Ontology Based Contextualization and Context Constraints Management in Web Service Processes

Kosala Gamini Yapa Bandara,

Yapa Mudiyanselage

Master of Science in Information Technology Security

Bachelor of Science in Computer Science (Special)

A Dissertation submitted in the fulfilment

of the requirements for the degree

of

Doctor of Philosophy (Ph.D.)

to the

Dublin City University

Faculty of Engineering and Computing, School of Computing

Advisor: Dr. Claus Pahl

January 2012

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Doctor of Philosophy is entirely my own work, that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed : ——————————

Student ID : 57 11 49 35

Date : ——————————-

Examiners:

Dr. Liam Tuohey	- School of Computing, Dublin City University, Ireland.
Prof. Dr. Wilhelm Hasselbring	- Head of Software Engineering Group, Christian-Albrechts-University of Kiel, Kiel, Germany.

Acknowledgments

I would like to express my profound gratitude to my supervisor, Dr. Claus Pahl for his continuous patience, encouragement, guidance and support. Thank you very much Claus for everything. You have been an exemplary supervisor and I have truly enjoyed working with you. I learnt a lot from you.

I would like to thank my colleagues in the Software and System Engineering Group and School of Computing for the given support, knowledge, humour and a friendly ear throughout my DCU life. Thanks Wang for all constructive discussions throughout this work. They were immensely helped to bring quality into this work. Pooyan, thank you very much for all the given supports. A special thank to your friends in the academy and industry, who gave their time and opinions to make this work successful. Aakash, Javed , Yalemisew, Paul and Murat; thank you very much for the given helps and happy times in DCU and Dublin. Oisin, thank you very much for all the given supports and sharing brilliant ideas and opinions about socity, life, politics and research. Dr. Declan McMullen and Oisin, I do not have enough words to say how much I thankful to you. Dr. Caroline Sheedy, thank you very much reading my reports. Dr. Mark Melia and Dr. Veronica Gacitua, thank you very much for the given support to set my things settled in Dublin and establish a firm foundation for this research. Your constructive discussions were massively helped me to make this work successful. Mr. and Mrs. Umagiliyage, thank you very much for the given support to make my living easy and enjoyable in Dublin.

I am greatly thankful to Science Foundation Ireland and Lero-The Irish Software Engineering Research Centre for the given financial support throughout this work. Last but not least, I would like to thank my parents and family. Words cannot express my gratitude to my father, mother, brother and two sisters for their love and encouragement. You have given me everything I need in life to succeed and I hope to make you proud.

Ontology Based Contextualization and Context Constraints Management in Web Service Processes

Abstract: The flexibility and dynamism of service-based applications impose shifting the validation process to runtime; therefore, runtime monitoring of dynamic features attached to service-based systems is becoming an important direction of research that motivated the definition of our work. We propose an ontology based contextualization and a framework and techniques for managing context constraints in a Web service process for dynamic requirements validation monitoring at process runtime. Firstly, we propose an approach to define and model dynamic service context attached to composition and execution of services in a service process at run-time. Secondly, managing context constraints are defined in a framework, which has three main processes for context manipulation and reasoning, context constraints generation, and dynamic instrumentation and validation monitoring of context constraints. The dynamic requirements attached to service composition and execution are generated as context constraints. The dynamic service context modeling is investigated based on empirical analysis of application scenarios in the classical business domain and analysing previous models in the literature. The orientation of context aspects in a general context taxonomy is considered important. The Ontology Web Language (OWL) has many merits on formalising dynamic service context such as shared conceptualization, logical language support for composition and reasoning, XML based interoperability, etc. XML-based constraint representation is compatible with Web service technologies. The analysis of complementary case study scenarios and expert opinions through a survey illustrate the validity and completeness of our context model. The proposed techniques for context manipulation, context constraints generation, instrumentation and validation monitoring are investigated through a set of experiments from an empirical evaluation. The analytical evaluation is also used to evaluate algorithms. Our contributions and evaluation results provide a further step towards developing a highly automated dynamic requirements management system for service processes at process run-time.

Keywords: dynamic service context, service-based applications, Web service process, service process run-time, ontology Web language, context manipulation, dynamic requirements, context constraints.

Contents

List of Figures

List of Tables

List of Publications

- Claus Pahl, Veronica Gacitua-Deca, Kosala Yapa Bandara and MingXue Wang. Ontology-based Compostion and Matching for Dynamic Cloud Service Co-ordination, International Journal of Metadata, Semantics and Ontologies (IJMSO), Inderscience, 2011.

- Claus Pahl, Veronica Gacitua-Deca, MingXue Wang and Kosala Yapa Bandara. Ontology-based Compostion and Matching for Dynamic Service Coordination. The 5th International Workshop on Ontology, Models, Conceptualization and Epistemology in Social, Artificial and Natural Systems, ONTOSE 2011, Co-located with 23rd International Conference on Advanced Information System Engineering, Springer, June 2011, London, England.

- Claus Pahl, Veronica Gacitua-Decar, Ming Xue Wang and Kosala Yapa Bandara. A Coordination Space Architecture for Service Collaboration and Cooperation. The 9th International Workshop on Systems/Software Architectures, IWSSA'11, Co-located with 23rd International Conference on Advanced Information System Engineering, Springer, June 2011, London, England.

- Claus Pahl, Kosala Yapa Bandara and MingXue Wang. Context Constraint Integration and Validation in Dynamic Web Service Compositions. M. Sheng, J. Yu, and S. Dustdar (Eds.) Enabling Context-Aware Web Services: Methods, Architectures, and Technologies, Chapman and Hall/CRC Press, 2010, Australia.

- MingXue Wang, Kosala Yapa Bandara and Claus Pahl. Process as a Service - Distributed Multi-tenant Policy-based Process Runtime Governance. IEEE International Conference on Services Computing, SCC 2010, July 2010, Miami, USA.

- MingXue Wang, Kosala Yapa Bandara and Claus Pahl. Distributed Aspect-Oriented Service Composition for Business Compliance Governance with Public Service Processes. The Fifth International Conference on Internet and Web Applications and Services, ICIW 2010, IEEE, May 2010, Barcelona, Spain.

- Kosala Yapa Bandara, MingXue Wang and Claus Pahl. Context Modeling and Constraints Binding in Web Service Business Processes. Workshop on Context-Aware Software Technology and Applications, CASTA 2009, Co-located with ESEC/FSE 2009, ACM Press, Aug. 2009, Amsterdam, The Netherlands.

- MingXue Wang, Kosala Yapa Bandara and Claus Pahl. Integrated Constraint Violation Handling for Dynamic Service Composition. IEEE International Conference on Services Computing SCC 2009, Sept. 2009, Bangalore,India.
- MingXue Wang, Kosala Yapa Bandara and Claus Pahl. Constraint Integration and Violation Handling for BPEL Processes. The Fourth International Conference on Internet and Web Applications and Services ICIW 2009, IEEE, May 2009, Venice, Italy.
- Kosala Yapa Bandara, MingXue Wang and Claus Pahl. Dynamic Integration of Context Model Constraints in Web Service Processes. International Software Engineering Conference SE2009, IASTED, Feb. 2009, Innsbruck, Austria.

List of Abbreviations

ATL	:	ATLAS Transformation Language
BPM	:	Business Process Management
CMO	:	Context Model Ontology
CC	:	Context Constraints
CLiX	:	Constraints Language in XML
Cset	:	Constraint Set
Cselector	:	Constraint Selector
DL	:	Description Logic
ECVC	:	Explicit Context Validation Constraint
EU	:	European Union
E-client	:	Enterprise Client
ICVC	:	Implicit Context Validation Constraint
ISO	:	International Organisation for Standardization
IEEE	:	Institute of Electrical and Electronic Engineers
IVS	:	Instrumentation and Validation Service
NFR	:	Non-Functional Requirements
OCL	:	Object Constraint Language
OWL	:	Ontology Web Language
OMG	:	Object Management Group
ODM	:	Ontology Definition Metamodel
PL	:	Process Level
QVT	:	Query / View / Transformation
QoS	:	Quality of Service Properties
RDF	:	Resource Description Framework
SCP	:	Service Context Profile
SOA	:	Service Oriented Architecture
SL	:	Service Level
SLA	:	Service Level Agreement
SWRL	:	Semantic Web Rule Language
UML	:	Unified Modeling Language
WSDL	:	Web Service Description Language
WS-BPEL	:	Web Services Business Process Execution Language
W3C	:	World Wide Web Consortium
XML	:	Extensible Modeling Language

Glossary of Definitions

- Context constraints - Dynamic requirements are implemented as assertions in terms of dynamic service context called context constraints.
- Constraints instrumented process - A service process equipped with context constraints.
- Constraints instrumentation - equip a service process with constraints as pre and post conditions.
- Context model ontology - A conceptualized model of dynamic service context and their relationships.
- Dynamic service context - Dynamic service context is client, provider or service related dynamic requirements, which enables or enhances effective composition and collaboration between them.
- Dynamic aspect - A process run-time relevant aspect attached to a service from a process perspective.
- Dynamic requirement - A requirement relevant to one dynamic aspect or combination of dynamic aspects.
- Dynamic validation - validation at process run-time.
- Dynamic instrumentation - instrumentation at process run-time.
- Dynamic service process - A Web service process in which services are combined at service process run-time.
- Explicit context category - A context category defined in a SCP called explicit context category.
- Implicit context category - A context category which is implicitly connected to an explicit context category.
- Inward perspective - How the service execution interacts with its environment is defined by inward perspective.
- Outward perspective - How the client or deployment environment impact on service execution is defined by outward perspective.
- Process - A combination of services to perform a specific task.
- Service - A Web service, which has one operation.
- Service level agreement(SLA) - A SLA is a formal definition of an agreement that exists between a service provider and a customer.

- Service context profile (SCP) - An instance level specification of context model ontology attached to a service.
- Transient context (TC) - An instance level specification of monitored context, derived context or an operator resulted context.
- Validation - A process which determines whether a context constraint is true or false.
- Validation monitoring - A process for collecting and storing the results of context constraints validation.
- Web service technology stack - A Web service architecture consist of discovery, description, packaging, transport and network layers and a set of protocols for each layer.

Chapter 1

Introduction

Contents

1.1 Motivation

Service-centric applications need monitoring at run-time because of numerous reasons, such as new versions of selected services, new services supplied by different vendors, different execution time contexts that hamper the correctness and quality levels of Web service applications, etc. Conventional applications are thoroughly validated before their deployment [Medjahed 2007], but often shifting the validation to run-time is needed to address monitoring and flexibility requirements [Wang 2009c, Baresi 2010b, Baresi 2011].

The validation monitoring of dynamic aspects attached to services and processes at run-time is becoming increasingly necessary and context-aware techniques are a possible method to address those challenges. The Software and Services Challenges Report - a contribution to the preparation of the Technology

Pillar on "Software, Grids, Security, and Dependability" of the 7th EU Framework Programme - introduces context as a central notion for future service platforms and technologies. The following paragraph is an excerpt from the report [Fitzgerald 2006]. *The desired state for software and services (third-generation services) is concerned with context-determined, consumer-driven, dynamically composed services. Mere enabling and integration of services alone is not sufficient. There is a dramatic increase in the threshold level of services management, quality, reliability, interoperability, security, and trust required across the heterogeneous enterprises involved. Also, the term 'context-determined' replaces the more commonly used term 'context-aware' to emphasize a more active stance on how software and services should help users to create new and meaningful knowledge through the use of third-generation services.* We observe that the third-generation Web services need to focus on operational features (e.g., services defined in WSDL), QoS features (e.g., services defined in OWL-S framework), and also features of the execution environment (e.g., ubiquitous services). Our focus is on context-aware third-generation Web services and applications. The notion of context can be used to define temporal and spatial properties of Web services in dynamic environments [Sheshagiri 2004]. In [Medjahed 2007], the notion of context is used to define functional and non-functional features of Web services. The recent trends in context-aware Web service compositions are compositions of spatial and temporal contexts in mobile service applications [Sheshagiri 2004], context communities and service discovery [Medjahed 2007, Doulkeridis 2006], context matching for service selection [Medjahed 2007] and context-based mediation approaches [Mrissa 2006b]. The basic Web service technology stack provides the interoperability and messaging mechanisms by which information can be transmitted between actors on the Web, but essentially more work is needed regarding how contextual information can be represented, where it can be stored, how and by whom it can be maintained, when it can be shared, what kind of reasoning is possible, and how to ensure the scalability and consistency of service applications.

