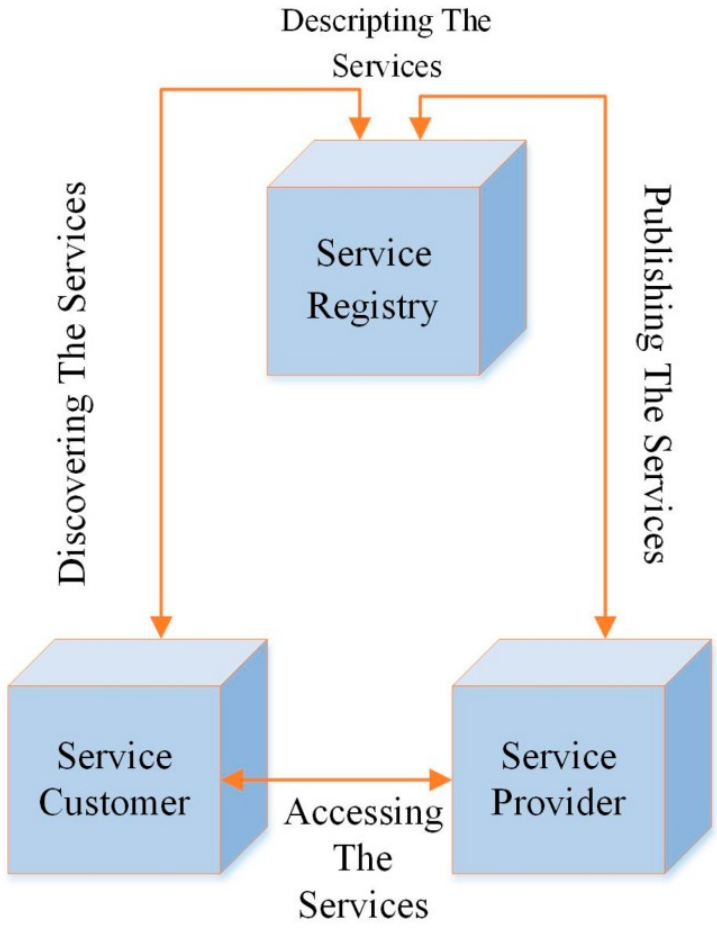


Figure 2.1: Service Oriented Architecture Components (Alshinina and Elleithy, 2017)

SOA as an architectural style describes how complex software systems can be built by exposing existing functionalities as services and putting these services together. SOA is a business-focused architectural pattern for achieving the integration of services (Calderón-Gómez et al., 2021). Web Services which have brought about the most practical implications of SOC concepts are considered the most common technology based on SOA (Mohebbi, Ibrahim, and Idris, 2012). One key principle of SOA insists that services be self-contained units of computation and as such should not maintain state across invocations (Bruijn et al., 2008, pp. 14–15). This principle promotes the heterogeneity and autonomy of services.

2.2 Web Services

Medjahed and Atif (2007) describe a Web service as a set of related functionalities, which is accessible through the Web. Concurring with this view, Filho and Ferreira (2009) describe the primary goal of Web Services as facilitating Web-accessibility of business functionalities. Vaculín (2009, pp. 18–19) provides a detailed analysis of the key aspects of Web Service definition as provided by the World Wide Web Consortium (W3C). These key aspects are:

- *Description* of Web service's public interfaces and bindings which captures the functional and non-functional aspects of the described service, using standards such as Web Service Description Language (WSDL) (Chinnici et al., 2007);
- *Identification* of Web services by means of Universal Resource Identification (URI);
- *Discoverability* which allows the descriptions of individual Web services to be discovered using standards such as Universal Description, Discovery and Integration (UDDI);
- *Network-orientation* which allows the Web service interaction by exchanging of messages using Internet protocol over a network such as Simple Object Access Protocol (SOAP).

Conventional Web Services are considered to be more syntactic and less dynamic thus making automated Web Service discovery and composition difficult (Dimitrov et al., 2007). This necessitates augmenting these syntactic Web Services with some semantic layer Vaculín (2009, p. 22). Berners-Lee (2003) expresses the vision to move from a human-only understandable and static Web where machines (e.g. computers) merely display the data (e.g. on Web browsers) to the kind of Web in which machines can understand and automatically process the data. Achieving this vision has been impeded by the lack of semantic descriptions within the traditionally static Web.

The semantic description of Web Services is the machine-understandable metadata necessary to support automatic Web Service discovery, composition and invocation (Siddiqui and Seth, 2017; Chifu, 2010). Ontologies may be viewed as connective structures that express the relationships between information resources on the Web as well as connecting these resources to their

formal terminologies (Fensel, 2003). Furthermore, ontologies represent the conceptualizations or understanding of the meaning of things within a particular domain (Acuna and Marcos, 2006). Since ontologies make imparting semantic data to Web resources possible, they are essential for enabling automatic service discovery, composition, and execution (Mtsweni, Biermann, and Pretorius, 2014).

With the additional semantic descriptions supported by ontologies as common conceptualization of domain knowledge (Mohebbi, Ibrahim, and Idris, 2012), the Web service descriptions are amenable to machine understanding. This annotation of Web Services by semantic descriptions (Stollberg and Fensel, 2010) leads to the concept of Semantic Web Services.

2.3 Semantic Web Services

The guiding principles and direction of Semantic Web Services are captured by the World Wide Web Consortium (W3C) in the Semantic Web Service Framework Overview (Battle et al., 2005). The key components of the Semantic Web Service Framework (SWSF) are the Semantic Web Service Language (SWSL) as well as the Semantic Web Service Ontology (SWSO).

The Semantic Web Service Language (SWSL), as explained by Konstantinou et al. (2010), specifies the formal characterizations of Web Service concepts as well as the descriptions of web Services themselves (Battle et al., 2005). SWSL comprises two sub-languages; one based on First Order Logic (FOL) which is aptly named SWSL-FOL and is used to express the ontology of Web Service concepts. The second sub-language (SWSL-Rule) is based on the Rules or Logic-Programming paradigm and is used to support reasoning over Web Service ontologies.

The Semantic Web Service Ontology, on the other hand, presents a conceptual model and the ontology for describing Web Services. To this end, SWSO defines two types of ontologies; one based on First Order Logic referred to as the First Order Logic Ontology for Web Services (FLOWS) and the other based on Rules/Logic Programming named the Rules Ontology for Web Services (ROWS). Battle et al. (2005) provides a cursory discussion of both SWSL and SWSO in the context of the Semantic Web Service Framework (SWSF) and links to W3C submission documents on each of these components.

The remainder of this section draws from Battle et al. (2005) together with

other resources and discusses the theoretical grounding of research efforts such as OWL, OWL-S, WSMO and WSML among others.

Semantic Web is a paradigm shift purported to address the lack of semantic descriptions in the current Web. According to Mohebbi, Ibrahim, and Idris (2012, p. 65) the goal has been to extend current Web by introducing machine-understandable semantics to Web resources. In the case of Semantic Web Services this extension is done by annotating Web Services with ontology-based descriptions to enable reasoning over these descriptions by ontology reasoning tools including software agents (Mohebbi, Ibrahim, and Idris, 2012, p. 66). According to Souza Neto, Moreira, and Musicante (2018), a Semantic Web Service is basically a Web Service with a description that defines its semantics in a computer-interpretable language (de Souza Neto, Moreira, and Musicante, 2018).

Semantic Web Service may be construed as the combination of Semantic Web and Web Service technologies (Bruijn et al., 2008). There are two broad directions in the technology development of Semantic Web Services for Internet-scale distributed computing which Slimani (2013) discusses; namely the formal specification driven *WS-** direction and the REpresentation State Transfer Protocol (REST) architectural direction of the World Wide Web. According to Slimani (2013), the WS-* set of specifications follows the messaging paradigm and specialized service interfaces, defining standardized infrastructure protocols for security and transaction management among others. On the other hand, the REST direction exploits the architectural style of the World Wide Web and considers Web services as sets of network-accessible resources which can be retrieved and consumed on the back of HTTP Internet infrastructure.

In discussing the classification of approaches to the semantic description of Web Services, Slimani (2013) depicted and contrasted the bottom-up and top-down approaches to semantic enrichment of Web Services in what he calls a Web Service description approaches taxonomy. Figure 2.2 depicts the taxonomy in terms of the two broad categories of approaches to semantic enrichment of Web Service descriptions.

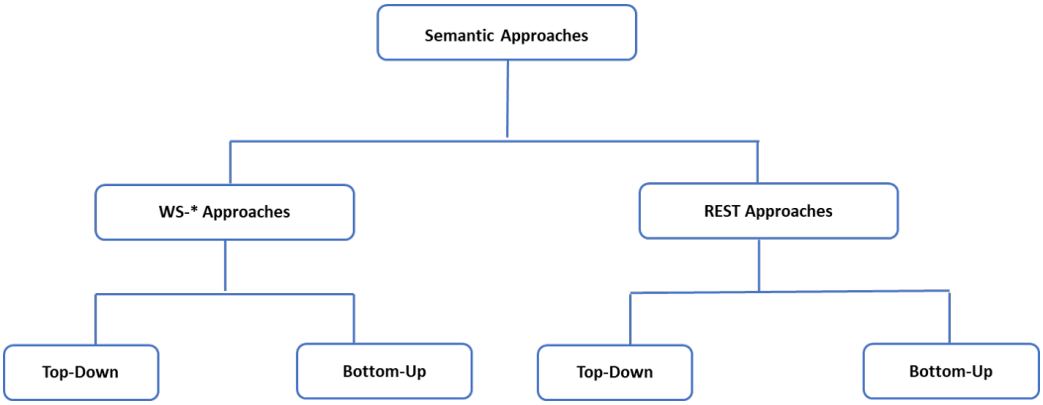


Figure 2.2: Semantic Web Service Approaches Taxonomy (adapted from Slimani, 2013)

Figure 2.3 shows the break-down of the top-down and bottom-up categories in the RESTful approaches taxonomy:

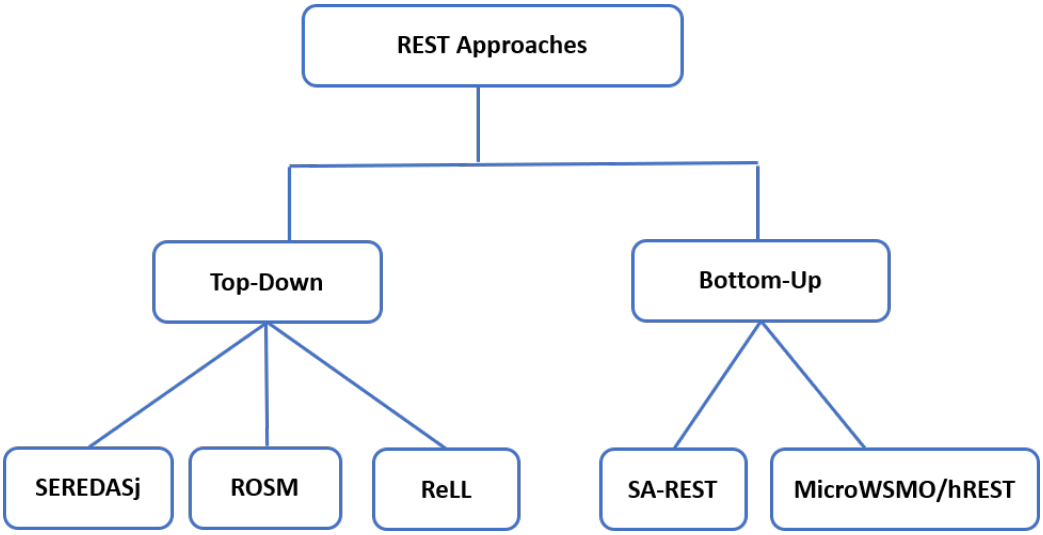


Figure 2.3: Approaches to RESTful Semantic Web Services (adapted from Slimani, 2013)

Following the REST architectural principles to the description of Semantic Web Services, a new breed of Semantic Web Services emerged and is called the RESTful Semantic Web Services (Hernández and Moreno García,

2010). Figure 2.3 above locates the two main approaches to achieving these RESTful Semantic Web Services. According to Lanthaler and Guetl (2011), the emergence of these RESTful Semantic Web Services has been chiefly motivated by the fact that the traditional RESTful Web Services closely resemble the document-based Web, and are thus perceived as less complex and less disruptive. Furthermore, the traditional RESTful Web Services provide read-write interfaces to the underlying data through the use of Application Programmer Interfaces (the REST APIs).

As far back as 2011, Lanthaler and Guetl (2011) noticed that many data publishers, and Web developers, chose to publish their data in the form of Web Services. The reasons given for this preference of Web Services over Semantic Web Services for publishing data were given as follows (Lanthaler and Guetl, 2011):

- No clear incentives for developers to use the Semantic Web Service technology stack;
- Developers feel overwhelmed by the complexities of the Semantic Web technology stack - the phenomenon that authors referred to as "Semaphobia";
- Enterprises perceive the Semantic Web as a disruptive technology, feeling that it will not be possible to build upon existing infrastructure investments to evolve their current business systems;
- Contemporary Semantic Web approaches usually provide just the read-only interfaces to the underlying data, thus inhibiting the networking effect and participation of the crowd.

In yet another study, Mtsweni, Biermann, and Pretorius (2014) discussed what they considered the main challenges that are contributing to the lack of real-world implementation and adoption of SWS:

- Lack of effective developer tools,
- Disruptive nature of SWS due to non-integration of SWS technologies into existing technologies,
- High costs associated with adopting the service-oriented architectures,

- The high complexity of prominent heavy-weight semantic models, resulting in a steep learning curve
- Concerns over contradictions and lack of consensus in semantic modeling standards.

This trend persists to this day (2020) and is responsible for the sluggish adoption of Semantic Web Services as mainstream enablers for Web of Data.

In highlighting the challenges impeding the adoption of SWS, Mtsweni, Biermann, and Pretorius (2014) proposed a model-driven approach for designing and developing SWS using semantic description languages and service description languages that a developer may prefer. The adoption of this promising approach by the developer community has not been reported.

For their contributions, Lanthaler and Guetl (2011), as well as Lanthaler (2012), focused on an approach to integrate RESTful Web Services into the Web of Data in which the underlying semantic graph data can be consumed through RESTful-API styled interfaces. For example, Lanthaler and Guetl (2011) introduced an algorithm to translate SPARQL queries to HTTP requests based on a semantic description language for RESTful data services called SEREDASj. According to Lanthaler (2012), SEREDASj specifies the syntactic structure of a specific JSON representation. Additionally, SEREDASj allows referencing JSON elements to concepts in an ontology and further describe these elements by semantic annotations.

In terms of the formal WS-* specification direction, and as depicted in the approaches taxonomy in Figure 2.4, the approaches to realizing Semantic Web Services can be classified into *bottom-up* and *top-down* (Bruijn et al., 2008). Figure 2.4 shows the break-down of the top-down and bottom-up categories in the WS-* approaches taxonomy:

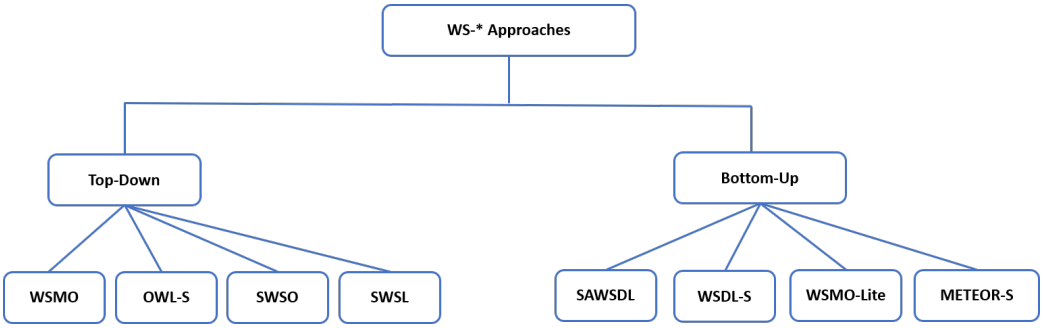


Figure 2.4: WS-* Approaches Taxonomy (adapted from Slimani, 2013)

The bottom-up approaches add semantic annotations to existing Web services standards such as WSDL with semantic descriptions (Mohebbi, Ibrahim, and Idris, 2012, p. 66) and thus leading to frameworks such as WSDL-S (Akkiraju et al., 2005), SAWSDL (Kopecký, Vitvar, et al., 2007), METEOR-S (Patil et al., 2004) and WSMO-Lite (Kopecký and Vitvar, 2008).

Whereas bottom-up approaches extend the existing technologies by introducing semantic descriptions, the top-down approaches introduce new independent semantic description frameworks that use ontology as a conceptualization formalism for describing Web services and then grounding these semantic-enriched descriptions to current Web services standards. Bruijn et al. (2008, pp. 1–3) describe two common top-down approaches, which are:

- **OWL-S** - the use of the Description Logic (DL) based Web Ontology Language (OWL) as a framework for describing services and which is aptly named OWL-S;
- **WSMO** - the use of a language-agnostic conceptual model for describing Web Service which is named Web Service Modeling Ontology (WSMO)

The research efforts that looked at comparative analysis of Semantic Web service frameworks include (Mohebbi, Ibrahim, and Idris, 2012; Ayadi and Ben Ahmed, 2011; Akkiraju, 2007). Slimani (2013) further looked at the classification of semantic description of Web Services, contrasting the top-down and bottom-up approaches to realizing the Semantic Web Services.

OWL-S presents an upper ontology that describes the properties and capabilities of Web services using OWL (Martin et al., 2004). According to this ontology a service is described in three main aspects (Mohebbi, Ibrahim, and Idris, 2012, p. 67). Firstly a *service model* captures how the service works and how it may be invoked. Secondly, the service also presents a *service profile* which describes what the service does in terms of its capabilities and thus supporting automated match-making and discovery. Service profiles include both functional and non-functional aspects of a Web service. Thirdly, and lastly the *service grounding* describes how a service can be accessed.

In terms of the three aspects of the upper ontology, the service model is of interest to this study because it enables, among other tasks, service enactment activities which include invocation and monitoring.

The second most common top-down approach to Semantic Web Service, also treated by Mohebbi, Ibrahim, and Idris (2012, pp. 68–69), is the Web

Service Modeling Ontology (WSMO). WSMO provides a conceptual model for semantically describing a Web service without prescribing any specific language but rather imposing that any language implementation of WSMO adhere to the conceptual model (Mohebbi, Ibrahim, and Idris, 2012, p. 68; Bruijn et al., 2008, p. 2).

Web Service Modeling Language (WSML) is a language implementation of WSMO (Bruijn et al., 2008; Slimani, 2013). The Web Service Modeling Language has to provide means for describing three key aspects of a service namely (1) *functional aspects* which captures the functionality of a service, (2) the *behavioural aspects* which describes messages (content) to be sent by or received by a service, and finally (3) the *non-functional aspects* (e.g. price, availability) which inform the selection of a particular service (Bruijn et al., 2008, pp. 20–21). In this sense, and according to Slimani (2013), WSML serves as a language for describing domain-specific semantic models. WSML provides a language for describing the four top-level elements of the WSMO conceptual model, namely Ontologies, Goals, Web Services and Mediators. Specifically, WSML provides sub-languages for the following three functions (Bruijn et al., 2008, p. 2):

- Sub-language for describing ontologies which provide the terminology and domain knowledge for service descriptions;
- Sub-language for describing functional aspects of a service which capture functionality required to satisfy consumer goals or the functionality provided by a service;
- Sub-language for describing the behavioural aspects of a service and service interfaces;

Bruijn et al. (2008, pp. 36–61) describe and thoroughly treat the five WSML sub-languages or variants as well as their surface syntaxes.

Mohebbi, Ibrahim, and Idris (2012) present a review and comparative analysis of the four Semantic Web Service approaches (OWL-S, WSMO, WSDL-S and SAWSDL). The results of their comparative analysis show that in terms of the criteria set out in Mohebbi, Ibrahim, and Idris (2012, pp. 71–73) both OWL-S and WSMO (top-down approaches) are superior to WSDL-S and SAWSDL (bottom-up approaches). Based on these results Mohebbi, Ibrahim, and Idris (2012, p. 75) concluded that WSMO and OWL-S represent semantically richer frameworks for describing Semantic Web Services.

In view of what Mohebbi, Ibrahim, and Idris (2012) had found in terms of the semantic richness of WSMO and OWL-S, Chapter 3 explores these frameworks as viable options to support Semantic Web Services Monitoring.

To summarize the preceding discussion on Semantic Web Services a modified Semantic Web services cartography (Nacer and Aissani, 2014, p. 136) is presented in Figure 2.5 and captures the relationships between technologies and standards that enable Semantic Web services.

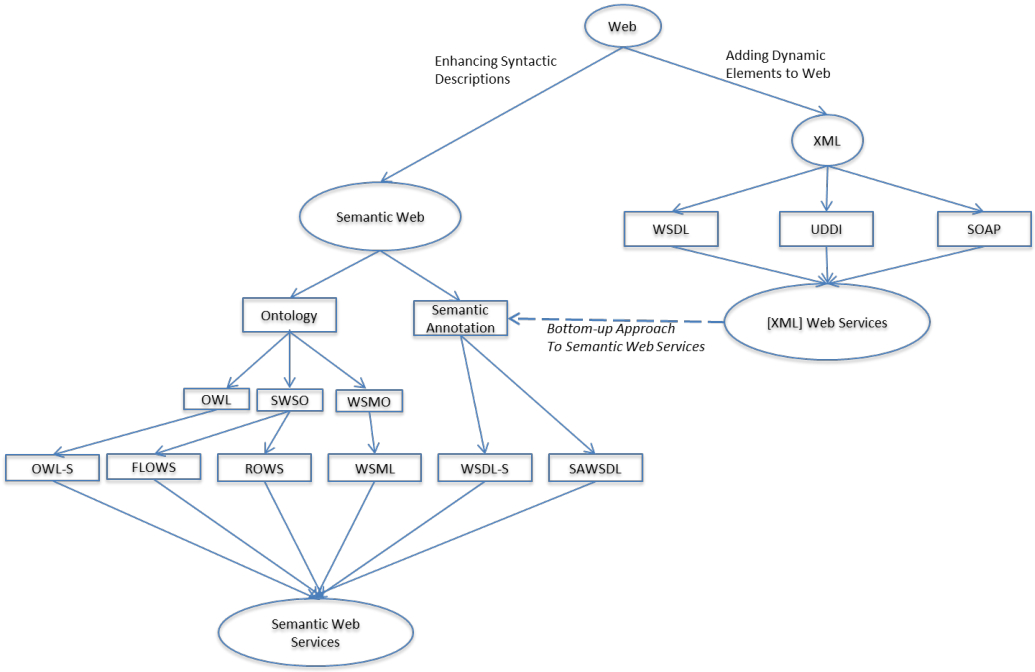


Figure 2.5: Semantic Web Service Cartography (adapted from Nacer and Aissani, 2014, p. 136)

The cartography in Figure 2.5 conveys the idea that the evolution of the traditional document Web into Web Services and the data Semantic Web, and finally the Semantic Web Services followed two trajectories (Nacer and Aissani, 2014, p. 136): (1) the *addition of dynamic elements (specifically XML) to the Web* leading to technologies such as XHTML, SOAP, WSDL, and UDDI, and (2) the *enhancement of the syntactic descriptions of Web resources* by adding formal characterizations (ontologies) and semantic annotations.

By way of concluding the discussion on the theoretical principles of Semantic Web Services and as a way of describing the state of art in Semantic Web Services research, the next section looks at the technologies that enable Semantic Web Services. Specifically, the section presents some of the ontology editing tools, reasoners and execution engines as well as technologies used to store the semantically enriched web of data.

2.4 Ontology Engineering Tools: Technologies Enabling Semantic Web Services

There is a focused discussion on ontologies at a more conceptual and theoretical level which covers the formalisms and languages used in the development of ontologies in Chapter 3 (see Section 3.4.1), the focus in the current section, however, is on the tools that enable the Semantic Web Services, including the capabilities to edit or create ontologies.

Slimani (2015) discuss several tools created for developing ontologies using languages or formats such as XML,RDF, RDFS (RDF Schema). Much earlier, Duineveld et al. (2000) referred to these tools as Ontology Engineering Tools. A relatively recent study was conducted by Malik (2017) and performed a usability evaluation of several such ontology engineering tools.

2.4.1 Resource Description Framework (RDF)

The Resource Description Framework (RDF) is recommended by the World Wide Web Consortium (W3C) as the standard for publishing descriptions of data resources on the Web. RDF is a unified data model for describing resources over the Web of data, reflecting how these resources are stored and interlinked (Lanthaler and Guetl, 2011). Essentially, RDF is a data model for representing information about resources on the Web (Ma, Capretz, and Yan, 2016).

RDF defines the triples consisting of resources, properties and values where a resource is an entity accessible by a URI, and a property defines a binary relation between resources or literals, and a value represents a literal or even a resource (Liu et al., 2019).

The RDF Schema (RDFS), according to McBride (2004), is the vocabulary description language for RDF. Liu et al. (2019) see RFDS as providing a data modeling vocabulary and syntax to RDF descriptions and enforces the

structural constraints to support the proper expression of semantics. The code listing in Figure 2.6 below shows an example of RDF in XML format usually denoted as RDF/XML:

```
<rdf:RDF ...>
  <owl:Class rdf:about="http://www.cc2masw.co.za/ontologies/qos#ServicePerformance">
    <rdfs:subClassOf>
      <owl:Class rdf:about="http://www.cc2masw.co.za/ontologies/qos#QoSAttribute"/>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Class rdf:about="http://www.cc2masw.co.za/ontologies/qos#MeasurableQoSAttribute"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="http://www.cc2masw.co.za/ontologies/qos#Units"/>
  <owl:Class rdf:about="http://www.cc2masw.co.za/ontologies/qos#MeasurableQoSAttribute">
    <rdfs:subClassOf rdf:resource="http://www.cc2masw.co.za/ontologies/qos#QoSAttribute"/>
  </owl:Class>
  <owl:Class rdf:about="http://www.cc2masw.co.za/ontologies/qos#Metric"/>
  <owl:Class rdf:about="http://www.cc2masw.co.za/ontologies/qos#UnMeasurableQoSAttribute">
    <owl:disjointWith rdf:resource="http://www.cc2masw.co.za/ontologies/qos#MeasurableQoSAttribute"/>
    <rdfs:subClassOf rdf:resource="http://www.cc2masw.co.za/ontologies/qos#QoSAttribute"/>
  </owl:Class>
  ...
  <owl:ObjectProperty rdf:about="http://www.cc2masw.co.za/ontologies/qos#hasMetric">
    <rdfs:range rdf:resource="http://www.cc2masw.co.za/ontologies/qos#ServiceResponseTime"/>
    <rdfs:domain rdf:resource="http://www.cc2masw.co.za/ontologies/qos#ServicePerformance"/>
  </owl:ObjectProperty>
</rdf:RDF>
```

Figure 2.6: RDF/XML Sample Ontology Description

The RDF data model forms the foundation for RDF-based technologies such as the SPARQL query language and the RDF-based data exchange formats such as RDF in XML syntax (RDF/XML). For instance, Liu et al. (2019) sought to exploit RDF to achieve the representation of machine-processable biomedical data as a possible application of this technology.

SPARQL is the standardized query language and protocol for accessing RDF and defines how to retrieve graph-based RDF data (Prud'hommeaux and Seaborne, 2008). SPARQL is developed by the W3C RDF Data Access Working Group [1].

Chooralil and Gopinathan (2015) conceded that recognizing the second or higher order associations between the resources on the Web of data has remained challenging. Their contribution was to develop Semantic Resource Description with Compound Protocol (SRD-CP) as an architectural framework for processing the expanded user queries and identifying the associations between the Web resources and user-fetched queries (Chooralil and Gopinathan, 2015, p. 725). The SRD-CP is based on RDF patterns and uses the RDF query language (i.e. SPARQL).

[1] http://www.w3.org/2001/sw/DataAccess/

The need to exchange large datasets, especially in the document-centric and human-readable Web, has exposed the drawbacks of traditional RDF representations. The key problems with these datasets are the high levels of verbosity/redundancy and the weak machine-processable capabilities in their descriptions. According to Fernández et al. (2013), addressing these problems requires efficient formats for publication and exchange of large datasets.

Fernández et al. (2013) developed an RDF representation that modularly partitions and efficiently represents three components of RDF datasets namely *(1)* **H**eader information, *(2)* a **D**ictionary, and *(3)* the actual **T**riples structure and called it HDT.

Another study that also looked at the challenge of processing larger RDF dataset is that of Huang, Abadi, and Ren (2011). They dealt with RDF datasets that needed to be partitioned across multiple machines to achieve reasonable query performance. To achieve this, Huang, Abadi, and Ren (2011) developed a scalable RDF data management system which consisted of three main tasks: (1) leveraging single node RDF-store technology (2) partitioning the data across multiple nodes and (3) decomposing SPARQL queries into high-performance query fragments.

Zeng, Yang, et al. (2013) introduced a distributed, memory-based graph engine for web-scale RDF data storage and processing called Trinity RDF. In their approach, the RDF data is stored in its native graph form as opposed to using the triple stores or bitmap matrices. According to Zeng, Yang, et al. (2013), storing RDF data in its native graph form like this achieves comparatively better SPARQL query performance.

Abdelaziz et al. (2017) suspected that the distributed SPARQL engines, such as those discussed above, exhibit trade-offs that are not well-understood because no comprehensive quantitative and qualitative evaluations had been conducted on them. In their study, Abdelaziz et al. (2017) surveyed 22 state-of-the-art distributed RDF processing systems or engines. A representative sample of these systems was evaluated based on preprocessing cost, query performance, scalability and workload adaptability. Based on this work, Abdelaziz et al. (2017) formulated a framework for the evaluation of distributed RDF processing engines.

In Semantic Web, and especially ontology engineering, the main usage of RDF is as a standardized format for the representation of an ontology. Ontology development/engineering requires careful considerations concerning the Formalisms, Tools and Languages to be used in the development. Slimani (2015, pp. 1–2) lists the following as some of the key considerations

for building ontologies from scratch:

- Specify the tool(s) for ontology development;
- Identify the ontology storage type and tool;
- Indicate if the required tool has an inference engine;
- Indicate if chosen tools have translators/adaptors for different ontology languages (OWL, OWL-S, WSMO, etc) or formats (XML,RDF,Turtle, etc);
- Specify which language(s) to use, including the preferred expressiveness of such language(s);
- Verify that the chosen tool supports the preferred language, and that the language is appropriate to support information exchange between different applications;
- Decide whether the envisaged ontology-driven application requires integrating with other ontologies implemented in different languages;

2.4.2 Ontology Editors

One category of ontology engineering tools is the ontology editors. The ontology editors are tools that allow users to visually manipulate, inspect, browse and modify ontologies. Apache Jena[2] is a free and open source Java framework for building Semantic Web and Linked Data applications. Linked Data applications use RDF data model to describe statements that link arbitrary data resources on the Internet and use these RDF links to infer new facts about the concepts in the ontology model (Hsu and Lyu, 2019). Another usage of Apache Jena is reported by Shafi et al. (2018) in which Apache Jena is used to parse the RDF ontology model file in order to extract Object Oriented Model. Apache Jena provides the facilities for authoring ontology models (RDF, RDFS and OWL, SPARQL) as well as rule-based inference engine for reasoning over these ontology models. Apache Jena framework also includes a query engine that supports SPARQL RDF query language. SPARQL is the query language developed by the W3C RDF Data Access Working Group. Apache Jena Fuseki is a SPARQL server.

[2]https://jena.apache.org/

A study conducted by Ranpara, Yusufzai, and Kumbharana (2019) compares ontology building tools for contextual information retrieval . The ontology building tools being compared are: Protégé [3], Swoop [4], Apollo[5], and IsaViz[6]. These are the same tools that Kapoor and Sharma (2010) also looked at earlier. These are the most popular open source tools in the category of ontology editors.

- **Protégé** - a knowledge-based ontology editor providing graphical user interface. It provides better flexibility for meta-modeling, and enables the construction of domain ontologies. It allows for the definition of classes, class hierarchies, variables, variable-value restrictions, and the relationships between classes and the properties of these relationships;

- **IsaViz** - is a visual environment for browsing and authoring RDF models as graphs. IsaViz supports the ability to import RDF/XML and N-Triples formats, and to export in the RDF/XML, N-Triples, Portable Network Graphics (PNG) and Scalable Vector Graphics (SVG) formats.

- **Apollo** is a user-friendly knowledge modelling application. The modelling is based around the basic primitives, such as classes, instances, functions, relations and so forth. The Apollo knowledge base consists of a hierarchy of ontologies.

- **SWOOP** is a Web-based OWL ontology editor and browser. It supports capabilities such as editing, comparing and merging ontologies. SWOOP contains OWL validation and offers various OWL presentation syntax views. It has reasoning support and provides a multiple ontology environment.

The list of ontology editing tools is not exhaustive and it is not the purpose of this study to discover and describe all existing tools. It suffices to show some of these tools as an indication of consistent attention to the task of ontology engineering. Studies such as Kapoor and Sharma (2010) and Slimani (2015) provide some coverage of other ontology engineering tools.

[3]http://protege.stanford.edu
[4]https://www.w3.org/2001/sw/wiki/SWOOP/
[5]http://apollo.open.ac.uk/index.html
[6]http://www.w3.org/2001/11/IsaViz/Overview.html

The next section illustrates the practical implications of Semantic Web Service technology by use of a Service Delivery Lifecycle as an illustrative framework.

2.5 Semantic Web Service Delivery Lifecycle Framework

The practical implications of Semantic Web Service technology can be understood better when illustrated by way of a Semantic Web Service Delivery Lifecycle (Kuropka et al., 2008) which captures the steps involved in the processing of a service request in a service-oriented computing scenario.

There are researchers who have done SOC related research within the health domain, for instance Calderón-Gómez et al. (2021) conducted performance evaluation of common SOC architectural patterns for developing eHealth applications. This study is not about eHealth applications at all and the health-related scenario in Figure 2.7 is chosen arbitrarily to illustrate the points around complex service composition. Similarly to our study, Zela Ruiz and Rubira (2016, p. 119) developed an experimental "Delivering Clinical Test Results System" around the clinical laboratory domain to demonstrate the case of conflict between performance and accuracy QoS attributes of a fictitious Web Service based application. Likewise a fictitious Semantic Web Service called “Book Doctor Appointment" is used for illustrative purposes only in our study.