Throughout this thesis, we discuss dynamic requirements of service processes at process execution time. *We define dynamic requirements as requirements relevant to the composition and execution of services at process run-time.* In SOA (Service Oriented Architecture) applications, requirements validation monitoring at process run-time is demanding. For example, SLA (Service Level Agreement) monitoring at provider end and client end. The service matching and service selection approaches at service composition support design-time validation for service-based applications [Medjahed 2007], but do not appropriately support run-time validation and monitoring. Web services appear from heterogeneous domains and environments. The latest trends in the service computing domain are hosting functionalities, data, software, infrastructure and even hardware as services. There are tens of thousands Web services available over the web, even for a single task there are many services available. The business needs are rapidly changing. The changes to available services are increasing because of technological enhancements and changing needs. The SLAs agreed between providers and clients need to be monitored taking into account the respective execution contexts. There are some aspects such as response time, availability, some business constraints, etc., which can only be measured at run-time and not at design-time. This shows that Web service processes demand dynamic requirements validation and monitoring at run-time.

According to our investigations, dynamic aspects are not appropriately addressed so far in the literature on Web service processes at process run-time. Figure 1.1 illustrates a few dynamic aspects in a simple example. In this example, we assume this process is a Web service process with three services. We also assume that service providers provide services and in return they charge from enterprise clients. In here, the E-client sets two constraints for this particular process instance. They are that the total cost of the process should be less than 0.3 Euro and that the process response time should be less than 2 seconds. The cost

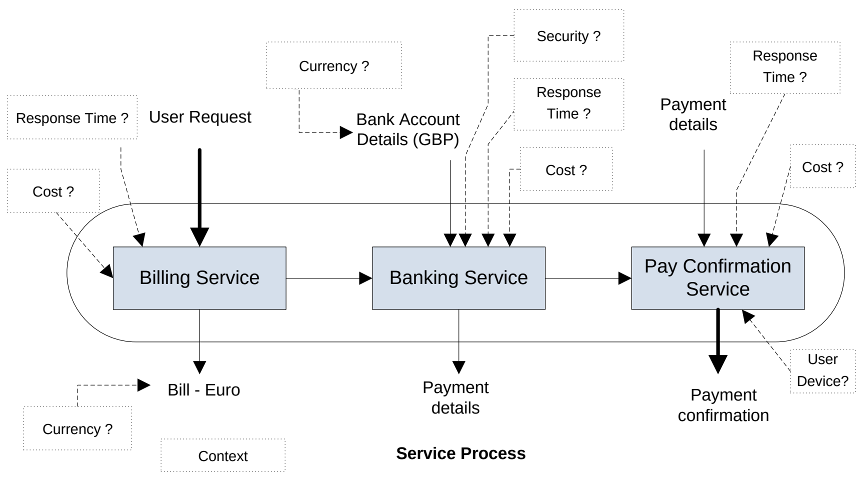


Figure 1.1: An example of dynamic aspects in a service process

of each service varies and the response time of each service can only be measured at process run-time. We can also see a semantic mismatch of output and input parameters of services one and two. If a service fails at run-time, the service needs to be replaced without violating these constraints. Moreover, we may find situations, such as the integrity of a banking service, needs to be captured from the defined security information (security is combination of integrity, authentication, non-repudiation, and confidentiality), payment confirmation needs to be sent to a users' mobile device, which supports MMS messaging, etc. We use the notion of context to define and capture these very different dynamic aspects. These create the following challenges on service composition and execution at process run-time. The dynamic aspects of services and processes can be defined and modeled using context representations. The dynamic requirements can be generated as context constraints. The context specifications attached to service processes can be manipulated, composed and reasoned about to support dynamic requirements. The dynamic requirements validation monitoring at process run-time needs constraints instrumentation and validation monitoring to be applied at process run-time.

1.2 Problem definition

We define objectives of our research based on motivations in section 1.1. Our main focus is firstly to develop a comprehensive context model that conceptualizes services and their dynamic requirements and secondly dynamic instrumentation and validation monitoring of dynamic requirements in a service process at process run-time. The objective of our research is defining an approach for conceptualizing dynamic service context, manipulating and reasoning about context aspects for dynamic requirements, generating context constraints for dynamic requirements, and instrumentation and validation monitoring context constraints at process run-time.
The overall research problem that we attempt to address is two-fold,

1. *What are process execution relevant context aspects (dynamic aspects) of services and processes?*

2. *How to manage process execution relevant aspects attached to a process instance to support monitoring the validation of process execution relevant requirements (dynamic requirements)?*

Managing implies conceptualization and manipulation of process execution relevant aspects, generation of process execution relevant requirements as constraints, and instrumentation and validation monitoring of context constraints.

1.2.1 Central hypothesis

The notion of context is widely used to address dynamic requirements in mobile and pervasive applications. We focus on using the notion of context to address dynamic requirements for Web service applications at run-time [Fitzgerald 2006]. We define our central hypothesis as, *Instrumenting context constraints at process run-time can validate dynamic requirements in Web service business processes.*

1.2.2 Detailed research problems

We expand the main research problem into detailed research problems and define their objectives and boundaries, having the focus on the central hypothesis. We identify four detailed research problems that need to be explored.

- How can we conceptualize (represent) dynamic aspects (contexts) attached to composition and execution time of Web service processes?
  The objective is to identify and model dynamic aspects attached to Web services and processes to provide a shared conceptualization. Dynamic aspects can be attached to functional features of services, quality of service features of services, and the environment in which the services are executed.

- How can we manipulate and reason about context specifications attached to services and processes?
  The objective is to manipulate and reason about context in context specifications attached to services and processes to support dynamic requirements.

- How can we define dynamic requirements as context constraints?
  The objective is to define dynamic requirements as context constraints, which can be instrumented and validated at process run-time.

- How can we instrument a service process with context constraints and perform validation monitoring at process run-time?
  The objective is to instrument and validate dynamic requirements at process run-time. The dynamic requirements can be validated against dynamic service context associated with services and monitored context.

1.3 Contributions

A context-aware approach to model and validate dynamic requirements in a Web service application at run-time is our central point of attention. This research results in four main contributions.

Dynamic service context modeling. A definition for dynamic aspects attached to composition and execution of services at process run-time (called dynamic service context) is needed. Moreover, a comprehensive classification of dynamic aspects and their formalisation in a proper conceptual model is necessary to enrich service-based applications with shared conceptualization and reasoning about dynamic aspects at process run-time. Dynamic service context is defined to represent composition and execution aspects of services at service process run-time. We define a comprehensive dynamic service context classification and its formalisation (formal description) in an OWL-based context model ontology (processable form), which provides shared conceptualization and reasoning facilities for service-based applications.

Context manipulation and reasoning. Some context aspects in a context specification need to be combined, renamed, restricted, and refined as independent aspects for dynamic requirements. Some context aspects in context specifications of a service process need to be composed on a service-by-service basis. These also include some situations that are difficult to address using available OWL-DL operators and some context specifications that need to be adapted at process run-time. The context reasoning can use the underlying description logic (DL) of the OWL-based context model ontology for context derivation and consistency checking. An operator calculus, which needs to be supported by context reasoning techniques is suitable for addressing context manipulation and composition. An operator calculus and reasoning techniques are proposed to manipulate, compose and reason about dynamic service context attached to services and processes for dynamic requirements.

Context constraints. Dynamic requirements need to be generated in terms of dynamic service context in the context model ontology to enrich them with the special featues provided by the context model ontology, such as shared conceptualization, reasoning capabilities, etc. We generate dynamic requirements as *context constraints* in terms of dynamic service context aspects in the context model ontology. The *context constraints* are context-based restrictions. The context constraints can be defined within the context model ontology, but that approach has major drawbacks, such as the complete ontology needs to be reasoned about even for a simple context validation, a simple change in a dynamic requirement requires changes in the context model ontology, etc. These drawbacks affect the flexibility and performance of the service process. Therefore, we contribute two types of context constraints, explicit context validation constraints (ECVCs use ontology-based context specifications) and implicit context validation constraints (ICVCs focus on context derivation and consistency checking using the context model ontology). These context constraints are separated from the context model ontology.

Instrumentation and validation monitoring at run-time. Dynamic requirements attached to composition and execution of services at process run-time need to be validated. The proposed architecture and components enable instrumentation and validation monitoring of ECVCs and ICVCs, without re-deploying the process. The performance of the instrumentation and validation monitoring needs to be acceptable. The *instrumentation* implies weaving context constraints, constraint services and data collectors to a deployed service process.

In summary, *contextualization and managing context constraints* in our approach implies:

- Conceptualization of dynamic service context (ontology-based context modeling)
- Context manipulation, composition, and reasoning about context specifica-

tions for dynamic requirements

- Context constraints generation for dynamic requirements
- Constraints instrumentation and validation monitoring at process run-time

Managing context constraints can result in service replacement or dynamic re-composition, which is beyond our scope.

1.4 Organization of the thesis

The organization of this thesis is described in figure 1.2.

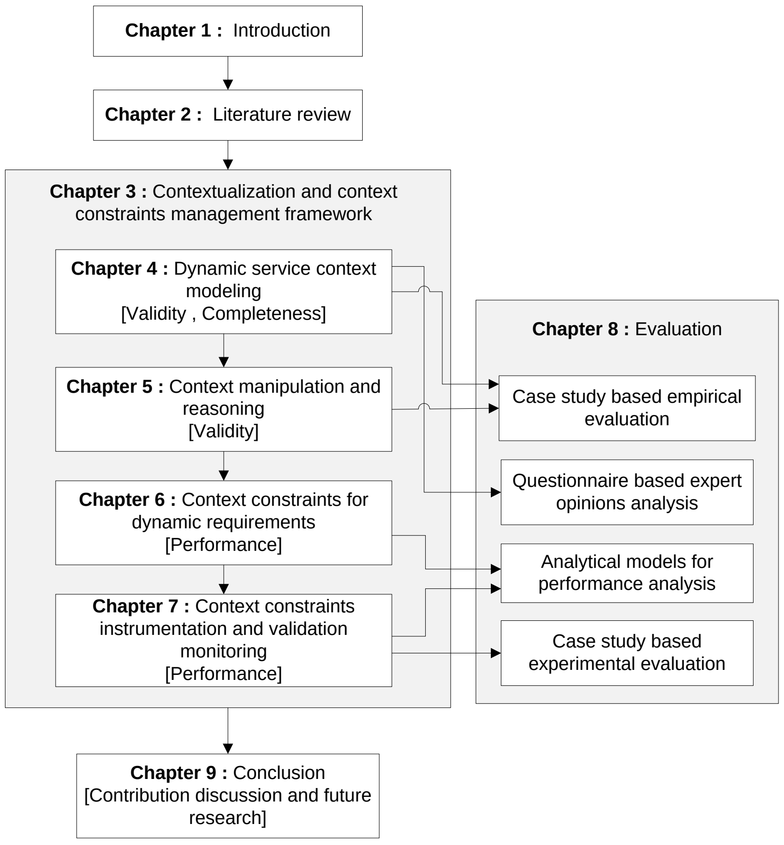

Chapter 1 : Introduction
Chapter 2 : Literature review
Chapter 3 : Contextualization and context constraints management framework
Chapter 4 : Dynamic service context modeling [Validity , Completeness]
Chapter 5 : Context manipulation and reasoning [Validity]
Chapter 6 : Context constraints for dynamic requirements [Performance]
Chapter 7 : Context constraints instrumentation and validation monitoring [Performance]
Chapter 8 : Evaluation
Case study based empirical evaluation
Questionnaire based expert opinions analysis
Analytical models for performance analysis
Case study based experimental evaluation
Chapter 9 : Conclusion [Contribution discussion and future research]

Figure 1.2: The organization of this thesis

Chapter 1 presents the motivation, research problems, and contributions.