The functionality of the “Book Doctor Appointment” service is such that, given the medical condition, preferred date and location, it locates a doctor and schedules an appointment with that doctor on behalf of the user. Figure 2.7 shows a simplified schematic of the service use case:

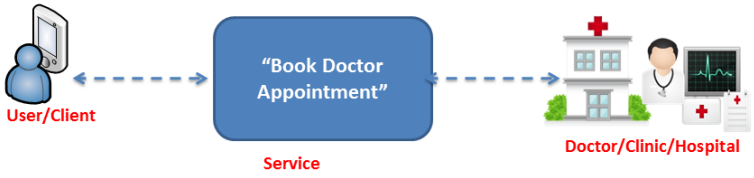


Figure 2.7: Book Doctor Appointment Service Scenario

In the illustrative scenario in Figure 2.7), the user/client interacts with a logical "service" which hides the complexities of handling the location of the available doctors and scheduling an appointment with one of the said doctors. These hidden complexities are what the Semantic Web Service Delivery Lifecycle will discuss in the following subsections.

The sub-cycles of the Semantic Web Service Delivery Lifecycle can be understood as web service usage tasks as described by Bruijn et al. (2008, pp. 17–18). The only difference is that in the delivery lifecycle framework, the publishing of the service description is not visible because publishing is not part of the provisioning of a service but rather the advertising of a service.

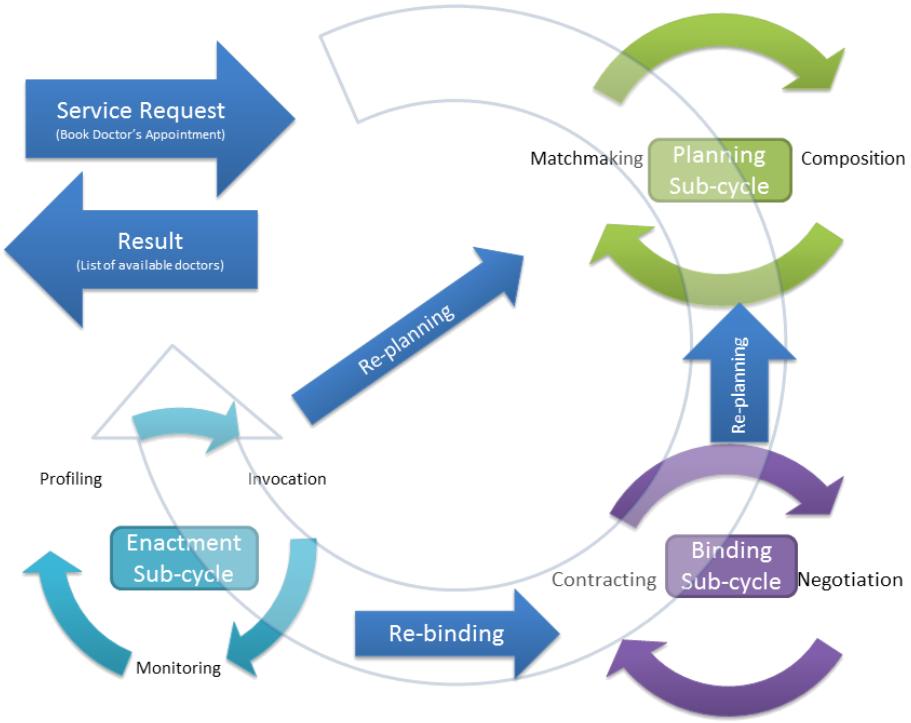


Figure 2.8: Service Delivery Lifecycle (adopted from Kuropka et al., 2008)

The individual sub-cycles of the Semantic Web Service Delivery Lifecycle are discussed in dedicated subsections that follow.

2.5.1 Planning Sub-cycle

The first step of service provisioning is to discover a service that matches the functionality required by the consumer (Bruijn, 2008, p. 17). Upon receiving a service request (e.g. Book a doctor's appointment), the service delivery platform searches the service space (e.g. Internet) for a service that matches the service request as best as possible; this is matchmaking also referred to as service selection. In the simplest case, such a service is found. However, if a matching service could not be located, then the service delivery platform searches the service space successively to discover those services that can satisfy some aspects of the request until the entire service request has been satisfied. This process is known as composition and also determines the order in which these multiple services should be invoked (Bruijn, 2008, p. 17).

Web Service Composition (WSC) is the dynamic integration of multiple Web Services to fulfill the requirements of the task at hand and is one of the most important areas in the Web Services research. In fact, the service composition plane of the Service Research Roadmap pertains to the roles and functionality that are needed for the aggregation of multiple services to form a composite service (Papazoglou, Traverso, Dustdar, and Leymann, 2008). Furthermore, Papazoglou, Traverso, Dustdar, and Leymann (2008) discuss some of the significant challenges to be overcome if Web Service Compositions and Service-Oriented approach to distributed applications are to be widely adopted by the industry. Some of these challenges are: the lack of technologies, methods and tools that support an effective, flexible, reliable, easy-to-use, low-cost, dynamic and time-efficient composition of services.

Papazoglou, Traverso, Dustdar, and Leymann (2008) further mention the autonomic composition of services as one of the key research challenges in Web Service Composition (WSC). When arranging atomic services, the correct order of executing these services are important and must be maintained to achieve the overall goal of the user request. To address this, Raj et al. (2015), proposed an architecture that exploits the abductive event calculus by using the abductive theorem prover to generate a plan for the correct order of execution of the atomic services.

Lara Calache and De Farias (2020) noticed that contemporary tools for semantic annotations using SAWSDL only support the annotation process at a low levels of abstraction, and as such expect the users to have extensive technical knowledge of both XML and WSDL technologies. In addressing this limitation, Lara Calache and De Farias (2020) developed a graphical

annotation tool to support collaborative composition of Semantic Web Services based off the SAWSDL. The work of Lara Calache and De Farias (2020) contributes towards addressing the challenge of developing easy-to-use tools for Semantic Web Service composition.

Autonomic Service Composition (ASC) is based on the tenet of SOC that services are the basic building block of distributed applications and that such distributed applications are developed by combining existing services. According to Agarwal et al. (2008), the automatic composition of Semantic Web Services can be accomplished in a four-stage nominal process: planning, locating or finding a candidate services, selecting the best candidate services and executing the selected service. In their proposal of a formal specification of adaptable Semantic Web Services Composition, Ben Lamine, Jemaa, and Amous (2018) use an ontology-based context model for extending Web services descriptions with metadata about the most suitable context for using the Web service being described. In their work on composition context-based Web Service similarity measure, Zhang, Zeng, et al. (2019) proposed the use of PersonalRank and SimRank++ algorithms to measure similarities between two Web Services to construct what they termed "composition context network".

A solution such as that proposed by Ben Lamine, Jemaa, and Amous (2018) with the usage context included in the description helps locate the most suitable candidate services. Louge et al. (2018) developed an automatic Semantic Web Service composition approach for the astrophysics domain which, as part of the composition, also includes non-Web Services. In terms of the selection criteria, Louge et al. (2018) apply both the non-functional requirements and the user's previous experience feedback to select the most suitable candidates for composition.

Siddiqui and Seth (2017) recognised the task of service selection as a case of solving the Service Selection Effort (SSE) estimation problem in service-based applications. In order to estimate the SSE, Siddiqui and Seth (2017) applied an algorithm based on the Information Entropy Weight (IEW), the fuzzy comprehensive evaluation model, in which case the synthesis performance of each candidate service is evaluated to select the most preferred service (Siddiqui and Seth, 2017). Using this model, the candidate service with the highest synthesis performance has a better chance of being selected.

In terms of publishing, discovering and composing services, Bruijn et al. (2008, p. 20) states that there needs to be functional descriptions of services; the means by which the functionality of service can be described.

Some of the research activities in this sub-cycle include a systematic review conducted by Huf and Siqueira (2019), the research on topics such as Quality of Services (QoS) aware service composition (Hayyolalam and Pourhaji Kazem, 2018), automatic service composition (Paik, Chen, and Huhns, 2014), service selection (Louge et al., 2018; Siddiqui and Seth, 2017; Akingbesote et al., 2013; Reiff-Marganiec, Yu, and Marcel, 2009), and Web Service similarity measure (Zhang, Zeng, et al., 2019), and match-making (Paulraj, Swamynathan, and Madhaiyan, 2011; Cimpian et al., 2008; Medjahed and Atif, 2007; Zeng and Ying, 2008).

2.5.2 Binding Sub-cycle

The most important aspect of service composition is the concept of late binding; the composite service is effectively an abstract service composed of standardized descriptions of atomic services which are subsequently bound to concrete executable instances of such services. This happens in the binding sub-cycle which involves negotiation of any negotiable non-functional properties of a service (e.g. response time) and then signing the digital contracts or agreements to guarantee the negotiated service properties. Bruijn et al. (2008, p. 21) indicate that to enable service binding the non-functional description of a service is necessary to capture properties such as Quality of Services (QoS) parameters like response time, price, and availability.

Liu (2011) explains Service Level Agreement (SLA) as an instrument, a digital contract that defines the relationship between the service provider and consumers (Kowalkiewicz, Ludwig, et al., 2008). These digital contracts are mainly realized through frameworks such as the Web Service Level Agreement (WSLA) (Keller and Ludwig, 2003) as well as the Web Service Agreement (WS-Agreement) (Andrieux et al., 2007). More recently, Telang, Kalia, and Singh (2014) presented a form of digital contract that employs Service Engagements and Commitments. Telang, Kalia, and Singh (2014) argue that such an approach more accurately captures the business relationships between the service participants. This is because a service engagement involves the interaction between two or more autonomous parties as they pursue their business relationships. Through a commitment, a service participant commits to the other participant to bring about some consequence if the prerequisite condition (antecedent) holds true (Telang, Kalia, and Singh, 2014, p. 47).

The agreements are usually established around a set of quality of service

(QoS) parameters. In this regard, Michlmayr et al. (2009) present the QoS model which classifies QoS parameters in terms of performance, dependability, security and trust, as well as cost and payment. Another work with a similar focus was that of Artaiam and Senivongse (2008), which is prompted by the observation that monitoring tools tend to focus on a limited number of QoS parameters and therefore unable to address requirements in practice. Artaiam and Senivongse (2008) proposed an enhanced QoS metric for availability, accessibility, performance, reliability, security, and regulatory aspects of a service. Their proposed solution aims to enhance service-side monitoring of Web Services; the QoS metrics are computed on the service-side.

Service binding can be done in two ways according to Kowalkiewicz, Ludwig, et al. (2008):

- **Binding by selection** – this is about selecting and integrating the most suitable atomic services into the end-to-end service composition.
- **Binding by Agent-Based Negotiation** – this approach uses software agents to negotiate terms of agreement. Software agents are autonomous computer systems that are capable of independent actions on behalf of the user. The negotiation follows certain set of rules that define the circumstances under which interaction between agents takes place – such rules form what is commonly known as the negotiation protocols. The most common of such protocols are Contract Net Protocol (CNP) and Iterated Contract Net Protocol (ICNP), developed by the Foundation for Intelligent Physical Agents (FIPA).

2.5.3 Enactment Sub-cycle

Once the binding has occurred, the functionality provided by the service or service composition can be invoked and the execution results are sent to the service consumer; this is done in the *enactment sub-cycle*. The task of invoking a service requires behavioural and interface descriptions of a service which capture the description of message content to be exchanged as well as the interactions (Bruijn et al., 2008, p. 21).

In terms of enactment of services or service compositions, Kowalkiewicz, Momotko, and Saar (2008) identified five main enactment strategies:

- *First-contract-all-then-enact* strategy – which requires that service selection and contracting of all atomic services is done before execution

so that functional and non-functional properties can be guaranteed (Babaee and Babamir, 2013). Some of the conditional branches might not get invoked, thus rendering the contracting and negotiation inefficient.

- *Step-by-step-contract-and-enact* strategy – instead of contracting all atomic services first, this strategy starts with the first atomic service in the service composition flow; contracting, execution and monitoring are done for each subsequent atomic service. This way, the contracting and negotiation can be done more accurately and efficiently because only invoked atomic services are contracted.

- *Late-contracting-then-enact* strategy – this strategy contracts and enacts only those atomic services that will undoubtedly be executed within a given composition. Thus, rather than contracting conditional branches that may never get invoked, it delays the contracting (hence late-contracting) until such a branch is known to being taken, in which case it contracts all atomic services on the satisfied branch.

- *First-contract-plausible-then-enact* strategy – this strategy relies on historical data of previous executions to select and contract all those atomic services that belong to one composition path which is most likely to be executed.

- *First-contract-critical-then-enact* strategy – this strategy selects and contracts first those atomic services for which there is small number of service candidates than for other atomic services in the service composition. Such services are critical in the sense that they are "hard" to be contracted dynamically, given the small number of candidates.

Kowalkiewicz, Momotko, and Saar (2008, p. 149) provide a set of simple rules that inform the decision to use each of the enactment strategies above. However, even with the aforementioned composition enactment and contracting strategies, there is still a need for a mechanism to perform continuous supervision of the ongoing service enactments Kowalkiewicz, Momotko, and Saar (2008, p. 149). This supervision is achieved through service monitoring and profiling.

Monitoring entails the collecting performance and operational data for services during execution time and thus represents the *source data* for service

profiling. Service profiling involves using the performance and operational data from the monitoring function to derive a condensed description of the service in terms of its non-functional properties. The conceptual principles as well as the monitoring mechanisms for Semantic Web Services are discussed in detail in Chapter 3.

In the enactment sub-cycle there are two possible exit points when a failure is detected, namely *re-binding* (see binding sub-cycle in Figure 2.8) and *re-planning* (see planning sub-cycle in Figure 2.8). Should any errors or failure be detected, the service delivery platform will attempt to redo the binding, which may involve *re-negotiation* and *contracting* possibly with the adjusted service properties (for example a longer response time). The earlier work done by Comuzzi and Pernici (2005), for instance, is about employing different negotiation strategies to negotiate the Quality of Service (QoS) parameters. In instances where the re-binding of a service is not possible or desirable, re-planning is done; this is the complete re-assembling of service components. In terms of the service provision lifecycle, a re-planning is performed as a strategy whereby a new service is found to replace the failing service and the entire process is repeated from planning through to enactment.

Some of research activities in the enactment sub-cycle and specifically on approaches to service monitoring will be looked at more critically in the next chapter dedicated to Semantic Web Service Monitoring. The most neglected area in developing the Semantic Web Service based applications is the area of testing to ensure a certain level of quality of the software itself. Souza Neto, Moreira, and Musicante (2018) conducted a Systematic Mapping to identify and characterize the existing initiatives for SWS testing.

2.6 Semantic Representation of Cloud Computing Services

Semantic representation of Cloud Computing services is an emerging area of research in the application of Semantic Web technology. Di Martino et al. (2014) contend that the application of Semantic Web technology and metadata can be applied to Cloud Computing and can yield benefits within and across the Cloud platforms. One such benefit, according to Di Martino et al. (2014, p. 647), is being able to provide machine-processable descriptions

of Cloud-based resources and cloud services. This capability will in turn enable the automatic evaluation, navigation and consumption of Cloud-based resources and services.

The work of Di Martino et al. (2014) focused on adding semantic and platform-agnostic description to functional as well as the non-functional aspects of Cloud service and enriching the API with meta-data to overcome syntactic differences and reconcile similar semantics between APIs from different providers. They used Microsoft Windows Azure APIs as their use case and created the semantic representation of these APIs to describe the functional and non-functional properties of the Cloud-based services. As reported by Di Martino et al. (2014), the OWL-based ontology was used to describe the Cloud-provider agnostic concepts and the OWL-S was used to describe the real Windows Azure Cloud Services. The work of Şimşek, Kärle, and Fensel (2018) is very similar to that of Di Martino et al. (2014) in that they focus on semantic annotation of Web APIs.

Noting that the focus of most research efforts into the semantic annotation was limited annotating the data, Şimşek, Kärle, and Fensel (2018) proposed the semantic annotation of Web APIs based on the Schema.org [7] in order to support automated service consumption.

Another area of application of Semantic Web technology in Cloud Computing is around the non-functional aspects of Cloud-based services. Greenwell, Liu, and Chalmers (2015) proposed a Description Logic (DL) driven approach to specifying cloud service agreements and modeling these Cloud service agreements as ontology. In responding to the problem of lack of standard nomenclature for Cloud Service Agreements (CSA) terms as outlined by the Cloud Standards Customer Council (CSCC), Greenwell, Liu, and Chalmers (2015) defined and developed an ontology for the Customer Agreements (CA), Acceptable Use Policies (AUP) both of which are the artifacts of CSA.

Also focusing on the semantic description of Cloud SLAs, Mittal et al. (2016) propose an automated process of extracting, managing and monitoring Cloud SLAs using Natural Language Processing (NLP) techniques and Semantic Web technologies. According to Mittal et al. (2016), the key limitations of ordinary Cloud SLAs is that these contractual documents are in plain text format suitable only for human readability. Their approach was to develop NLP-based techniques for extracting SLA details from the contrac-

[7] https://schema.org/

tual documents, describing the extracted SLA information using an ontology that they had developed to support reasoning over the Cloud SLAs. Mittal et al. (2016, p. 140) contend that a critical step in automating Cloud Service Management is to make the Cloud SLAs machine-readable so that software agents can be used for service monitoring, ensuring that the service performs as expected in terms of service contracts. This ability to reason over the SLA also means that the software agents can be used to interpret and enforce the policy rules such as the Acceptable Use Policies (AUP).

Likewise, this study proposes and develops a Monitoring Information eXchange (MIX) protocol for exchanging monitoring information which is based on the idea of making the monitoring information machine-processable by having this information enriched with semantic annotations. Using the Cloud SLA as an example, the MIX protocol could provide a mechanism for software agents to enforce the contracted SLA by monitoring the Cloud service based on the published SLA.

2.7 Summary

This chapter followed the nominal structure of a literature survey on Semantic Web Service. It discussed the classical Web services and its related standards and technologies. The discussion then moved to highlight the limitations of the classical Web services and thus making a case for Semantic Web Services. Discussion on Semantic Web Service then followed and presented the more theoretical principles of Semantic Web Services. The chapter then used the Semantic Web Service Delivery Lifecycle as an illustrative framework to capture the overview of the Service Oriented Computing (SOC) domain and to capture the practical implications of Semantic Web Service technology. In keeping with the Semantic Web Service Delivery Lifecycle, the chapter discussed each sub-cycle in detail, from the Planning Sub-cycle, and the Binding Sub-cycle, and finally the Enactment Sub-cycle.

The next chapter presents a focused literature on Semantic Web Service Monitoring concepts and approaches.

Chapter 3

Semantic Web Service Monitoring

This chapter discusses the conceptual principles of Semantic Web Service Monitoring (Section 3.1) and the actual monitoring mechanisms in Section 3.2. The fundamentals of Quality of Service (QoS) are discussed in Section 3.3 to develop a working definition of QoS, which is itself a key subject of monitoring. This is followed by a discussion, in Section 3.4, on the approaches to semantically describing QoS of Web Services. The review of related works is presented in Section 3.5. Having reviewed existing approaches to Semantic Web Service Monitoring and highlighted shortcomings of related works, this study makes a case, in Section 4.2, for collaborative and corroborative monitoring of Semantic Web Services.

3.1 Conceptual Principles of Semantic Web Service Monitoring

Monitoring involves the collection of performance and operational data for services during execution time. It will be seen in the conceptual principles of Semantic Web Service Monitoring in Section 3.1 that Semantic Web Service Monitoring is essentially about reasoning over the semantically enriched performance and operational data of a service. This section spells out what is meant by Semantic Web Service Monitoring.

Semantic Web Service Monitoring may be understood as simply the use of existing monitoring techniques to monitor the execution of Semantic Web Services. This view of Semantic Web Service Monitoring de-emphasizes the introduction of "semantic" as if the change in the nature of services being

monitored (Semantic Web Services instead of simply Web Services) has no bearing on the nature of monitoring and the techniques thereof. It is argued in this study that the monitoring of Semantic Web Services has serious implications on the nature of monitoring itself. In view of this, Semantic Web Service Monitoring (SWSM) is understood as the use of Semantic Web concepts and technologies in the monitoring of services. Using this definition, the approaches to Semantic Web Service Monitoring tend to take following forms:

- *Using ontologies to formally describe the quality of service (QoS) attributes of services* - this is useful to enable automatic service selection since the software agents will be able to interpret the non-functional aspects of the service. An example of this is the QoSOnt as well as the OWL-Q discussed in Section 3.4.3 and Section 3.4.6 respectively;

- *Using ontologies as a formal conceptualization framework for describing service performance* - using ontologies this way addresses the terminological confusion that may arise during post-execution analyses. The Performance Ontology (Gonsalves and Itoh, 2008) represents one example of this approach;

- *Enriching the service execution event logs with semantic annotations* - these are particularly useful in enabling software agents to reason over historical service execution logs to determine whether the service performed as expected or not. A good application of this is proposed by Vaculín (2009).

The following section draws from contemporary literature on Semantic Web Service Monitoring in terms of the three forms discussed above. Necessarily, the discussion will cover monitoring aspects of both traditional Web Services and Semantic Web Services.

Regardless of which perspective of Semantic Web Monitoring is adopted, any monitoring system has to accomplish three critical functions (Vaculin, 2012, p. 3):

- Measuring of run-time metrics and key performance indicators;

- Measuring and evaluation of Quality of Services (QoS) metrics such as response time;

- Enforcement of Service Level Agreements (SLA);

Therefore, monitoring entails collecting performance and operational data for services during execution time (Wang et al., 2009). Due to the fact that monitoring collects performance and operational data for a service during execution, it provides the *source data for both negotiation and service profiling*. This is because during service selection and match-making when functionally equivalent services are found, their non-functional properties are considered for negotiation – especially the historical performance data.

Monitoring also provide the source data for service profiling. Service profiling entails the use of the performance and operational data from the monitoring function in order to derive a condensed and data-driven description of the service in terms of its non-functional properties. In view of this, a service profile is considered an up-to-date description of a service at any given time (Kowalkiewicz, Momotko, and Saar, 2008, p. 153). By deriving an *up-to-date* description of a service based on its historical performance and operational data, service profiling enables the qualitative comparison of candidate services during service discovery. The ability to perform this qualitative comparison of candidate services is a critical requirement for dynamic service selection.

The dynamic selection of Web Services based on their historical performance and other operational data is significantly improved by using ontologies to describe these data. With specific reference to monitoring, Vaculín (2009, p. 122) lists the following as advantages of Semantic Web Service Monitoring:

- Semantically enriched monitoring data can be shared and processed by software agents;
- Semantic reasoning can be used to support more flexible service execution events detection;
- Semantically enriched execution traces or logs can be reasoned over during post-execution analyses.

The health of the services is monitored for the lifetime of these services in order to *detect* any deterioration in service quality which may lead to breaching of the SLA.

Quality of a Service (QoS) as defined according to the Quality Model for Web Services is what is being monitored. Kowalkiewicz, Momotko, and

Saar (2008, p. 149) warn that it is not practical to formulate the ultimate set of monitoring attributes because of the domain-dependent nature of QoS. Deriving such an ultimate set of monitoring attributes is a unique challenge for generic service monitoring and profiling frameworks.

The classification of monitoring values (run-time metrics) that have a bearing on the QoS is provided by Kowalkiewicz, Momotko, and Saar (2008, p. 149) in terms four quadrants:

- *Quality of Results* – this captures the characteristics of the result content in terms of correctness or accuracy.
- *Quality of Operation* – this captures the quality of result delivery in terms of the service delivery infrastructure. Most common QoS values are related to this class of values as all services share this class of properties.
- *Resource-Level* – this captures the characteristics of the execution environment itself.
- *Implementation-Level* – this captures the characteristics of the service implementation.

This study confines its focus to *performance-related quality attributes* of Semantic Web Services in which the performance and operational data being collected are semantically enriched using ontologies. In this sense, the study focuses on monitoring values in the *Quality of Operation* quadrant.

Having explored the conceptual aspects of Semantic Web Service Monitoring in the preceding discussion, the next subsection looks at the actual mechanism for performing monitoring.

3.2 Semantic Web Service Monitoring Mechanisms

Service monitoring approaches are generally based on two common service level agreement standards, specifically the *WSLA* (Keller and Ludwig, 2003) and the *WS-Agreement* (Andrieux et al., 2007). Both WSLA and WS-Agreement provide the ability to define monitoring parameter types.

Web Service monitoring approaches can be categorized into three main groupings based on which component of the computing infrastructure carries the responsibility of monitoring execution:

- *Service-side monitoring* – in this case, the monitoring is done by the service provider, and all QoS parameters are understood from the perspective of the service provider;

- *Client-side monitoring* – in this case, the monitoring is done by the service consumer or client, and QoS parameters are understood from the client's perspective;

- *Off-site monitoring* – in this case, the monitoring is outsourced, usually by the service provider, to an impartial component (a separate process and infrastructure) that is neither the client-side nor the service-side.

Although there is no formal framework, as of this writing, for characterizing approaches to Web Service Monitoring, the current approaches to monitoring can be readily categorized according to the above groupings. Zela Ruiz and Rubira (2016, p. 116) discussed monitoring configurations which are based on where the "monitor" component is running relative to the Service Consumer and Web Service (producer). These configurations mirror very closely the monitoring approaches as discussed above.

Literature analysis led to the identification of the following classes of service monitoring:

- *Monitoring by proxy* - suggested by Jurca, Faltings, and Binder (2007), here the proxy or broker runs the monitoring process/code - this is easier to manage as the monitoring process is centralized. However, it is costly because the proxy (being a go-between) increases overhead. This can be considered as *off-site monitoring* and as such a dedicated monitoring infrastructure is required.

- *Monitoring using sampling* - this is a variant of monitoring by proxy whereby the monitor takes a sample (mainly historical performance/profiling information) and deduces the quality of service - this reduces overhead; however, the problem with this is that it is imprecise and relies solely on historical performance information (Jurca, Faltings, and Binder, 2007). This is also an *off-site monitoring* approach.

- *Provider-Self-Monitoring* (internal to the service provider, as part of a service) - here, the service provider runs the monitoring process or code. The mechanics of monitoring a service are built into the service, eliminating any noticeable overhead. The client is at the mercy of the service provider who is the sole source of performance information; this is likely to be biased or untrustworthy (especially where penalties may be incurred for violating performance agreements such as QoS). This is a *service-side monitoring* approach.

- *Monitoring by Client Feedback* - as explicated by Jurca, Faltings, and Binder (2007), in this client-side monitoring approach, the client runs the monitoring code. The *client-side monitoring code* embeds preferred performance metrics to check the services against. The benefit of this, as opposed to service-side monitoring, is that it is independent of the service provider who now cannot temper with the monitoring process. There is no overhead and in terms of precision/accuracy - it may be precise, but there is no reason not to believe that clients can deliberately report incorrect information in an attempt to claim violation and receive compensation through penalties - which may then make the service significantly cheaper for them (they get some money back with each penalty). This necessitates incentivising correct reporting so that clients stand to gain nothing by lying about their service experiences (in this instance, they will get a certain amount of money back with each "honest report").

- *Monitoring by surveillance ("Monitoring-as-Service")* - in this case, there is a dedicated monitoring service infrastructure that performs surveillance, closely monitoring interactions between the service consumer and provider, checking this behaviour against the agreed-upon QoS/SLA. This is similar to passive monitoring strategy as discussed by Zela Ruiz and Rubira (2016). In this approach neither the client nor the service provider is burdened with monitoring concerns; it is less intrusive. However, this is very expensive as it requires duplication of service provisioning infrastructure for a diagnostic/monitoring function. We refer to this approach to monitoring as an off-site or outsourced monitoring; it is outsourced by either the consumer or the service provider – but usually the latter. Monitoring tools in this category apply two monitoring strategies according to Cabrera and Franch

(2012), namely *passive monitoring* by sniffing the interactions between service consumers and providers, and *active monitoring* by actively requesting performance data from the service being monitored.

In addition to the three main classes discussed above, there is a specialised technique for QoS monitoring which expresses the measurable QoS attributes in terms of probability quality property in a way discussed by Grunske (2008). The technique is aptly named probabilistic monitoring and is concerned with monitoring the probabilistic properties of QoS (Grunske and Zhang, 2009). Using this technique, the probabilistic property of the QoS parameter, such as the response time, can be expressed as *the probability that the response time of a particular Web Service is within certain duration (example, 5 seconds)*. In their experiment Zhang, Jin, et al. (2018) applied this technique to consider the probabilistic characteristics of QoS.

Monitoring tools for Semantic Web Services, or Web Services more generally may fall into each of the categories or classes discussed above. According to Zela Ruiz and Rubira (2016), for instance, some monitoring tools adopt either a *passive* monitoring strategy which is essentially a monitoring by surveillance, or an *active* monitoring strategy which is a combination of client-side and service-side monitoring approaches.

The monitoring tools were developed to perform a continuous and accurate assessment of Semantic Web Services to ensure that the agreed SLA is being honored (Cabrera and Franch, 2012). In their work on the use of quality model for analysis of Web Service monitoring tools, Cabrera and Franch (2012) discussed several of these monitoring tools. Zela Ruiz and Rubira (2016) also discussed monitoring tools but from the perspective of potential risks and problems that can be introduced by intrusive monitoring tools. For the purpose of this research, it is worth mentioning that none of the monitoring tools discussed supported collaborative approach to monitoring or the exchange of semantically enriched monitoring information between the service consumer and service provider.

The preceding discussion made reference to Quality of Service (QoS) as the basis of quality models for Web Services and thereat a key subject of Semantic Web Service Monitoring. The next section (Section 3.3) discusses the fundamentals of QoS.

3.3 Quality of Service (QoS) Fundamentals

Quality management and quality assurance vocabulary defines Quality of Service (QoS) as a set of all the features and characteristics of a service that have a bearing on the service's ability to satisfy user/consumer needs (ISO IEC 8402, 1994). Kritikos and Plexousakis (2007) aligned closely to this definition, in that they view QoS as a set of non-functional properties encompassing performance and network-related characteristics of resources. Sabata et al. (1997) contend that an effective definition for QoS must reconcile the intrinsic "subjectivity" of quality definition and the "objectivity" required whenever two or more different perspectives of quality have to be compared. Oriol, Marco, and Franch (2014) conducted a systematic review of existing perspectives and proposals on quality models for Web Services purporting to facilitate the reconciliation of existing definitions of quality factors applicable to Web Service usage. Oskooei and Mohd Daud (2014) explored the use of technical and non-technical QoS attributes for the evaluation of Web Services.

A hierarchy of quality concepts (Figure 3.1) captures the relationship between QoS, quality characteristics, quality attributes and the metric to measure these quality attributes.

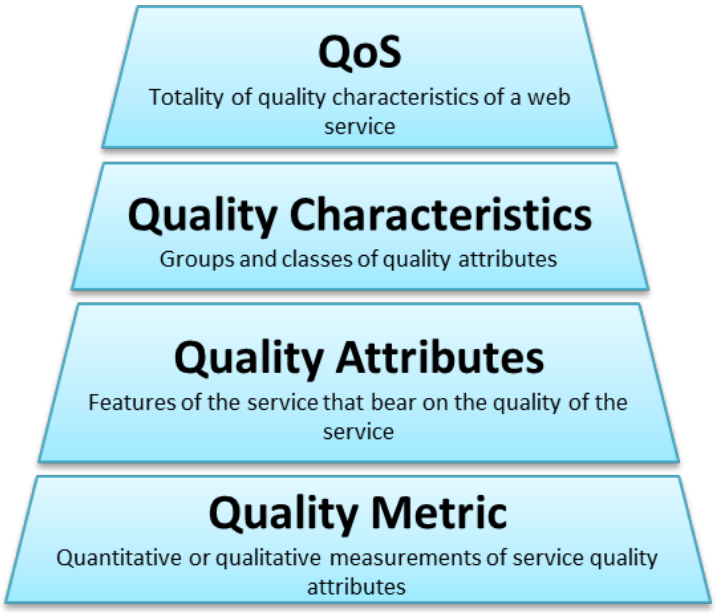


Figure 3.1: Hierarchy of quality concepts (adapted from Oriol, Marco, and Franch (2014))

The relationship between the quality concepts depicted in the hierarchy is such that QoS expresses the overarching quality features and characteristics of a service. The quality characteristics of a service encompass all classes of those features of a service that have a bearing on the objective service quality. Quality attributes are those features of a service that have a bearing on the quality of a service. In this regard, a key contribution of Michlmayr et al. (2009) is a QoS model which classifies quality characteristics of a service in terms of *performance*, *dependability*, *security and trust*, as well as *cost and payment*. Quality metrics are the quantitative or qualitative measurements of service quality attributes. The work of Artaiam and Senivongse (2008) proposes enhanced QoS metrics for *availability*, *accessibility*, *performance*, *reliability*, *security*, and *regulatory aspects* of a service.

Although the hierarchy of quality concepts (Figure 3.1) makes the relationship between these concepts explicit, terminological confusion still occurs whereby researchers use the concepts such as quality attributes and quality characteristics interchangeably. Oriol, Marco, and Franch (2014, p. 1169) generalize the concepts of quality attribute and quality characteristic into

what they refer to as a quality factor.

QoS is comprised of a multitude of attributes as described above, this study focuses specifically on those quality attributes pertaining to the performance of a service. In this regard the service's *response time* is considered an important QoS attribute that characterizes the performance of a service.

Response time is a good candidate for collaborative and corroborative monitoring because it is understood from two very different perspectives by service providers and service consumers.This means that service providers measure response time in a very different way from service consumers. From the perspective of a service provider, the response time of a service is measured as the time between receiving service request to the time that the response is sent out; in other words, service providers view response time as an execution time. However, from the perspective of service consumers, the response time of a service is measured as the time between sending the request and receiving the response.

As shown in Makitla and Mtsweni (2015), these two perspectives on response time lead to two different values when the same execution is observed. The problem, in this instance, is that the same QoS parameter is understood differently by both the service provider and the consumer (e.g. response time vs execution time). This problem arises due to the fact that there is no *generalized metric for calculating the response time*. Such a generalized metric would help to develop consensus on contracted QoS parameters. Makitla and Mtsweni (2015) applied the descriptive statistical technique of linear regression to develop a generalized metric for response time in the context of Semantic Web Services. The generalized metric is given by Equation (3.1) :

$$Response - Time(generalized) = srt + (az).\dot{x} \tag{3.1}$$

The generalized response time in Equation (3.1) is arrived at by adding the server-side response time (srt) and the product of the historical mean-error ($\dot{x}$) and the historical average deviation of each error value from the mean-error (az). As suggested in Makitla and Mtsweni (2015, p. 376), the advertised QoS parameter for response time should be based on this generalized metric.