Chapter 2 presents the literature review within the scope of the thesis and discusses the gaps and challenges, which we address in this thesis.

Chapter 3 presents the overall architecture and research approach to address the main challenges identified in chapter 2.

Chapter 4 presents a classification of dynamic service context with the objective of creating a more complete taxonomy and formalisation of context into a context model ontology.

Chapter 5 presents context manipulation, composition, and reasoning. New operators are introduced for manipulation and composition.

Chapter 6 presents context constraints modeling for dynamic requirements. A new architecture and an algorithm for context constraints generation are introduced.

Chapter 7 describes context constraints instrumentation and validation monitoring at process run-time. A new architecture and algorithms for constraints instrumentation and validation monitoring are introduced.

Chapter 8 presents the evaluation approaches and evaluation results of main contributions.

Chapter 9 summarises the contributions presented in this thesis and provides an overview of future work and research directions.

Chapter 2

Literature review

Contents

2.1 Introduction

This section analyses the body of work related to context-aware Web services, context modeling and manipulation, context constraints generation, and constraints instrumentation and validation monitoring in Web service business processes. The analysis focuses on identifying possible gaps and the challenges in the literature.

Web services can be classified into three generations [Fitzgerald 2006]. First-generation Web services are described using functional properties from the service process design perspective. Second-generation Web services are described using functional and quality of service properties from the service process design perspective. Dynamic service composition, which is service composition at process run-time [Agarwal 2008, Dustdar 2008] is becoming a vital aspect for service process design in emerging service computing applications. Web service applications need changes at process run-time; therefore, process design-time perspectives are not enough and process run-time perspectives are necessary. The requirements can be frequently changed at service process run-time (dynamic requirements) due to various reasons, such as dynamic user needs, various quality of service features, a service update, service execution middleware update, server updates, various networks, devices, etc. Third-generation Web services need to be described using functional properties, quality of service properties, and environmental aspects on service execution from the perspective of service process run-time. Our particular concern is the composition and execution aspect of Web services for third-generation Web service applications.

A notion of context is used in Web service applications, such as business applications, mobile applications, ubiquitous applications, etc. In section 2.2, we analyse the notion of context of Web services in the literature to identify gaps in defining context towards composition and execution aspects of services and processes at run-time. In other words, we need to define dynamic service context from the process run-time perspective. Context modeling, manipulation, and reasoning are explored in service-based applications, such as mobile applications and ubiquitous applications to address dynamic aspects. We carry out a detailed analysis of the previous work related to context modeling, manipulation, and reasoning in section 2.3. In 2.4, we analyse in detail the previous work related to constraints generation, instrumentation, and validation monitoring towards dynamic service applications. The gaps in the state of the art along with useful approaches and technologies are described. Finally, at the end of the chapter in section 2.5, we identify some weaknesses and limitations of the state of the art, which defines where this work will build on. Each piece of related research is considered with respect to its relevance and significance.

2.2 Notion of context for Web services

The notion of context has been modeled and exploited in many areas of informatics since the early 1960s [Coutaz 2005]. The notion of context has been defined by the scientific community as having application domain perspectives without reaching a clear consensus. However, it is commonly agreed that context is about evolving, structured, and shared information spaces, which are designed to serve a particular purpose [Coutaz 2005]. In this section, we explore the notion of context defined in the state of the art associated with Web service domains in section 2.2.1, and context categorizations in section 2.2.2 examining gaps related to dynamic aspects attached to the composition and execution of services.

2.2.1 Context definition

We explore the context definitions used in previous work related to context-aware service applications. Perhaps the most highly cited work on context definition is [Dey 2000]. They define context as "any information that can be used to characterize the situation of an entity, in which an entity can be a person, place, physical or computational object that is considered relevant to the interaction between an entity and an application, including the application and the user themselves". Service composition should not be confused with functional and non-functional properties [O'Sullivan 2002]. In [Medjahed 2007], context is any information that can be used by a Web service to interact with clients and clients to interact with Web services. Web services that use context, whether available to clients or providers are called context-aware. In [Chen 2006], the authors define the term context as having a service requester perspective and a service perspective. In the service requester perspective, context is defined as the surrounding environment affecting the requester's Web service discovery and access, such as the requester's profiles and preferences, network channels and devices that the requesters are using to connect to the web, etc. In the services' perspective, context is defined as the surrounding environment affecting Web services delivery and execution, such as service profiles, networks and protocols for service binding, devices and platforms for the service execution, etc. In [Maamar 2006], authors define two abstract types of context, C-context for the context of a composite service, and W-context for the context of a single Web service. According to them, the context of Web services has fine-grained content, whereas the context of composite services has coarse-grained content. Their C-context permits monitoring the deployment progress of a composite service specification from a temporal perspective (such as, what occurred, what is occurring, and what might be occurring with the component Web services), and the W-context caters for participating service compositions. The authors in [Rosemann 2008] point out that another area for investigating the notion of context in process modeling is the requirements engineering discipline. They introduce the notion of an immediate context of a business process, which includes elements that go beyond the constructs that constitute the pure control flow, and covers the elements that directly facilitate the execution of a process. They suggest extending this view towards a wider perspective on the environment of a business process. In [Doulkeridis 2006], context is defined as the implicit information related to both the requesting user and the service provider that can affect the usefulness of the returned results, having the focus on service discovery for pervasive computing.

In particular, the notion of context can be used to define dynamic aspects of Web service processes. The available context definitions of Web services do not sufficiently describe dynamic aspects associated with the composition and execution of Web services at process run-time such as response time, dynamic business features, dynamic execution environments (different middleware), etc.

2.2.2 Context categorization

Context categorization in the general Web services domain. A Web service description is only complete once the non-functional aspects are also expressed [O'Sullivan 2002, O'Sullivan 2006]. The non-functional properties of services include availability, channels, charging styles, settlement models, settlement contracts, payment, service quality, security, trust, and ownership. The service matching framework proposed by [Medjahed 2007] is a promising work for Web service composition in the context-aware Web services domain. This framework combines the concepts Web service, policy-based context modeling, and policy-based service matching focusing on service composition at process design phase. Their context categorization is detailed, but the categorization has inconsistencies. For example, most of their value-added contexts are quality of service contexts and the service execution perspective is not sufficiently described, such as environmental context where services are executed, performance of services, availability of services, etc. A general structure of a Web services community proposed in [Medjahed 2005] enables clustering Web services based on their domain of interest. This classification is more detailed, compared to the classification in [Medjahed 2007], particularly the quality of operation in [Medjahed 2005] is comprehensive in addressing quality of service properties. However, some properties need to be added, for example the trust property in [Hasselbring 2006]. Moreover, their dynamic semantics can also provide some inputs for defining dynamic properties of Web services.

The context of Web services is categorized as provider-related context, customer-related context and collaboration-related context, which uses the concept of Web service community and a goal template to facilitate service selection [Boukadi 2009, Boukadi 2008]. The customer-related context is a set of available information as well as meta-data used by service providers to adapt their services, for example, customer profile, environmental context, customer preferences, etc. The provider-related contexts are conditions under which providers can offer their Web services for external use, for example, service capability, history related to previous events and situations, quality of service, etc. The collaboration-related context referring to the general context in which the enterprise collaboration will be created and changed over time, for example, data, location, business domain, etc. They consider only explicit contexts, which are explicitly defined by service providers and customers. The implicit contexts are out of scope in their work. The goal templates in [Boukadi 2009, Boukadi 2008] contain both functional and contextual parameters used to select services. Their focus is on static context and service selection for Web services composition at process design-time.

How context permits determining the semantics of Web service interfaces is motivated and addressed by [Mrissa 2006b, Mrissa 2006a, Mrissa 2007, Mrissa 2008]. Their focus is on data heterogeneities that arise when Web services from different origins take part in a composition. They propose annotating Web service descriptions with data semantics. This annotation is integrated into a semantic mediation architecture that handles information heterogeneities using semantic objects and Web service contexts. Context is proposed to integrate with WSDL by amending the WSDL message part through adding new context attributes. Their context model revolves around the notion of a semantic object. A semantic object has a data part and semantic part. A data part has a value v of type t described in a type system language, a semantic part has a concept c of the application domain, and a context C is represented as a tree of meta-attributes referred to as modifiers. These modifiers make explicit the semantic properties of semantic objects. However, their work is more about context integration with the focus on mediation towards semantic Web service composition and does not address context in detail. However, this idea emphasises the need for context descriptions for service interfaces. Web services can originate from various providers, and that leads to context heterogeneity for service composition [Sattanathan 2006]. The focus in [Sattanathan 2006] is on context reconciliation among different Web services while defining Web service context as a set of common meta-data about the current execution status of a Web service and its capability of collaborating with peers. Like in [Maamar 2006, Mrissa 2006b, Mrissa 2006a, Mrissa 2007, Mrissa 2008], [Sattanathan 2006] addresses context at a very abstract level.

A comprehensive context model of context classifications for business processes is proposed in [Rosemann 2008, Rosemann 2006]. Their concern is exploring extrinsic drivers for process flexibility, which is an important requirement in modern business processes. The extrinsic drivers for flexibility are studied in the context of a process, and they argue that a stronger and more explicit consideration of contextual factors can make the process more adaptive. They discuss why context matters and how context can be conceptualized, classified, and integrated with existing approaches to business process modeling. They focus on a goal-oriented process modeling approach to identify relevant context elements and propose a framework and a meta-model for classifying relevant context. They propose a stratified layer framework by incorporating and differentiating four types of context. They are the immediate, internal, external, and environmental context. The immediate context of a business process describes elements that go beyond the constructs that constitute the pure control flow and covers those elements that directly facilitate the execution of a process. However, their classification is more conceptual. They emphasise that the concept of immediate context is prevalent in process modeling and suggest to extend this view in a wider perspective for an environment of a business process.

Context categorization in pervasive computing domain. The notion of context is used to define dynamic aspects in pervasive computing applications. The notion of context is explored for physical and conceptual objects including person, activity, computational entity, and location in pervasive computing applications

[Wang 2004, Roy 2010]. The authors in [Fujii 2009] propose a semantics-based context-aware dynamic service composition framework, which allows users to request applications in an intuitive manner (e.g. user request in a natural language). Applications adapt to different users by utilizing user context information when composing applications, and adapt an application for dynamic environments by autonomously composing a new application and migrating to it when the user context changes. They use user context information such as location, time, and history to achieve adaptability at dynamic service composition. These are application dependent dynamic aspects for pervasive applications. A set of elements, user context, computing context, time context, physical context, and context history, which make up a full definition of context [Mostéfaoui 2003]. The user context focusses on user role, identity, age, location, preferences, social situation, etc. The computing context focuses on network connectivity and nearby resources such as printers, displays, etc. The time context focuses on time of day, week, month, seasons of the year, etc., The physical context focuses on weather, temperature, etc. The context history focuses on recording the above context across a time span. These context types focus on service discovery and composition in pervasive environments. The focus on pervasive applications is providing the right information to the right users, at the right time, in the right place, and on the right device [Bikakis 2008]. Therefore, it is necessary for a system to have thorough knowledge about people and devices, their interests and capabilities, and tasks and activities being undertaken. All this information falls under the notion of context in [Bikakis 2008]. A number of observations about the nature of context information in pervasive computing systems are context information has a range of temporal characteristics, context information is imperfect, context has many alternative representations and context information is highly interrelated [Henricksen 2002].

The challenges of large-scale ubiquitous computing can be tackled with a structured and flexible approach to context [Coutaz 2005]. For them, the context is not simply the state of a predefined environment, which has a fixed set of interaction resources. It is part of a process of interacting with an ever changing environment composed of reconfigurable, migratory, distributed, and multi-scale resources. In [Lee 2007], the authors define context types for context-aware frameworks in ubiquitous environments. They define four types of definitions - sensed context (context gathered from sensors), combined context (context calculated from sensed context), inferred context (context inferred from sensed context), and learned context (context made from a learning algorithm). Two kinds of context are identified as service context and user context in [Doulkeridis 2006] for context-aware service discovery in mobile services domain. The service context can be the location of a service, its version, the provider's identity, type of the returned results, and its cost of use. The user context defines the user's current situation, such as location, time, temporal constraints, as well as device capabilities, and user preferences. Their concern is matching user context against service context in order to retrieve relevant services with respect to context-awareness.