The following section (Section 3.5) explains the relationship between service description and service monitoring. This is important specifically because the OLW-S and WSMO semantic frameworks are actually considered in terms of their semantic richness in *describing* Semantic Web Services and

their non-functional properties. An important feature that ought to be part of any service description is the *Quality of Service* (QoS) because it is a major factor for consideration during service discovery and selection. Information contained in the QoS may relate to performance (e.g. response time, latency, and throughput), availability, accessibility, and aspects of trust and security among others. In Semantic Web Services, the idea is for the semantic annotations to be extended to the non-functional (QoS) aspects of the service description as well.

3.4 Semantic Description of Quality of Services

This section discusses the rationale and the approaches to enriching non-functional descriptions of Semantic Web Services using ontologies. With the proliferation of Web Services came the problem that prompted the need to enrich non-functional descriptions of Web Service with semantic annotations. The problem was that service discovery was not sufficiently accurate and Web Service registries will return many functionally-equivalent Web Services as matches for the user request, or as candidates for Web Service composition.

Kritikos and Plexousakis (2007) proposed a possible solution to this by way of an ontological specification that enables the semantic description of QoS for Web Services. Such a semantic QoS-based Web Service description complements a semantic framework like OWL-S which provides the semantics for describing functional aspects of a Web Service. Although introducing QoS as part of the service description may improve service discovery by facilitating the quantitative comparison of functionally equivalent Web Services, there may still be lingering issues of semantic interoperability. These interoperability issues arise due to the fact that, according to Zhang and Yang (2013), different service providers and consumers may use different QoS concepts and models for describing service quality information (Tran, Tsuji, and Masuda, 2009).

The application of semantic technology, specifically the ontology, is seen as an enabler for semantic interoperability across various languages, concepts and models that are used to describe QoS information (Tran, Tsuji, and Masuda, 2009, p. 1379).

In order to make sense of both the case for semantically enriching non-functional aspects of Semantic Web Service as well as the approaches to doing so, it is necessary to discuss *ontologies* as they apply in Semantic Web

technology.

3.4.1 Ontologies

According to Vaculín (2009), an ontology is a *formal explicit specification* of terms in the domain of interest and *relations* among them. In this sense, an ontology serves as a common conceptualization of domain knowledge (Mohebbi, Ibrahim, and Idris, 2012). Put differently, ontologies represent the conceptualizations or understanding of the meaning of things or concepts within a particular domain (Acuna and Marcos, 2006). Ontologies are also often conceived as a set of *entities*, *relations*, *axioms* and *instances* within some domain (Lera, Juiz, and Puigjaner, 2006, p. 28). Furthermore, and as used in Semantic Web technology, ontologies represent the *connective structures* that express the relationships between information resources on the Web as well as connecting these resources to their formal terminologies (Fensel, 2003).

Ontologies are important because they enable the communities of interest (i.e. *people* or *software agents*) to perform the following key functions (Lera, Juiz, and Puigjaner, 2006;Kritikos and Plexousakis, 2007):

- Sharing of common understanding,
- Reuse of domain knowledge - building ontologies based on other well-defined ontologies,
- Making domain assumptions explicit,
- Separating domain knowledge from the operational knowledge, and
- Reasoning about concepts within a domain by performing logical inference.

In order to support intelligent analysis of and reasoning about Semantic Web Service performance, it is necessary to develop performance domain ontologies. As mentioned in Section 3.3, QoS represents a set of all the features and characteristics of a Semantic Web Service, enabling it to satisfy user requests and preferences. In view of this, the QoS expresses the overarching quality features and characteristics of a Semantic Web Service chief among which are the performance-related characteristics. Verily the semantic specification of QoS is invariably achieved through the development of

performance ontologies. Consequently, the QoS ontologies being discussed in the following subsections represent the basis for the semantic description of performance characteristics of Semantic Web Services.

For the semantic description of QoS aspects of Semantic Web Services to be possible, a high-level service ontology is required which will serve to promote consensus around quality aspects of Semantic Web Services. It is important to note that such an ontological specification for QoS has to serve three key roles:

- Express a set of *constraints on a certain subset of QoS metrics* as part of a user-defined preference during service request - this is then used to filter candidate Web Services returned by Web Service registries;
- Express a set of *commitments by the service providers to guarantee certain non-functional properties of the advertised Web Service* as a differentiator from functionally-equivalent Web Services;
- Characterize *actual or derived performance of a Web Service* which can be checked against the published QoS specifications of a Web Service to determine whether the service actually performed as advertised;

The subsections that follow explore a number of such QoS specification ontologies.

3.4.2 QoS Ontology Language

The *QoS Ontology language* is concerned with the development of *QoS ontology* and *vocabulary* for semantic specification of QoS information for Web Services. This work has been motivated by the need to efficiently represent the variety of information on QoS parameters in such a way as to support machine processability and the ability to reason over this QoS information (Papaioannou et al., 2006). Verily the QoS ontology language (Lilien et al., 2011) provides a standard ontological model to formally describe arbitrary QoS parameters including the relationships between QoS parameters as well as the methods of measuring these parameters (Papaioannou et al., 2006, p. 102).

The main classes of the QoS ontology language are *QoSParameter*, *Metric*, *QoSImpact*, *Type*, *Nature*, *Aggregated*, *Node*, and *Relationship*. Papaioannou et al. (2006, pp. 103–104) provide full descriptions of all these classes.

As will be seen in subsequent sections, different ontological models refer to similar concepts by different names. The two key concepts in the QoS ontology language are *(1)* the QoS parameter (*QoSParameter*) which represents a non-functional property of a Web Service which may be measurable or static, and *(2)* the metric (*Metric*), which specifies how a specific QoS parameter gets assigned a value (i.e measured). The *hasMetric* property is assigned to those QoS parameters that are measurable and its value is expressed in certain units of measure. The *QoSImpact* class is used in the model to describe the impact of a particular QoS parameter on the overall service quality. Using this technique it is possible, for instance, to express the impact of high throughput on response time.

One of the earliest efforts with regard to QoS specification ontologies is the development, by Dobson, Lock, and Sommerville (2005), of a base ontology for QoS aptly named QoSOnt. The key aspects of QoSOnt which are relevant to this study are discussed next in Section 3.4.3.

3.4.3 QoSOnt: Basic QoS Ontology

According to Dobson, Lock, and Sommerville (2005), QoSOnt was developed to promote consensus on QoS concepts amongst service consumers and service providers. Although the work of Dobson, Lock, and Sommerville (2005) on QoS ontology (QoSOnt) predates the conceptualisation of the hierarchy of quality concepts in Figure 3.1 by Oriol, Marco, and Franch (2014), the QoSOnt does *codify* the relationships between quality concepts using ontologies as a formal conceptualisation. QoSOnt provides a base set of useful concepts which cover common use cases for QoS and specifically for monitoring. The base QoS ontology is depicted in Figure 3.2.

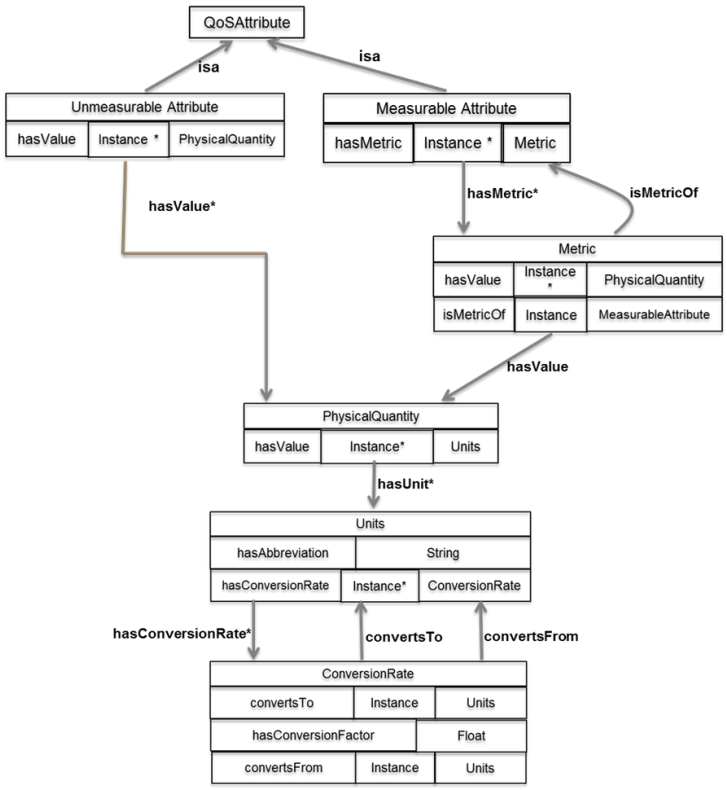


Figure 3.2: Base QoS Ontology (adapted from Dobson, Lock, and Sommerville (2005))

As can be seen from the base ontology (Figure 3.2), *QoSAttribute* is the base class of all QoSOnt concepts and can represent either a *Measurable Attribute* or an *Unmeasurable Attribute*. The significant difference between these sub-classes of QoSAttribute (i.e. Measured and Unmeasured attributes) is that by definition a measurable attribute must have a *Metric*. A metric is essentially a specification of a measurement method and includes such details as value ranges and units (Kritikos and Plexousakis, 2007). In this sense, a metric has a value which is a physical quantity. A *Physical Quantity* is distinguishable from any other quantity such as a number (e.g. '3'). This number does not convey any meaning in and of itself; however, by adding a unit of measure to this number (e.g minutes), then suddenly the number represents time. As such a Physical Quantity must have a *Unit*. This also means that a *metric represents a way of measuring some QoS at-*

tribute using some unit of measure. The Unit may be converted to and from other units using a *ConversionRate* - for example from minutes to seconds with a ConversionRate of 60 (number of seconds in a minute). This conversion becomes very helpful when two collaborating systems use different units. Finally, and as depicted in Figure 3.2 an Unmeasurable Attribute has a Physical Quantity but does not specify any method of measure (i.e. a metric) because it is not being measured.

In Semantic Web Service monitoring, the main focus is on the measurable performance parameters. In order for the measurements of performance parameters to be amenable to logical inference and reasoning it is necessary to describe these parameters semantically. Since the standard of measure for these parameters is expressed as a QoS metric, by the same token it is necessary to be able to describe these QoS metrics semantically. Furthermore, because QoS metrics are essentially formulae describing how the values of these performance parameters are derived, then the requirement is to describe these formulae semantically. This aspect of QoS is largely ignored in most of the literature on QoS and is mainly described generally as non-functional properties of Semantic Web Service. Section 3.4.4 presents related literature that pertains to the semantic description of arithmetic or mathematical formula expressions as would be used in QoS metric description.

3.4.4 QoS Metrics and the Semantic Descriptions of Arithmetic Expressions

According to Papaioannou et al. (2006), the key function of a QoS ontology language is to provide a facility in the form of a standard ontological model to formally describe, among other things, the methods of measuring the QoS parameters. To describe these QoS parameters is essentially to describe their mathematical formulae. For example, to formally describe the metric for "throughput" would be to describe the mathematical expression in Equation (3.2) below:

$$Q_{throughput} = 1 - (Q_{RequestsNotCompletedSuccessfully}/Q_{TotalRequestsProcessed}) \quad (3.2)$$

Ferre (2012) looked at techniques to enable the representation of complex mathematical and logical expressions in RDF so that these rich expressions can be explored through semantic search. The advantage of this, as suggested by Ferre (2012), is that it becomes possible to search for these formal

expressions by their contents and thus to support the mathematical search. Ferre (2012) proposes *(1)* an RDF vocabulary for mathematical expressions to support expressive structured search, *(2)* syntactic extension of Turtle and SPARQL syntaxes to enable notation of descriptions and queries, and *(3)* generating labels for the expressions that are close to their mathematical symbols.

Although not directly related to semantic representation of performance metrics of a service, the work of Benvenuti et al. (2017) provides a good example of how computational semantics can be made more explicit. Using the formula concept of the KPIOnto [1] ontology, which is an ontology for semantically describing Key Performance Indicators (KPI), Benvenuti et al. (2017) demonstrated how formally express one KPI as a function of other KPIs. Their framework uses **PR**olog **E**quation **S**olving **S**ystem (PRESS) to support reasoning over the mathematical expressions (Benvenuti et al., 2017).

As seen in the preceding discussion, QoSOnt provides a base set of very useful concepts for describing QoS and was built specifically for this purpose. Furthermore, it is possible to see how a service parameter in the OWL-S framework can make use of QoSOnt whereby the service parameter of a service is linked to a concept (*ServiceParameter*) in the QoSOnt model. This study adopts QoSOnt as a base ontology for describing quality attributes of a Semantic Web Service during service publishing as well as annotating service performance data during post-execution and monitoring. QoSOnt is preferred because of its generic nature and simplicity.

The next section describes another QoS ontology which bears some similarities to QoSOnt but also has some important concepts not found in the QoSOnt model.

3.4.5 WS-QoSOnto: Web Service QoS Ontology

WS-QoSOnto is used to describe detailed QoS information and allows service producers to express their QoS offers and the service consumers to express their QoS demands at different levels of expectation (Tran, Tsuji, and Masuda, 2009).

QoS Property is the key concept in the WS-QoSOnto model much as the *QoS Attribute* is the central concept in QoSOnt model. According to

[1] https://kdmg.dii.univpm.it/kpionto/specification/

Tran, Tsuji, and Masuda (2009, p. 1379) the QoS property describes the non-functional aspects of a Web Service. Tran, Tsuji, and Masuda (2009) distinguish between two types of QoS properties that represent quality aspects of a Semantic Web Service, namely the *technical* QoS properties and *managerial* QoS properties. According to this categorization, the technical QoS properties encompass those properties that are related to operational aspects of Semantic Web Services such as *usability*, *efficiency*, *reliability*, *availability*, *accessibility* and *performance*. The managerial QoS properties on the other hand relate to those descriptive properties that capture the service management information such as *ownership*, *provider*, *contractual details*, and *methods of payment*. For the purpose of this study, the focus is on the technical QoS properties and, more specifically, on performance-related QoS properties.

According to the WS-QoSOnto model, *Performance* is one of the direct subclasses of QoS property (*QoSProperty*). Ontologically, this means that Performance *is-a* QoS property. Furthermore, Performance has *Latency*, *Throughput*, and *Response Time* as its direct sub-classes and therefore suggests the existence of an *is-a* relationship between these and *Performance*. This relationship can be read as stating that Latency, Throughput, and Response Time together characterize the performance of a Semantic Web Service. Characterizing the performance in this manner also suggests that these subclasses (Latency, Throughput, and Response Time) are the measures of performance. In other words, they represent the metrics of Semantic Web Service performance.

WS-QoSOnto has the concept of metric (*QoSMetric*) which is used in evaluating specific QoS properties. The metric may have units of measure or it may be unitless. In the WS-QoSOnto model each metric has a measurement directive (*MeasurementDirective*) as well as the QoS formulation (*QoSFormulation*) both of which provide the necessary information regarding how to measure specific QoS value (*QoSValue*). This conceptualization differs slightly from the QoSOnt model where a metric has a value representing a physical quality (*PhysicalQuantity*) which by definition has a unit (*Unit*).

Whereas the measurement mechanisms (i.e. formulae) are not fully described in the QoSOnt model, the *QoSFormulation* of WS-QoSOnto describes the formula used to calculate a specific metric. Similarly to QoSOnt, the units in WS-QoSOnto can be converted to other units (e.g. from minutes to seconds).

Based on the similarities and significant differences between WS-QoSOnto and QoSOnt models, the most reasonable approach would be to merge these two models through subclassing and equivalence axioms where applicable in order to derive benefits from both these models.

Although both WS-QoSOnto and QoSOnt models represent QoS aspects of a Semantic Web Service the extension points for either WSMO or OWL-S are not sufficiently emphasised. The next subsection describes an OWL-S extension that provides a QoS-based description of Web Services.

3.4.6 OWL-Q: QoS-based Web Service Description

OWL-S provides a semantically rich framework for describing the functional aspects of Semantic Web Services but this semantic richness did not extend to the QoS information. The OWL-Q is an OWL-S extension that provides a QoS description model for Semantic Web Services (Kritikos and Plexousakis, 2007).

In OWL-Q, a number of sub-ontologies called *facets* are defined and can be extended independently and thus providing for syntactical separation and refinement of QoS specifications (Kritikos and Plexousakis, 2007, p. 126). Each aspect of the QoS description has a dedicated facet. According to Kritikos and Plexousakis (2007) OWL-Q is essentially an *upper ontology* made up of a total of eleven (11) facets: *OWL-Q(main facet)*, *Measurement Directive*, *Time*, *Goal*, *Function*, *Measurement*, *Metric*, *Scale*, *QoSSpec*, *Unit* and *ValueType*. The description of each of these facets is given by Kritikos and Plexousakis (2007, pp. 126–130). In this subsection, only the Measurement Directive, Metric, Scale and QoSSpec facets are discussed in relation to the other ontological models of QoS described in preceding subsections.

The *Metric Facet* describes the QoS metric model, capturing all the classes and properties that are necessary to provide formal definition of QoS metric (Kritikos and Plexousakis, 2007, p. 128). In the OWL-Q ontology the *QoSMetric* class represents a QoS metric and its value is provided by an *Actor*. QoS metric measures QoS attributes (*QoSAttribute*) that are measurable (*MeasurableAttribute*) on a specific ServiceElement (Kritikos and Plexousakis, 2007, p. 129). Axiomatically this means that *a QoSAttribute that has a QoSMetric is by definition a MeasurableAttribute*. The same axiom is expressed in QoSOnt.

QoSSpec is the facet that enables the standardised way for Semantic Web Service providers and consumers to define their QoS constraints (Kritikos and

Plexousakis, 2007, p. 127). The two sub-classes in this facet are *QoSOffer* and *QoSDemand*. Through the QoSOffer the Semantic Web Service providers can define their QoS guarantees. On the other hand, Semantic Web Service consumers can specify their QoS preferences or constraints by way of QoS-Demand. Furthermore, whereas QoSOffer is used during the publishing of the Semantic Web Services, QoSDemand is used during discovery whereby multiple functionally equivalent Semantic Web Services are filtered based on how well their particular QoSOffers satisfy the current QoSDemand. In terms of monitoring, the Semantic Web Service consumer agent will observe the execution of the service to ascertain that it behaves and performs in a manner that is consistent with the published QoSOffer.

The *Measurement Directive* facet specifies how QoS metrics are measured. This is equivalent to measurement directive (*MeasurementDirective*) and QoS formulation (*QoSFormulation*) that provide information on how to measure specific QoS values in the WS-QoSOnto. However, in the OWL-Q model the Measurement Directive facet further specifies the mode of getting the QoS value from the managed resources (Kritikos and Plexousakis, 2007, p. 128). In the *pull* mode, the party responsible for measurement will ask for the value and in the *push* mode the QoS value is made available by the managed resource (*Actor*) itself.

For each metric of a QoS attribute the *Scale* facet controls the value type as well as the set of operations that are permissible (Kritikos and Plexousakis, 2007, p. 129). Examples of such scales are *nominal*, *ordinal*, *interval*, *ratio* and *absolute*. The Scale facet thus specifies how value expressions on a particular scale can be transformed to the value expression of a compatible scale. The value expressions in any scale (for instance *interval*) represent a particular unit of measure (time). In this sense, and according to Kritikos and Plexousakis (2007, p. 129), *Scale* is a general notion of *Unit*.

3.4.7 WSMO-QoS

WSMO-QoS is a framework extension to the Web Service Modeling Ontology (WSMO) and describes the non-functional aspects of Semantic Web Services based on WSMO. According to Li and Zhou (2009, p. 70), their work on WSMO-QoS is motivated by the acknowledgement that whereas WSMO and OWL-S frameworks sufficiently describe the functional aspects of Semantic Web Services, they both fall short of providing accurate service discovery mechanism that selects candidate Semantic Web Services with consideration

of their respective QoS values. In view of this, WSMO-QoS augments the WSMO framework with an *ontology-based meta-model* of QoS (Li and Zhou, 2009, p. 70).

The central concept in the WSMO-QoS meta-model is *QoS Metric* which is defined by four main elements; namely its *Name*, *Unit*, *ValueType* and *URI* (Li and Zhou, 2009, p. 72). Similarly to WS-QoSOnto and QoSOnt models where QoS metric represents a standard of measurement for some QoS attributes, here a QoS attribute is also measured by one or more such QoS metrics. Unlike WS-Onto, the WSMO-QoS meta-model does not explicitly distinguish between QoS offers and QoS demands but instead uses the concept of role (*Role*) to indicate the origin of the QoS information. Possible values of Role are *Service Provider*, *Service Requestor* and *Third Party*. Therefore, implicitly, if Role is indicated as Service Provider then the QoS information expresses an offer. If Role is indicated as Service Requestor, then the QoS information is interpreted as a QoS demand or requirement. Third parties also issue QoS information as offers - these third parties are usually partners to service providers.

The concept of *QoS Relation* is introduced to express the relationships between QoS parameters (Li and Zhou, 2009, p. 71). It is possible for two QoS parameters to be inversely proportional, such as in the case of response time and the throughput; the higher the throughput, the lower the response time or vice versa.

The WSMO-QoS model further defines two broad categories of QoS attributes; *static* and *dynamic* attributes. The static QoS attributes are specified during the designing/publishing of the service and do not change between executions. Dynamic QoS attributes on the other hand may vary during service execution. An example of dynamic QoS attribute which is also relevant in our study is *performance*. The performance sub-ontology of the WSMO-QoS framework is composed of four QoS parameters; *latency*, *response time*, *throughput*, and *error-rate* (Li and Zhou, 2009, p. 71). These will be revisited during the development of our performance ontology for collaborative monitoring in Chapter 4.

3.4.8 SMOnt: Service Monitoring Ontology

Masood, Cherifi, and Moalla (2016) developed what they call a performance monitoring framework for service-oriented system lifecycle. They argue in their work that in order to accurately monitor the performance of a service-

oriented system (SOS), it is necessary to monitor performance at different levels of the SOS lifecycle; service level, business process level, server level and the binding or network messaging level. They designed a Service Monitoring Ontology (SMOnt) which defines performance ontologies for the technical performance indicators at the different levels of the SOS lifecycle to derive what they call a service performance profile (Masood, Cherifi, and Moalla, 2016, pp. 803–805).

The main shortcoming of the ontologies that Masood, Cherifi, and Moalla (2016) developed is that they are not explicitly linked to the main QoS concepts using the subclassing rule (*see* sub-sections 3.4.2 and 3.4.3). The inability to conceptually link SMOnt to the fundamental concepts in QoS makes it difficult to reuse it.

Having discussed the ontological foundations of QoS in the preceding subsections, it is worth emphasizing that QoS was being discussed because it is actually the *monitoring* of QoS that is the subject of Semantic Web Service Monitoring. So whereas QoS in the service description represents the service provider's promise to potential service consumers, this *advertised* QoS is also used during service monitoring to ascertain if the service performs consistent with its supposed QoS.

3.5 Related Works

This section describes some of the research efforts that are closely related to this research. These works tackle the challenge of Semantic Web Service Monitoring using the perspectives as described in Section 3.1. The selection criteria for these related works are as follows:

- The work has to be about monitoring of Web Services or more strictly Semantic Web Services;
- The work has to demonstrate appropriate application of Semantic Web technology for monitoring;
- There is a clear focus on developing (even if only conceptually) some mechanism for monitoring Semantic Web Services;
- The work focuses on mechanisms by which service providers *collect*, *aggregate* and *analyze* the service performance data;

- The researcher(s) developed (even if only conceptually) mechanism for publishing or exchanging of monitoring data (i.e. service performance data);

The selection criteria is helpful in filtering out most of the research works which, even though they discussed QoS ontologies (see Subsections 3.4.2 - 3.4.7), did so only for the purposes of supporting the discovery of Semantic Web Services. This study is mainly interested in the monitoring of Semantic Web Service executions. The research works that met at least one of the selection criteria above are discussed in the following subsections.

Zhang, Jin, et al. (2018) responded to the problem of how to monitor QoS parameters of Web Service timely and accurately in dynamic environments by developing **I**nformation **g**ain and **S**liding **w**indow based weighted naive **B**aye**S**ian **R**untime **M**onitoring (IgS-wBSRM) - a weighted naive Bayesian runtime monitoring approach which incorporates the sliding window mechanism combined with information gain theory. This combination of sliding window mechanism and information gain theory, according to Zhang, Jin, et al. (2018, p. 15), provides an improved algorithm for dynamically updating the weights of QoS impact factors (such as service location versus client location) as more and more performance data is collected. The sliding window mechanism introduces the time-awareness whereby historic and out-of-date sampled performance data is discarded. In this manner the technique enables the monitoring results to always consider the latest performance data.

Pradhan et al. (2016), in their literature study, gave a summary of research developments and challenges in the Web Service Based Systems (WSBS) monitoring domain, including the research work of Baresi and Guinea (2013).

The work of Baresi and Guinea (2013) is motivated by the need for cross-layer monitoring, which is a paradigm shift from the traditional approach to software monitoring, which always monitored the service-oriented systems at the application layer and treating lower layers (i.e. hardware layer) as constants. Baresi and Guinea (2013) proposes a cross-layer monitoring of service-based systems. The key aspect from their work is the customizable and extensible approach to collecting, aggregating and analysing data from across the layers in order to identify the root causes of unexpected behaviours. To achieve this, Baresi and Guinea (2013, p. 84) developed an extensible declarative language that allows designers to define the various data they want to collect from all the layers, how to aggregate the data sets and analyse them to discover undesirable behaviour. The collection of performance data

is an essential part of monitoring and this is why the proposal of Baresi and Guinea (2013) is important. Whereas our study focuses on the exchange of this monitoring information between the consumer and the service provider, the work of Baresi and Guinea (2013) focuses on how the service provider collects, aggregates and analyzes the service performance data.

In his work, Vaculín (2009) observed that contemporary event monitoring and filtering systems rely on instances of primitive events that are described at the syntactic level. A major drawback of such systems is that the detection mechanisms of primitive events are restricted only to exact event types detection via syntactic match (Vaculin, 2012 ; Vaculín, 2009, pp. 121–122). One of the key contributions made in this regard is the development of a semantic monitoring framework based on OLW-S (Vaculín, 2009, pp. 121–144). The approach developed by Vaculín (2009) entails applying the semantic annotations to descriptions of event types and event instances emitted during interactions with Semantic Web services. The event types are organized into a taxonomy and this taxonomy of events, along with event parameters and associated data are defined in an ontology. This has been achieved by developing an ontology of event types where each event type is represented by one OWL class. The events types are specific to the execution of OWL-S based Web Services.

A sample event definition is shown in Figure 3.3 :

```
<AtomicProcessEndEvent>
  <timestamp>2007-03-12T12:35:12</timestamp>
  <process rdf:resource="&shoppingService;Login"/>
  <input>
    <ParameterValueBinding>
      <toParameter rdf:resource="&shoppingService;username"/>
      <dataValue>John</dataValue>
    </ParameterValueBinding>
  </input>
  <input>
    <ParameterValueBinding>
      <toParameter rdf:resource="&shoppingService;password"/>
      <dataValue>secret word</dataValue>
    </ParameterValueBinding>
  </input>
  <output>
    <ParameterValueBinding>
      <toParameter rdf:resource="&shoppingService;status"/>
      <dataValue>true</dataValue>
    </ParameterValueBinding>
 </output>
</AtomicProcessEndEvent>
```

Figure 3.3: Sample Event Instance (adapted from Vaculín (2009, p. 128))

The benefits of using ontologies this way, as suggested by Vaculín (2009,

p. 122), are that:

- Events and their content can be easily processed and shared by software agents;
- Semantic reasoning can support more flexible even detection which is better than pure syntactic matching;
- Semantically enriched interaction traces/logs can be reasoned over as part of post-execution analyses;

The emphasis of the semantic monitoring in Vaculín (2009) is on event detection and the author adopted an event detection algebra as the formalism for the definition and detection of complex event patterns (Vaculín, 2009, p. 128). Event detection algebra entails using a number of *operators* and the *primitive events* to formulate *event expressions* that describe an event pattern to be used to detect event occurrences.

Whereas Vaculín (2009), as further expanded in Vaculin (2012), proposes annotating the execution trace with ontology concepts, he does not specify how this monitoring data (i.e. execution trace) generated by service provider can be shared with service consumers. This study proposes a Monitoring Information eXchange (MIX) protocol whereby the service provider would include, as part of the service response, a URL of the ontology document that contains the semantically enriched service performance data. This service performance data is described using the Resource Description Framework (RDF). The URL is sent instead of the actual document to avoid increasing the payload size of the service response. A Semantic Web Service consumer agent can retrieve the semantically enriched document and reason over the service performance data. Chapter 4 presents full discussion on the proposed Monitoring Information eXchange (MIX) protocol as well as the technical infrastructure necessary to enable the storage and exchange of the semantically-annotated monitoring information.

Singh and Liu (2016) developed a cloud service architecture for analyzing big monitoring data. The architecture, called *TraceAnalyzer*, exposes REST APIs that enables the monitoring data to be accessed by different analysis modules.

The *TraceAnalyzer* does not provide real-time monitoring but instead offers a powerful collection of tools for performing post-monitoring analyses. In terms of the category of monitoring approaches (see Section 3.2), the work

of Singh and Liu (2016) follows an *Off-site monitoring* approach in which case the analyses of monitoring data are done after execution and whereby the process that collects the monitoring data and exposing it through REST APIs for analysis is separate from the actual process that is running and generating the monitoring data. In terms of the semantic enrichment of the monitoring data, the *TraceAnalyzer* uses the Semantic Media Wiki (SMW). The SMW adds additional markup into the wiki-text to enable semantic annotations of the text that allow a wiki to function like a collaborative database (Singh and Liu, 2016, p. 56). The added semantic annotations allows for the wiki documents to become machine-processables so that the user can query the monitoring data wikis to make sense of the analysis results coming from the architecture. This is done on the top layer of the *TraceAnalyzer* architecture as depicted in Figure 3.4:

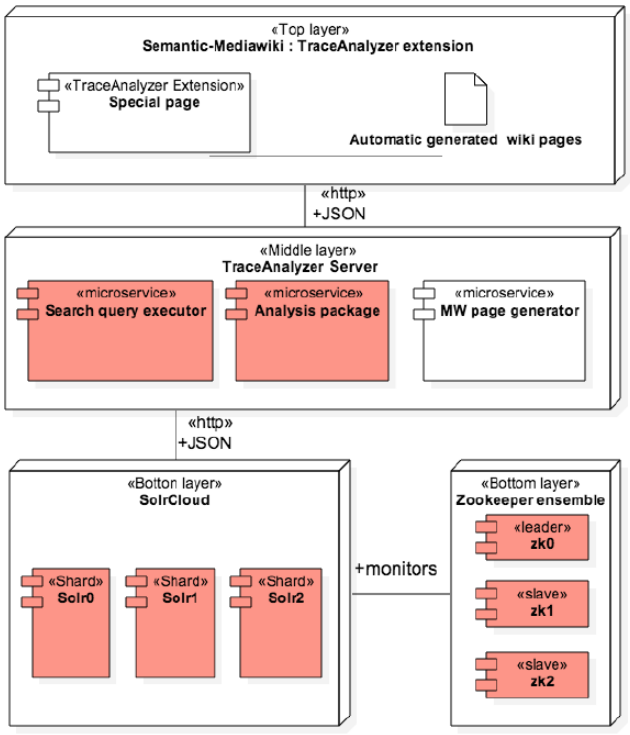


Figure 3.4: TraceAnalyzer Core Architecture (Singh and Liu (2016, p. 57))

The work of Keeney et al. (2011) is motivated by the lack of a structured

approach to monitoring diverse heterogeneous systems as though they were one homogeneous end-to-end system. Specifically Keeney et al. (2011, p. 658) identify and aim to address three challenges of managing diverse heterogeneous systems namely *(1)* cost-effectively harmonizing monitoring data from diverse devices, systems and services, *(2)* inability to express and fuse end-to-end service, system and network information in an interoperable manner, and *(3)* inability to facilitate meaningful participation of service consumers in the quality of experience control loop. This last point relates most closely to this study that proposes active participation of service consumers in the collaborative and corroborative monitoring of Semantic Web Services.

In terms of the harmonizing of monitoring data, Keeney et al. (2011, p. 658) suggest that mechanisms are required to semantically enrich the large data produced by the monitoring systems. The meaningful participation of service consumers in monitoring the quality of experience is enabled through personalized visualization of harmonized monitoring data (Keeney et al., 2011, p. 659).

As part of their research, Keeney et al. (2011) developed a testbed monitoring and management framework which is composed of several functional components. The adapted and simplified diagram of their testbed framework is depicted in Figure 3.5

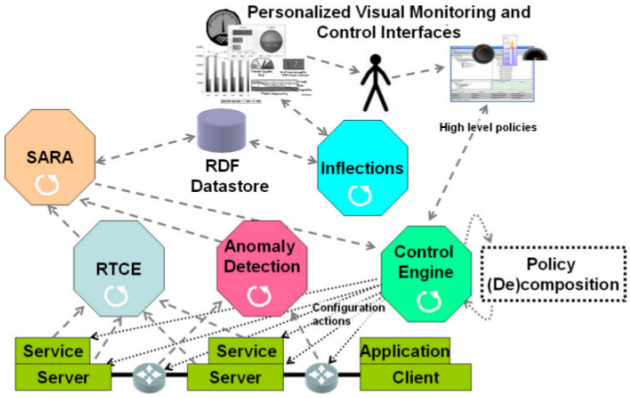


Figure 3.5: Simplified Framework for End-to-End Monitoring and Management (Keeney et al. (2011, p. 659))

The three components that are of interest to this study are *(1)* INFLECTIONS, *(2)* Semantic Attribute Reconciliation Architecture(SARA), and *(3)*

Semantic-based Service Control Engine.

The *Semantic Attribute Reconciliation Architecture (SARA)* provides mechanisms for monitoring data harmonization by semantically *lifting*, *reconciling* and *mapping* information from heterogeneous sources into a common semantic structure (Keeney et al., 2011, p. 660).