Context categorization in e-learning domain domain. The context, adaptivity, context-aware systems, and ontologies are explored in e-learning applications. The authors in [Soylu 2009b, Soylu 2009a] use the constructed theory and practices in context-aware literature to enable anywhere and anytime adaptive e-learning environments. They propose eight categories for context-aware settings where they need to achieve an optimal granularity and need to represent main actors of a typical pervasive computing setting. They also emphasise that this categorization provides an initial step towards a generic conceptualization. Their context categories include user context (internal, external), device context (hard, soft), application context, information context, environmental context (physical, digital), time context, historical context, and relational context.

In contrast to conventional applications (validated before deployment), service-centric applications can heavily change at run-time, for example, they can bind to different services according to the context in which they are executed or providers can modify internals of their services [Baresi 2005]. A more extensive context classification has to be done having the perspective of service process run-time. The literature does not appropriately discuss a comprehensive categorization of dynamic aspects attached to the composition and execution of services at process run-time.

2.3 Context modeling and manipulation

In Web service business processes, notion of context is mostly described for service matching, selection, and mediation applications focusing on service composition at process design-time. We review context definitions for service applications in section 2.2 and Web service context categorization in section 2.2.2. In this section, we further explore context modeling in section 2.3.1 and context manipulation and reasoning in section 2.3.2 to identify gaps in the literature.

2.3.1 Context modeling

The pervasive computing community increasingly understands that the development of context-aware applications should be supported by adequate context information modeling and reasoning techniques to reduce the complexity of context-aware applications [Bettini 2010]. The learning techniques, such as Bayesian networks, fuzzy logic, etc. [Khedr 2004, Ngo 2004], statistical methods [Cakmakci 2002], and ontologies [Wang 2004] can be used to model context information [Soylu 2009b, Soylu 2009a]. In [Strang 2004], the authors survey the most relevant current approaches to model context for ubiquitous computing systems. The approaches are classified relative to their core elements, and evaluated with respect to their appropriateness. The authors in [Strang 2004] classify modeling approaches based on a scheme of data structures, which are used to exchange contextual information in the respective system. Their categorization include, key-value models, markup scheme models, graphical models, object-oriented models, logic-based models, and ontology-based models. Context modeling and implementation use XML, XML based CC/PP [Doulkeridis 2006], UML [Kapitsaki 2009], Topic Maps [Goslar 2004], RDF [Medjahed 2007], and OWL [Wang 2004] in various application domains. The authors in [Ibáñez 2010] describe how an RDF vocabulary is used to model functional and non-functional properties of basic activities of XML-based BPEL processes [Alves 2006]. However, based on a survey on the most important existing context modeling approaches, authors in [Strang 2004] conclude that the most promising approach, which satisfies requirements in ubiquitous computing applications is ontology-based context modeling. [Wang 2004] shows several reasons to use ontologies for context modeling, such as logical inference, knowledge sharing, knowledge reuse, etc. Moreover, authors of [Hervás 2010] have listed significant benefits and functionalities of ontology-based modeling, such as explicit knowledge representation, expressive power, reasoning and inference, detection of inconsistencies, simplifying functional complexities, etc.

There are complete semantic ontology frameworks to model semantic descriptions of Web services, such as WSMO [Lausen 2005], SWSF [Battle 2005], and OWL-S [Martin 2004]. These model semantic Web services in a top-down fashion. They define complete frameworks for describing the semantics of Web services. The authors in [Kopecky 2008] emphasise the importance of ontology-based modeling of semantics on top of WSDL and propose SAWSDL [Farrell 2007] annotations with concrete semantic constructs. In the METEOR-S project [Aggarwal 2004], a semantic Web service designer tool supports the design and development of semantic Web services. The incorporation of semantic descriptions into a service is achieved by means of service source code annotations. The complete description of the semantics of an operation involves the description of inputs, outputs, constraints to be satisfied, exceptions thrown by each operation, and the functional description of the operation. The METEOR-S architecture incorporates data semantics, functional semantics, and quality of service semantics to support activities in the complete Web process life cycle, and also uses WSDL annotations, WSDL-S, and OWL-S for describing semantics.

Ontologies are used to build consensual terminologies for the domain knowledge in a formal way so that they can be more easily shared and reused [Ye 2007]. Ontology is referred to as a shared understanding of some domains, which has a set of entities, relations, functions, axioms and instances [Gu 2004]. The authors in [Gu 2004] describe their reasons for selecting OWL [McGuinness 2004] to realize their context model, which is that OWL is more expressive than RDF [Brickley 2004]. OWL supports semantic interoperability to exchange and share context knowledge between different systems and OWL is a W3C standard. The authors of [Chen 2003, Chen 2004] emphasise the suitability of OWL for building a common knowledge representation for context-aware systems to share and reason with contextual knowledge.

OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with formal semantics [McGuinness 2004]. OWL has three expressive sublanguages. They are OWL-Lite, OWL-DL, and OWL-Full. OWL-Lite supports classification hierarchy and simple constraints. OWL-DL supports maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations finish in finite time). OWL-DL includes all OWL language constructs, but they can be used only under certain restrictions, such as a class cannot be an instance of another class. OWL-Full has maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, a class can be treated simultaneously as a collection of individuals and as an individual on its own. OWL-Full also allows an ontology to augment the meaning of the pre-defined RDF/OWL vocabulary. However, it is unlikely that any reasoning software will be able to support complete reasoning for every feature of OWL-Full [McGuinness 2004].

Description logic is used for the decidable fragment of OWL, called OWL-DL [Baader 2007]. In [Serrano 2007], the focus is on an OWL-based ontology for creation, delivery, and management of context-aware services, and also for the integration of user context information in service management operations for heterogeneous networks. The person, place, task, and object are considered as fundamental contextual data. In [Lee 2007], the focus is on a context ontology for ubiquitous environments using OWL and SWRL. Their context types are sensed context, combined context, inferred context and learned context. [Hong 2008] also proposes a context ontology focused rule-based approach for ubiquitous learning environments. An interesting comparison of available context-aware systems modeling, and reasoning can be found in [Bikakis 2008]. OWL is a common formalism for context representation. A number of query languages (e.g. RDQL [Seaborne 2004], RQL [Karvounarakis 2002], TRIPLE [Sintek 2002]), and reasoning tools (e.g. FaCT++ [Tsarkov 2007], RACER [Haarslev 2001], Pellet [Sirin 2007]) have been developed to support development of complete OWL-based applications. The query languages and reasoning tools can retrieve relevant information, check the consistency of the available data, and derive implicit ontological knowledge. OWL-DL is based on the $\mathcal{SHOIN}(\mathcal{D})$ description logic [Baader 2003]. OWL-DL also has the inference ability [Hong 2008].

The dynamic aspects attached to the composition and execution of services need shared conceptualization and reasoning facilities such as consistency checking, deriving implicit dynamic aspects from explicit dynamic aspects, etc. for addressing dynamic requirements at service process run-time. According to our understanding, the literature does not appropriately discuss a comprehensive conceptual model which addresses dynamic requirements attached to composition and execution of services at process run-time.

2.3.2 Context manipulation and reasoning

The ontology-based context aspects and specifications need to be manipulated, composed and reasoned about to address dynamic requirements. For example, to capture integrity aspect of a service from the given security context. In this section, we review the state of the art for manipulation and composition aspects on specifications and ontology-based reasoning methods.

Context manipulation and composition. The research towards manipulating context specifications in dynamic environments and requirements is not new. Static QoS specifications are valuable, but they have limitations, such as QoS characteristics and requirements change dynamically due to changing user preferences or changes in the environment. They do not allow objects of a system to be aware of the QoS they require or provide [Frølund 1998b]. In contrast, an object, which is QoS-aware is able to communicate and manipulate its QoS information. The authors in [Frølund 1998b] provide a run-time representation of QoS specification for dynamic systems that change and evolve over time. This representation enables systems to create, manipulate, and exchange QoS information, and thereby negotiate and adapt to changing QoS requirements and conditions. The authors introduce a language called QML (QoS Modeling Language)[Frølund 1998a] to capture component-level QoS properties and design a QML-based run-time QoS fabric called QRR (QoS run-time Representation). The QRR allows dynamic creation, manipulation, communication, and comparison of QoS specifications [Frølund 1998b]. The authors in [Cámara 2008] present an approach to the flexible composition of possibly mismatching behavioural interfaces in systems where context information can vary at run-time. This approach simplifies the specification of composition/adaptation by separation of concerns, and is able to handle context-triggered adaptation policies. However, manipulation and composition operators for dynamic aspects and specifications, which can work with the description logic of ontology-based specification are still lacking.

The service profile in OWL-S [Martin 2004] provides the information needed for an agent to discover a service. This representation includes a description of what is accomplished by the service, limitations on service applicability, quality of service, and requirements that the service requester must satisfy to use the service successfully. This approach does not discuss about manipulation and composition of dynamic aspects in ontology-based specifications for dynamic requirements. The authors in [Kaiya 2005] propose a method of requirements analysis by using ontology techniques where they establish a mapping between requirement specifications and ontological elements. Ontology-based manipulation of architecture styles can be found in [Pahl 2009, Pahl 2007]. The authors propose an ontological approach for architectural style modeling based on description logic as an abstract meta-level modeling instrument. They use description logic to define an ontology for the description and development of architecture styles that consist of an ontology to define architecture styles through a type constraint language, and an operator calculus to relate and combine architecture styles. They give an overview of the central operators renaming, restriction, union, intersection and refinement, and define the semantics of an operator calculus. Instead of general ontology mappings, they introduce a notion of a style specification and define style comparison and development operators on it. They also introduce three types of composite elements for architecture style specifications, called structural composition, sequential composition, and behavioural composition. This is interesting work, which can provide some inputs for developing an operator calculus to manipulate and compose ontology-based context aspects and specifications towards dynamic requirements. However, more work is still necessary, such as composing dynamic aspects for various dynamic requirements at process run-time. For example, the security of a process is the weekest security of a constituent service, a language aspect of a process is the language common to all services, etc.

Context reasoning. A key concern in the study of context-aware systems is reasoning about context [Wang 2004, Lee 2007]. The context reasoning can be OWL-DL reasoning or rule-based reasoning [Bikakis 2008]. OWL-DL reasoning takes advantage of the class relationships, property characteristics, and property limitations while rules are used to combine information about various contexts using algebra [Hong 2008, Ranganathan 2002]. Ontology-based models provide logical characterization for the interpretation of objects, classes, and relationships [Pessoa 2007]. This type of modeling permits the specification of a domain to allow semantically consistent inferences and assure shared and reusable representation of context information among context providers, service platforms and applications. This further enables the development of systems that are able to derive implicit information through the analysis of information represented explicitly [Pessoa 2007]. Rule languages provide a formal model for context reasoning. They are easy to understand, widely used, and there are many systems that integrate them with the ontology model [Bikakis 2008]. SWRL is based on a combination of OWL-DL and unary/binary datalog [Boley 2010] sublanguages of the Rule Markup Language [Boley 2010, O'Connor 2005a]. SWRL [Horrocks 2004b] rules (Horn-like rules) reason about OWL individuals, primarily in terms of OWL classes and properties [O'Connor 2005b, O'Connor 2005a]. The rules can be used to infer new knowledge from existing OWL knowledge bases [O'Connor 2005a]. The aim of context reasoning is to deduce new knowledge based on available context data [Bikakis 2008]. The ultimate goal is to make the service more intelligent and closer to the specific needs of their users [Bikakis 2008]. As a reasoning tool, Pellet is the first sound and complete OWL-DL reasoner with extensive support for reasoning with individuals (including nominal support and conjunctive query), user-defined data types, and debugging support for ontologies [Sirin 2007]. Pellet implements several extensions to OWL-DL, such as a combination formalism for OWL-DL ontologies, OWL/Rule hybrid reasoning, etc. It has proven to be a reliable tool for working with OWL-DL ontologies and experimenting with OWL extensions [Sirin 2007]. However, reasoning methods applied on dynamic aspects of service processes for dynamic requirements at process run-time are still lacking.