The *INFLECTIONS* component on the other hand supports the exploration of semantically enhanced information (from SARA) and enables domain experts to define reusable views of harmonized information.

According to Keeney et al. (2011, p. 660), the *Semantic-based Service Control Engine* enables the efficient management of semantically encoded monitoring information and events. The semantically encoded information is taken from SARA, while the events represent triggers that cause the enactment of configuration actions by service components so that their performance is consistent with the agreed quality of experience (Keeney et al., 2011, p. 660).

Semantically enriching monitoring information, as proposed by Keeney et al. (2011), is a critical step in facilitating semantic monitoring, but without the support for the exchanging of this information between participating service-oriented systems, it falls short of enabling collaborative and corroborative monitoring as proposed by this study.

Al-Hazmi and Magedanz (2015) developed a **M**onitoring **O**ntology for **F**ederated **I**nfrastructures (MOFI) ontology to enable the exchange of sensory monitoring data across the federated infrastructures. The work of Al-Hazmi and Magedanz (2015) involved creating an ontology-based information model to semantically describe the federated infrastructures and their underlying resources in order to facilitate, among other life-cycle events, the discovery, selection, reservations, provisioning and monitoring of the resources (i.e. sensors or devices).

The work of Al-Hazmi and Magedanz (2015), as well as more recent experimental work of Nejkovic et al. (2020), are all based in the Internet of Things (IoT) domain and their use of semantic annotation is mostly to enrich the high-level descriptions of sensor data. There is no evidence of IoT devices themselves monitoring each other or exchanging their monitoring sensory data. Instead, the IoT platform itself is responsible for collecting the sensory data from the devices or sensors connected to the platform.

Benvenuti et al. (2017) noted that the Public Transport Systems (PTS) are finding it difficult to maintain high quality of public transport services to improve citizens mobility in their respective regions of operation. In response

to the difficulties experienced by PTS, Benvenuti et al. (2017) developed an ontology-based framework aimed at supporting performance monitoring in public transport systems. KPIOnto is the ontological representation of performance indicators and their formulas (Benvenuti et al., 2017). According to Benvenuti et al. (2017), KPIOnto is an ontoloty that provides main concepts and properties for semantic descriptions of the KPIs. A specialized reasoning framework then developed to process and reason over the semantically enriched KPI data of the PTS. The formula in the KPIOnto model enables a formal representation of a KPI as a function of one or more KPIs and by so doing it makes the computational semantics of the referenced indicator more explicit.

The work of Benvenuti et al. (2017) does not explicitly refer to QoS but rather considers the Key Performance Indicators (KPI) which are mainly used for diagnosis purposes. Specifically, their work focused on monitoring the physical transport services, as opposed to Semantic Web Services. Furthermore, reasoning over the semantically enriched KPI data is done within the system (PTS) in order to report how the transportation system actually performs in relation to the KPIs.

Letia and Marginean (2011) proposed a client-side monitoring framework for the non-intrusive monitoring of service behavior. This approach requires that the service provider introduces a SOAP message handler which captures these SOAP messages in a request-response pairs and makes them available for analysis. The SOAP messages, according to Letia and Marginean (2011), are to be interpreted based on the consumer's current context using the **DOLCE** [2] ontology situations, descriptions, and expectations.

The monitoring in the context of the work of Letia and Marginean (2011) focuses on observing client-service interactions in terms of functional requirements and service expectations of the client. This means that performance and other QoS related metrics are not being monitored, but rather the SOAP messages which carry the functional aspects of the service are enriched with the ontology-based information model. This client-side monitoring relies the client's perception of what the service does and therefore focuses solely on service functionality. Furthermore, the use of ontology is for enriching the request-response SOAP message pairs that are generated by client (requests) and the service (responses). The monitoring is collaborative in the sense that both the client and the service provider contribute to the logging of request

[2] https://triplydb.com/vocabulary/dolce

and response SOAP messages respectively.

Although not specifically focusing on monitoring, the works of Amir (2018) and Sigwele et al. (2018) provide a good conceptual model for understanding how semantic annotation can facilitate the exchange of information between participating systems, whether humans or machine agents. The purpose of their study was instead to develop a generic collaboration framework which can support any type of sensor and any type of Web of Things (WoT) application. Amir (2018) conceptualize a collaboration framework in the context of the Web of Things (WoT) as being made up of three phases: *(1)* Capture and represent data, *(2)* Generate knowledge, and *(3)* Collaboration; share and exchange information and knowledge.

The top-level conceptual architecture of a collaboration framework is depicted in Figure 3.6

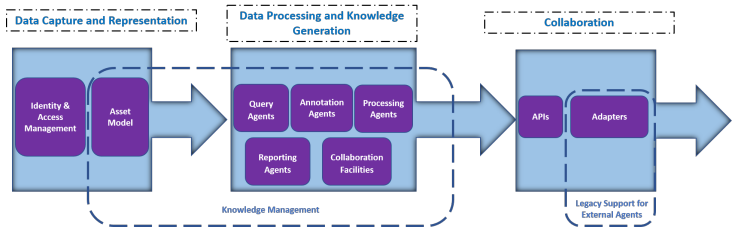


Figure 3.6: Architecture of a Collaboration Framework (adapted from Amir (2018, p. 11))

In the *Capture and Represent Data* phase, the sensor data is captured from local sensor networks and from repositories that expose their data through APIs. This phase is comprises two components, namely (1) Identity and Access Management, and (2) Asset Model. The Identity and Access Management component handles the authentication and authorization of internal and external actors who want to access and interact with the data stored in the asset model. On the other hand, the Asset Model handles the modeling and representation of the sensing devices and data (or readings) in a platform-specific manner. The purpose of the Asset Model is to make the sensing devices' readings or data available to the other components of the collaboration framework for further processing and semantic enrichment.

The second phase of the collaboration framework is the *Data Processing and Knowledge Generation* phase. The phase is comprised of five (5) components which represent the (1) Query Agents, (2) Annotation Agents,

(3) Processing Agents, (4) Reporting Agents and (5) Collaboration Facilities. During this phase, the data read from local sensor network or imported from other repositories is contextualized to represent some meaningful information. The process of transforming raw data into useful information is done through a variety of methods, including semantic annotation and enrichments. An example of this might be assigning meaning to a numerical sensory value based on the sensing device, so that value of 35 coming from the temperature sensor is converted into 35 degrees (35°.) which carries more meaning once the units (degree) is augmented. See the section discussing the ontological model called *QoSOnt* in Section 3.4.3.

The third and final phase of the collaboration framework is the *Collaboration* phase and as shown in Figure 3.6 comprises two components; the (1) APIs and (2) Adapters. The process of collaboration and information exchange is enabled by developing an API and/or legacy adapters. The API exposes the semantically enriched information whilst the legacy or platform-specific adapters are used to collaborate with systems which may be using some proprietary schemata.

Lara Calache and De Farias (2020) noted that contemporary semantic annotation tools focus mainly on the lower levels of abstraction which requires the users to have technical knowledge of XML and WSDL. Noticing this gap, Lara Calache and De Farias (2020) proposed a graphical collaborative semantic annotation support tool which is aimed at facilitating the semantic annotation of WSDL-based Semantic Web Services. Their proposed tool enables different people to make their individual contributions to the semantic annotation. Although the work of Lara Calache and De Farias (2020) focuses on collaboration within Semantic Web Service domain, its focal area is in the description of the Semantic Web Service. Our study, by contrast focuses on the collaborative collection and corroboration of semantically enriched monitoring information.

Sigwele et al. (2018) drew extensively from the work of Amir (2018) to propose an intelligent semantic gateway which is used by collaborating healthcare systems to expose semantically enriched healthcare information through a RESTful API. The relevance of this work to our study is that it highlights the importance of semantic annotation in facilitating collaboration across heterogeneous systems. The gap, however, is that the information being exchanged in the framework proposed by Sigwele et al. (2018) is healthcare information and not semantically enriched monitoring information.

Table 3.1 summarizes the related works by indicating, for each work, the

manner in which ontologies are being used in monitoring, the monitoring approach used, and whether the monitoring approach is collaborative and corroborative.

Research Work	**Ontology Usage**	**Monitoring Approach**	**Collaborative** and **Corroborative**
Keeney et al. (2011)	Semantic annotations to enrich monitoring information	Service-side monitoring	No.
Letia and Marginean (2011)	Ontology-enriched request-response SOAP message pairs	Client-side monitoring	Partially.
Vaculín (2009)	Semantic annotations to descriptions of event types and event instances emitted during services execution	Service-side monitoring	Partially.
Singh and Liu (2016)	Semantic annotations to enrich monitoring data analysis results	Off-site monitoring	No.
Zhang, Jin, et al. (2018)	Relies on underlying ontological models of QoS and introduces probabilistic characteristics of QoS	Service-side monitoring	No.
Al-Hazmi and Magedanz (2015)	Ontology-based information model for monitoring federated infrastructures in IoT	Off-site monitoring (IoT platform service)	No.

Benvenuti et al. (2017)	Ontology used to formally describe key performance indicators (KPIs) and their formulae	Off-site monitoring (PTS platform service)	No.
Amir (2018)	No reference to QoS but deals with generic collaboration framework in Web of Things (WoT)	N/A	Partially.
Sigwele et al. (2018)	No reference to QoS but provides potential applications of semantic annotation of healthcare information to facilitate collaboration across heterogeneous healthcare systems	N/A	Partially.
Lara Calache and De Farias (2020)	Semantic annotations to Web Service description using a graphical collaborative semantic annotation support tool	N/A	Partially.

Table 3.1: Summary of Related Works

Although it is not mentioned explicitly in the research, the monitoring approach developed by Vaculín (2009) could potentially support collaborative monitoring since the data can be shared as part of the post-execution analysis. The practical difficulty with this approach is that the service execution logs will include traces from multiple executions from multiple consumers. This study proposes the exchange of monitoring data during service execution so that the post-execution analyses can be done separately by both the service providers and the service consumers.

3.6 Summary

This chapter has looked at the conceptual principles of Semantic Web Service Monitoring and also discussed monitoring mechanisms that adhere to some of these principles. The fundamentals of Quality of Service (QoS) were discussed in view of it being a key subject of Semantic Web Service Monitoring. The approaches to semantically enriching the QoS information in the descriptions of Semantic Web Services were discussed. The discussion on semantically enabled QoS models was followed by a review of related research works that focused on the monitoring functions of Semantic Web Services. Having reviewed existing approaches to Semantic Web Service Monitoring and highlighted shortcomings of related works, a case is made in the next chapter for collaborative and corroborative monitoring of Semantic Web Services.

The next chapter discusses the proposed solution to achieving Collaborative and Corroborative Semantic Web Services Monitoring.

Chapter 4

A Monitoring Information Exchange (MIX) Protocol

This chapter firstly addresses the research objective **(RO-1)** which pertains to the development of a generalized metrics for specific QoS parameters (such as Response Time) to enable both the service consumers and service providers to reach consensus on the meaning of each of these QoS parameters. Secondly, the chapter also addresses research objective **(RO-2)** which is about developing a Monitoring Information eXchange (MIX) protocol as a technology-agnostic protocol for service consumers and service providers to exchange their respective monitoring information. The chapter specifically presents the conceptual aspects of the approach for Collaborative and Corroborative Monitoring of Semantic Web Services as the solution proposed by this study.

4.1 Introduction

The key enabler for the proposed Collaborative and Corroborative Monitoring approach is the exchange of monitoring information between participating systems. This exchange of monitoring information imposes *three* key requirements as discussed in Section 1.3:

- Developing a mechanism by which participating monitoring systems can develop *consensus* on observable QoS parameters and attributes;
- Mechanism by which monitoring information can be *exchanged*;

- Technology-neutral *negotiation protocol for conflict resolution* in case of Service Level Agreement (SLA) breach or inconsistencies in the monitoring data;

The remainder of this chapter is structured as follows: Section 4.2 re-emphasizes the case for Collaborative and Corroborative Monitoring based on Semantic Web technology in order to address the monitoring information exchange requirements. Section 4.3 describes the development of a generalized metric for computing the response time of a Web Service which is a key feature for collaborative and corroborative monitoring. Section 4.4 then presents performance ontology as the basis for semantic description for the performance characteristics of a Semantic Web Service. The use of ontologies in this way is purported to promote consensus on observable QoS parameters and attributes among participating monitoring systems. Indeed the generalized metric for response time will be described semantically as part of this performance ontology. Finally, Section 4.5 presents the mechanism by which the semantically enriched monitoring information can be exchanged between the service consumer and service provider during, or after service execution.

4.2 Case for Collaborative and Corroborative Monitoring Using Semantic Web Technology

The utility of being able to reason over the semantically enriched service performance data has long been recognized (Vaculín, 2009; Kritikos and Plexousakis, 2007; Dobson, Lock, and Sommerville, 2005). Contemporary researchers do not seem to have considered an even greater advantage of being able to exchange or share this semantically enriched monitoring data with other participating components or systems in a service-oriented computing environment. Specifically, the exchange of semantically enriched monitoring data between service providers and service consumers has not been considered at all. As already mentioned in Section 1.1, the benefits of being able to exchange the monitoring data are as follows:

- It *eliminates the potential for conceptual confusion* resulting from the fact that both the service consumer and the service provider may have

different perspectives of the same QoS parameter (Zela Ruiz and Rubira, 2016, p. 118) and thus having no consensus on the meaning of individual QoS parameters.

- Enables the *support for a flexible pricing model*, which is a potential competitive advantage for service providers; consumers feel empowered to ensure that they receive the appropriate quality of service for the money they pay. This is a desirable alternative to paying penalties; the service provider may negotiate with the consumer to rather pay an amount appropriate for the quality of service experienced.

- It *eliminates the need for incentive schemes* aimed at encouraging honest reporting by service consumers. Both the service consumer and service provider exchange and corroborate each other's monitoring data which, due to semantic annotations, they can both reason over the monitoring data.

- There is *no need for a dedicated infrastructure for monitoring by surveillance* because the service consumer and service provider exchange their environment-dependent monitoring information directly.

The following section discusses the generalized metric for response time of a Web Service. The generalized metric is necessitated by the need to develop consensus on what constitutes the response time since both the service provider and service consumer have a different perspective of this QoS parameter.

4.3 Generalized Response Time Metric

This section draws from the work, originally published in (Makitla and Mtsweni, 2015), describing the development of a *generalized metric for computing response time* of a Web Service. This generalized metric is useful in developing consensus around the meaning of contracted QoS parameters. In doing this, the generalized metric helps to avoid the confusion that may arise when the same QoS parameter is understood differently by both the service provider and the consumer (e.g. response time versus execution time). In the absence of this generalized metric for response time, contemporary monitoring approaches resort to incentivising the service consumers to honestly report their perception of service quality.

The main objective of the Makitla and Mtsweni (2015, pp. 375–376) experiment was to develop a generalized metric which *characterizes* the performance (in terms of response time) of a Web Service under specific resource conditions.

The response time values were measured on both Web Service provider and consumer ends. A lightweight monitoring library based on the Java programming language called JAMon [1] was used. The JAMon library allows the developer to provide a label for each monitor. The monitors are named using the developer-specified labels. The JAMon API also exposes *start* and *stop* methods used for starting and stopping monitors, respectively. When called, the "stop" method marks the end of monitoring and the performance statistics are available through the API. These performance statistics include the average response time per monitor (i.e. label) which is the main focus of this study. Since the experiment involved a unique monitor per request, the average response time per monitor is the same as just the response time (dividing by "1" has no effect).

In the data-driven experiment a total of ten thousand (10,000) uniquely labelled requests were made to the same Web Service resource. Response times were recorded for both client (Web Service consumer) and the server (Web Service provider). For each request a unique monitor label was derived by combining the method name "getDoctors" and the request number (1 to 10000), for example, *getDoctors1*, *getDoctors2*, up to *getDoctors10000*. Using this technique, we specified a unique label for each web service request at the client-side and noted the monitoring values per request.

To ensure that both the client and service response times are recorded for the same request – the client passes its monitor label along with the request (as a parameter) – the web service then adopts this very label for its own monitor for this request. With this technique, it was possible to collect 10000 pairs of client-side response times and server-side response times independently. Due to the fact that tabulating the entire set of 10000 pairs would take up too much space in this thesis, Table 4.1 only lists a few pairs for illustration purposes. The response times are all in milliseconds.

[1] http://jamonapi.sourceforge.net/

Label	**Client-side (ms) Response Time (ms)**	**Service-side Response Time**
getDoctors1	105.0	100.0
getDoctors2	105.0	100.0
...	...	...
getDoctors9998	102.0	100.0
getDoctors9999	102.0	100.0
getDoctors10000	103.0	100.0

Table 4.1: Response-time pairs per request

Statistical analyses were performed to characterize the *variance* of client-side and server-side monitoring values for the same request across all 10000 requests. As can be seen in Table 4.1 there are differences between the client-side response time (crt) and the server-side response time (srt) values. We refer to this difference as "error" and is given by $crt \dot{-} srt$. This is an "error" because the Web Service provider claims a different response time (e.g. 100.0ms) from what the consumer is experiencing (e.g. 105.0 ms). By using the statistical variability of these "error" values it is possible to get to a server-side response time value which will have the smallest "error" possible.

First we compute the "mean error" ($\dot{x}$) as the average of differences between crt and srt for all requests (N). The mean error is given by Equation (4.1):

$$\dot{x} = [\sum_{i=1}^{N}(crt_i \dot{-} srt_i)] \div N \tag{4.1}$$

Applying Equation (4.1) to the experimental data in Table 4.1 for all the 10000 pairs of data points given by ($crt \dot{-} srt$), the value of "mean error" (for $N = 10000$) is:

$$\dot{x} = [\sum_{i=1}^{10000}(crt_i \dot{-} srt_i)] \div 10000 = 2.2616 \tag{4.2}$$

Having calculated the "mean error" in Equation (4.2) we now compute the deviation (dev) of each of the "error" values (ev) from the "mean error"

above. For this we used *absolute* values (ignored negative signs):

$$dev = [\sum_{i=1}^{10000} (ev_i \dot{-} \dot{x})] \div 10000 = 1.7384 (ev_i = [crt_i \dot{-} srt_i]) \tag{4.3}$$

In order to understand the spread of the monitoring values, we computed the average squared deviation (i.e. *variance*):

$$variance = [\sum_{i=1}^{10000} (ev_i \dot{-} \dot{x})^2] \div 10000 = 1.1693 \tag{4.4}$$

The standard deviation (σ) then works out to be: $\sqrt{variance} = 1.0813$.

Finally, each error value (crt – srt) is examined to see how many standard deviations each of these values are from the population mean (of all 10000 values). For this we calculate the z-score for each error-value for all 10000 entries:

$$z = (ev_i - \dot{x})/\sigma \tag{4.5}$$

However because the purpose of the experiment was to characterize the response time value over a number of Web Service interactions (i.e. 10,000) it is not meaningful to look at the individual z values instead we compute the average *z-value* (az) using absolute values:

$$az = [\sum_{i=1}^{10000} |Z_i|] \div 10000 = 2.5323 \tag{4.6}$$

The value of az from Equation (4.6) indicates the "more or less" value that the Web Service client should expect given the response time from the server-side. Based on this, a generalized metric for response time, or the Generalized Response Time Metric (GRTM) can be formulated as:

$$GRTM = srt + (az).\dot{x} \tag{4.7}$$

In Equation (4.7), both az and $\dot{x}$ are "historical" values known from previous executions or, in the case of this study, from our experiment of 10000 identical requests.

These descriptive statistical values characterize the performance of the *getDoctors* Web Service method under certain resource conditions. The characterization suggests a linear regression model, where the variability (beta

value) or the average-z value (az) is taken from memory (historical or previous executions) to predict the most likely client-side response time value. However, it must be noted that in this experiment, we were not interested in what independent variables had an effect on the response time. As such, we did not attempt to attribute the observed response time on the client-side to any variable; we mainly wanted to develop a generalized metric which characterizes the performance of a system under specific conditions of our experiment. In fact, Zhang, Jin, et al. (2018, p. 15) caution that the influence of environmental factors may lead to incorrect monitoring results and thus a misleading performance characterization of the Web Service being monitored. QoS values such as response time are affected by multitudes of factors. For this reason, Zhang, Jin, et al. (2018) proposed a weighted naive Bayesian run-time monitoring approach which combines information gain and the sliding window mechanism to dynamically re-adjust the weights of different environmental factors impacting on the response time.

The point of developing this generalized metric is that the QoS parameter for response time advertised by the Web Service providers should be based on this generalized metric. Furthermore, the generalized metric describes, mathematically, a formulation of how to compute response time through collaborative monitoring. The performance ontology (Section 4.4) then describes this formulation semantically so that software agents may understand how to compute response time when they receive the performance data during monitoring information exchange (see Section 4.5).

In Section 3.4.1, it was explained why the discussion on QoS ontologies invariably focuses on the semantic description of performance characteristics of Semantic Web Services. Actually, the semantic specification of QoS is achieved through the description of those concepts which are the basis for performance characteristics. Section 4.4 brings together the concepts discussed in Sections (3.4.2 - 3.4.7) to develop a performance ontology. The performance ontology will, in turn, describe the semantically enriched performance information to be exchanged by participating systems as part of collaborative and corroborative monitoring.

4.4 Semantic Web Service Performance Ontology for Monitoring

The performance ontology provides the basis for the semantic description of the performance characteristics of a Semantic Web Service. In terms of the WSMO-QoS model (Li and Zhou, 2009), such a performance ontology is comprised of four QoS parameters namely the *latency*, *response time*, *throughput*, and *error-rate*. To provide a comprehensive ontology for performance it is necessary to locate this performance ontology as a sub-ontology in the larger QoS ontology. To do this, QoSOnt is considered as the basis for QoS ontology and is expanded to include those concepts found in ontological models discussed in Sections (3.4.2 - 3.4.7).

For collaborative and corroborative monitoring purposes, we consider only those performance-related QoS parameters that are measurable, and for which the service provider and service consumer may have different perspectives. Only the *response time* parameter meets this criteria. In Service Oriented Computing (SOC), *response time* is an important aspect for characterizing performance.

Figure 4.1 below depicts a basic ontology for Performance.

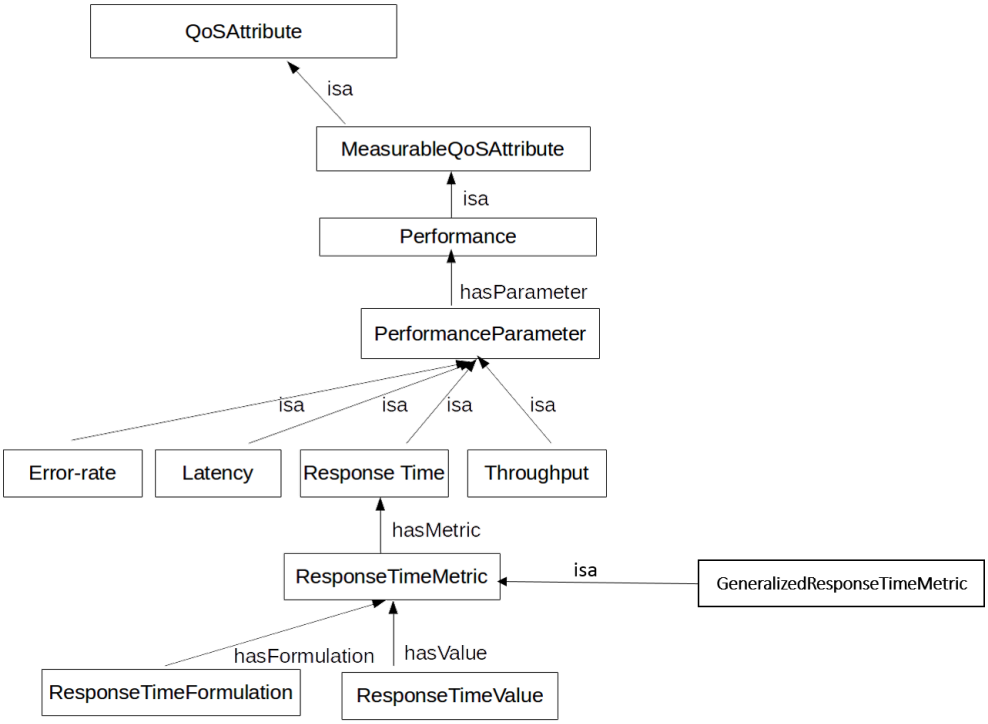


Figure 4.1: Performance Ontology for Monitoring Information Exchange

The simplified performance ontology in Figure 4.1 conveys the idea that *Response Time* is a measurable *parameter* of performance which in itself is a *measurable* QoS attribute. This is inferred from the fact that the Response Time has a *metric* (i.e *ResponseTimeMetric*). The value assigned by this response time metric, ResponseTimeValue, is measured in *Time* units. Ontologically, *Generalized Response Time Metric* is a specialization, or a subclass, of the *ResponseTimeMetric* which is a metric of Response Time. This can be seen in the *"isa"* relationship between *ResponseTimeMetric* and *GeneralizedResponseTimeMetric* concepts.

The performance ontology in Figure 4.1 also illustrates how the response time, as a performance parameter, relates to the performance characteristics of a service. Importantly, the depicted performance ontology positions the *Generalized Response Time Metric* in relation to the four other performance parameters which are *latency*, *response time*, *throughput*, and *error-rate*.

Sachan, Kumar Dixit, and Kumar (2014, p. 89) accurately describe performance as a composite attribute comprising the weighted sum of latency,

throughput, and response time. Defined this way, and with the inclusion of error-rate, the logical expression of performance is :

$$Q_{performance} = \sum_{i=1}^{4} W_i * Q_i \tag{4.8}$$

Where Q_i =$Q_{latency}, Q_{throughput}, Q_{response-time}, Q_{error-rate}$.

The Latency, denoted as $Q_{latency}$ above, refers to the delay before a transfer of data begins following an instruction for its transfer. In the context of this study latency refers specifically to a network latency which signifies a delay in the transfer of data along the connectivity network. Latency affects both the request arrival time (at the service side) and the response arrival time (at the client/consumer side). In this sense, Latency is defined as:

$$Q_{latency} = Q_{responsetime} - ExecutionTime \tag{4.9}$$

Throughput is defined as the total number of requests/transactions that a Web Service completes over a period of time and has values such as $2000/sec$ to denote 2000 requests/transactions completed per second. In their work of formalizing the QoS parameters, Sachan, Kumar Dixit, and Kumar (2014, p. 91) defined *Throughput* as:

$$Q_{throughput} = 1 - (Q_{incomplete}/Q_{experience}) \tag{4.10}$$

Where $Q_{incomplete}$ represents the number of requests/transactions that the Web Service did not complete successfully, and $Q_{experience}$ denotes the number of times that the Web Service was selected to handle a request. In fact, the quotient $(Q_{incomplete}/Q_{experience})$ can be understood as the *error rate* of a Web Service as it shows the percentage of failed Web Service executions. In this sense, the metric for *Error-Rate* is given by:

$$Q_{error-rate} = Q_{incomplete}/Q_{experience} \tag{4.11}$$

Finally, the response time ($Q_{response-time}$), which is the focus of this study, refers to the total delay from when the request is sent and when the response is received. In this sense, Equation ((4.9)) can be restated as follows:

$$Q_{responsetime} = Q_{latency} + ExecutionTime \tag{4.12}$$

Practically $Q_{latency} + ExecutionTime$ is what the service provider would have submitted as its response time. To generalize this response time metric (as discussed in 4.3) it is necessary to take the consumer-side response time values into account and then compute the average of their differences across multiple executions. Having done that we can restate the Generalized Response Time Metric (GRTM) as :

$$Q_{GRTM} = (Q_{latency} + ExecutionTime) + (az).\dot{x} \tag{4.13}$$

Where $Q_{latency} + ExecutionTime$ refers to the service-side response time in collaborative monitoring, and as such, it can be denoted as srt. Then the Generalized Response Time Metric (GRTM) is finally stated as:

$$Q_{GRTM} = srt + (az).\dot{x} \tag{4.14}$$

The graphical representation in Figure 4.1 is humanly easy to understand; however, for a real application scenario where software agents must be able to understand the ontological model, it is necessary to semantically describe the performance ontology using appropriate syntaxes. Describing the performance ontology further necessitates encoding the logical expressions for each metric of the four performance parameters; $Q_{latency}$, $Q_{throughput}$, $Q_{response-time}$, and $Q_{error-rate}$. The semantic description of these logical expressions or formulae is necessary to support the corroboration of performance data among the monitoring agents. These logical expressions, being semantically described, can be understood by the monitoring agents. The monitoring agents are then able to execute these logical expressions (*arithmetic formulae*) to reproduce the same performance values as those produced by other participating monitoring agents. This is how corroboration of performance data among the monitoring agents is achieved.

This study focuses on the Response Time parameter of performance, and as such, Sub-section (4.4.1) discusses and presents the semantic description of the Generalized Response Time Metric (Q_{GRTM}). This is what the monitoring agents will reason over in order to corroborate the response time values reported by the other participating monitoring agent.

4.4.1 Semantic Description of Generalized Response Time Metric

Whereas Section 4.3 describes the mathematical formulation of how the *Generalized Response Time* is computed collaboratively, Section 4.4.1 provides

the semantic description of the logical expression or formula of Generalized Response Time Metric (i.e. $srt + (az).\dot{x}$) using the Turtle as well as the RDF syntaxes.

The work of Ferre (2012) forms the basis of this aspect of the study. Whereas the main thrust of Ferre (2012), as discussed in Section 3.4.4, was to solve the difficulties relating to mathematical search, the interest of our study is in the ability of monitoring agents to understand the mathematical expressions. The monitoring agents will then be able to *parse* these mathematical expressions into their associated mathematical operations and subsequently perform those operations. For instance, given an expression $total = 3 + 2$, the monitoring agent should be able to interpret this expression to mean that *total* is the sum of 3 and 2 and then compute the answer correctly as 5. Specifically for this study, the monitoring agents should be able to interpret the expression for Generalized Response Time Metric and compute the answer correctly based on the supplied values. The "answer" of such computation informs the conclusion as to whether the SLA has been breached or not, and whether it is necessary to enact a conflict resolution protocol because the values arrived at by the participating monitoring agents differ.

The Generalized Response Time Metric is given by Equation (4.14) as :
$Q_{GRTM} = srt + (az).\dot{x}$

The above mathematical expression or formula for computing the generalized metric for response time can be represented semantically in the Turtle as:

```
[ a math: Plus ;
  rdf:_1 _:x1;
  rdf:_2
       [a math: Multiplication;
          [rdf:_1 _:x2 ;
           rdf:_2 _:x3 ];
       ]
 ] .
 _:x1 rdfs:label "srt"
 _:x2 rdfs:label "az"
```

```
_:x3 rdfs:label "$\dot{x}$"
```

Listing 4.1: Generalized Response Time metric in Turtle

It should be noted that in the context of this study, the labels are actually ontological concepts and so the Turtle syntax can be rewritten to incorporate these concepts with their associated meanings. Specifically *srt* refers to the server-side response, *az* refers to the historical average-z scores based on all historical request a server has handled, and finally $\dot{x}$ refers to the *"mean error"* which is the difference between client-side response-time and server-side response-time for each historical client-server interaction. These concepts are kept as labels here for brevity.
The graphical RDF syntax for the Generalized Response Time Metric is depicted in Figure 4.2:

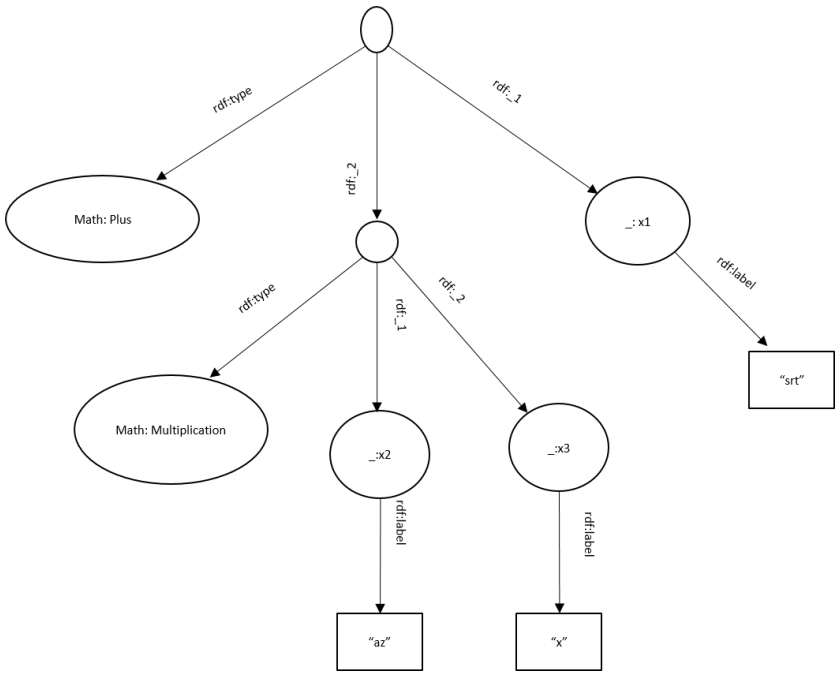


Figure 4.2: Generalized Response Time Metric in graphical RDF syntax

For the performance ontology to be usable in collaborative and corroborative monitoring, it must provide explicit descriptions of how values are computed. Specifically, the method or formula of measuring response time

(i.e. $srt + (az).\dot{x}$) should be semantically described so that the participating monitoring agents can understand how to derive the performance values (response time) conveyed in the monitoring information. Corroboration is facilitated by the ability of monitoring agents to derive performance values. In this sense, the service provider provides both the performance data as well as the metadata explaining how the values in the performance data were arrived at so that this might be corroborated by the receiving monitoring agent (i.e. service consumer's monitoring agent).

The mechanism by which the service provider's monitoring agents provide both the performance and metadata is the Monitoring Information eXchange protocol discussed in Section 4.5.

4.5 Monitoring Information eXchange (MIX) Protocol

In view of the related works discussed in Chapter 3, the motivation for this study is that current approaches to Semantic Web Service Monitoring do not facilitate collaborative and corroborative exchange of monitoring information between the service consumer and the service provider. Collaborative means that both the service consumer and service provider are aware of each other's monitoring activities and also that the interaction between them is based on sending to and receiving monitoring information from each other. This section discusses **M**onitoring **I**nformation e**X**change (MIX) protocol as a mechanism to facilitate the exchange of semantically enriched monitoring information between the service consumer and the service provider.