2.4 Constraints generation and validation monitoring

Globalization and the need for accessibility causes a significant shift in business processes from static solutions to flexible processes that can address rapidly changing business needs [Ardagna 2011]. The industry has a major interest in the QoS of a workflow and workflow systems [Cardoso 2004]. Currently, ad hoc techniques are applied to estimate the QoS of a workflow [Cardoso 2004]. Most of the previous work can be viewed as QoS-based design, QoS-based selection and execution, QoS-based monitoring and QoS-based adaptation [Cardoso 2004]. In SOA, constraint-based service selection has been explored from process design-time perspectives [Alrifai 2009, Boukadi 2008, Aggarwal 2004, Rajasekaran 2004]. The constraints validation that traditionally pertains to development time, now needs to be extended to run-time for addressing emerging challenges in Web service applications [Bianculli 2008], such as SLAs monitoring.

2.4.1 Constraints generation

In this section, we explore the previous work related to constraints generation (requirements or QoS-based constraints) and specification technologies towards service-based applications.

A context constraint specifies that certain context attributes must meet certain conditions to permit a specification operation [Strembeck 2004]. A multilevel context model in [Boukadi 2008] defines how context data can become context assertions, then composite context assertions, and finally context constraints, which can be used in service selection. They define a context assertion as a predicate that checks the validity of a condition and consists of an operator and two operands. Their context constraint consists of context data and one or more context assertions including composite context assertions. Context constraints generation is restricted to context instances and their assertions. The assertions can be simple assertions and composite assertions. This fundamental idea of context constraints is interesting for context ontology integrated service-based applications. The idea behind the service selection approaches in [Boukadi 2008] and [Medjahed 2007] are similar. The constraints representation module in [Channa 2005, Aggarwal 2004] is a set of classes, attributes, instances, and relationships between them that allow users to represent business constraints in ontologies. Their concern is business constraints for selecting Web services for a given process. However, constraints generation and specification need to be more precise than their detailed ontologies to make them more flexible in modifications and validations.

The constraints are successfully applied in numerous domains such as computer graphics, natural language processing, database systems, resource allocation, planning, and scheduling [Karakoc 2009]. In [Karakoc 2009], constraint translator approach converts a work flow of a Web service composition and constraint specifications into a constraint language. Service constraints are constructed from the service templates so that service variables are mapped to constraint variables and a flow model is converted into a set of constraints in the constraint language of the constraint engine [Karakoc 2009]. These constraints are quality conditions that reflect the users requirements on a composite service. The service selection should satisfy these specified constraints. The service process is then deployed and executed. They assume that the constituent services are stable at process run-time. Moreover, this approach does not address process run-time aspects (e.g. service response time, reliability, availability, etc.); therefore this work is less suitable for dynamic service applications.

In [Frølund 1998b], the authors provide a run-time representation of QoS specification, which allows dynamic changes in dynamic systems (systems, which change due to changing user preferences or changes in the environment). These specifications are developed using object-oriented programming constructs; therefore less compatible for service-based applications, which are based on XML-based specifications. Formal methods are also used in some approaches to specify constraints, such as in [Mahbub 2005] where constraints are specified in an event calculus. The UML [Priestley 2003] standard is already equipped with the Object Constraint Language (OCL) [OCL 2010] to express constraints, which cannot be conveyed by the graphical elements in UML. OCL is a declarative language (first order logic language), which can be used to specify functional behaviour of elements in a software system (e.g. invariants on class diagrams) [Cacciagrano 2006].

SoaML integrates modeling capabilities to support SOA [SoaML 2009]. For example, ServiceContract and ServiceInterface elements of the SoaML profile support contract-based and interface-based approaches respectively. USDL is made up of a set of modules including service level agreements and each addresses different aspects of the overall service description [USDL 2011]. These modules have dependencies among each other, as they reuse concepts from other modules.

The XML schema does not allow to express complex constraints and define inter-dependencies. Therefore, it is necessary to have more expressive technologies [Cacciagrano 2006]. The constraints language in XML (CLiX) is gaining increased momentem for constraint specifications [Cacciagrano 2006, Jungo 2007, Nentwich 2002, Dui 2003, Nentwich 2005]. It allows the specification of static and dynamic integrity constraints of Web service parameters by means of logic formulas. The existential and universal quantifiers are used to iterate over sets of nodes and boolean operators that allow to build more complex formulas, which we find suitable for defining dynamic requirements that need to be validated at process run-time. The xlinkit validator engine facilitates validation of CLiX logic formulas [Nentwich 2002].

We observe some challenges not appropriately addressed so far. The constraint representation modules define constraints in ontologies [Aggarwal 2004, Boukadi 2008]. However, defining dynamic requirements as context constraints within the ontology has major drawbacks, such as the complete context ontology is involved in reasoning for a simple context validation (e.g., cost of a service need to be less than 0.2 USD), the complete context ontology is repeatedly involved for each and every context validation, a change in one dynamic requirement needs a change in the ontology, etc. These drawbacks affect the flexibility and performance of the service process. But the complexity of some context constraints, which need context derivation and consistency checking, can be reduced by implementing them using the underlying description logic of the context model ontology. However, context reasoning is more costly than validation against context specifications, which are small in size. Dynamic requirements are specific to application scenarios (e.g. security of a service process attached to one scenario is integrity context of constituent services) and the context model ontology is common to all service processes. Therefore, it is necessary to separate context constraints from the context model ontology. These context constraints need to be supported by the shared conceptualization and context reasoning facilitated by the context model ontology. In our scope, most of the dynamic requirements are not complex and need to bind dynamic aspects with simple logic. However, some complex dynamic requirements can be simplified using the context reasoning capabilities provided by the context model ontology. The state of the art does not sufficiently address dynamic requirements in service processes, which can be supported by dynamic aspects reasoning and shared conceptualization.

2.4.2 Constraints instrumentation and validation monitoring

In this section, we explore previous work related to requirements (or QoS) instrumentation and validation monitoring in service-based applications.

Web service composition lets developers create applications on top of the description, discovery, and communication capabilities of service-oriented computing [Milanovic 2004]. Composition of Web services has received much interest to support business-to-business or enterprise application integration [Srivastava 2003]. XML-based standards have been introduced to formalise the specification of Web services and their flow-composition. WS-BPEL (or BPEL) is a OASIS standard used for specifying business process behaviour based on Web Services [Alves 2006], which exports and imports functionality by using Web Service interfaces exclusively. WS-BPEL defines a model and a grammar for describing the behavior of a business process based on interactions between the process and its partner services [Alves 2006]. Service composition can be static composition or dynamic composition. Combining services for a composite service or process in a pre-determined manner is called static composition. Combining component services in a composite service at process run-time is called dynamic composition [Dustdar 2008, Bastida 2008].

The service selection algorithm in [Yu 2007] focuses on selecting individual services that meet QoS constraints and also provide the best value for the user-defined utility function. The user-defined utility function is defined by a weighted sum of system parameters including system load, cost, and other attributes. This can be seen as constraints validation in service selection at process design-time. The research work on validation at design-time always assumes that the services are error-free at run-time. In the METEOR-S project [Aggarwal 2004], a constraint analyzer deals with constraints. Any constituent set of services of a process, which satisfies the given constraints is a feasible set. Their focus is on using business constraints, technological constraints, and partnerships to select services for a Web service process at process design-time. The METEOR-S allows the process designers to bind Web services to an abstract process. However, the use of SWRL and OWL to provide descriptive rules for specifying constraints is planned, but not properly realized.

In [Mahbub 2005], the authors present a framework to support run-time verification of requirements for service-based systems. Their framework supports the run-time monitoring of behavioural properties of service based systems. The behavioural properties (e.g. a conditional structure of service execution), which are specified using an XML schema that represents formulas of an event calculus (EC) [Shanahan 1999] are extracted from the process specification. Event calculus is a first-order temporal logic language that can be used to specify the events that occur within a system and the effects of these events [Mahbub 2005]. The authors in [Mahbub 2005] classify five types of events attached with behaviour, invocation events (service-level invocation), reply events (service-level reply), return events (process-level return), request event (process-level request), and assignment events. In [Moser 2008b, Moser 2008a], the authors propose a tool for monitoring and dynamic adaptation of BPEL processes. Their monitoring focuses on computing quality of service (QoS) data for various service selection strategies to support dynamic service adaptation. Their service adaptation happens at run-time by dynamically selecting an alternative service for an existing service used in the process, without any changes to the deployed BPEL process. The tool further provides transformation components to compensate for service interface mismatches, which are likely to occur when using alternative services offered by different providers. However, this approach assumes that the quality of service used in the service selection is guaranteed for service executions, which is not the case for dynamic aspects and more exploration is still necessary for validation monitoring of dynamic aspects at process run-time. The authors in [Ehlers 2011] present a framework that allows for autonomic on-demand adaptation of the monitoring coverage at run-time. Their objective is self-adaptive monitoring to investigate performance anomalies in component based software systems. They use goal-oriented monitoring rules specified in OCL. The continuous evaluation of the monitoring rules enables to zoom into the internal realization of a component, if it behaves anomalous. Their monitoring and analysis process contains probe injection, probe activation, data collection, data processing, visualisation and self-adaptation. In order to monitor the internal behaviour of components, probes can be instrumented at various measuring points in the control flow of the components. This is similar to an idea; dynamic instrumentation of constraints as pre / post conditions of services in a service process. However, it goes beyond fixed assignment of monitoring probes.

Run-time monitoring of web service compositions has been widely acknowledged as a significant and challenging problem [Barbon 2006]. [Foster 2006] provides an approach to specify, model, verify and validate the behaviour of web service compositions based on interaction requirements. The compatibility of Web services is important for service compostion. An approach for compatibility analysis for Web services based on sequences of messages, exchanged data flow, constraints attached to data flow and temporal requirements is proposed in [Guermouche 2008]. WS-BPEL does not allow to define temporal constraints in service composition [Kallel 2009]. A formalism to capture the timed behavior of a composite Web services process is proposed in [Kazhamiakin 2006]. An approach for monitoring temporal constraints (such as availability, throughput, reliability, etc.) by translating them into timed automata is proposed in [Raimondi 2008].

Dynamic monitoring using a code weaving approach is proposed in [Baresi 2005]. Their code weaving blends monitoring rules with an abstract service process before the process deployment, assuming all candidate services are stable. However, in this approach, a service failure or monitoring rule failure needs process redeployment. This approach does not sufficiently address late-binding [Canfora 2005], in which services are selected and combined at process run-time. The authors in [Leitner 2010] propose a system that integrates event-based monitoring, prediction of SLA violations using machine learning techniques, and automated run-time prevention of those violations by triggering adaptation actions in service composition. They use predictions of SLA violations, which are generated using regression from monitored run-time data to trigger adaptations in the service composition.

Web service technology aims to reuse distributed functionalities. However, there are challenges to realize its full potential, such as to ensure that the behaviour of Web services is consistent with their requirements [Wang 2009c]. The authors in [Wang 2009c] propose an online monitoring approach for Web service requirements. The monitoring approach covers five kinds of system events relevant to client request, service response, application, resource, and management. Though these five parts are separated, they are closely related, such as lacking resources leads to bad response time, malicious request results in failure of application, etc. Their monitoring model tries to answer "What should be monitored?", while their monitoring framework tries to answer "How to monitor?". The monitoring framework is composed of distributed probes (probes extract events from the target system), an agent (active events collector), a central analyzer (process different kinds of monitored events), and a management center (human-friendly management interface). Their probes are generated from the information contained in the service constraint specification and this approach focuses on the monitoring of individual Web services, including client and operating environment. The monitoring framework begins monitoring a service for specified constraints, once all the monitoring codes are successfully deployed. The instrumentation is the most widely used monitoring mechanism. Traditionally, instrumentation code is inserted manually and in [Wang 2009c] the monitoring code is embedded inside the target code. In recent years, some new instrumentation mechanisms (such as Javaassit and AspectJ) have been developed to instrument a java code automatically according to configuration information [Wang 2009c].

However, previous work does not sufficiently address instrumentation and validation monitoring of dynamic requirements at process run-time, which is a necessary requirement for Web service applications. In particular, the explicit information about dynamically managing requirements instrumentation and validation monitoring in a BPEL process at process run-time is not appropriately addressed.

2.5 Discussion

This discussion focuses on analysing the most important previous work to highlight gaps in the literature. The gaps are discussed and clearly defined at the end of the discussion.

There is no specific definition for context in information science. The notion of context has been defined in various application domains from more general to more specific terms [Dey 2000, Coutaz 2005, Maamar 2006, Medjahed 2007, Fujii 2009, Rosemann 2008, Hong 2008, Lee 2007, Boukadi 2008]. The notion of context has been widely used in mobile [Sheshagiri 2004] and ubiquitous [Lee 2007, Hong 2008] applications to define spatial and temporal aspects. Recently, the notion of context has been explored in service matching [Medjahed 2007] and service selection [Boukadi 2008] research in the Web services domain. However, these context definitions are not complete in addressing the notion of context of third-generation Web services. In particular, to address service composition and execution aspects attached to dynamic requirements at process run-time. In other words, the available context definitions do not focus on dynamic service context. A proper definition to define dynamic service context is still lacking.

According to our investigations, a context classification to address dynamic aspects of Web services is needed. The authors in [Rosemann 2008] propose a conceptual context model for context classification for business processes in general and state the importance of further research on business process execution aspects, called immediate context. A more detailed solution related to context and context-aware Web services for Web service process domain is proposed in [Medjahed 2007]. This is context-based service selection for service composition at process design-time. Their context categorization is detailed, but it is about static context and does not sufficiently address dynamic requirements based aspects such as run-time context, execution environment context, etc. Their previous work in [Medjahed 2005] presents Web service clusters in a detailed classification. They detail static semantics, dynamic semantics, and also quality of operations. However, their focus is on semantic clusters for services, which reinforces the concept of a service registry. Dynamic service context is increasingly demanding and dynamic requirements validation at service process run-time is a challenging direction. However, a comprehensive solution towards this direction is still lacking.

There is some work on context models associated with Web service applications [Chen 2004, Wang 2004, Lee 2007, Chen 2006]. Most of these context models are domain-specific and application-dependent. A more general context model to address dynamic aspects associated with service composition and execution from run-time perspective is needed. In this thesis, we focus on process run-time aspects. Consequently, a context model ontology to formalise dynamic service context is a challenge. It is necessary to have a shared conceptualization of dynamic aspects attached to composition and execution of services, having a particular focus on process run-time aspects. Context modeling uses XML, XML-based CC/PP [Doulkeridis 2006], UML [Kapitsaki 2009], Topic Maps [Goslar 2004], RDF [Medjahed 2007], and OWL [Wang 2004] in various application domains. However, ontology-based context modeling is the most promising approach for Web service applications [Strang 2004, Bikakis 2008, Hervás 2010]

A run-time representation of QoS specification for dynamic systems is proposed in [Frølund 1998b]. A mapping between a requirement specification and ontological elements is proposed in [Kaiya 2005] for requirement analysis. The authors in [Pahl 2009, Pahl 2007] propose an approach for manipulating architectural style modeling based on description logic as an abstract meta-level modeling instrument. Context-aware systems support context reasoning, such as OWL-DL reasoning and rule-based reasoning [Wang 2004, Lee 2007, Bikakis 2008]. In Web service applications, context aspects in context specifications need to be combined, renamed, restricted and refined as independent aspects for dynamic requirements. For example, some context aspects in context specifications of a service process need to be composed on a service-by-service basis. These include some functionalities, which are difficult to address using the available OWL-DL operators, such as the security of a process is the weakest security of all individual services, some context specifications need to be adapted at process run-time, etc. These aspects are not sufficiently addressed in the literature. An operator calculus is a promising solution to address context manipulation and composition. An operator calculus needs to be supported by context reasoning. For example, operators can make context specifications inconsistent.

There is a significant shift in business processes from static solutions to flexible processes that can address rapidly changing business needs [Ardagna 2011]. So far in the literature of ontology-based specifications, constraints are defined within the ontology [Channa 2005, Aggarwal 2004]. However, defining dynamic requirements as context constraints within the ontology has major drawbacks, which can affect the flexibility and performance of the service process. The context reasoning is more costly than validation against context specifications. However, some context constraints, which need context derivation and consistency checking need context reasoning. A standard constraint specification language can be adapted for defining context constraints. Constraint language in XML (CLiX) is gaining increased momentum in defining constraints [Cacciagrano 2006, Jungo 2007, Nentwich 2002, Dui 2003, Nentwich 2005] for rapidly changing business needs. Context constraints need to be supported by shared conceptualization and context reasoning.

Conventional applications are usually validated before deployment and testing is the usual means to discover failures before release. In contrast, service-based applications can heavily change at run-time and testing activities cannot foresee all these changes [Baresi 2005]. Therefore, shifting validation to run-time and introducing the idea of validation monitoring is important. In their work, monitoring rules are blended with an abstract service process. Service composition is separated from rule integration, which is of our interest. However, they instrument the abstract service process before deployment. In [Baresi 2005], the authors assume that all services in an abstract service process are stable, but if a service fails at run-time, they need to redeploy the whole process. If a monitoring rule fails, then the process designer needs to change priorities or redeploy the whole process. Therefore, this approach is not sufficient for dynamic requirements instrumentation and validation monitoring. In [Wang 2009c], the monitoring framework begins to monitor a service against specified constraints, once all the monitoring codes are deployed successfully. However, the service process needs to be redeployed for a requirement change (to instrument a new requirement) at process run-time. This approach monitors individual Web services for attached requirements, but process level requirements are missing. The proposed approach in [Mahbub 2005] discusses run-time verification of behavioural properties, while the approach in [Moser 2008b] addresses dynamic behaviour based on monitored QoS data attached to various service selection strategies. Both approaches address the behaviour of service processes, such as verifying behavioural requirements of a service process, selecting behaviour based on monitored QoS data, etc. However, both approaches do not sufficiently address dynamic instrumentation of requirements. Although requirements monitoring at process run-time is addressed in the state of the art [Baresi 2005, Wang 2009c, Mahbub 2005, Moser 2008b], addressing run-time instrumentation of requirements for run-time validation monitoring, which we consider necessary for Web service applications is still lacking.

We identify gaps in the literature and define challenges as follows:

- The available context definitions are not rich enough to precisely address service composition and execution related aspects at process run-time. A new context definition for dynamic service context is required.
- The available context categorizations and modelings do not sufficiently describe and conceptualize dynamic service context. A complete context model ontology to conceptualize dynamic service context is needed.
- The manipulation and reasoning about dynamic service context specifications at service process run-time is necessary for dynamic requirements. This is not sufficiently addressed in the literature.
- The dynamic requirements can be defined as context constraints, and need to be supported by shared conceptualization and context reasoning features of the ontology. This is not sufficiently addressed in the literature.
- The available constraint instrumentation and validation monitoring approaches do not sufficiently address run-time instrumentation and validation monitoring of dynamic requirements.

We analyse some closely related work on context operationalization in service processes at process design-time and process run-time as in figure 2.1 to locate our work. The notion of context attached to Web service processes and its operationalization to address dynamic requirements in a service process at process run-time is necessary. The illustration mainly focuses on context operationalization and service composition. The related work on context operationalization is illustrated from domain specific to more general Web service processes. The context exploration from abstract to more detail is considered for both domain specific and

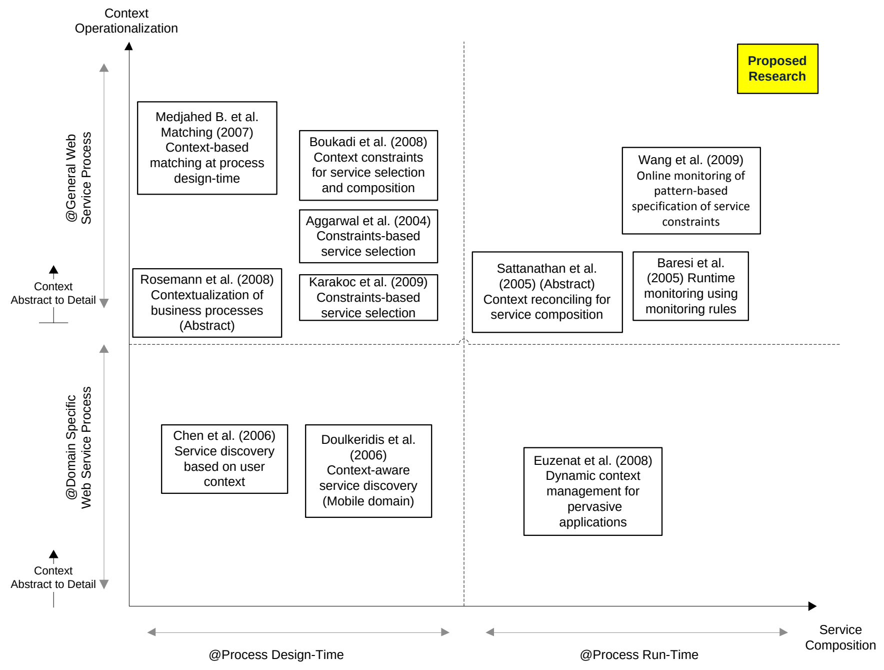
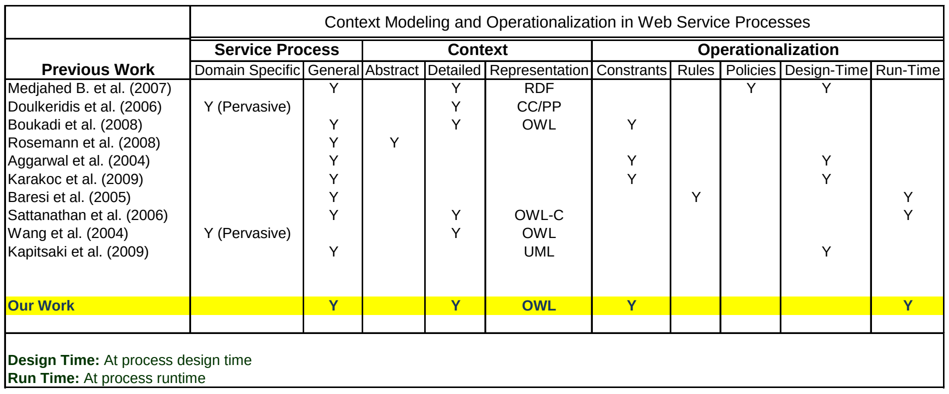

Context Operationalization
Proposed Research
@General Web Service Process
Context Abstract to Detail
@Domain Specific Web Service Process
Context Abstract to Detail
Medjahed B. et al. Matching (2007) Context-based matching at process design-time
Boukadi et al. (2008) Context constraints for service selection and composition
Aggarwal et al. (2004) Constraints-based service selection
Rosemann et al. (2008) Contextualization of business processes (Abstract)
Karakoc et al. (2009) Constraints-based service selection
Sattanathan et al. (2005) (Abstract) Context reconciling for service composition
Wang et al. (2009) Online monitoring of pattern-based specification of service constraints
Baresi et al. (2005) Runtime monitoring using monitoring rules
Chen et al. (2006) Service discovery based on user context
Doulkeridis et al. (2006) Context-aware service discovery (Mobile domain)
Euzenat et al. (2008) Dynamic context management for pervasive applications
@Process Design-Time
@Process Run-Time
Service Composition

Figure 2.1: Related research review

more general Web service processes. The related work on service composition is illustrated from process design-time to run-time. Finally, the proposed work is located as in figure 2.1.