The high-level architecture of the MIX protocols is described next in Subsection 4.5.1 followed by the discussion on the functional principles of the protocol in Subsection 4.5.2.

4.5.1 High-Level Architecture of MIX Protocol

The MIX protocol is an open standard specification protocol. It specifies the minimal set of messages, the flow of these messages and what content they convey. Similarly, the high-level architecture of the MIX protocol describes the main layers of the protocol stack without being prescriptive on how each protocol layer should be implemented.

The four-layer architectural stack of the MIX protocol consists of the *Transport Layer*, *Session Initialization and Management Layer (SIML)*, the *MIX Message Encoding Layer (MMEL)*, and finally, the *MIX Application Layer*. The high-level architecture of the MIX protocol is depicted in Figure 4.3:

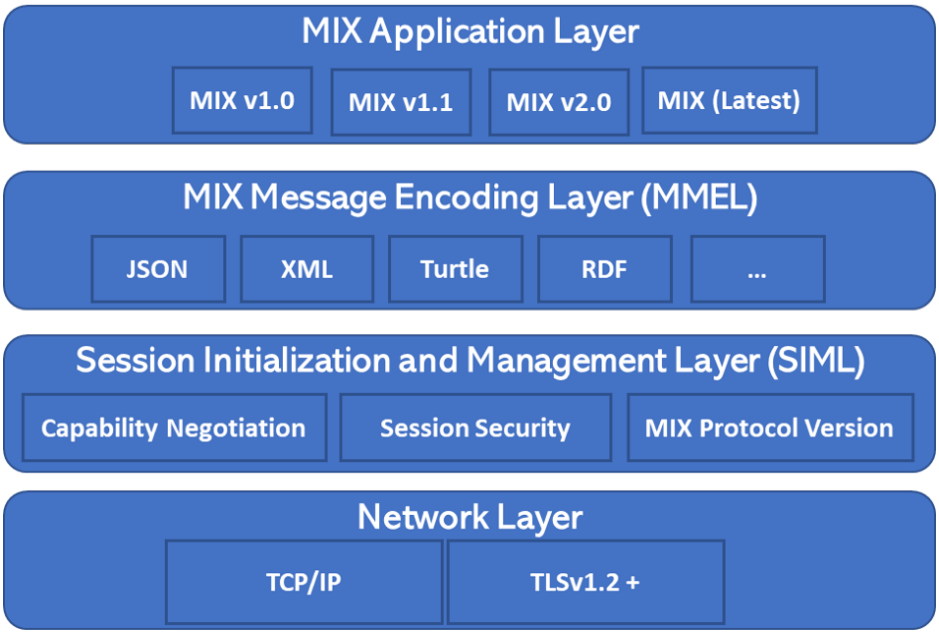


Figure 4.3: High-Level MIX Protocol Architecture

The *Transport Layer* is responsible for managing the network connectivity between the participating MIX agent systems. The network protocols at this level are not specific to the MIX protocol *per se*, but are universal IP-based protocols such as the TCP/IP. Furthermore, for secure communication between the MIX agent systems, Transport Layer Security (TLS) can be used. A MIX agent in the context of this study is a software program that runs the implementation of the MIX protocol.

The *Session Initialization and Management Layer (SIML)* is responsible for handling the establishment of the interaction sessions between the MIX agent system. At this level, there are MIX protocol specific functions such as specification for the MIX protocol version, the creation and termination of sessions and agent capability negotiations between the two participating

MIX agent systems. For example during session initialization the two MIX agents have to agree on the version of the protocol to be used.

The *MIX Message Encoding Layer (MMEL)* is responsible for packaging the protocol messages in the agreed format. Furthermore, the semantic enrichment of MIX protocol messages takes place at this level. Example of message encoding standards include JSON, XML, and Turtle among others. Even though multiple standards may be supported by a particular implementation of MIX protocol, a single format is agreed upon during the session initialization in the SIML layer.

The *Application Layer* of the MIX protocol is where the actual features and functionalities of the specific MIX agent implementations reside. Messages coming from the lower layers of the protocol stack are received and processed by the MIX agent. The collaborative and corroborative monitoring is achieved by the logic that is built into the application. The application reacts to different protocol events such as when a session is initiated (connecting with another MIX agent), when the monitoring data is received, when there is an alert for SLA breach, receiving ACK and NACK messages and when the participating MIX agent terminates the session. The protocol events represent the phases of the MIX protocol (see Figure 4.5).

Although the lower layers of the architecture may remain unchanged, different features can be added at the Application Layer and thereby provide a much richer set of capabilities that cater for more complex use cases of the MIX protocol.

The functional principles of the MIX protocol are discussed in sub-section 4.5.2 to capture the inner workings of the protocol.

4.5.2 Functional Principles of MIX Protocol

One of the key functional principles of the MIX protocol is that the performance description data is populated by both the service provider and service consumer. For each request, the arrival time of the request and the response dispatch time (time that the response is sent) are recorded at the service provider. The service consumer records the request dispatch time and the response arrival time. These recorded "times" use the ontological concept of *Time*, or more formally, *Duration* to represent the unit of measure. The recorded values are then used to calculate *response time* based on the equation: $srt + (az).\dot{x}$.

Another principle is that the descriptions of the Semantic Web Services

are modified to include the MIX endpoint as well as the version of the protocol supported by the service. The MIX endpoint [2] specifies a Universal Resource Locator(URL) to which monitoring data should be sent. The presence of MIX endpoint in the description of the service indicates that the service has capabilities for MIX.

Upon discovering a service and binding to it, the client also establishes a connection to the service's monitoring agent via the MIX endpoint (see Figure 4.4).

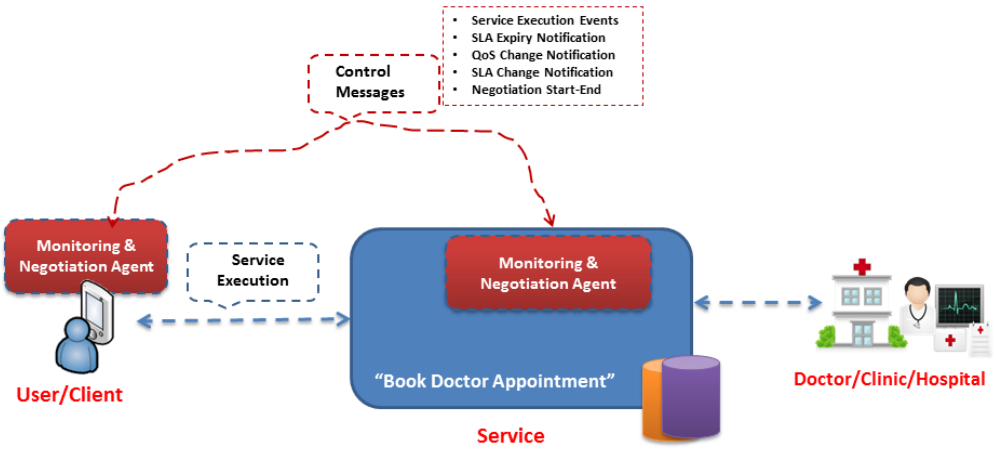


Figure 4.4: MIX-enabled Book Doctor Appointment Service Scenario

Once a MIX connection is established, the protocol gets initiated and control messages are being exchanged between the monitoring and negotiation agents (MIX agents) of both the service and the consumer or client. This happens over the "monitoring plane" which is represented by the dotted line in Figure 4.4. The diagram depicts the fact that the service execution happens on one plane while the monitoring and potential negotiations happen on a different plane which is where the MIX protocol resides.

MIX protocol is divided into four main phases, namely the *Initialization* phase, *Monitoring* phase, *Tear-Down* phase and finally the *Post-Execution* phase as depicted by Figure 4.5 below:

[2] mix://agent-ip-address:mix-port/

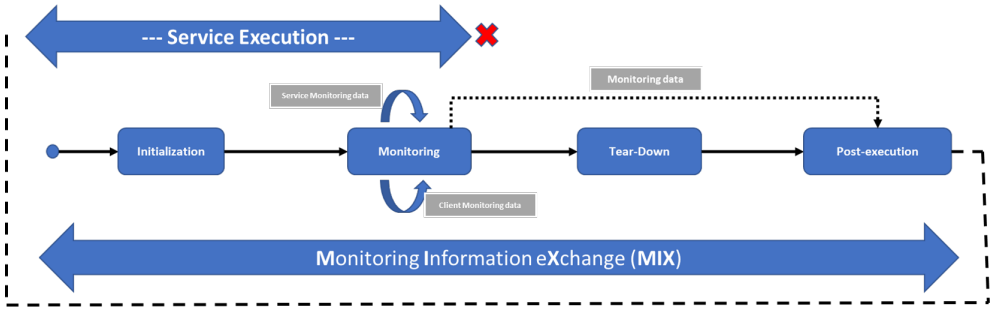


Figure 4.5: The MIX protocol phases

During the *Initialization* phase, the monitoring agents (MIX agents) of both the service and the consumer establish a session over which they will later exchange their monitoring information. It is during the initialization phase that the Service Level Agreement (SLA) negotiations take place. Once the session is established, monitoring begins and the service MIX agent and client MIX agents start exchanging monitoring information from their respective processes. This happens during the *Monitoring* phase. When the Semantic Web Service's consumer or client process receives a response from the Semantic Web Service, it notifies its MIX agent and the MIX agent terminates its MIX session with the service MIX agent. The *Tear-Down* phase is necessary to demarcate the point at which the collection of monitoring data is concluded, and the service response has been received by the client process. Finally, the *Post-Execution* is where the monitoring agents start evaluating their collected performance data and corroborating each other's findings and enacting conflict resolution processes if necessary.

During the initialization phase the client sends its baseline data through the *Initialize()* call - this is essentially the timestamp of its service request and includes the time the request was sent, the expected response-time (as per advertisement), and the interval (in seconds) at which the client's MIX agent will be sending/receiving monitoring data. The interval should always be less than the quoted response-time; monitoring data is only exchanged during service execution. The sequence of interactions during initialization phase is shown in Figure 4.6 below:

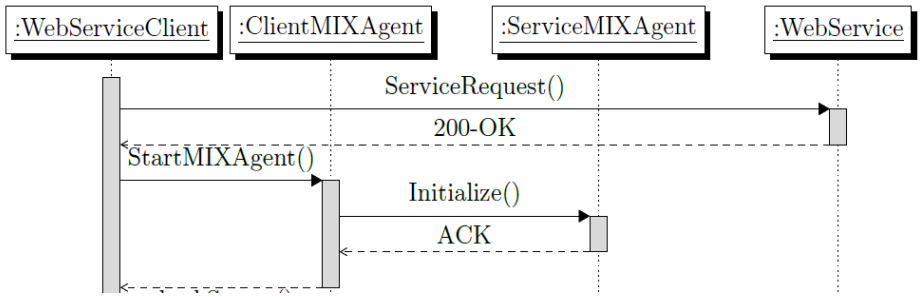


Figure 4.6: The MIX protocol initialization phase

An example of baseline data carried in the *Initialize()* call is shown in Listing 4.2 using the Java Simple Object Notation (JSON) format:

```
{
  requestTimeStamp: '1564584478745',
  requestId: '39d0b90c'
  advertisedResponseTime: '10',
  updateInterval: '2'
}
```

Listing 4.2: Sample baseline data in JSON format

In Listing 4.2, the client's MIX agent sends its baseline data indicating that it expects the advertised response time of 10 seconds and that it will be sending its monitoring data every 2 seconds. Messages such as this one shown in Listing 4.2 are carried in what is depicted as "control messages" over the monitoring plane in Figure 4.4.

The first corroboration occurs when the service provider's MIX agent acknowledges the response-time value quoted by the client's MIX agent by sending an *ACK* message. It is possible that the service might have improved or degraded in performance. In this case the service MIX agent will, after sending the ACK message to acknowledge receipt of the baseline data, send another protocol message (NACK) to indicate that it wishes to communicate contradictory information back to the client's MIX agent. In the NACK message the service provider's MIX agent indicates the client's quoted response-time and the service's *actual* response-time (what it commits to). It also supplies an indicator as to whether the actual service response time is better ("B") or worse ("W") than the client's MIX agent's expectations as reflected in its baseline data.

A sample NACK message is shown in Listing 4.3 below:

```
{
  requestId: '39d0b90c',
  responseId: '39d00000',
  response: 'NACK',
  data: { expectedResponseTime: '10',
          currentResponseTime: '12',
          indicator: 'W'
        },
  notifications: {
          contactPerson: 'Support Contact Centre',
          email: 'support@service-name.domain'
        }
}
```

Listing 4.3: NACK response data with 'worse' current response time

In Listing 4.3 above, the service provider's MIX agent indicates that it has contradictory information regarding the client MIX agent's baseline data (see *requestId*) in terms of its expected response time (*expectedResponseTime*). It indicates that its current response time (*currentResponseTime*) is 12 seconds which it classifies as being worse than the expected response time of 10 seconds quoted by the client MIX agent.

If the actual service response time is the same as the value quoted by the client, then the NACK message would not have been sent. The ACK and NACK messages also include contact information to be used when sending notifications of SLA breaches. The MIX protocol does not specify how the sending of notifications is done. Ideally, the MIX agent should be able to automatically send an email to the email address specified in the ACK or NACK messages.

A sample ACK message is shown in Listing 4.4 below:

```
{
  requestId: '39d0b90c',
  responseId: '39d00000',
  response: 'ACK',
  notifications: {
          contactPerson: 'Support Contact Centre',
          email: 'support@service-name.domain'
```

```
        }
    }
```

Listing 4.4: ACK response data

This initial exchange of messages (ACKs and NACKs) simply ensures that the client's expectations are aligned with what the current situation is at the service side in terms of the response time it is able to commit to. This also gives the client the opportunity to refresh its baseline data. The client can now evaluate the current execution based on the service response time specified in the NACK message.

During the monitoring phase and after receiving the first ACK or NACK message from the service provider's MIX agent, the client MIX agent iteratively gathers monitoring data from the Web Service Client's process and sends the updates to the MIX agent on the service side at the intervals specified in the baseline data. This real-time exchange of monitoring information happens through the *MIX-UPDATE* calls and is depicted in the sequence diagram in Figure 4.7:

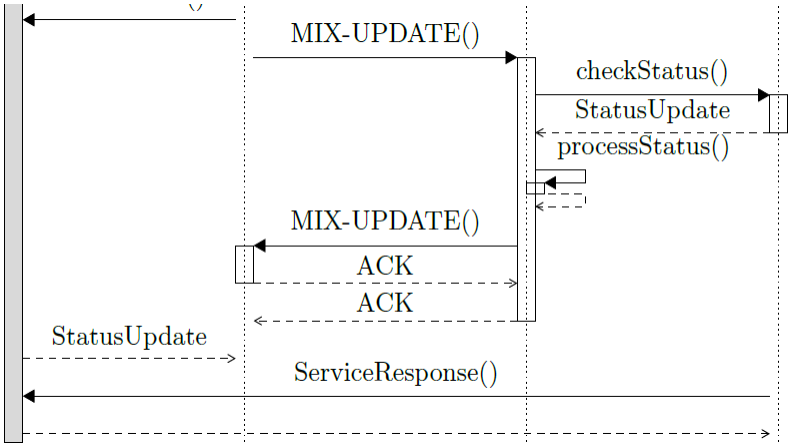


Figure 4.7: The MIX protocol monitoring phase

When the Web Service Client's process receives a response from the Semantic Web Service, it notifies the client's MIX agent via *StatusUpdate* call. The client's MIX agent then computes the response-time. The computation is based on the time that the response is received and the time that the request is sent. This computation uses the Generalized Response Time Metric (GRTM) - see Equation (4.14).

The result of the above computation should match what the service MIX agent had indicated in its ACK or NACK message. In case of any deviation or violations, the conflict resolution protocol is activated and notifications are sent to the email address specified in the ACK/NACK message from the service's MIX agent (see Listing 4.3). This is done during the final phase of the MIX protocol before the MIX session is terminated or torn down.

Figure 4.8 below depicts an interaction diagram for the MIX protocol to show the interactions between the MIX agents throughout the phases of the MIX protocol:

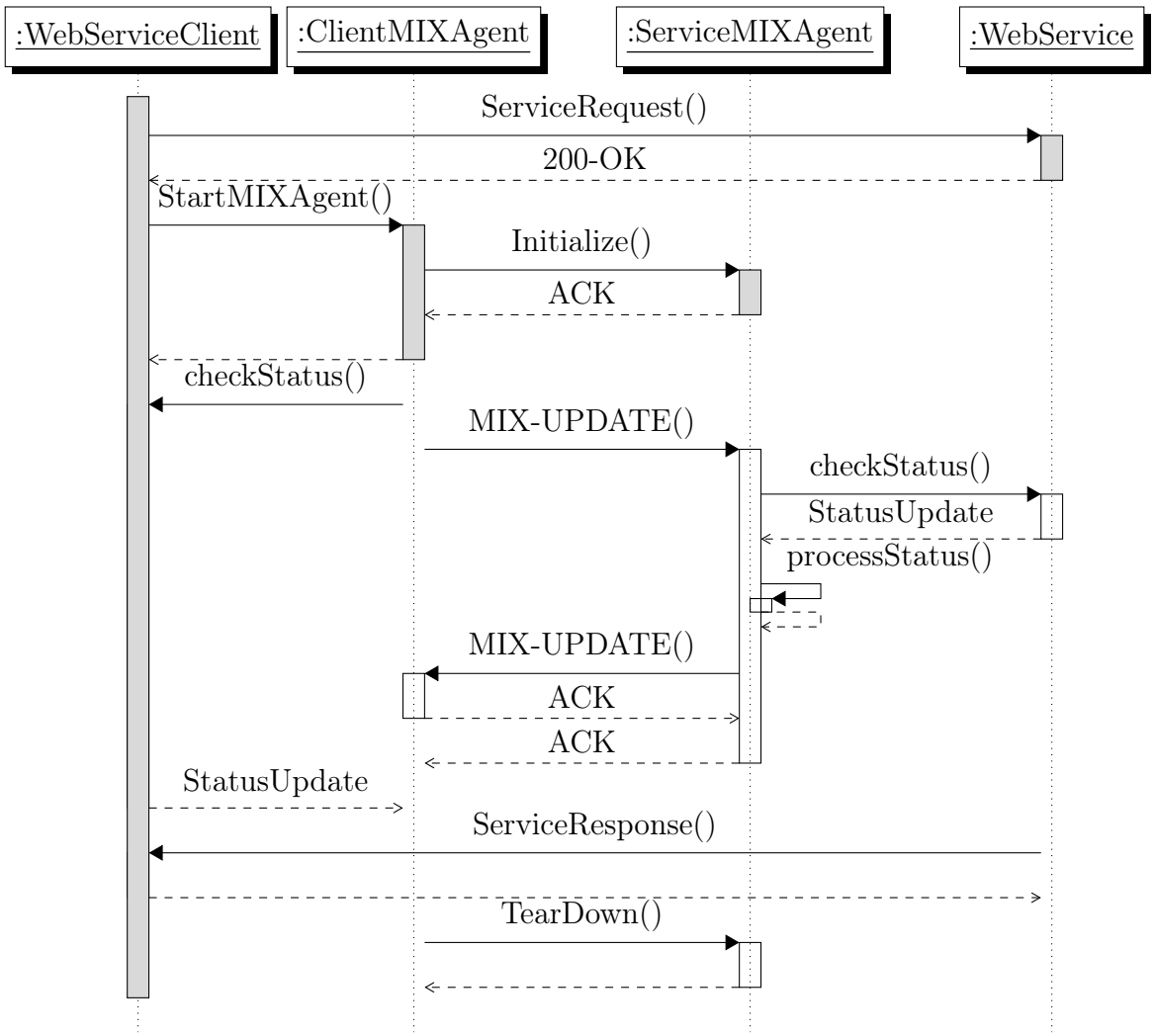


Figure 4.8: Monitoring Information eXchange Sequence Diagram.

The post-execution is the final phase of the four MIX protocol phases (see Figure 4.7). This phase takes place "outside" of the monitoring function and is meant to be mostly administrative. This is because it follows after the MIX session is already torn down (see *TearDown* call). During this time, the MIX client and MIX service agents can start performing post-execution tasks, including conflict resolution if necessary in the event of SLA breaching or discrepancies in the collected performance data. The MIX protocol does not specify the functionality within this phase. The developer can decide

how this is handled.

4.6 Summary

This chapter presented the development of a Generalized Response Time Metric (GRTM) which is a major conceptual contribution by itself and then demonstrated how this metric is formalized using semantic annotations. A case was made as to why it is necessary to describe the logical/mathematical expression of performance parameters using the appropriate syntaxes for semantic annotation (e.g. RDF and Turtle). The chapter also discussed the exchange of the semantically enriched monitoring data through the MIX protocol. The functional principles, and the sequence of interactions, as well as the nature and contents of the monitoring data being exchanged, were discussed.

The next chapter presents a reference implementation of the MIX protocol as part of a collaborative and corroborative monitoring of Semantic Web Services. This reference implementation will also serve to verify the technical feasibility of collaborative and corroborative monitoring as a technique.

Chapter 5

Proof of Concept Implementation of MIX Protocol

This chapter addresses the research objective **(RO-3)** which entails developing a Semantic Web Service Monitoring tool as a prototype to demonstrate the technical feasibility of collaborative and corroborative utility in Semantic Web Service monitoring. In terms of the chosen research methodology of Design Science Research (DSR), this chapter represents the instantiation of the design artifact. Hevner et al. (2004, p. 79) make a case for the reference implementations, arguing that these reference implementations serve to demonstrate the feasibility of the artifacts of design science.

The reference implementation as discussed in this chapter is achieved using the already-available Java technologies. It is worth noting that although the MIX protocol is technology-agnostic, Java was only chosen for convenience based on the researcher's proficiency in using the Java programming language.

5.1 Introduction

As part of our study, we propose the appropriate semantic annotations for Web Service performance data (which constitute monitoring information),and a technical infrastructure that enables the exchange of this semantically enriched performance data. The focus of this chapter is on the technical infrastructure and the reference implementation of the **M**onitoring **I**nformation e**X**change (MIX) protocol.

The technical infrastructure for collaborative and corroborative monitoring provides for the mechanism by which the service consumer and the service provider can perform their local monitoring tasks and share the data. This implies that as far as the service consumer is concerned, it is implementing client-side monitoring (for example *see* Jurca, Faltings, and Binder (2007)) and similarly, the service provider feels as though it is performing service-side monitoring (for example *see* Jurca, Faltings, and Binder (2007)). This is the essential aspect of collaborative and corroborative monitoring; all the monitoring is done locally by each party (consumer/provider), and then the collected monitoring data is shared (post-execution) or exchanged (real-time). Furthermore, the technical infrastructure enables the storage and exchange of the already-semantically-annotated monitoring information.

It is expected that the service consumer and the service provider may be implemented using totally different technology solutions, and therefore use different software tools to perform monitoring. In view of this, the actual data generated by these monitoring tools should be annotated with some sort of metadata to facilitate interoperability. This is the justification for the semantic annotation of Web Service monitoring data. This semantic annotation can be achieved by using performance ontology suggested in this study (for Semantic Web Service performance data). There are Quality of Service (QoS) related ontologies, as discussed in Chapter 3 and Chapter 4, which can be used to describe and annotate the performance data of a Semantic Web Service to enable collaborative and corroborative monitoring.

The remainder of this Chapter is structured as follows: Section 5.2 presents a scenario for the use case of collaborative and corroborative monitoring of Semantic Web Services, and later Section 5.3 presents a technically viable architecture for collaborative and corroborative monitoring of Semantic Web Services. Section 5.4 then discusses the reference implementation of the Monitoring Information eXchange (MIX) protocol. Finally, Section 5.5 describes the experimental execution of the MIX protocol implementation.

5.2 Proof of Concept Use Case for MIX

The doctor booking use case described earlier in this thesis is used in this chapter to illustrate how the MIX protocol can be implemented in real-life applications.

There are two levels of interactions that take place when the end user

of a service (consumer or client) invokes the services ("Book Doctor Appointment"): firstly there is the high-level or front-end interaction where the service client and service provider applications interact following the request-response interaction style, and secondly there is the low-level or back-end interaction between the MIX agents on both sides; the client MIX agent interacting with the service MIX agent using the MIX protocol.

The use case scenario depicted in Figure 5.1 shows the high-level interactions between the service client and the Book Doctor Appointment service. Under the hood, however, there are interactions between the client's MIX agent and the service's MIX agent and these happen in the monitoring plane which is depicted in the diagram as "Monitoring Information Exchange".

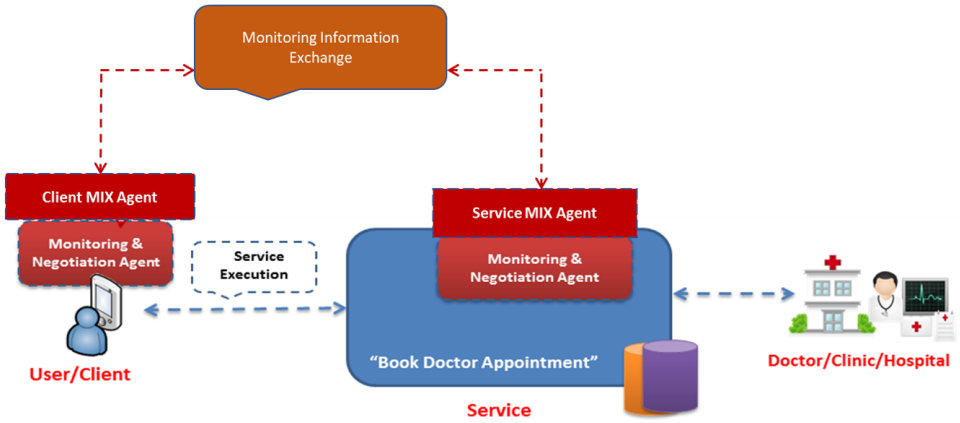


Figure 5.1: Book Doctor Appointment with MIX plane

The end user, looking for a service for booking an appointment with a doctor, will invoke the Book Doctor service. Upon discovering and binding with the "book doctor" service, the MIX client running on the end-user device will establish a connection with the service's MIX agent. To establish this connection, the client's MIX agent sends the "initialize" protocol message in which it indicates its last known response time of the service. Upon receiving the initialize message, the service MIX agent will respond by sending its baseline message which indicates its current or actual response time. The simulation of this exchange is presented later in Section 5.4.

The architecture in Figure 5.2 indicates which components are required to support collaborative and corroborative monitoring. It does not represent the architecture of the MIX protocol - the MIX agents that implement the MIX

protocol reside in the Semantic Web Service components (see *ServiceA* and *ServiceZ* instances). The exchanges of monitoring information between the MIX agents are facilitated by the technical components of this architecture.

5.3 Collaborative and Corroborative Monitoring Architecture

Figure 5.2 below shows the high-level view of the architecture as realized using the Java technologies; Apache JENA [1] , and Java-based agent platform (JADE [2]). The components are described hereafter.

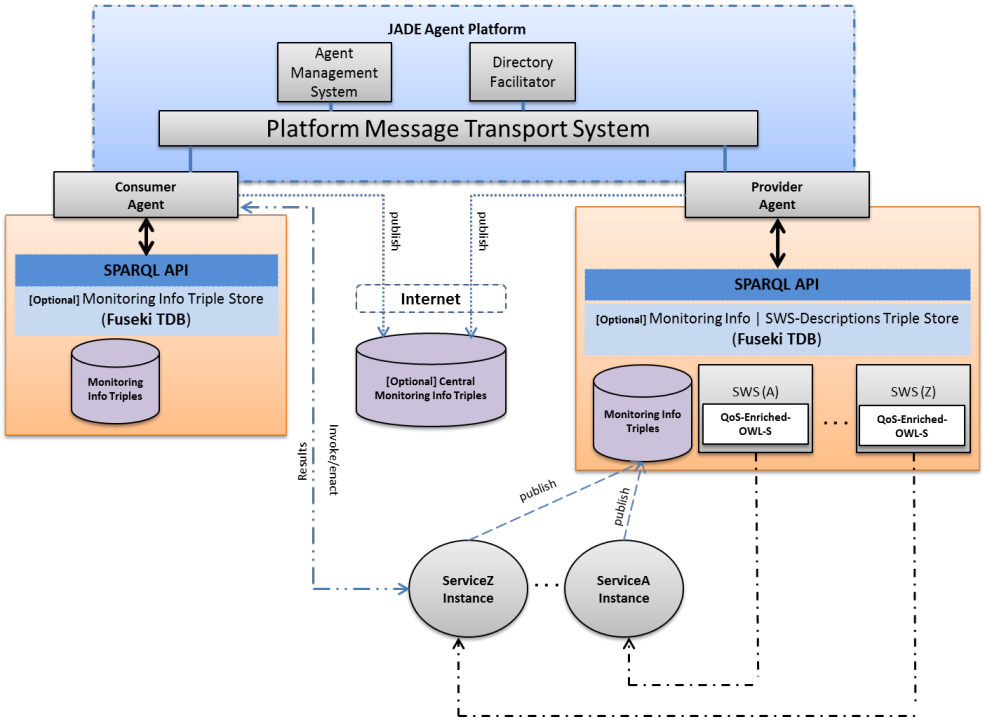


Figure 5.2: Collaborative and Corroborative Monitoring Enabled Architecture

It must be noted that the purpose of this chapter is not to assess how well

[1] https://jena.apache.org/
[2] http://jade.tilab.com/

the chosen technologies, such as the components in the above architectural diagram, are able to implement the MIX protocol. Instead, the objective is to illustrate that the MIX protocol can be implemented and that it is technically feasible. Readers and researcher are encouraged to explore other technologies for implementing the same MIX protocol. This chapter uses JADE technology which is freely available and provides an agent execution environment which closely mimics the behaviour required for MIX agents. The choice of JADE, however, is not a prescription, any other technology that provides similar capabilities as JADE can be used. Balaji B, N K, and R S (2018) chose JADE for their experiment because of its support of peer-to-peer communication capability.

The following subsections describe the key components of the experimental technical architecture for facilitating collaborative and corroborative monitoring of Semantic Web Services.

5.3.1 Java Agent DEvelopment (JADE)

In the experimental set-up, the Consumer Agent and Provider Agent are hosted on the **J**ava **A**gent **DE**velopment (JADE) framework and use the platform's message transport system to exchange messages. These messages are used during the discovery and session initiation phase. The Consumer Agent uses the Directory Facilitator of JADE to locate an interoperable Provider Agent. Once located, the Provider Agent sends a message to the discovering Consumer Agent – this message is a URL to a Web Service Description Language (WSDL) file. The Consumer Agent uses the URL it received and then automatically invokes the instance of a Web Service that is described by the WSDL file. The Consumer Agent then activates its client-side monitoring to observe how the remote peer (the service) is performing while handling its request. The Consumer Agent then annotates and stores this performance data. The facility for doing this annotation in the Java programming language is provided by the Apache JENA's Ontology API.

5.3.2 Apache JENA

Apache Jena is a free and open source Java framework for building Semantic Web and Linked Data applications. It provides the facilities for authoring ontology models (RDF, RDFS and OWL, SPARQL) as well as a rule-based inference engine for reasoning over these ontology models. Through these

reasoners it is possible to derive additional truths about the concepts in the ontology model. The code that uses the Apache Jena facilities, specifically the Jena Ontology API, is shown in Figure 5.8 and Figure 5.9.

5.3.3 FUSEKI - SPARQL Server

SPARQL [3] is the query language developed by the W3C RDF Data Access Working Group [4]. Apache Jena framework also includes a query engine that supports SPARQL RDF query language.

For storing semantically enriched performance data, the proposed technical architecture uses the Apache JENA Fuseki [5] which is a SPARQL server. Specifically, the monitoring information is stored using Resource Description Framework (RDF) [6] format which is a standard model for data interchange on the Web. JENA Fuseki server includes a native Triples Database (TDB) and this is where the monitoring information gets stored. The query engine provided by Apache Jena is then used to interrogate the performance data stored by the monitoring agents.

Although in the design we have included the Monitoring Info Triple store (see Fuseki TDB) as a dedicated component, it is not necessarily required. An agent (especially a consumer) can store its own performance information on a log file or even in memory – as long as the content is semantically enriched and is amenable to machine processing, it can still work. The advantage of using a storage facility like TDB (triple store) is that there are available tools that can be used to query the store during post-execution analysis. For instance, JENA has a dedicated API to support querying of the triple store using RDF Query Language called SPARQL [7].

5.4 Monitoring Information eXchange (MIX) Protocol

As shown in Figure 5.2, the Provider Agent is not involved after service execution – instead, the instance of the service being invoked (e.g. ServiceA)

[3] http://www.w3.org/TR/sparql11-query/
[4] http://www.w3.org/2001/sw/DataAccess/
[5] https://jena.apache.org/documentation/fuseki2/index.html
[6] https://www.w3.org/RDF/
[7] http://www.ibm.com/developerworks/xml/library/j-sparql/

generates its own service-side monitoring data and publishes/stores the data into the Monitoring Info Triple store. Figure 5.3 shows the MIX-protocol plane as part of the technical architecture.

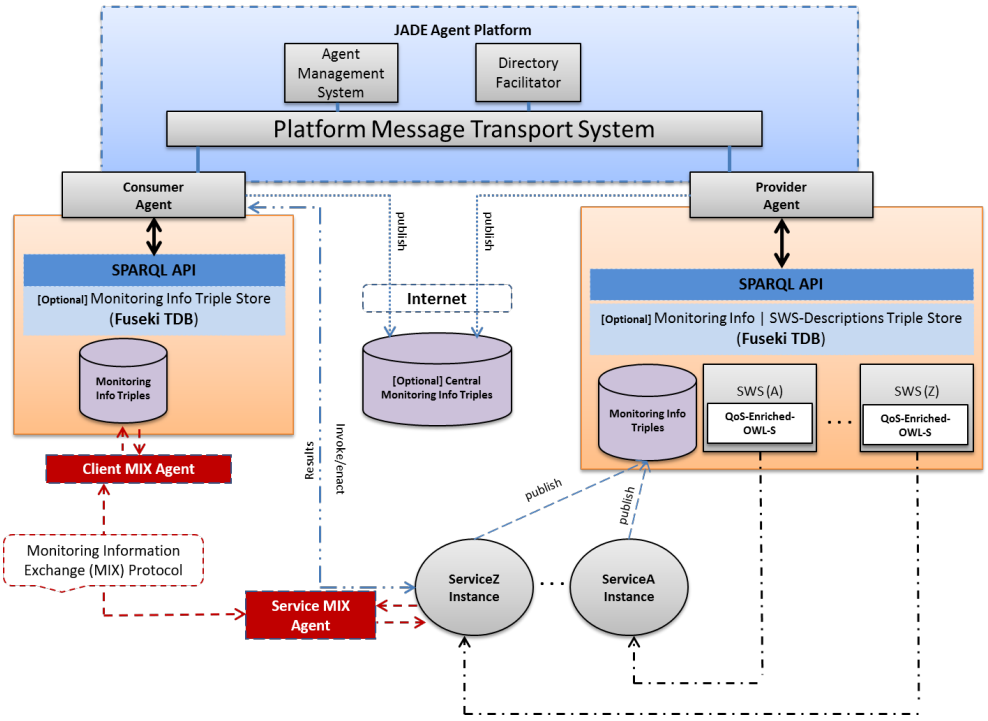


Figure 5.3: MIX Protocol in Collaborative and Corroborative Monitoring Enabled Architecture

The MIX agents interact with the Triple Store to save and retrieve monitoring information as part of the monitoring information exchange protocol. The diagram illustrates that the interaction between the MIX agents is separate from the client-service interaction. This is significant as it indicates that the MIX protocol is not intrusive to the pre-existing Semantic Web Service based applications.

In the collaborative and corroborative monitoring architecture, a Monitoring Info Triple store is included as a component on both the Consumer and Provider sides. Furthermore an optional, central, and presumably Internet-hosted, Monitoring Info Triple store is also included.

The Monitoring Info Triple stores are used to store historical service performance information and can be queried by any of the participating agents

(Consumer or Provider). Each request for a service is uniquely identified and the service execution information generated during the processing of this request is linked to the unique identifier which the consumer (who made the request) can query the service provider for its performance information identified by the same identifier as the request. Likewise, the service provider can, upon handling the request from a consumer, query the consumer-generated monitoring information pertaining to its handling of the request. This way, the two parties can collaborate and exchange their monitoring information. This is the essence of the **M**onitoring **I**nformation e**X**change (MIX) protocol.

Furthermore, by querying each other's monitoring information which is already semantically annotated, both parties can corroborate each other's monitoring information. Figure 5.4 below shows sample contents of Monitoring Info Triple store after some executions had taken place:

QUERY RESULTS

Raw Response | Table

Search: Show 50 entries

	monitoringDataUri	publisher	transactionId	responseTime	units
1	performance:servicex/performancedata/E2772076-D5BA-4B9D-A939-FC39190DA45B	<http://www.cc2masw.co.za/services/ExperimentalDoctorBookingService>	"E2772076-D5BA-4B9D-A939-FC39190DA45B"	312.0	"ms"
2	performance:servicex/performancedata/3F0D845C-DE1F-492D-AFE1-54C4A51D6818	<http://www.cc2masw.co.za/services/ExperimentalDoctorBookingService>	"3F0D845C-DE1F-492D-AFE1-54C4A51D6818"	312.0	"ms"
3	performance:servicex/performancedata/3A146E6D-7293-4674-812E-560E8ADA8FC5	<http://www.cc2masw.co.za/services/ExperimentalDoctorBookingService>	"3A146E6D-7293-4674-812E-560E8ADA8FC5"	200.0	"ms"
4	performance:servicex/performancedata/3F5FE224-75B6-4A42-A773-A39063A9FA49	<http://www.cc2masw.co.za/services/ExperimentalDoctorBookingService>	"3F5FE224-75B6-4A42-A773-A39063A9FA49"	200.0	"ms"
5	performance:servicex/performancedata/D3E34A06-D63E-4914-A66B-8B4A704625C8	<http://www.cc2masw.co.za/services/ExperimentalDoctorBookingService>	"D3E34A06-D63E-4914-A66B-8B4A704625C8"	200.0	"ms"

Figure 5.4: Sample Monitoring Data in Monitoring Info Triple Store

The **"monitoringDataUri"** is what uniquely identifies the service performance record and is used to query for performance information during corroboration to enable the service consumer and the service provider to perform the "comparing of notes".

The **"publisher"** is the URL to the actual Semantic Web Service. In this case, it points to the experimental "Book a Doctor Appointment" service.

The **"transactionId"** uniquely identifies the set of interaction actions between the service and the client. This represents a service request identifier on the service side and is used to tie the specific performance data to an interaction between service and client.

The **"responseTime"** represents the response time as reported by the service for all the requests/transactions it has executed. In the collaborative and corroborative scenario, we expect the client to have similar information for all the requests it would have made.

The **"Units"** shows the unit of measure for the reported response time (responseTime). This means, looking at the first row, the responseTime is 312 milliseconds (ms).

The next section discusses the experimental execution that demonstrates further technical details of how the most basic collaborative and corroborative monitoring mechanism can be accomplished using JADE, JENA and Java application monitoring library called JAMon [8].

5.5 Experimental Execution of the MIX Protocol

In order to illustrate the experimental execution of the MIX protocol, it is necessary to connect this experiment to the theoretical foundations of the MIX protocol. Specifically, the mapping between the phases of the MIX protocol and the actual protocol messages and the events that get emitted as part of the interactions between participating MIX agents is necessary.

Figure 5.7 below depicts the four phases of the MIX protocol and overlays the interactions between the client MIX agent and the service MIX agent by indicating the series of events emitted and processed by the respective MIX agents (*see* dotted lines).

[8] http://jamonapi.sourceforge.net/

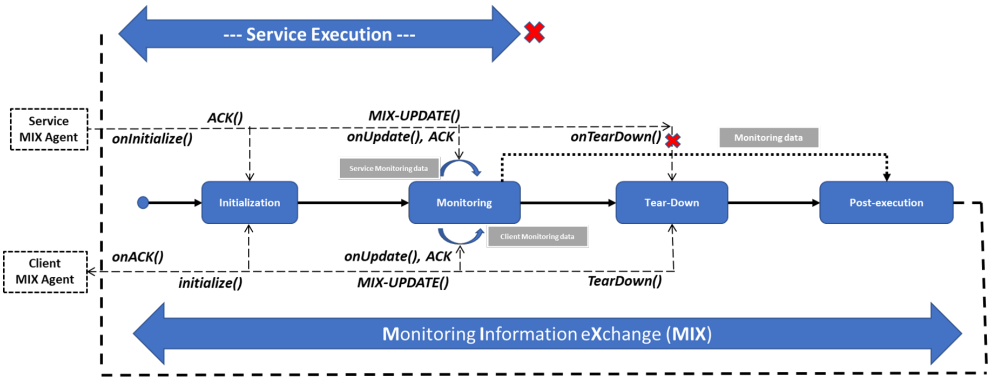


Figure 5.5: MIX Agent Interactions During MIX Protocol Phases

The illustration in Figure 5.7 depicts the relationship between the phases of the MIX protocol and the protocol messages that get sent or received by the participating MIX agents. This also shows the MIX protocol's state-machine; some events are only emitted at specific phases in the protocol life-cycle. For instance a *MIX-UPDATE* event is only permissible during the *Monitoring* phase of the protocol.

The order in which the events are emitted within each of the MIX protocol phases is best represented in a sequence or activity diagram. Figure 5.6 below depicts the sequence diagram of a typical MIX protocol interaction:

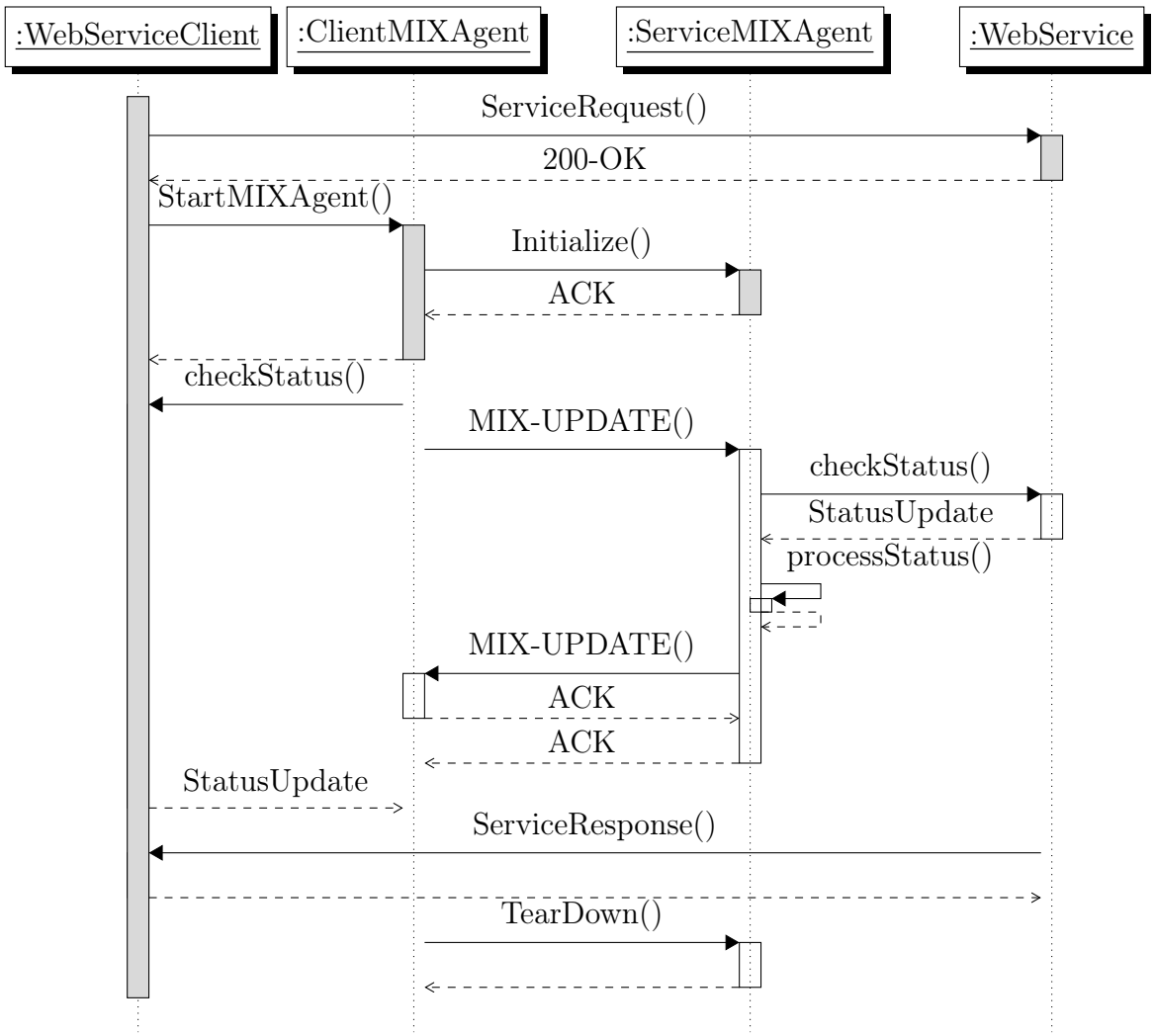


Figure 5.6: MIX Protocol Sequence Diagram.

In addition to showing the order of events during the interactions between the client MIX agent and the service MIX agent, the sequence diagram further shows the interactions between the MIX agents and the underlying Semantic Web Service process. For instance, during the monitoring phase, if the Semantic Web Service's client process receives results, it notifies the client MIX agent and then invokes the tear-down call to terminate the MIX session.

The first activity in Service-Oriented Computing is service discovery; Fig-

ure 5.7 shows agent communication which simulates the initial hand-shake that constitutes service discovery.

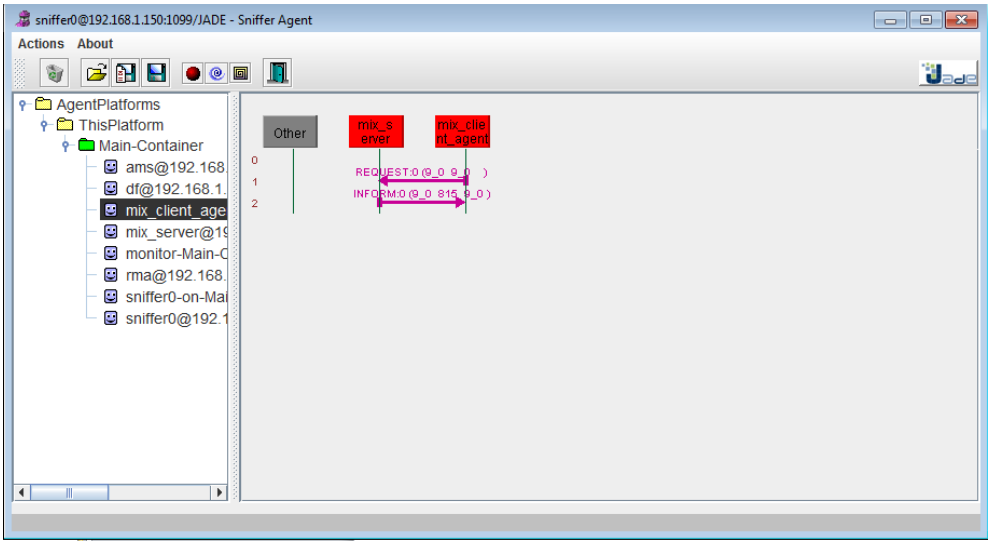
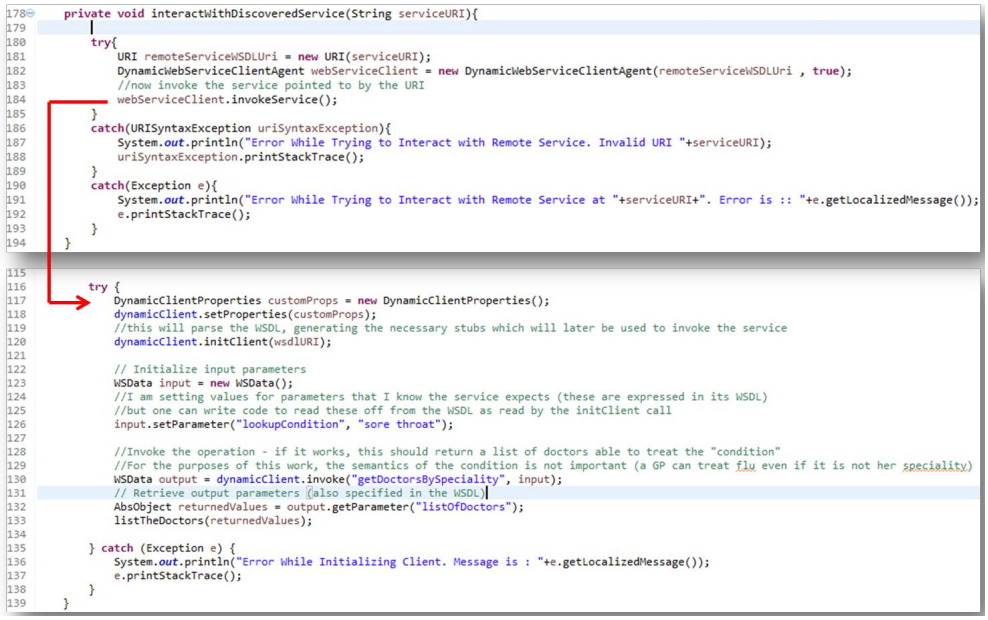


Figure 5.7: Agent Communication during Service Discovery

In Figure 5.7 above, "mix_client_agent" is the Service Consumer Agent and first sends the “Find Service” Agent Action REQUEST. The Service Provider Agent is represented by "mix_server" and responds to client request with an INFORM message. Upon receiving “Find Service” request, "mix_server" responds with a URI to the WSDL file of a service to handle customer request. The sample payload of the INFORM message sent by Service Provider Agent ("mix_server") is below:

http://localhost:8080/cc2masw/experimentaldoctorbookingservice?wsdl

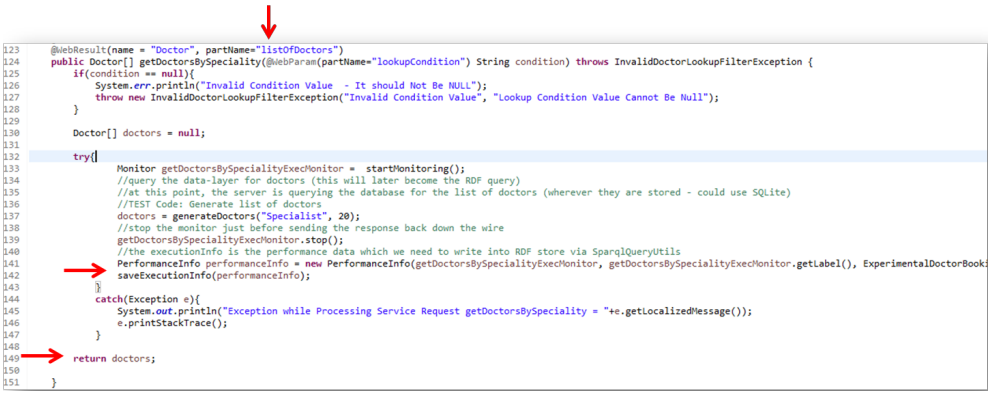
The receipt of the INFORM message by the Consumer Agent concludes the agent-communication portion which represents “service discovery”. The Consumer Agent dynamically invokes the service pointed to by the URI received from the Provider Agent during “service discovery”. See Figure 5.8 for the code listing:



```
178    private void interactWithDiscoveredService(String serviceURI){
179        
180        try{
181            URI remoteServiceWSDLUri = new URI(serviceURI);
182            DynamicWebServiceClientAgent webServiceClient = new DynamicWebServiceClientAgent(remoteServiceWSDLUri , true);
183            //now invoke the service pointed to by the URI
184            webServiceClient.invokeService();
185        }
186        catch(URISyntaxException uriSyntaxException){
187            System.out.println("Error While Trying to Interact with Remote Service. Invalid URI "+serviceURI);
188            uriSyntaxException.printStackTrace();
189        }
190        catch(Exception e){
191            System.out.println("Error While Trying to Interact with Remote Service at "+serviceURI+". Error is :: "+e.getLocalizedMessage());
192            e.printStackTrace();
193        }
194    }
115
116        try {
117            DynamicClientProperties customProps = new DynamicClientProperties();
118            dynamicClient.setProperties(customProps);
119            //this will parse the WSDL, generating the necessary stubs which will later be used to invoke the service
120            dynamicClient.initClient(wsdlURI);
121
122            // Initialize input parameters
123            WSData input = new WSData();
124            //I am setting values for parameters that I know the service expects (these are expressed in its WSDL)
125            //but one can write code to read these off from the WSDL as read by the initClient call
126            input.setParameter("lookupCondition", "sore throat");
127
128            //Invoke the operation - if it works, this should return a list of doctors able to treat the "condition"
129            //For the purposes of this work, the semantics of the condition is not important (a GP can treat flu even if it is not her speciality)
130            WSData output = dynamicClient.invoke("getDoctorsBySpeciality", input);
131            // Retrieve output parameters (also specified in the WSDL)
132            AbsObject returnedValues = output.getParameter("listOfDoctors");
133            listTheDoctors(returnedValues);
134
135        } catch (Exception e) {
136            System.out.println("Error While Initializing Client. Message is : "+e.getLocalizedMessage());
137            e.printStackTrace();
138        }
139    }
```

Figure 5.8: Dynamic Invocation of the Discovered Service

For our experiment, all that the discovered remote service does is to return a list of doctors, and then it saves/publishes its performance/execution monitoring data into its own Monitoring Info Triple store. The code-listing in Figure 5.9 shows the lines of code responsible for this:



```
123    @WebResult(name = "Doctor", partName="listOfDoctors")
124    public Doctor[] getDoctorsBySpeciality(@WebParam(partName="lookupCondition") String condition) throws InvalidDoctorLookupFilterException {
125        if(condition == null){
126            System.err.println("Invalid Condition Value  - It should Not Be NULL");
127            throw new InvalidDoctorLookupFilterException("Invalid Condition Value", "Lookup Condition Value Cannot Be Null");
128        }
129
130        Doctor[] doctors = null;
131
132        try{
133                Monitor getDoctorsBySpecialityExecMonitor =  startMonitoring();
134                //query the data-layer for doctors (this will later become the RDF query)
135                //at this point, the server is querying the database for the list of doctors (wherever they are stored - could use SQLite)
136                //TEST Code: Generate list of doctors
137                doctors = generateDoctors("Specialist", 20);
138                //stop the monitor just before sending the response back down the wire
139                getDoctorsBySpecialityExecMonitor.stop();
140                //the executionInfo is the performance data which we need to write into RDF store via SparqlQueryUtils
141                PerformanceInfo performanceInfo = new PerformanceInfo(getDoctorsBySpecialityExecMonitor, getDoctorsBySpecialityExecMonitor.getLabel(), ExperimentalDoctorBook
142                saveExecutionInfo(performanceInfo);
143            }
144            catch(Exception e){
145                System.out.println("Exception while Processing Service Request getDoctorsBySpeciality = "+e.getLocalizedMessage());
146                e.printStackTrace();
147            }
148
149        return doctors;
150
151    }
```

Figure 5.9: Service-side execution and monitoring

The above code-listing illustrates how the monitoring information gets

collected by the service MIX agent. However this monitoring information needs to be exchanged with the client MIX agent and this is done through the MIX protocol. The communication between this service MIX agent and the client MIX agent is done by sending and receiving specific MIX protocol messages. For the purpose of our experimentation, this exchange was simulated using the JADE platform where we implemented the logic for receiving and processing the protocol messages.

The next series of screenshots describe how a MIX agent is instantiated in the JADE platform, how a MIX protocol message is authored using the JSON format in terms of the Message Encoding Layer and how such a message is received by a participating MIX agent.

Figure 5.10 shows the simulation of the start of an agent on the JADE platform:

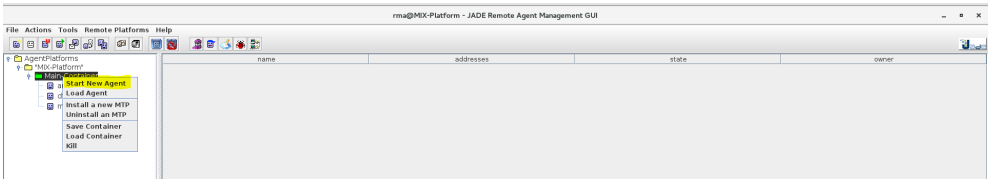


Figure 5.10: Start JADE MIX Agent

The JADE platform prompts you to indicate the parameters of the agent you intend to start; the parameters include the name of the agent and the class that implements the agent-specific logic. Figure 5.11 shows the agent-creation dialog screen:

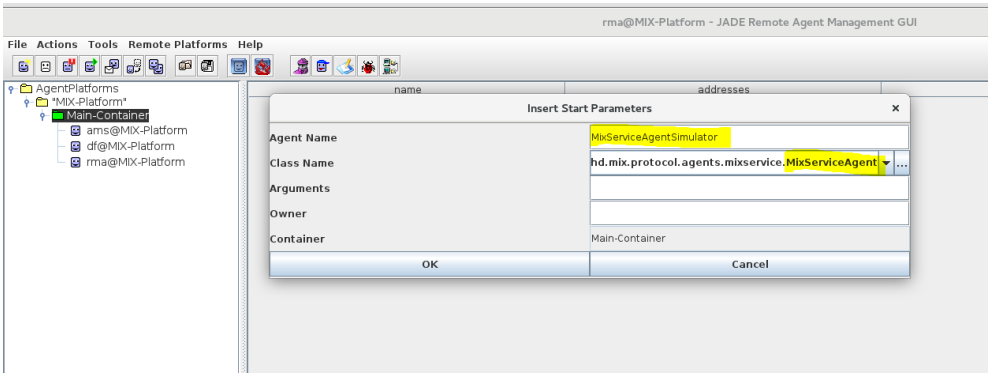


Figure 5.11: Set JADE MIX Agent Parameters

On the JADE platform, once you create an Agent, you need to select the implementation of the Agent from a list of hosted JADE agent. In the case of this simulation, we select the MIX agent class from the list (see Figure 5.12); this is the implementation of the MIX agent.

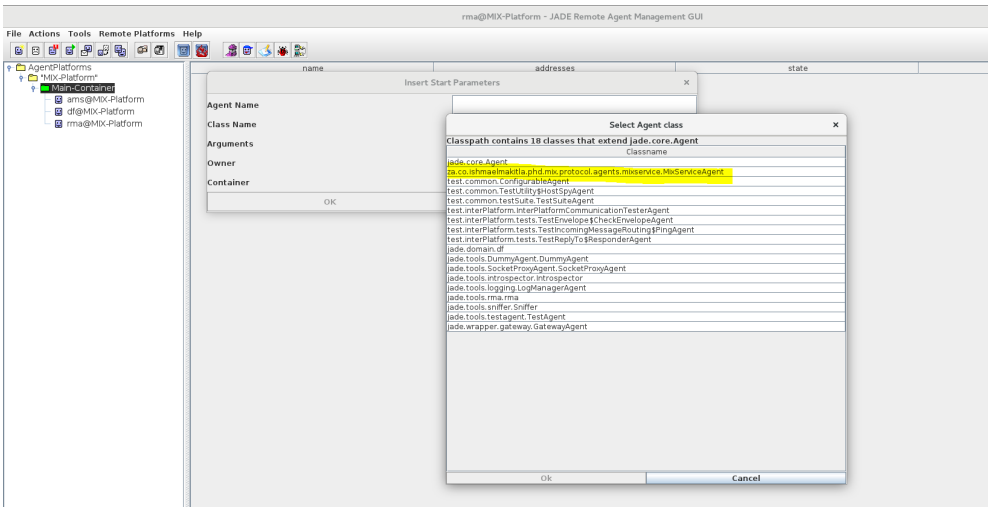


Figure 5.12: Select JADE MIX Agent Implementation

Once the parameters of the intended agent are set, JADE starts the agent in the selected container. Figure 5.13 shows a running JADE agent for MIX simulator:

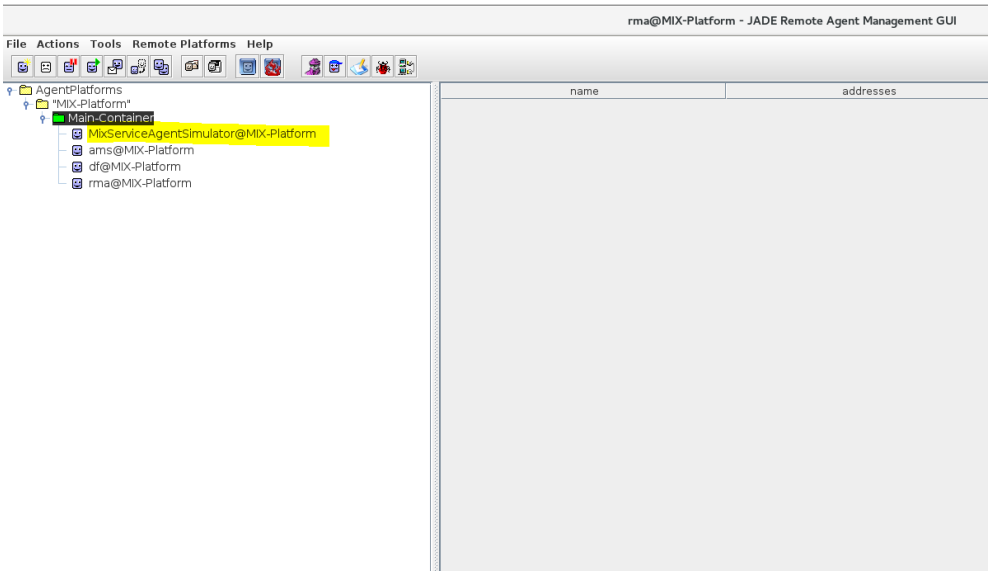


Figure 5.13: JADE MIX Agent Running

With the MIX agent now running, we are able to author protocol messages and send these to the running agent. On the JADE user interface, one can use the "Send Message" feature of the platform. Figure 5.14 illustrates how a protocol message can be authored - notice that the message content is in JSON format:

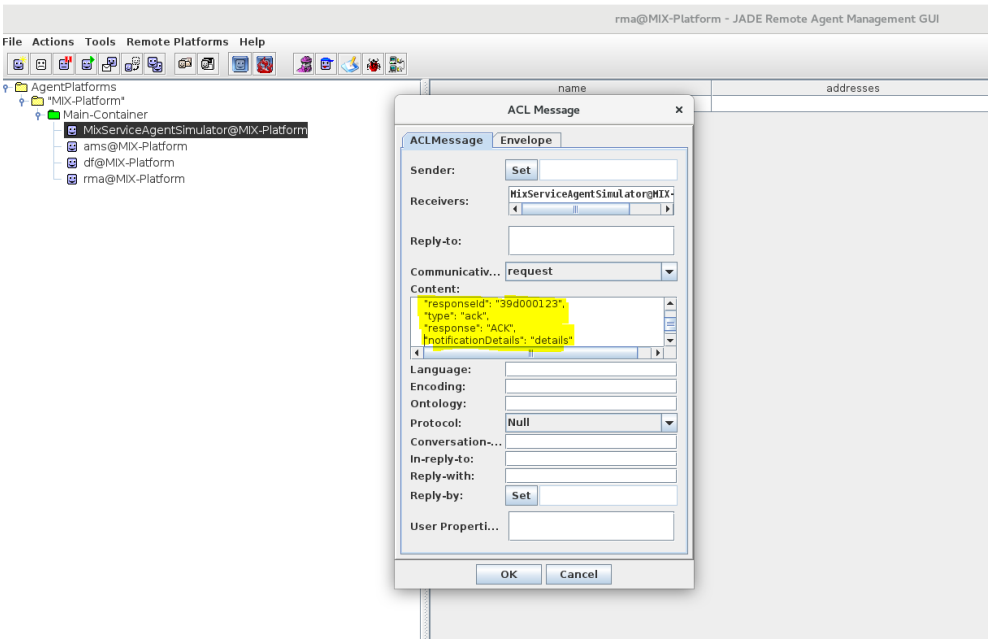


Figure 5.14: Send ACK MIX Protocol Message in JSON

In the case of this simulation, JSON would have been the selected message format in the MIX Message Encoding Layer (MMEL) in terms of the MIX architecture presented in Chapter 4 (see Figure 4.3).

When sending a message from the JADE user interface as shown in Figure 5.14, the message gets delivered onto the running JADE platform and the registered agent receives the message. To illustrate how this works, we implemented the message handler that receives the message and then prints out the contents of the received message. See Figure 5.15 :

Figure 5.15: Receive MIX Protocol Message in JSON

Upon receiving a protocol message such as above, the participating MIX agent would respond according to the MIX protocol state machine. For example, in the above scenario, the MIX agent receiving the ACK message would proceed with its processes such as sending a correlated ACK message, or interpreting the ACK message to mean that it should confirm its previous state proposal. For instance, if the MIX agent had sent a MIX-UPDATE message, it would have received an ACK message to indicate its MIX-Update message was received and processed successfully.

5.6 Summary

This chapter focused on the technical infrastructure and the reference implementation of the Monitoring Information eXchange (MIX) protocol. The chapter has drawn from the conceptual design presented in Chapter 4 and developed a technology-specific demonstrator using the Java programming language as well as the Apache Jena framework which is specifically built for developing Semantic Web and Linked Data applications. JADE was used to illustrate the MIX protocol interaction between participating MIX agents. As can be expected for a chapter such as this, there are many references to technology tools; these are the tools that were chosen solely for the purpose of demonstrating the technical feasibility of the proposed solution to the research problem. It was not the intent of this chapter to prescribe any technologies. There are many alternative technologies that are functionally equivalent and those can be explored by other researchers. Finally, this chapter presented results from an experimental prototype of a Semantic Web Service monitoring tool developed as a proof-of-concept implementation of the Monitoring Information eXchange (MIX) protocol.

The next chapter presents the findings of the study and discusses the results from the evaluation of the collaborative and corroborative monitoring technique for Semantic Web Services as proposed by this study.

Chapter 6

Evaluation and Results

This chapter reports on the technical feasibility demonstrations presented in the previous chapter (Chapter 5). For the proposed solution of a collaborative and corroborative monitoring of Semantic Web Services, the demonstrations focused on three main aspects; firstly is the basic collection of the service performance data by the monitoring library or tool, secondly is the storage of the collected monitoring data, and finally, is the exchanging of the monitoring data by the participating monitoring agents. This chapter addresses the fourth research objective **(RO-4)** which involves evaluating technical feasibility demonstrator or prototype. In terms of the chosen research methodology of Design Science Research (DSR), this chapter represents the Design Evaluation research activity.

Hevner et al. (2004, p. 78) state, with regards to the evaluation of Design Science artifacts, that since these artifacts are built to address hitherto unsolved problems, they are to be evaluated on the basis of the utility that they provide in solving those problems. In view of this, the reference implementation of MIX, as presented in Chapter 5, should likewise be evaluated on the basis of its technical feasibility and its utility in supporting collaborative and corroborative monitoring.

This chapter discusses the results of the evaluation of the collaborative and corroborative monitoring technique for Semantic Web Services as proposed by this study. Finally the chapter discusses the methodological aspects of the study as a way of appraising the technical validity and academic rigor of the work.

6.1 Introduction

This study has proposed a collaborative and corroborative monitoring approach for Semantic Web Services. Underpinning this approach is the ability of the participating monitoring agents or systems to exchange their monitoring information. This exchanging of monitoring information necessitated addressing the three critical requirements, and these were introduced in Chapter 1 (Section 1.3) and discussed in detail in Chapter 4. The three requirements are:

- Developing a mechanism by which participating monitoring systems can develop *consensus* on observable QoS parameters and attributes;
- Mechanism by which participating monitoring systems can *exchange* their monitoring information;
- Technology-neutral *negotiation protocol for conflict resolution* in case of SLA breaches or inconsistencies in the monitoring data;

The following subsections discuss each of the three requirements above, explaining how each requirement was met and what the implications, if any, are for both theory and practice.