A selected set of previous work related to context operationalization for service processes is illustrated in figure 2.2. We review the state of the art in three



	Context Modeling and Operationalization in Web Service Processes									
	Service Process		**Context**			**Operationalization**				
Previous Work	Domain Specific	General	Abstract	Detailed	Representation	Constrants	Rules	Policies	Design-Time	Run-Time
Medjahed B. et al. (2007)		Y		Y	RDF			Y	Y	
Doulkeridis et al. (2006)	Y (Pervasive)			Y	CC/PP					
Boukadi et al. (2008)		Y		Y	OWL	Y				
Rosemann et al. (2008)		Y	Y							
Aggarwal et al. (2004)		Y				Y			Y	
Karakoc et al. (2009)		Y				Y			Y	
Baresi et al. (2005)		Y					Y			Y
Sattanathan et al. (2006)		Y		Y	OWL-C					Y
Wang et al. (2004)	Y (Pervasive)			Y	OWL					
Kapitsaki et al. (2009)		Y			UML				Y	
Our Work		**Y**		**Y**	**OWL**	**Y**				**Y**

Design Time: At process design time
Run Time: At process runtime

Figure 2.2: Context operationalization review

perspectives, which are service process, notion of context and operationalization. The service processes are considered from general and domain specific perspectives. The notion of context is reviewed from abstract, detailed and representation language perspectives. The context operationalization in a service process is considered from constraints, rules, policies, design-time and run-time perspectives. Our focus is highlighted as illustrated in figure 2.2. The blank spaces represent either not applicable or undiscussed.

Chapter 3

Contextualization and context constraints management framework

Contents

3.1 Motivation

The overall objective is ontology-based contextualization and context constraints management for Web service processes to support dynamic requirements validation monitoring at process run-time.

Web services are independent components that can be composed and invoked in order to satisfy business needs. A Web service is a self-contained, self-describing, modular application that can be published, located, and invoked across the Web [Rao 2004]. The number of services available for a single task is increasing. Available services are frequently updated. Web services provided by different providers bring heterogeneous dynamic features with them, such as security, cost, availability, platform features, etc. Moreover, business needs and client (or user) needs are rapidly changing, service providers and service users may need to monitor their SLAs [Ehlers 2011]; therefore, a Web service process needs to be flexibly related to its dynamic aspects to monitor the changing nature of business processes. Monitoring SLAs in a service process is important for service users and service providers, figure 3.1. Moreover, in multi-tenant service applications

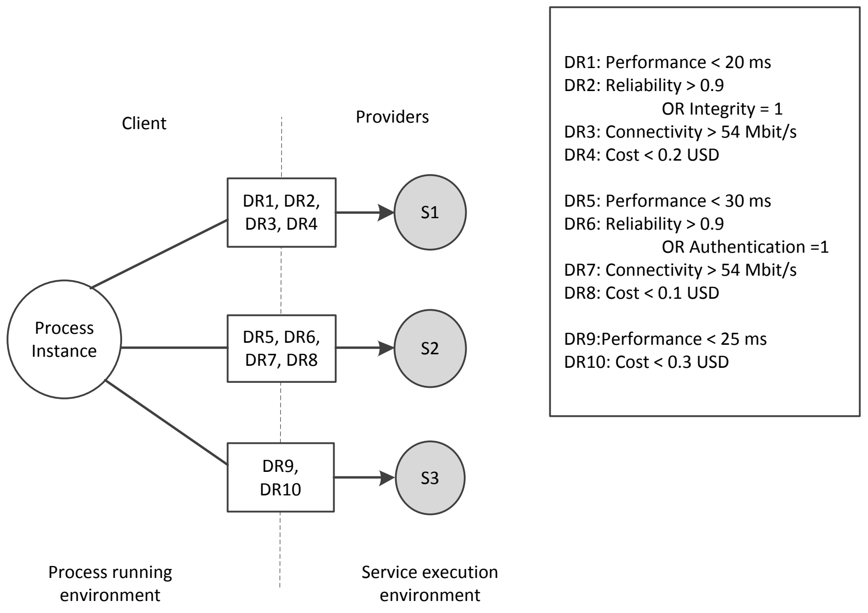


Figure 3.1: SLAs attached to different services with dynamic requirements (DR)

[Aulbach 2008], one process instance can be used for various clients who are attached to different SLAs, consequently constraint sets generation and instrumentation is required to support validation monitoring of SLAs at process run-time. SLAs address aspects that we capture in our context model later on.

Service-based applications are developed by combining heterogeneous services to perform business tasks. The service composition can be a static composition or a dynamic composition. For static composition, services to be composed are decided at process design time. For dynamic service composition, services to be composed are decided at process run-time [Dustdar 2008]. The pre-deployment validations, such as service matching and service selection at process design time are inadequate for tackling rapidly changing aspects in which service-oriented applications are deployed [Baresi 2005, Zhao 2007]. The monitoring of functional and non-functional behaviour of service applications needs to be shifted towards run-time [Baresi 2005]. Services are provided by heterogeneous platforms and environments that can result in service failures, SLAs may not be met as agreed, constituent services can be updated and better services can be made available. These are a few examples to show the importance of run-time validation monitoring of dynamic requirements.

At the present time, business and user needs are becoming more and more dynamic and new dynamic business applications are emerging. Service providers provide services of various qualities. For example, providing software services is itself a business and each service invocation is charged by the provider, hardware is provided as services, and software is provided as services. With these emerging trends, monitoring *dynamic requirements attached to composition and execution of services at process run-time* is one of the emerging needs, which is not adequately addressed so far in the literature. We explore these emerging needs and introduce a contextualization of dynamic requirements and a context constraints management framework to support validation monitoring of dynamic requirements at process run-time.

3.2 Contextualization and context constraints management framework

This section defines a framework in figure 3.2 that integrates different services involved in the contextualization and context constraints management problem.

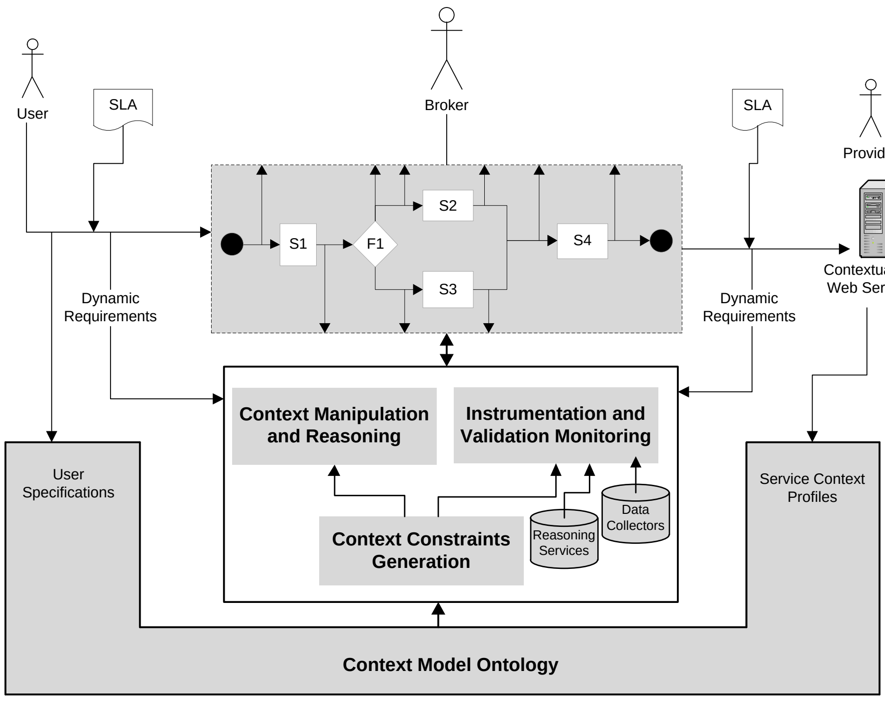


Figure 3.2: Contextualization and context constraints management framework

In this framework, the contextualized Web services are provided by service providers. A context model ontology to conceptualize context aspects is described in chapter 4. The service context profiles, which semantically enhance WSDL to define provider specifications about services, chapter 5. In the proposed approach, dynamic requirements are captured from SLAs (service level agreements [Verma 2004]) and generated as context constraints in terms of context aspects in the context model ontology, see figure 6.1 in chapter 6. A contextualized Web service process at the broker needs to be instrumented with context constraints for validation monitoring of dynamic requirements at run-time (e.g., some dynamic requirements in figure 1.1), see figure 7.1 in chapter 7. The data collectors and reasoning services, which support context constraints validation monitoring are implemented and maintained at the broker, chapter 7. An instrumented service process instance can have services (e.g., S1, S2, S3, and S4) and constraints, including process behaviour constraints (e.g., F1). The overall framework consists of four main activities, which address challenges identified in section 2.5.

- Conceptualize dynamic service context in a context model ontology
- Manipulate and reason about ontology-based context specifications
- Dynamic requirements are generated as context constraints
- Constraints instrumentation and validation monitoring

Our main concerns within the framework are defined by the highlighted process components. The subsequent methodologies in addressing and evaluating the above four activities are described in each sub-section. These sub-sections are further detailed in chapters 4, 5, 6 and 7, respectively. We assume, dynamic requirements can be captured from service level agreements, but discussing service level agreements is beyond our focus.

3.2.1 Dynamic service context modeling

Dynamic service context is defined to represent composition and execution aspects of services at service process run-time. A comprehensive dynamic service context classification, and it's formalisation (formal description) in an OWL-based context model ontology (processable form) is our primary concern. This context model ontology provides shared conceptualization and reasoning features for Web service applications. The relationships of context categories are defined based on description logic [Baader 2003].

We investigate and organise dynamic aspects attached to composition and execution of services at process run-time. The empirical analysis of different case study scenarios is used in the investigation [Pahl 2008]. The available classifications such as [Rosemann 2008], [Medjahed 2007], [Medjahed 2005], etc. provide some input. The orientation of dynamic aspects in a more general classification is identified as an important concern. We observe dynamic aspects relevant to service interfaces, quality of service properties and process execution environments are three general perspectives. The orientations provided by literature (for example, [Chung 2009], [Medjahed 2005], [Rosemann 2008], etc.) also provide some input.

A properly formalised context model is defined using comprehensive context categories and their relationships. The dynamic service context has taxonomic relationships and non-taxonomic relationships, which we define in a context model ontology (for example, trust is a combination of technical assertions and relationship-based factors [Hasselbring 2006], so that trust can be a combination of security and reputation context). The shared conceptualisation of dynamic service context is needed. The ontology web language (OWL) is used as the formalisation language, which has relevant merits [ODM 2009], such as tractability, interoperability, etc. We used Protégé as a tool, which is a free, open-source platform that provides a growing user community with a suite of tools to construct domain models and knowledge-based applications with ontologies [1].

We evaluate context categories to check whether they represent dynamic requirements (validity), and they cover all necessary dynamic requirements (completeness). These aspects are evaluated based on empirical case study scenarios and expert's opinions through a survey.

3.2.2 Manipulation and reasoning context specifications

Context aspects and specifications attached to services (service context profiles) and processes need to be manipulated, composed, and reasoned out in service composition and execution at process run-time. The idea of relating and manipulating architecture descriptions and composite elements in architecture de-

[1] http://protege.stanford.edu/overview/

scriptions proposed in [Pahl 2009] is relevant work and we extend this work for ontology based context specifications to address dynamic requirements.

Some context aspects in a context specification need to be combined, renamed, restricted and refined as independent aspects for dynamic requirements. Some context aspects in context specifications of a service process need to be composed on a service-by-service basis. These include some situations (e.g., the security of a process is the weakest security of all individual services), which are difficult to address using the available OWL-DL operators and some context specifications that need to be adapted at process run-time are proposed as manipulation and composition operators. An operator calculus is a suitable way for addressing context manipulation and composition. An operator calculus needs to be supported by context reasoning. The OWL-based context ontology is based on description logic, which facilitates context reasoning [Wang 2004, Horrocks 2003]. The context reasoning has different functionalities such as subsumption, consistency checking, and context derivation. Context reasoning can be adapted to reason about dynamic aspects in context specifications using the context model ontology.