6.1.1 Consensus on Observable QoS Parameters

The need for consensus emanates from the fact that, especially for measurable performance-related QoS parameters, the service provider and service consumer may have different perspectives; for example, response time versus execution time and which value should be reported for QoS monitoring purposes. This requirement pertains to the need to reconcile the differences in perspectives relating to measurable QoS parameters which are measured independently by both the service consumers and the service providers.

The performance ontology provides the basis for semantic description of performance characteristics of a Semantic Web Service. In view of this, the use of ontological models to describe QoS parameters provides the most appropriate facility for developing the required consensus on observable QoS parameters.

For this study, the consensus on observable QoS parameters has been achieved by using ontology, specifically the performance ontology. The performance ontology comprises four QoS parameters; the *latency*, *response*

time, *throughput*, and *error-rate*. A comprehensive ontology for performance was developed and integrated as a sub-ontology in the larger QoS ontology. Existing QoS ontologies were extended using the sub-classing mechanism in order to integrate the new concept of Generalized Response Time Metric (GRTM). As shown in Section 4.3, the GRTM was developed through a data-driven collaborative and corroborative technique so that it represents the best view of the response time for both the consumer (Semantic Web Service client) and the producer (Semantic Web Service provider). The GRTM is the outcome of a consensus-seeking process to address the issue whereby the service consumer and service provider have different perspectives on the same response time QoS parameter. Furthermore, to make the GRTM amenable to semantic reasoning and unlock the power of existing reasoners and inference engines, a semantic description of the mathematical expression that represents this generalized form of response time metric was developed. In this way, both service consumers and service providers will be able to reason over the performance data. The corroborative aspect of the proposed monitoring technique, as will be seen in Section 6.1.2, is accomplished by re-computing the result of the generalized form of response time metric (the GRTM) and ascertaining that both the service consumer and the service provider arrive at the same value within some error threshold. The GRTM, which was developed through the consensus-seeking process in Chapter 4, is given by:

$$Q_{GRTM} = srt + (az).\dot{x}$$

The graphical RDF syntax for GRTM is represented as:

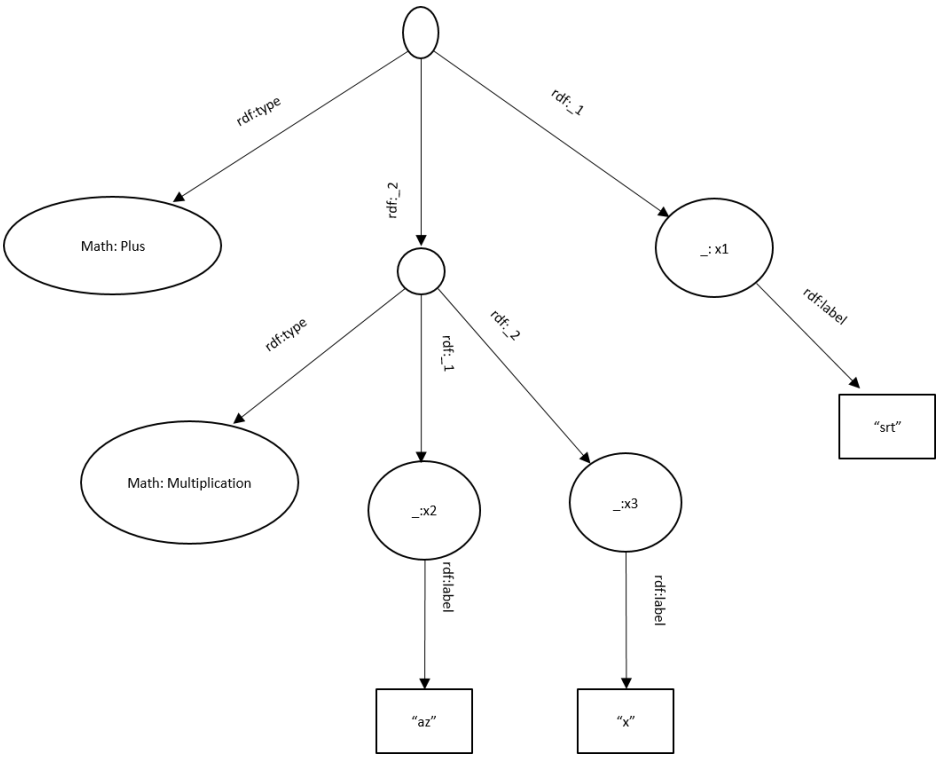


Figure 6.1: Graphical RDF Syntax for GRTM

The GRTM represents the conceptual contribution made by this study and, as of this writing, is the first such concept in the domain of Service Oriented Computing.

6.1.2 Mechanism for Monitoring Information Exchange

This requirement pertains to the mechanism that would enable the monitoring systems on the Semantic Web Service client-side and the Semantic Web Service server-side to share their monitoring data.

Section 4.5 introduced and discussed the **M**onitoring **I**nformation e**X**change (MIX) protocol as a mechanism to facilitate the exchange of semantically enriched monitoring information between the service consumer and the service provider. One of the fundamental principles of this protocol is that the performance data is collected independently by both the monitoring agent of the service consumer and the monitoring agent of the service provider. Each

monitoring agent then semantically enriches the performance data and sends it to another. For the purposes of this study, the service provider's monitoring agent would provide both the performance data as well as the metadata explaining how the values in the performance data were arrived at. The metadata is essentially the semantic descriptions of the metrics of each of the measurable QoS parameters. Most importantly, the metadata is the semantic description of the Generalized Response Time Metric (GRTM). The performance values can then be re-computed, based on the supplied metadata, and be corroborated by the receiving monitoring agent (i.e. service consumer's monitoring agent) to determine whether it too will arrive at the same value, for the same metric, as what the service provider had reported.

The implication of this corroborative feature is what necessitated the use of metadata in that the structure of the monitoring data being exchanged needed to be semantically annotated to enable the participating agents to process and reason over the data. The mechanism for the exchange of monitoring data, on the other hand, needed to define the exchange protocol in terms of the sequence and content of data being exchanged.

MIX protocol is divided into four main phases, namely the *Initialization* phase, *Monitoring* phase, *Tear-Down* phase and finally the *Post-Execution* phase:

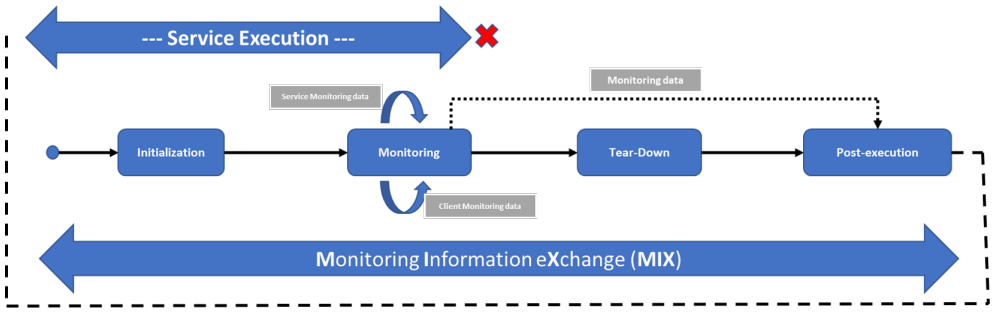


Figure 6.2: Four Main Phases of the MIX Protocol

The MIX protocol is made to be very lightweight and extensible. As a consequence, it specifies a limited set of interactions so that there is sufficient scope for customization by engineers who may want to introduce a number of interactions within one or more phases of the MIX protocol. Most importantly, the protocol is technology-agnostic, and does not prescribe any specific technology for implementing any of its components, including the

message formats and encoding schemes. The protocol only specifies the order of interactions and what the payload of each interaction ought to be.

The nominal sequence of interactions according to the MIX protocol is depicted in Figure 6.3 below:

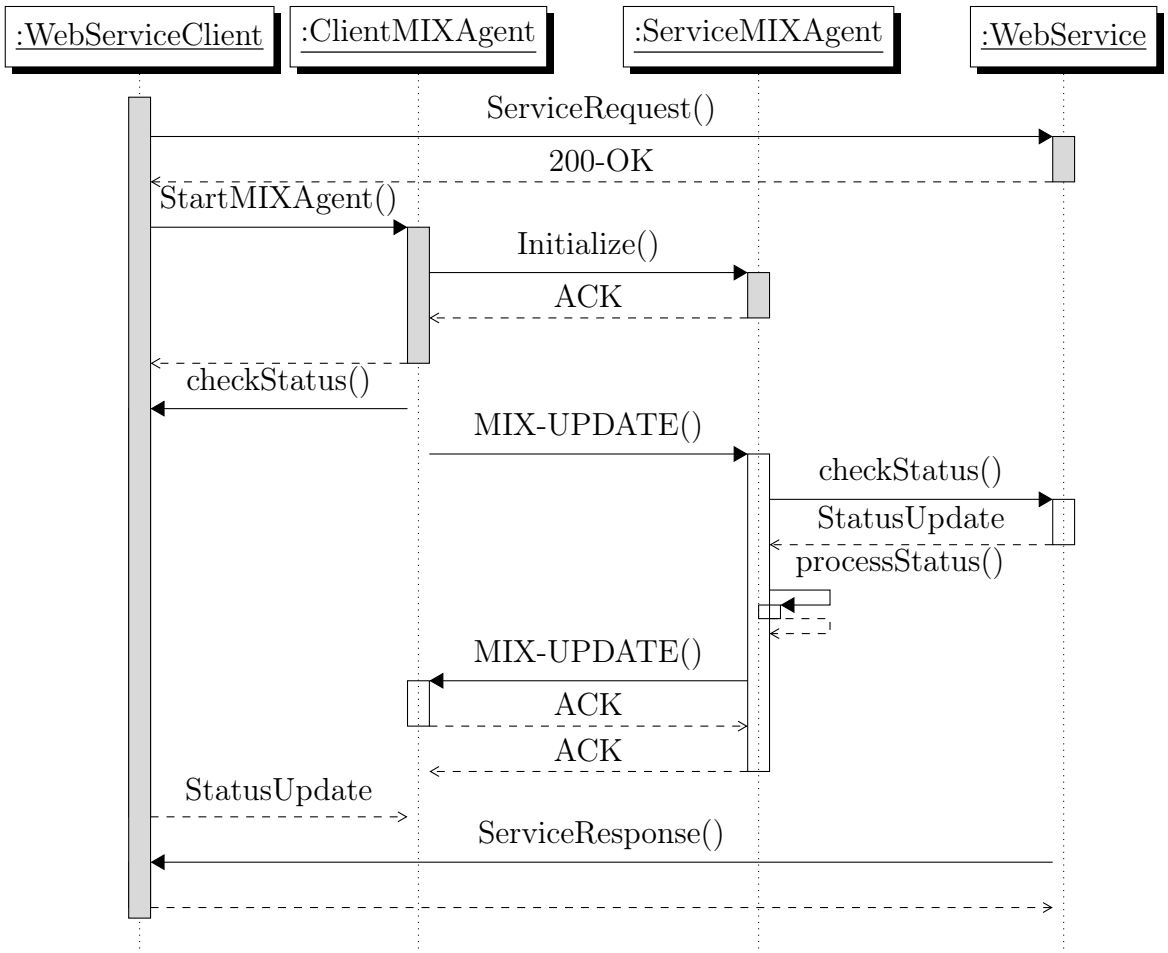


Figure 6.3: Nominal Interactions Sequence for MIX Protocol.

The MIX protocol is a significant conceptual contribution made by this study and, as of this writing, is the first such protocol in the general domain of Web Service monitoring, and the more specific Semantic Web Service monitoring.

6.1.3 Technology-neutral Negotiation Protocol

This requirement pertains to the capability that enables the Semantic Web Service client's and the Semantic Web Service server's monitoring systems to participate in a negotiated conflict resolution in the event of inconsistencies between their respective performance characterization or execution monitoring data.

The initial exchange of the baseline data (see Figure 6.4) represents negotiation. It can result in the MIX agents deciding to abort the entire process, or continuing based on the agreed-upon performance expectations and commitments. The sequence of interactions during the negotiation of the baseline data is shown in Figure 6.4 below:

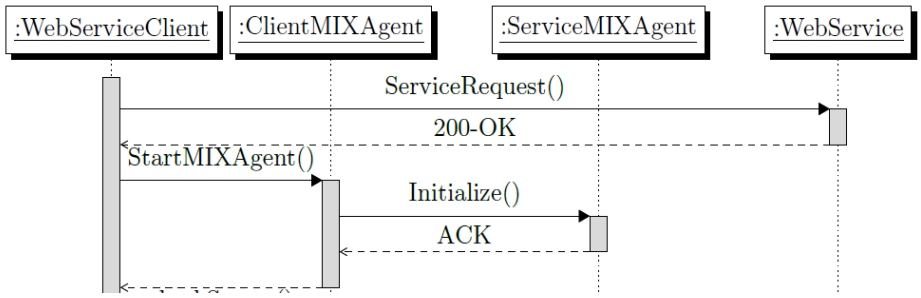


Figure 6.4: MIX Protocol Baseline Data Negotiation

This study developed a very light-weight negotiation protocol with greater scope for expansion to handle more complex use cases. The current design of the protocol is deliberately abstract. For instance, decision-making by the MIX agents is not specified or dictated. This means, for example, that upon receiving the baseline data feedback NACK message, the client MIX agent can be implemented such that it negotiates pricing if the new baseline data is worse than expected (service quality degraded). The pricing can be based on percentages - for instance the client MIX agent could use the percentage degradation in quality as a percentage price reduction. For example a Response Time that changed from 10ms to 13ms (i.e. 30% degradation) for a service that charges R2.00 will now cost R1.40. Likewise the service MIX agent can also re-negotiate pricing due to improvements in its baseline data (for instance, Response Time improving from 10ms to 7ms). These behaviors are not dictated by the protocol design, hence the protocol is open

to more specialized implementations as long as both the participating MIX agents are compatible.

As a design principle for MIX protocol, the post-execution processing is taking place "offline" - otherwise it is felt that it introduces a lot of overhead for a real-time monitoring process. These "contractual" negotiations are best handled "offline" - the tear-down phase of the MIX protocol is meant to demarcate this "offline" part of the process where post-execution negotiations can take place, even between people instead of MIX agents.

6.2 Evaluation

In terms of evaluation, the chapter makes a case for the method of evaluation adopted for this study, namely technical feasibility demonstration. The difference between comparative analysis and technical feasibility is discussed next, and a case is made for why technical feasibility was adopted for this study.

6.2.1 Comparative Analysis vs. Technical Feasibility

In the case where the proposed solution is an improvement on existing solutions, perhaps because it implements superior techniques and methods, it becomes necessary for the researcher to demonstrate that his or her proposed solution is indeed superior to any other existing solution. This is done through comparative analysis whereby the researcher compares his or her solution to existing solutions on a number of metrics. For example, comparison could be done based on accuracy, speed, scalability, performance and reliability among others.

However in other cases where there are no other solutions similar to the one the researcher is proposing, then the researcher is obligated to demonstrate that his or her proposed solution is actually workable; that it can be implemented with technologies or tool that are readily available. This is done through proofing the technical feasibility of the solution by doing a reference implementation, or through a proof of concept. Failure to do so, the researcher would not have given his or her solution any credibility.

Since there is no other implementation of collaborative or corroborative monitoring of Semantic Web Services to compare with, it is futile to report on comparative analysis. Instead, Table 6.1 contrasts how the related works

dealt with issues relating to the use of ontologies in monitoring and whether the approaches supported collaborative and corroborative monitoring.

Table 6.1 reproduces the tabulation of related works presented initially in Chapter 3 (see Table 3.1) and compares these related works against the current study by indicating, for each work, the manner in which ontologies are being used in monitoring, the monitoring approach used, and whether the monitoring approach is collaborative and corroborative.

Research Work	Ontology Usage	Monitoring Approach	Collaborative	Corroborative
Keeney et al. (2011)	Semantic annotations to enrich monitoring information	Service-side monitoring	No.	No.
Letia and Marginean (2011)	Ontology-enriched request-response SOAP message pairs	Client-side monitoring	Yes.	No.
Vaculín (2009)	Semantic annotations to descriptions of event types and event instances emitted during services execution	Service-side monitoring	Yes.	No.
Singh and Liu (2016)	Semantic annotations to enrich monitoring data analysis results	Off-site monitoring	No.	No.
Zhang, Jin, et al. (2018)	Relies on underlying ontological models of QoS and introduces probabilistic characteristics of QoS	Service-side monitoring	No.	No.
Amir (2018)	No reference to QoS but deals with generic collaboration framework in Web of Things (WoT)	N/A	Yes.	No.
Al-Hazmi and Magedanz (2015)	Ontology-based information model for monitoring federated infrastructures in IoT	Off-site monitoring (IoT platform service)	No.	No.

Benvenuti et al. (2017)	Ontology used to formally describe key performance indicators (KPIs) and their formulae	Off-site monitoring (PTS platform service)	No.	No.
Sigwele et al. (2018)	No reference to QoS but provides potential applications of semantic annotation of healthcare information to facilitate collaboration across heterogeneous healthcare systems	N/A	Yes.	No.
Lara Calache and De Farias (2020)	Semantic annotations to Web Service description using a graphical collaborative semantic annotation support tool	N/A	Yes.	No.
MIX	Semantic annotations to the descriptions of QoS parameters and their metrics (e.g. Response Time) and semantic annotations to enrich the service's performance data	Both Client-side and Service-side monitoring	Yes.	Yes.

Table 6.1: Comparison Against Related Works

Table 6.1 shows that the most closely related works have either not supported collaborative and corroborative monitoring or did so partially. It also indicates that only the current study (i.e. MIX) provides for a fully collaborative and corroborative monitoring mechanism as well as the semantic annotations of QoS parameter descriptions to enrich the Semantic Web Service performance data.

Comparative analysis against existing implementations of collaborative or corroborative monitoring of Semantic Web Services could not be conducted because, as stated earlier and as at the time of this writing, there were no existing implementations to compare with. This study sought to demonstrate the technical feasibility of the proposed collaborative and corroborative monitoring technique for Semantic Web Service. This was achieved through a reference implementation of the MIX protocol using only the technologies that are readily available and in active use by the Internet engineering communities. A Java-based monitoring library (JAMon) was used to demonstrate how to technically collect the monitoring data. A Semantic enrichment library, called Apache JENA, was used to annotate the monitoring data to enable reasoning over the same data as part of collaborative and corroborative monitoring. Finally, the exchange of the monitoring data was enabled by the use of yet another Java-based agent communication platform called JADE. The JADE platform provides the technical capability to implement MIX agent functionality and provide the communication infrastructure that allows the exchange of messages between agents.

In addition to the three key requirements imposed by the need to facilitate monitoring information exchange (Sections 6.1.1,6.1.2 and 6.1.3), a more utilitarian requirement is that of technical feasibility and is necessary for the validity of this study as discussed in Section 6.2.1 above. The next subsection explains how the technical feasibility requirement was satisfied.

6.2.2 Technical Feasibility of Collaborative and Corroborative Monitoring

With regards to the technical feasibility of collaborative and corroborative monitoring, the following have been achieved:

- It was demonstrated that with the least intrusion to the current Semantic Web Services frameworks, the Semantic Web Service descriptions or

their QoS properties can be extended to include connectivity directives for MIX protocol by way of the MIX endpoint URL;

- The ability to collect the performance or monitoring data was demonstrated using JAMon library which recorded how long a service took to process a request (for booking a doctor's appointment);
- The ability to semantically enrich the monitoring data was demonstrated with the use of Apache JENA which provides the ontology authoring capability;
- The exchanging of monitoring information between MIX agents was demonstrated using the JADE platform which allows communication between agents by sending and receiving protocol messages; these were in the JSON format in the demonstration;
- The design decision of keeping the MIX protocol technology-agnostic meant that the implementation of the MIX protocol is open to whichever technology the prospective SWS provider has access to, and the expertise within their organization. The proof of concept (see Chapter 5) demonstrated that MIX can be implemented using open source technology (or Java in this case).

6.3 Results and Their Implications for SWS Monitoring

This section discusses the results and their implications for the future of Semantic Web Service monitoring.

A generalized metric for response time, named the Generalized Response Time Metric (GRTM), represents a consensus on the meaning of the response time QoS parameter as seen from both the Semantic Web Service provider and the Semantic Web Service consumer. This generalized metric was found to be given by the following equation: $Q_{grtm} = srt + (az).\dot{x}$.

It was found, and demonstrated, that it is *technically feasible* to support and implement collaborative and corroborative monitoring for Semantic Web Services. The collaborative and corroborative monitoring is made possible by implementing a monitoring information exchange mechanism based on the technology-agnostic **M**onitoring **I**nformation e**X**change (MIX) protocol.

The following are the implications of this technical feasibility as demonstrated in this study:

- The conceptualization of the Generalized Response Time Metric (GRTM) implies that a "response time", as is currently used in the non-functional characteristic of Semantic Web Services, is not sufficiently accurate. A more accurate response time should be very dynamic in nature. It further implies that in most descriptions of response time, there is confusion between *execution time* (which is how long the service takes before rendering results to the consumer) and the *response arrival time* (which is the time it takes before the consumer receives the results). A more accurate response time should account for the time difference between execution time and response arrival time; this is what the consumer considers to be the real response time of a service).

- The functionality of sharing service performance data, which was demonstrated as technically doable, can be made a regulatory requirement for a Semantic Web Service based interactions, entitling the service consumer to such information as it pertains to their specific interactions. This is very important as functionally equivalent services put a premium on their non-functional properties such as performance.

- The ability to exchange monitoring information means that there is no need for incentive schemes aimed at encouraging honest reporting of service experience especially from consumers.

- Due to the functionality provided by the MIX protocol, monitoring and exchanging of the monitoring data become part of the service provision and consumption infrastructure. Therefore, there is no need for dedicated infrastructure for monitoring by surveillance because service consumer and service provider exchange their environment-dependent monitoring information directly.

6.4 Methodological Aspects

This section reflects on the methodological aspects of the study and makes a case for the academic rigor and the appropriateness and soundness of the research approach adopted by this study. Section 1.5 gave a full discussion of the research methodology.

The stated purpose of this study is to address the evident lack of a mechanism to support or facilitate collaborative and corroborative monitoring of Semantic Web Services by prescribing a Monitoring Information eXchange (MIX) protocol. The prescriptive nature of this study precludes a similarly prescriptive research approach. Consequently this study adopts a design science research paradigm which, as Gregor and Hevner (2010) argued, is particularly suited to research projects that entail the development, design and building of artifacts as outputs of the research while adhering to sound theoretical and scientific principles.

March and Smith (1995) noted in categorizing research approaches that there are basically two main categories, namely natural science and design science. The fundamental difference between these two categories is in the objective of each research approach. Whereas natural science, or behavioral science research, attempts to understand the natural environment (reality), the design science research is concerned with the building and evaluating of Information Technology artifacts that serve real-world human purposes (March and Storey, 2008, p. 725). The differences between these research approaches also have a bearing on how outputs or contributions are to be evaluated or appraised. Natural science related research is evaluated mainly in terms of the explanatory power of the theories the research develops. Design science, on the other hand, is evaluated in terms of the utility of the design artifacts that the researcher creates. This also means that such design artifacts have to be technically viable to be implemented in real life (March and Storey, 2008, p. 726).

Design science, for the purpose of this study, is understood as a scientifically rigorous engineering-inclined research approach which involves the use of existing scientific and engineering knowledge base as a foundation for creating useful design artifacts that serve real human needs. In view of this, and epistemologically, the study also adopted pragmatism as its philosophical orientation which is compatible with design science research (Iivari, 2007). This philosophical stance advances an argument that knowledge and a better understanding of design problems and the solutions to these problems can be established through building, applying and evaluating purposeful artifacts (Hevner et al., 2004). The development of the MIX protocol, and the theoretical foundations it is based on are valid contributions in terms of practical utility and extending the body of knowledge. Collaborative and corroborative monitoring for Semantic Web Service is a new concept and represents a conceptual departure from the traditional approaches to Semantic Web

Service monitoring which were discussed in the literature.

6.5 Summary

This chapter addressed the fourth research objective **(RO-4)** involving the evaluation of the technical feasibility demonstrator or prototype. This chapter reported on the technical feasibility demonstration of the basic collection of the service performance data by the monitoring library or tool, the storage of the semantically enriched monitoring data, and finally on the exchanging of the monitoring data by the participating monitoring agents. This chapter represented the thesis statement, the objectives that emerged from the thesis statement and subsequently discussed how these objectives were met. The chapter also discussed the evaluation of the collaborative and corroborative monitoring technique for Semantic Web Services as a solution proposed by this study. The chapter finally discussed the methodological aspects of the study as a way of appraising the technical validity and academic rigor of the work. Specifically, the methodological aspects of the study made the case that the development of the MIX protocol and the theoretical foundations it is based on, such as the generalized response time metric, are valid contributions in terms of practical utility and extending the body of knowledge.

The next and final chapter of this thesis presents the conclusions of the study.

Chapter 7

Summary, Conclusions and Recommendations

Based on the evaluation and research results discussed in the previous chapter (Chapter 6), this chapter presents the conclusions that the study deduces from the work accomplished by the entire research project. Specifically, this chapter links the thesis statement and the conclusions that can be drawn based on the findings of this study. For this purpose, the chapter recaps the thesis statement, the objectives that emerged out of the thesis statement and then discusses how these objectives were met. This chapter also presents a summary of findings of the study.

7.1 Introduction

This study made a case for collaborative and corroborative monitoring of Semantic Web Services. This need for monitoring was discussed as emanating from the desire by Semantic Web Service providers to be able to detect and correct any deviations from the contracted QoS parameters. Monitoring also allows the service providers to differentiate themselves based on non-functional properties such as response time.

The study also contended that the problem with current approaches to monitoring Semantic Web Services is their inability to facilitate the collaborative and corroborative exchange of monitoring information between the service provider and the service consumer.

The study made a hypothetical proposition that collaborative and corrob-

orative monitoring of Semantic Web Services is a viable alternative approach to contemporary monitoring approaches such as client-side, service-side and 3rd-party/dedicated monitoring. The thesis statement was further delineated by clarifying what is meant by *viable alternative* such as (1) transparency of the monitoring process, (2) potential savings on infrastructural costs, (3) support for flexible pricing model, and (4) eliminating the need to incentivise honest reporting by service consumers.

The conclusions of the study are presented in Section 7.2 on the basis of the above thesis statement and what the study has been able to achieve in terms of the evidence to support the hypothetical proposition.

7.2 Conclusions

Since the assertions in the thesis statement pertain to viability, this section discusses the four (4) aspects of viability as used to delineate the thesis statement. Each subsection, in the following sequence of subsections, is dedicated to each aspect of viability as a way of drawing conclusion on each.

7.2.1 Transparency of the monitoring process

Transparency is achieved through the MIX protocol in two important ways. Firstly by supporting the exchange of monitoring information between the MIX agents of the service consumer and the service provider. Secondly, by the semantic enrichment of the monitoring information being exchanged which solves the potential conceptual confusion which arises when the same QoS parameter is understood differently by the service consumer and the service provider. The semantically annotated monitoring information, along with the metadata, enables the participating MIX agents to see how the performance values were arrived at; the metadata specifies the semantic description of the QoS metric for measurable QoS parameters. In the case of this study, this is the Generalized Response Time Metric (GRTM).

7.2.2 Eliminating the need to incentivise honest reporting by service consumers

A point was made in Chapter 1, Section 1.1, that since both the service consumer and service provider exchange and corroborate each other's monitoring

information, there is no need for incentive schemes such as those proposed by Artaiam and Senivongse (2008), which are aimed at encouraging honest reporting.

The design feature of the MIX protocol, supporting the direct exchange of monitoring information between the monitoring agents on the client-side (the service consumer) and on the server side (the service provider) eliminates the need to incentivize the service consumer to report their experience of service quality honestly. The protocol requires that the client MIX agent should send its monitoring information to the service MIX agent, which can then corroborate this information with the data from its own monitoring processes. There is also a post-execution process within the protocol (see phases of the MIX protocol in Figure 6.4), which empowers both the Semantic Web Service client and the Semantic Web Service server monitoring systems to participate in a negotiated conflict resolution in the event of inconsistencies in their respective performance monitoring data.

Hofstee (2006, .p 157) stated that academic work is judged on the reliability of its conclusions and not on what the work concludes. This being the case because the hypothetical propositions represented by the thesis statement are presented without evidence and that the research is conducted expressly to find evidence to either support the assertions or dispute them. The following two assertions (Subsections 7.2.3 and 7.2.4) were not directly supported by the evidence and are presented here as potential avenues for further research.

7.2.3 Potential savings on infrastructural costs

This aspect of the viability of the collaborative and corroborative monitoring approach suggests that there is no need for dedicated infrastructure for monitoring by surveillance because service consumer and service provider exchange their environment-dependent monitoring information directly.

There is no direct evidence to support this assertion - also, practical limitations necessitate the need for a triple-store for the staging of semantically enriched monitoring information which introduces some costs. Although there might be potential savings, scaling these might offset those gains. Furthermore, since the study could not provide a full implementation of the collaborative and corroborative monitoring (MIX) solution to an existing service provider and then compare the costs, there is no evidence to support this assertion and should be considered only as a "potential" until such time that a study is conducted with the express intention of doing a

comparative evaluation of implementation costs between this solution and existing solutions.

7.2.4 Support for flexible pricing model

Again, no direct evidence from the technical feasibility standpoint. The ability for service providers to support flexible pricing, based on the actual service quality experienced by the consumer, and which the service provider could corroborate during the MIX interactions, is a natural extension of this work; *see* Section 6.1.3 on possible extensions in terms of price negotiation capabilities. However, since this avenue was not explored in the reference implementation, it is unreasonable to support this claim directly from the available evidence. Therefore, the conclusion is that it is impossible, on the strength of evidence, to state categorically that collaborative and corroborative monitoring supports a flexible pricing model for Semantic Web Services. Owing to the open nature of the MIX protocol the implementation of this price-negotiation feature can be taken up as part of future research works.

7.3 Summary of Contributions

In order to achieve the objectives of the solution proposed by this study (see Section 1.3), as well as addressing the key requirements imposed by the need to support the exchanging of monitoring information, this study made both conceptual and technical contributions. In terms of the conceptual or theoretical contributions, this study has made two key contributions:

- Generalized Response Time Metric: the method of measuring response time of a Semantic Web Service which takes into account the response time values measured at both the service and the client side. This makes the generalized response time metric a collaborative effort.
- Monitoring Information eXchange (MIX) - the protocol for exchanging monitoring information between participating Semantic Web Services' monitoring agents.

In terms of the technical or practical contributions, this study has made two contributions in its endeavor to illustrate technical feasibility of the conceptual contributions in relation to the Monitoring Information eXchange (MIX) protocol discussed above.

- A technology-agnostic architecture for a Collaborative and Corroborative Semantic Web Service Monitoring.
- Semantic Web Service Monitoring Tool (software tool) which implements the Monitoring Information Exchange (MIX) protocol.

The next and final section presents the recommendations that emanate from this study in terms of their implications for both theory and practice and what further research still needs to be done in the area of collaborative and corroborative monitoring of Semantic Web Services.

7.4 Recommendations

Recommendations for further research and application are provided here as natural extensions to the contributions made by this study.

7.4.1 Semantic Descriptions of Performance Metrics in QoS Ontology

This study focused mainly on Generalized Response Time Metric (GRTM) as one of the parameters that characterize the performance of a Semantic Web Service. The recommendation is to include the semantic descriptions of the metrics in terms of their mathematical or arithmetical expressions. This means developing the correct syntax for expressing the mathematical formulae for the performance metrics namely *latency*, *response time*, *throughput*, and *error-rate*. For example: $Q_{latency} = Q_{responsetime} - ExecutionTime$ is a mathematical difference (result from subtraction), but in this case it represents latency as a concept. A fully expressive ontology model for performance needs to include the logical expressions of these performance parameters.

7.4.2 MIX-Based Semantic Web Service Monitoring Tool

The implementation of the MIX protocol as part of this study was done mainly for proof of concept and technical validation. For a real-life usage, the Semantic Web Service Monitoring Tool based on the MIX protocol would need to provide extensible interfaces which software engineers can implement to support MIX functionality. For instance, the message format for the protocol is not defined, but for a real-life application, this needs to be either XML

or JSON format and the software engineer would write either the client MIX agent or the service MIX agent to support these message formats. The recommendation is for using JSON as this is lightweight and will not introduce unnecessary overhead to the monitoring process.

7.4.3 MIX Agent Negotiation Protocol

Another area for further development is regarding to the negotiation protocol as specified. This study developed a very light-weight negotiation protocol. The current design of the protocol is deliberately abstract and leaves more scope for expansion to handle more complex use cases. Software engineers and researchers are encouraged to explore possible negotiation strategies and implement these as part of the post-execution processes of the MIX protocol, but also during the initial exchange of the baseline data when the MIX session is initialized between the client MIX agent and the service MIX agent. Currently, the decision-making by the MIX agents is not specified or dictated. The MIX protocol does not specify what the MIX agent should do when it receives the baseline data. In practice, the behavior needs to be fully specified even if the MIX agent has to apply some artificial intelligence logic to decide on best course of action.

Appendices

Appendix A

The 10th International Conference for Internet Technology and Secured Transactions (ICITST -2015), London, UK, December 14-16, 2015

A Generalized Web Service Response Time Metric to Support Collaborative and Corroborative Web Service Monitoring

Makitla, I
Mtsweni, J

Abstract

In this paper, we describe the development of a generalized metric for computing response time of a web service. Such a generalized metric would help to develop consensus with regards to the meanings of contracted Quality of Service (QoS) parameters; this also avoids the confusion that may arise when the same QoS parameter is understood differently by both the service provider and the consumer (e.g. response time versus execution time). Without a generalized metric for response time, contemporary monitoring approaches employ measures such as incentivizing the service consumers to honestly report their perceived service quality. Other measures involve using a dedicated monitoring service to observe the service execution and enforce the contracted QoS. This is a costly duplication of infrastructure. Having a generalized metric for each QoS parameter eliminates the need for incentive schemes or costly dedicated monitoring infrastructure.

Appendix B

Proceedings of the 2014 IST-Africa Conference , May 2014.

Towards a Collaborative Approach to Web Service Monitoring: In appreciation of connectivity challenges in Africa

Makitla, I
Mtsweni, J

Abstract

Current approaches to Quality of Service (QoS) monitoring using Service Level Agreements (SLAs) fall short of appreciating the many possible causes of breaching these digital contracts. These approaches are mainly concerned with the enforcement of these contracts without paying attention to what aspects of the service oriented environment the service provider has no control over and which have direct impact on his ability to meet the contracted QoS. Paying attention to these aspects is very crucial in the African context with intermittent connectivity issues, including limited Internet speed and electricity outages among others. It is very likely therefore that most of the advertised QoS parameter, especially those with obvious dependency on connectivity, will be breached. Current approaches to Web Service monitoring do not consider this at all. This paper proposes a collaborative QoS monitoring approach that allows for mid-way renegotiation of contracted QoS/SLAs based on consumer's initial service experience.

Bibliography

Abdelaziz, I. et al. (Sept. 2017). "A survey and experimental comparison of distributed SPARQL engines for very large RDF data". In: *Proceedings of the VLDB Endowment* 10, pp. 2049–2060. DOI: 10.14778/3151106.3151109.

Acuna, C. J. and E. Marcos (2006). "Modeling semantic Web services: a case study". In: *6th international conference on Web engineering (ICWE'06)*. Palo Alto, California, USA.

Agarwal, V. et al. (Jan. 2008). "Understanding approaches for web service composition and execution". In: *Proceedings of the 1st Bangalore annual Compute conference on - Compute '08*. New York, New York, USA: ACM Press, p. 1. ISBN: 9781595939500. DOI: 10.1145/1341771.1341773. URL: http://dl.acm.org/citation.cfm?id=1341771.1341773.

Akingbesote, A. et al. (Nov. 2013). "A quality of service aware multi-level strategy for selection of optimal web service". In: *2013 International Conference on Adaptive Science and Technology*. IEEE, pp. 1–5. ISBN: 978-1-4799-3067-8. DOI: 10.1109/ICASTech.2013.6707518. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6707518.

Akkiraju, R. (2007). "Semantic Web Services". In: *Semantic Web Services: Theory, Tools, and Applications*. Ed. by J. Cardoso and N. Hershey, pp. 191–216.

Akkiraju, R. et al. (2005). "Web Service Semantics - WSDL-S". In.

Alshinina, R. and K. Elleithy (Apr. 2017). "Performance and Challenges of Service-Oriented Architecture for Wireless Sensor Networks". In: *Sensors* 2017, pp. 536–575. DOI: 10.3390/s17030536.

Amir, M. (2018). "Semantically-enriched and semi-Autonomous collaboration framework for the Web of Things. Design, implementation and evaluation of a multi-party collaboration framework with semantic annotation and

representation of sensors in the Web of Things and a case study on disaster management". PhD. University of Bradford. URL: http://hdl.handle.net/10454/14363.

Andrieux, A. et al. (2007). "Web Services Agreement Specification (WS-Agreement)".

Artaiam, N. and T. Senivongse (2008). "Enhancing Service-Side QoS Monitoring for Web Services". In: *2008 Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*. IEEE, pp. 765–770. ISBN: 978-0-7695-3263-9. DOI: 10.1109/SNPD.2008.157. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4617464.

Ayadi, N. and M. Ben Ahmed (Feb. 2011). "An Enhanced Framework for Semantic Web Service Discovery". In: vol. 82, pp. 53–67. DOI: 10.1007/978-3-642-21547-6_5.

Babaee, R. and S. M. Babamir (Aug. 2013). "Runtime Verification of Service-Oriented Systems: A Well-Rounded Survey". In: *Int. J. Web Grid Serv.* 9.3, pp. 213–267. ISSN: 1741-1106. DOI: 10.1504/IJWGS.2013.055699. URL: https://doi.org/10.1504/IJWGS.2013.055699.

Balaji B, S., K. N K, and R. K. R S (2018). "Fuzzy service conceptual ontology system for cloud service recommendation". In: *Computers & Electrical Engineering* 69, pp. 435–446. ISSN: 0045-7906. DOI: https://doi.org/10.1016/j.compeleceng.2016.09.013. URL: https://www.sciencedirect.com/science/article/pii/S0045790616302804.

Baraldo, V., A. Zuccato, and T. Vardanega (Mar. 2015). "Reconciling Service Orientation with the Cloud". In: pp. 195–202. DOI: 10.1109/SOSE.2015.26.

Baresi, L. and S. Guinea (June 2013). "Event-Based Multi-level Service Monitoring". In: *2013 IEEE 20th International Conference on Web Services*. IEEE, pp. 83–90. ISBN: 978-0-7695-5025-1. DOI: 10.1109/ICWS.2013.21. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6649565.

Battle, S. et al. (2005). *Semantic Web Services Framework (SWSF) Overview*. URL: http://www.w3.org/Submission/SWSF/ (visited on 08/30/2014).

Ben Lamine, R., R. Jemaa, and I. Amous (Oct. 2018). "Formal Specification of Adaptable Semantic Web Services Composition". In: *International Journal of Information Technology and Web Engineering* 13, pp. 14–34. DOI: 10.4018/IJITWE.2018100102.

Benvenuti, F. et al. (2017). "An ontology-based framework to support performance monitoring in public transport systems". In: *Transportation Research Part C: Emerging Technologies* 81, pp. 188–208. ISSN: 0968-090X. DOI: https://doi.org/10.1016/j.trc.2017.06.001. URL: https://www.sciencedirect.com/science/article/pii/S0968090X17301547.

Berners-Lee, T. (2003). *The Semantic Web and Challenges*. URL: http://www.w3.org/2003/Talks/01-sweb-tbl/Overview.html.

Bokhari, S. and F. Azam (Apr. 2015). "Limitations of Service Oriented Architecture and its Combination with Cloud Computing". In: *Bahria University Journal of Information and Communication Technologies (BUJICT)* 8.

Bouguettaya, A. et al. (Mar. 2017). "A service computing manifesto: The next 10 years". In: *Communications of the ACM* 60, pp. 64–72. DOI: 10.1145/2983528.

Boukaye Boubacar, T. et al. (July 2018). "Service-Oriented Computing for intelligent train maintenance". In: *Enterprise Information Systems* 13, pp. 1–24. DOI: 10.1080/17517575.2018.1501818.

Bruijn, J. de et al. (2008). *Modeling Semantic Web Services*. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-540-68169-4. DOI: 10.1007/978-3-540-68172-4. URL: http://link.springer.com/10.1007/978-3-540-68172-4.

Bruijn, J. de. (2008). *Modeling Semantic Web services : the web service modeling language*. Springer, p. 192. ISBN: 9783540681724.

Cabrera, O. and X. Franch (May 2012). "A quality model for analysing web service monitoring tools". In: *2012 Sixth International Conference on Research Challenges in Information Science (RCIS)*, pp. 1–12. DOI: 10.1109/RCIS.2012.6240444.

Calderón-Gómez, H. et al. (May 2021). "Evaluating Service-Oriented and Microservice Architecture Patterns to Deploy eHealth Applications in Cloud Computing Environment". In: *Applied Sciences* 11, p. 4350. DOI: 10.3390/app11104350.

Carlsson, S. A. et al. (Oct. 2010). "Socio-technical IS design science research: developing design theory for IS integration management". In: *Information Systems and e-Business Management* 9.1, pp. 109–131. ISSN: 1617-9846. DOI: 10.1007/s10257-010-0140-6. URL: http://link.springer.com/10.1007/s10257-010-0140-6.

Chifu, V. (2010). "Automatic Web Service Composition". PhD. Technical University of Cluj-Napoca.

Chinnici, R. et al. (June 2007). "Web Services Description Language (WSDL)". In.

Chooralil, V. S. and E. Gopinathan (2015). "A Semantic Web Query Optimization Using Resource Description Framework". In: *Procedia Computer Science* 70. Proceedings of the 4th International Conference on Eco-friendly Computing and Communication Systems, pp. 723–732. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2015.10.110. URL: https://www.sciencedirect.com/science/article/pii/S1877050915032743.

Cimpian, E. et al. (2008). "Ontologies and Matchmaking." In: *Semantic Service Provisioning*. Ed. by D. Kuropka et al. Berlin Heidelberg: Springer, pp. 19–54.

Comuzzi, M. and B. Pernici (2005). "An Architecture for Flexible Web Service QoS Negotiation". In: *Ninth IEEE International EDOC Enterprise Computing Conference (EDOC'05)*. IEEE, pp. 70–82. ISBN: 0-7695-2441-9. DOI: 10.1109/EDOC.2005.4. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1540669.

de Souza Neto, J. B., A. M. Moreira, and M. A. Musicante (2018). "Semantic Web Services testing: A Systematic Mapping study". In: *Computer Science Review* 28, pp. 140–156. ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2018.03.002. URL: https://www.sciencedirect.com/science/article/pii/S1574013717301648.

Di Martino, B. et al. (2014). "Semantic Representation of Cloud Services: A Case Study for Microsoft Windows Azure". In: *2014 International Conference on Intelligent Networking and Collaborative Systems*, pp. 647–652. DOI: 10.1109/INCoS.2014.76.

Dimitrov, M. et al. (2007). "WSMO Studio - a semantic web services modelling environment for WSMO." In: *4th European Semantic Web Conference, ESWC 2007, Innsbruck, Austria, June 3-7, 2007*. Innsbruck, Austria.

Dobson, G., R. Lock, and I. Sommerville (2005). "QoSOnt: a QoS Ontology for Service-Centric Systems". In: *31st EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE, pp. 80–87. ISBN: 0-7695-2431-1. DOI: 10.1109/EUROMICRO.2005.49. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1517730.

Duineveld, A. et al. (June 2000). "WonderTools? A comparative study of ontological engineering tools". In: *International Journal of Human-Computer Studies* 52, pp. 1111–1133. DOI: 10.1006/ijhc.1999.0366.

Ellis, T. and Y. Levy (2010). "A Guide for Novice Researchers: Design and Development Research Methods". In: *Informing Science & IT Education Conference (InSITE)*. Cassino, Italy, pp. 107–118.

Fensel, D. (2003). *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*. Springer-Verlag New York, Inc. ISBN: 3540003029.

Fensel, D. et al. (Apr. 2011). *Semantic Web Services*. Springer-Verlag New York, Inc. ISBN: 978-3-642-19192-3. DOI: 10.1007/978-3-642-19193-0.

Fernández, J. D. et al. (2013). "Binary RDF representation for publication and exchange (HDT)". In: *Journal of Web Semantics* 19, pp. 22–41. ISSN: 1570-8268. DOI: https://doi.org/10.1016/j.websem.2013.01.002. URL: https://www.sciencedirect.com/science/article/pii/S1570826813000036.

Ferre, S. (Jan. 2012). "An RDF Vocabulary for the Prepresentation and Exploration of Expressions with an Illustration on Mathematical Search". In.

Filho, O. F. F. and M. A. G. V. Ferreira (2009). "Semantic Web Services: a RESTful approach". In: *IADIS International Conference WWW/Internet 2009*. Rome, Italy.

Gonsalves, T. and K. Itoh (2008). "Performance Simulation And Design Of Petri Net Systems". In: *Journal of Integrated Design & Process Science* 12.4, pp. 27–37. ISSN: 1092-0617.

Greenwell, R., X. Liu, and K. Chalmers (2015). "Semantic description of cloud service agreements". In: *2015 Science and Information Conference (SAI)*, pp. 823–831. DOI: 10.1109/SAI.2015.7237239.

Gregor, S. and A. R. Hevner (Nov. 2010). "Introduction to the special issue on design science". In: *Information Systems and e-Business Management* 9.1, pp. 1–9. ISSN: 1617-9846. DOI: 10.1007/s10257-010-0159-8. URL: http://link.springer.com/10.1007/s10257-010-0159-8.

Grunske, L. (2008). "Specification patterns for probabilitic quality properties". In: *International Conference on Software Engineering*, pp. 31–40.

Grunske, L. and P. Zhang (Aug. 2009). "Monitoring probabilistic properties". In: *Joint Meeting of the European Software Engineering Conference and th ACM Sigsoft International Symposium on Foundations of Software Engineering, Amsterdam, 2009*. ACM, pp. 183–192.

Hayyolalam, V. and A. A. Pourhaji Kazem (2018). "A systematic literature review on QoS-aware service composition and selection in cloud environment". In: *Journal of Network and Computer Applications* 110, pp. 52–74. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2018.03.003. URL: https://www.sciencedirect.com/science/article/pii/S1084804518300845.

Al-Hazmi, Y. and T. Magedanz (2015). "MOFI: Monitoring ontology for federated infrastructures". In: *2015 IEEE International Workshop on Measurements & Networking (M&N)*, pp. 1–6. DOI: 10.1109/IWMN.2015.7322988.

Hernández, A. and M. Moreno García (Jan. 2010). "A formal definition of RESTful semantic web services". In: pp. 39–45. DOI: 10.1145/1798354.1798384.

Hevner, A. R. et al. (2004). "Design Science in Information Systems Research". In: *MIS Quarterly* 28.1, pp. 75–105.

Hofstee, E. (2006). *Constructing a Good Dissertation. A Practical Guide to Finishing a Masters, MBA or PhD on Schedule*. Exactica.

Hsu, I.-C. and S.-F. Lyu (2019). "A Lightweight Linked Data Reasoner Using Jena and Axis2". In: *IEA/AIE*.

Huang, J., D. J. Abadi, and K. Ren (Aug. 2011). "Scalable SPARQL Querying of Large RDF Graphs". In: *Proc. VLDB Endow.* 4.11, pp. 1123–1134. ISSN: 2150-8097. DOI: 10.14778/3402707.3402747. URL: https://doi.org/10.14778/3402707.3402747.

Huf, A. and F. Siqueira (2019). "Composition of heterogeneous web services: A systematic review". In: *Journal of Network and Computer Applications* 143, pp. 89–110. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2019.06.008. URL: https://www.sciencedirect.com/science/article/pii/S108480451930205X.

Huhns, M. N. (2005). "Service-Oriented Computing : Key Concepts and Principles". In: 9.1, pp. 75–81.

Iivari, J. (2007). "A Paradigmatic Analysis of Information Systems as a Design Science". In: *Scandinavian Journal of Information Systems* 19.2, pp. 39–64.

Jurca, R., B. Faltings, and W. Binder (2007). "Reliable QoS monitoring based on client feedback". In: *Proceedings of the 16th international conference on World Wide Web*. WWW '07, pp. 1003–1012. URL: http://doi.acm.org/10.1145/1242572.1242708.

Kapoor, B. and S. Sharma (2010). "A Comparative Study of Ontology building Tools in Semantic Web Applications". In: *International Journal of Webz '&' Semantic Technology* 1, pp. 1–13.

Keeney, J. et al. (May 2011). "A semantic monitoring and management framework for end-to-end services". In: *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*. IEEE, pp. 658–661. ISBN: 978-1-4244-9219-0. DOI: 10.1109/INM.2011.5990649. URL: http://ieeexplore.ieee.org/document/5990649/.

Keller, A. and H. Ludwig (2003). "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services". In: *Journal of Network and Systems Management* 11.1, pp. 57–81.

Konstantinou, N. et al. (Jan. 2010). "Technically approaching the semantic web bottleneck". In: *International Journal of Web Engineering and Technology* 6, pp. 83–111. DOI: 10.1504/IJWET.2010.034761.

Kopecký, J. and T. Vitvar (Aug. 2008). "WSMO-Lite: Lowering the Semantic Web Services Barrier with Modular and Light-Weight Annotations". In: *2008 IEEE International Conference on Semantic Computing*. IEEE, pp. 238–244. DOI: 10.1109/ICSC.2008.54. URL: http://dx.doi.org/doi:10.1109/ICSC.2008.54%20http://ieeexplore.ieee.org/document/4597197/.

Kopecký, J., T. Vitvar, et al. (Dec. 2007). "SAWSDL: semantic annotations for WSDL and XML schema". In: *Internet Computing, IEEE* 11, pp. 60–67. DOI: 10.1109/MIC.2007.134.

Kowalkiewicz, M., A. Ludwig, et al. (2008). "Service Composition and Binding". In: *Semantic Service Provisioning*. Ed. by D. Kuropka et al. Berlin Heidelberg: Springer, pp. 73–143.

Kowalkiewicz, M., M. Momotko, and A. Saar (2008). "Service Composition Enactment". In: *Semantic Service Provisioning*. Ed. by D. Kuropka et al. Berlin, Heidelberg: Springer, pp. 144–162.

Krichen, M. and S. Tripakis (2004). *Model Checking Software*. Ed. by S. Graf and L. Mounier. Vol. 2989. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 109–126. ISBN: 978-3-540-21314-7. DOI: 10.1007/b96721. URL: http://www.springerlink.com/index/10.1007/b96721.

Kritikos, K. and D. Plexousakis (2007). "OWL-Q for semantic QoS-based Web Service description and discovery". In: *CEUR Workshop Proceedings*. Vol. 243, pp. 123–137. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.93.9067.

Kuropka, D. et al., eds. (2008). *Semantic Service Provisioning*. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-540-78616-0. DOI: 10.1007/978-3-540-78617-7. URL: http://www.springerlink.com/index/10.1007/978-3-540-78617-7.

Lanthaler, M. (Feb. 2012). "Seamless Integration of RESTful Services into the Web of Data". In: *Advances in Multimedia* 2012. DOI: 10.1155/2012/586542.

Lanthaler, M. and C. Guetl (Sept. 2011). "Aligning Web Services with the Semantic Web to Create a Global Read-Write Graph of Data". In: pp. 15–22. DOI: 10.1109/ECOWS.2011.17.

Lara Calache, M. de and C. De Farias (Mar. 2020). "Graphical and Collaborative Annotation Support for Semantic Web Services". In: pp. 210–217. DOI: 10.1109/ICSA-C50368.2020.00044.

Lera, I., C. Juiz, and R. Puigjaner (June 2006). "Performance-related ontologies and semantic web applications for on-line performance assessment of intelligent systems". In: *Science of Computer Programming* 61.1, pp. 27–37. ISSN: 01676423. DOI: 10.1016/j.scico.2005.11.003. URL: http://linkinghub.elsevier.com/retrieve/pii/S016764230600013X.

Letia, I. A. and A. N. Marginean (2011). "Service monitoring with ontology based expectations". In: *2011 IEEE 7th International Conference on Intelligent Computer Communication and Processing*, pp. 111–114. DOI: 10.1109/ICCP.2011.6047851.

Li, S. and J. Zhou (Dec. 2009). "The WSMO-QoS Semantic Web Service Discovery Framework". In: *2009 International Conference on Computational Intelligence and Software Engineering*. IEEE, pp. 1–5. ISBN: 978-1-4244-4507-3. DOI: 10.1109/CISE.2009.5366383. URL: http://ieeexplore.ieee.org/document/5366383/.

Lilien, L. et al. (May 2011). "Quality of Service in an Opportunistic Capability Utilization Network". In: *Mobile Opportunistic Networks: Architectures, Protocols and Applications*. ISBN: 978-1-4200-8812-0. DOI: 10.1201/b10904-8.

Liu, J. et al. (Jan. 2019). "An effective biomedical data migration tool from resource description framework to JSON". In: *Database : the journal of biological databases and curation* 2019. DOI: 10.1093/database/baz088.

Liu, L. (2011). "Organic service-level management in service-oriented environments". In: KIT Scientific Publishing.

Louge, T. et al. (Aug. 2018). "Semantic Web Services Composition in the astrophysics domain: Issues and solutions". In: *Future Generation Computer Systems* 90, pp. 185–197. DOI: 10.1016/j.future.2018.07.063..

Luthria, H. and F. Rabhi (Apr. 2009). "Service Oriented Computing in Practice: An Agenda for Research into the Factors Influencing the Organizational Adoption of Service Oriented Architectures". In: *Journal of theoretical and applied electronic commerce research* 4.1, pp. 39–56. ISSN: 0718-1876. DOI: 10.4067/S0718-18762009000100005. URL: http://www.scielo.cl/scielo.php?script=sci%7B%5C_%7Darttext%7B%5C&%7Dpid=S0718-18762009000100005%7B%5C&%7Dlng=en%7B%5C&%7Dnrm=iso%7B%5C&%7Dtlng=en.

Ma, Z., M. A. M. Capretz, and L. Yan (2016). "Storing massive Resource Description Framework (RDF) data: a survey". In: *The Knowledge Engineering Review* 31.4, pp. 391–413. DOI: 10.1017/S0269888916000217.

Makitla, I. and J. Mtsweni (Dec. 2015). "A generalized web service response time metric to support collaborative and corroborative web service monitoring". In: *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*. IEEE, pp. 373–377. ISBN: 978-1-9083-2052-0. DOI: 10.1109/ICITST.2015.7412124. URL: http://ieeexplore.ieee.org/document/7412124/.

Malik, Z. (July 2017). "Usability evaluation of ontology engineering tools". In: pp. 576–584. DOI: 10.1109/SAI.2017.8252154.

March, S. and G. Smith (1995). "Design and natural science research on information technology". In: *Decision Support Systems* 15.4, pp. 251–266.

March, S. and V. Storey (2008). "Design science in the information systems discipline: an introduction to the special issue on design science research." In: *MIS Quarterly* 32.4, pp. 725–730.

Martin, D. et al. (2004). "OWL-S: Semantic Mrkup for Web Services". In: *W3C Member Submission* 22.4.

Masood, T., C. Cherifi, and N. Moalla (Feb. 2016). "Performance monitoring framework for service oriented system lifecycle". In: *2016 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pp. 800–806.

McBride, B. (2004). *Handbook on ontologies*.

Medjahed, B. and Y. Atif (2007). "Context-based matching for Web service composition". In: *Distrib Parallel Databases* 21.1, pp. 5–37. URL: http://download.springer.com/static/pdf/615/art%7B%5C%7D3A10.

1007%7B%5C%7D2Fs10619-006-7003-7.pdf?auth66=1409667468%7B%5C_%7Deea04805cb286d7fb69dc2352ad534c0%7B%5C&%7Dext=.pdf.

Michlmayr, A. et al. (2009). "Comprehensive QoS monitoring of Web services and event-based SLA violation detection". In: *Proceedings of the 4th International Workshop on Middleware for Service Oriented Computing MWSOC 09*, pp. 1–6. URL: http://portal.acm.org/citation.cfm?doid=1657755.1657756.

Mittal, S. et al. (Apr. 2016). "Automatic Extraction of Metrics from SLAs for Cloud Service Management". In: pp. 139–142. DOI: 10.1109/IC2E.2016.14.

Mohebbi, K., S. Ibrahim, and N. B. Idris (Oct. 2012). "Contemporary Semantic Web Service Frameworks: An Overview and Comparisons". In: p. 12. arXiv: 1210.3320. URL: http://arxiv.org/abs/1210.3320.

Mtsweni, J., E. Biermann, and L. Pretorius (June 2014). "iSemServ: A model-driven approach for developing semantic web services". In: *South African Computer Journal* 52. DOI: 10.18489/sacj.v52i0.185.

Nacer, H. and D. Aissani (2014). "Semantic web services: Standards, applications, challenges and solutions". In: *Journal of Network and Computer Applications* 44, pp. 134–151. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2014.04.015. URL: http://www.sciencedirect.com/science/article/pii/S1084804514001143.

Nejkovic, V. et al. (2020). "Semantic approach to RIoT autonomous robots mission coordination". In: *Robotics and Autonomous Systems* 126, p. 103438. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2020.103438. URL: https://www.sciencedirect.com/science/article/pii/S0921889019306414.

OATES, B. (2006). *Researching information systems and computing*. Sage Publications Ltd.

Olivier, M. (2004). *Information technology research: A practical guide for computer science and informatics*. 2nd ed. Pretoria, South Africa: Van Schaik.

Oriol, M., J. Marco, and X. Franch (Oct. 2014). "Quality models for web services: A systematic mapping". In: *Information and Software Technology* 56.10, pp. 1167–1182. ISSN: 09505849. DOI: 10.1016/j.infsof.2014.03.012. URL: http://linkinghub.elsevier.com/retrieve/pii/S0950584914000822.

Oskooei, M. and S. Mohd Daud (Sept. 2014). "Quality of service (QoS) model for web service selection". In: *2014 International Conference on Com-*

puter, Communications, and Control Technology (I4CT), pp. 266–270. DOI: 10.1109/I4CT.2014.6914187.

Paik, I., W. Chen, and M. N. Huhns (Jan. 2014). "A Scalable Architecture for Automatic Service Composition". In: *IEEE Transactions on Services Computing* 7.1, pp. 82–95. ISSN: 1939-1374. DOI: 10.1109/TSC.2012.33. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6357179.

Papaioannou, I. et al. (2006). "A QoS ontology language for Web-services". In: *20th International Conference on Advanced Information Networking and Applications - Volume 1 (AINA'06)*. IEEE, pp. 101–106. ISBN: 0-7695-2466-4. DOI: 10.1109/AINA.2006.51. URL: http://ieeexplore.ieee.org/document/1620177/.

Papazoglou, M. P., P. Traverso, S. Dustdar, F. Leymann, and B. J. Krämer (2006). "Service-Oriented Computing : A Research Roadmap". In: *Dagstuhl Seminar Proceedings 05462 Service Service Oriented Computing (SOC)*. April, pp. 1–29. URL: http://drops.dagstuhl.de/opus/volltexte/2006/524.

Papazoglou, M. P., P. Traverso, S. Dustdar, and F. Leymann (June 2008). "SERVICE-ORIENTED COMPUTING: A RESEARCH ROADMAP". en. In: *International Journal of Cooperative Information Systems* 17.02, pp. 223–255. ISSN: 0218-8430. DOI: 10.1142/S0218843008001816. URL: http://www.worldscientific.com/doi/abs/10.1142/S0218843008001816.

Patil, A. et al. (May 2004). "METEOR-S Web Service Annotation Framework". In: pp. 553–562. DOI: 10.1145/988672.988747.

Paulraj, D., S. Swamynathan, and M. Madhaiyan (2011). "PROCESS MODEL ONTOLOGY-BASED MATCHMAKING OF SEMANTIC WEB SERVICES". In: *International Journal of Cooperative Information Systems* 20.04, pp. 357–370. DOI: 10.1142/S0218843011002262. URL: https://doi.org/10.1142/S0218843011002262.

Peffers, K. et al. (2007). "A design science research methodology for information systems research". In: *Journal of Management Information Systems* 24.3, pp. 45–77.

Pradhan, S. et al. (Jan. 2016). "Research Issues on WSBS Performance Evaluation". In: pp. 123–129. DOI: 10.1109/CINE.2016.29.

Prud'hommeaux, E. and A. Seaborne (Jan. 2008). "SPARQL Query Language for RDF". In: URL: http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/.

Raj, P. et al. (Jan. 2015). "Service Composition and Execution Plan Generation of Composite Semantic WEB Services Using Abductive Event Calculus". In: *Computational Intelligence* 32, n/a–n/a. DOI: 10.1111/coin.12080.

Ranpara, R., A. Yusufzai, and C. Kumbharana (2019). "A Comparative Study of Ontology Building Tools for Contextual Information Retrieval". In.

Reeves, T. (2006). "Design research from a technology perspective". In: *Educational Design Research*. Ed. by J. Van Den Akker et al. London, UK: Routledge, pp. 52–66.

Reiff-Marganiec, S., H. Yu, and T. Marcel (2009). "Service selection based on non-functional properties". In: *ICSOC 2007 Workshops, LNCS 4907*. Ed. by E. DI NITTO and M. RIPEANU. Vol. 4907, pp. 128–138.

Rojas, H., K. A. Arias, and R. Renteria (2021). "Service-oriented architecture design for small and medium enterprises with infrastructure and cost optimization." In: *Procedia Computer Science* 179. 5th International Conference on Computer Science and Computational Intelligence 2020, pp. 488–497. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2021.01.032. URL: https://www.sciencedirect.com/science/article/pii/S1877050921000375.

Sabata, B. et al. (1997). "Taxomomy of QoS Specifications". In: *Proceedings of the 3rd Workshop on Object-Oriented Real-Time Dependable Systems - (WORDS '97)*. WORDS '97. Washington, DC, USA: IEEE Computer Society. ISBN: 0-8186-8046-6. URL: http://dl.acm.org/citation.cfm?id=523512.837716.

Sachan, D., S. Kumar Dixit, and S. Kumar (Oct. 2014). "QOS Aware Formalized Model for Semantic Web Service Selection". In: *International journal of Web & Semantic Technology* 5.4, pp. 83–100. ISSN: 09762280. DOI: 10.5121/ijwest.2014.5406. URL: http://airccse.org/journal/ijwest/papers/5414ijwest06.pdf.

Sedayao, J. (Nov. 2008). "Implementing and operating an internet scale distributed application using service oriented architecture principles and cloud computing infrastructure". In: *iiWAS'2008 - The Tenth International Conference on Information Integration and Web-based Applications Services*, pp. 417–421. DOI: 10.1145/1497308.1497384.

Shafi, U. et al. (2018). "Parsing RDFs to Extract Object Oriented Model Using Apache Jena". In.

Shaukat Dar, K. et al. (July 2016). "An overview of Service Oriented Architecture, Cloud Computing and Azure Platform". In: *International Journal of Computer Science and Information Security* 14, pp. 891–896.

Siddiqui, Z. and K. Seth (June 2017). "Study on service selection effort estimation in service oriented architecture-based applications powered by information entropy weight fuzzy comprehensive evaluation model". In: *IET Software* 12. DOI: 10.1049/iet-sen.2016.0141.

Sigwele, T. et al. (Aug. 2018). "An Intelligent Edge Computing Based Semantic Gateway for Healthcare Systems Interoperability and Collaboration". In: DOI: 10.1109/FiCloud.2018.00060.

Şimşek, U., E. Kärle, and D. Fensel (2018). "Machine Readable Web APIs with Schema.org Action Annotations". In: *Procedia Computer Science* 137. Proceedings of the 14th International Conference on Semantic Systems 10th – 13th of September 2018 Vienna, Austria, pp. 255–261. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2018.09.025. URL: https://www.sciencedirect.com/science/article/pii/S1877050918316302.

Singh, S. and X. Liu (Feb. 2016). "A cloud service architecture for analyzing big monitoring data". In: *Tsinghua Science and Technology* 21, pp. 55–70. DOI: 10.1109/TST.2016.7399283.

Slimani, T. (Jan. 2013). "Semantic Description of Web Services". In: *International Journal of Computer Science Issues* 10, pp. 368–377.

— (Feb. 2015). "Ontology Development: A Comparing Study on Tools, Languages and Formalisms". In: *Indian Journal of Science and Technology* 8, pp. 1–12. DOI: 10.17485/ijst/2015/v8i1/54249.

Souza Neto, J., A. Moreira, and M. Musicante (May 2018). "Semantic Web Services testing: A Systematic Mapping study". In: *Computer Science Review* 28, pp. 140–156. DOI: 10.1016/j.cosrev.2018.03.002.

Stollberg, M. and D. Fensel (2010). "Semantics for Service-Oriented Architectures". In: *Agent-Based Service-Oriented Computing*. Ed. by N. Griffiths and K.-M. Chao. London: Springer London, pp. 113–139. ISBN: 978-1-84996-041-0. DOI: 10.1007/978-1-84996-041-0_5. URL: https://doi.org/10.1007/978-1-84996-041-0_5.

Telang, P. R., A. K. Kalia, and M. P. Singh (May 2014). "Engineering Service Engagements via Commitments". In: *IEEE Internet Computing* 18.3, pp. 46–54. ISSN: 1089-7801. DOI: 10.1109/MIC.2013.86. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6579598.

Tran, V. X., H. Tsuji, and R. Masuda (Sept. 2009). "A new QoS ontology and its QoS-based ranking algorithm for Web services". In: *Simulation Modelling Practice and Theory* 17.8, pp. 1378–1398. ISSN: 1569190X. DOI: 10.1016/j.simpat.2009.06.010. URL: http://linkinghub.elsevier.com/retrieve/pii/S1569190X09000859.

Vaculin, R. (Jan. 2012). "Semantic Monitoring of Service-Oriented Business Processes". In: pp. 467–494. DOI: 10.4018/978-1-4666-0146-8.ch022.

Vaculín, R. (2009). "Process Mediation Framework for Semantic Web Services". PhD. Charles University.

Venable, J. (2010). "Design science research post hevner et al.: criteria, standards, guidelines, and expectations". In: *the 5th International Conference on Global Perspectives on Design Science Research (DESRIST '10)*. St. Gallen, Switzerland: Springer- Verlag, Berlin / Heidelberg, pp. 109–123.

Wang, Q. et al. (2009). "An Online Monitoring Approach for Web Service Requirements". In: *IEEE Transactions on Services Computing* 2.4, pp. 338–351. DOI: 10.1109/TSC.2009.22.

Welman, J. and S. Kruger (2004). *Research Methodology*. Oxford University Press.

Zela Ruiz, J. and C. M. Rubira (2016). "Quality of Service Conflict During Web Service Monitoring: A Case Study". In: *Electronic Notes in Theoretical Computer Science* 321. CLEI 2015, the XLI Latin American Computing Conference, pp. 113–127. ISSN: 1571-0661. DOI: https://doi.org/10.1016/j.entcs.2016.02.007. URL: https://www.sciencedirect.com/science/article/pii/S1571066116300081.

Zeng, K., J. Yang, et al. (Feb. 2013). "A Distributed Graph Engine for Web Scale RDF Data". In: *Proc. VLDB Endow.* 6.4, pp. 265–276. ISSN: 2150-8097. DOI: 10.14778/2535570.2488333. URL: https://doi.org/10.14778/2535570.2488333.

Zeng, Z. and S. Ying (2008). "The Usage of Semantic Condition Expression for Semantic Web Service Matchmaking". In: *International Symposium on Intelligent Information Technology Application Workshops*.

Zhang, F., Q. Zeng, et al. (2019). "Composition Context-Based Web Services Similarity Measure". In: *IEEE Access* 7, pp. 65195–65206. DOI: 10.1109/ACCESS.2019.2915371.

Zhang, L. and Y. Yang (2013). "Dynamic web service selection group decision-making method based on hybrid QoS". In: *International Journal of High Performance Computing and Networking* 7.3, pp. 215–226. DOI: 10.1504/

IJHPCN.2013.056517. URL: https://www.inderscienceonline.com/doi/abs/10.1504/IJHPCN.2013.056517.

Zhang, P., H. Jin, et al. (2018). "IgS-wBSRM: A time-aware Web Service QoS monitoring approach in dynamic environments". In: *Information and Software Technology* 96, pp. 14–26. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2017.11.003. URL: http://www.sciencedirect.com/science/article/pii/S0950584917301817.