The context manipulation and composition operators and context reasoning are evaluated for validity (validity means that context manipulation operators and context compositions can process contexts for dynamic requirements) using the empirical analysis of application scenarios.

3.2.3 Context constraints generation

Dynamic requirements in composition and execution of services at process runtime need to be defined as context constraints. The context constraints can use the features of the context model ontology, such as shared conceptualization, reasoning, etc., see section 2.3.2. Our context constraint generation combines logical relations and dynamic aspects (context) to define context constraints. Consequently, a constraint generation process and its components need to be defined.

The constraint representation modules define constraints within ontologies [Aggarwal 2004, Boukadi 2008]. However, defining dynamic requirements (DRs) as context constraints within the ontology has major drawbacks, which we discuss in detail in section 2.4.1. In our context constraint generation process, the context constraints are separated from the context model ontology. However, the features of the context model ontology are used for explicit context validation constraints (ECVCs) and implicit context validation constraints (ICVCs). The ECVCs (e.g., use shared conceptualization) define dynamic requirements attached to ontology-based context specifications of constituent services and processes. An ICVC define a dynamic requirement, which needs to derive implicit contexts from the explicit contexts and validate them using the context model ontology. A standard constraint specification language can be adapted for defining ECVCs. This approach reduces the above mentioned drawbacks and provides flexibility for instrumentation and validation. However, defining ICVCs using reasoning features of the context model ontology is more practical than a seperate constraint specification language.

We propose the constraints language in XML (CLiX)[2] [3] as a possible specification language [Nentwich 2005] for ECVCs. CLiX is a constraint language that combines first order logic with XPath [4], which is a W3C standard. However, this approach is not rich enough to use the reasoning features of a context model ontology, which is necessary to validate implicit contexts in ICVCs. Therefore; we use OWL API to implement reasoning services for ICVCs. More details about these languages and tools are described in chapter 6. The performance of ECVCs generation is important for a practical solution. In this approach, we propose an ECVC generation process and an algorithm to define the generation process. The time complexity of the ECVC generation algorithm is evaluated analytically.

[2]http://clixml.sourceforge.net/
[3]http://code.google.com/p/openclixml/
[4]http://www.w3.org/TR/xpath/

3.2.4 Constraints instrumentation and validation monitoring

A Web service application needs constraints instrumentation to enable dynamic requirements validation monitoring at process run-time. We shift the constraint instrumentation and validation monitoring to process run-time.

The service-based systems can consistently change at run-time [Baresi 2010b]. For example, service applications can bind to different services according to the context they are executed in, providers can modify their services, and these aspects can hamper the correctness and quality levels of these service applications. The pre-deployment validations, such as service matching and service selection at process design-time are inadequate for tackling changing aspects in which service-oriented applications are deployed [Baresi 2005, Zhao 2007, Baresi 2010b]. Web service technology aims to reuse distributed functionalities. However, there are challenges to realize its full potential, such as to ensure that the behavior of Web services is consistent with their requirements [Wang 2009c]. However, these approaches do not suffciently address dynamic instrumentation of requirements with a service process and their validation monitoring at process run-time. In Web service applications, it is necessary dynamic requirements are added to service processes for validation monitoring at process run-time (section 3.1). Consequently, an architecture and methodologies are required. We generate context constraints based on dynamic requirements and propose architecture and components to instrument a deployed service process for validation monitoring of context constraints at process run-time. The proposed architecture enables instrumentation and validation monitoring of ECVCs and ICVCs without re-deploying the process. We use WS-BPEL [Alves 2006] as the service execution engine and xlinkit validator [5] as a validator engine for CLiX-based constraints. We implement this archiecture and components using Java as the programming language. More information about these engines, languages and tools can be found in chapter 7.

[5] http://www.messageautomation.com/

The context model (section 3.2.1), its application in the form of service context profiles and the operators on context specifications (section 3.2.2), and their integration with dynamic requirements through context constraints (section 3.2.3) need to be supported at process run-time focusing on context constraints validation monitoring. The context constraints generation, instrumentation, and validation monitoring at the broker in figure 3.2 is detailed in figure 3.3.

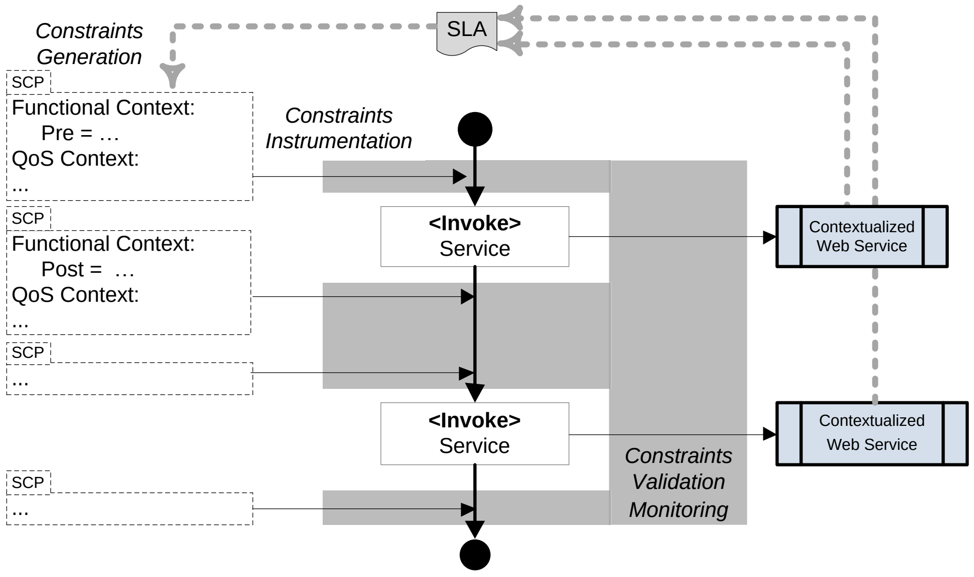


Figure 3.3: Context model utilisation overview

Service context profiles on the server-side define the provider specifications, which are updated dynamically. The server- and client-side specifications define SLAs captured in terms of a context model. The client-side specifications specify the minimum requirements, whereas the provider-side specifications specify the maximum range of capabilities. Dynamically, the client requirements are validated against profile or monitored data. Corresponding constraints are generated.

An architecture for dynamic instrumentation of context constraints and context constraints validation monitoring at process run-time is proposed, chapter 7. The instrumentation process is defined using algorithms and algorithms are analytically evaluated for performance. The instrumentation and validation monitoring are implemented and also evaluated for performance using case study-based controlled experiments.

3.3 Chapter summary

This chapter describes the overview of contextualization and context constraints management in a service process for validation monitoring of dynamic requirements at process run-time. We propose a contextualization and context constraints management framework. The framework has four main activities. The context modeling process focuses on conceptualization of dynamic service context. The other activities of the framework demonstrate the implementation of the formalised conceptualization and its benefits. The context manipulation and reasoning activity focuses on manipulation, composition and reasoning about context specifications and monitored contexts of a process instance to support dynamic requirements. We generate context constraints for dynamic requirements. The instrumentation and validation monitoring activity focuses on instrumentation of context constraints at process run-time for validation monitoring of dynamic requirements. While most context aspects are oriented towards services, our framework demonstrates the need to look at these from the perspective of process run-time. We detail context modeling in chapter 4, context manipulation and reasoning in chapter 5, context constraints generation in chapter 6, and context constraints instrumentation and validation monitoring in chapter 7.

Chapter 4

Context modeling

Contents

4.1 Introduction

Service-oriented computing has the capacity to address the dynamic nature of business applications. The notion of context is becoming a central element for addressing dynamic aspects, particularly in pervasive and mobile application domains [Roy 2010, Wang 2004], which cover external (environmental) aspects of services such as location, time, client's requirements, etc. Recently, the notion of context has been explored in service matching and service selection research focusing on service composition at process design-time [Medjahed 2007].

We extend these perspectives for Web service business applications, focusing on a context-aware approach to address dynamic aspects in Web service processes. We particularly focus on dynamic requirements in composition and execution of services at service process run-time. Consequently, a definition for dynamic service context is needed (the challenge identified in section 2.5). The explicit formalisation of dynamic aspects relevant to composition, deployment, and execution of Web services at process run-time in distributed and heterogeneous environments is still lacking, which we refer to as contextualization (the challenge identified in section 2.5). The contextualization output, a context model ontology provides a reusable set of context categories and relationships for context-aware service-based applications. In the context model ontology, context categories are fomalised as concepts (classes in OWL terminology), basic relationships are formalised as roles (properties in OWL terminology), and the complex relationships are formalised in rules. We use the term instances for individuals because in our work, most of the individuals are instances.

The proposed dynamic service context model addresses runtime aspects of services and processes from inward and outward perspectives (section 4.2.4) whereas service description models such as USDL [USDL 2011], WSMO [Lausen 2005] and OWL-S [Martin 2004] describe functional and non-functional service properties from only one perspective; what a service provides to its environment but not how an environment impacts on a service or process. OWL-S and USDL are complementary [Kona 2009]. OWL-S strength lies in describing the structure of services in a formal way, while USDL is good for describing services in various perspectives. USDL describes a service in terms of port type and message. USDL is made up of a set of modules: legal, pricing, participants, service level, technical, functional and interaction [USDL 2011]. The semantics are given in terms of how a service affects the external world [Kona 2009]. However, how the external world affects a service and process is missing. Integrity of a service / process can be defined in numerous ways, such as transaction integrity (executing transactions consistently with the ability to recover as required), interaction integrity (providing users with up-to-date, secure access to information and content) and information integrity (ensuring reliable, complete and manageable information). Trust of a service / process can be defined in numerous ways in different applications, such as WS-Trust (defines a trust model for Web services), a trust vector in [Park 2005] includes direct trust, authenticity and reputation relationships, etc. The direct trust is trust that is obtained by entities from direct interaction [Li 2009]. It is not the intention of our framework to provide a complete description of all these definitions and relationships, which are application dependent and practically impossible to formalise within the scope of our work. Our dynamic service context model provides a comprehensive set of high-level terms for dynamic service context, a framework and technologies, which can be used by software architects to address runtime aspects of a service process. For example, monitoring SLA violations at process runtime.

We discuss the contextualization of Web service processes as a major goal in this chapter. The technical contribution of this chapter is a semantic model of context (context model ontology) that formalises dynamic aspects of Web services and processes, facilitates reasoning, and supports shared conceptualization. In section 4.2, we describe the context model taxonomy exploring dynamic service context, taxonomy development methodology, context taxonomy definition, and non-taxonomic relationships. In section 4.3, we describe context modeling including ontology-based context modeling principles, description logic - $\mathcal{SHOIN}(\mathcal{D})$, and ontology-based dynamic service context formalisation in detail. In section 4.4, we describe a case study to illustrate the context model ontology integration with a Web service and technological tools. Finally, we conclude the chapter in section

4.5.

4.2 Context model taxonomy

4.2.1 Overview

The notion of context is extensively investigated in mobile and pervasive applications to define locative and temporal aspects in dynamic applications [Hong 2008, Lee 2007, Sheshagiri 2004]. CONON [Wang 2004] and SOUPA [Harry 2004] are widely used context models in pervasive computing environments. They have the fundamental context classifications, such as device, location, person, and activity for capturing information about the execution situation. While these context models do not characterise dynamic aspects of Web services as software entities embedded into business processes, their formal context representation, knowledge sharing, and reasoning aspects provide some inputs for our research. The notion of context and classifications are used to define functional and non-functional features of Web services [Medjahed 2007, Medjahed 2005] focusing on context matching for service selection, but only statically for the process design stage. The authors in [Rosemann 2008] address context in business processes in general and propose a conceptual context taxonomy. They state the importance of further research on process execution aspects, called immediate context [Rosemann 2008].

The previous work on pervasive and ubiquitous applications uses context ontologies, which are tightly coupled with individual applications [Euzenat 2008]. In their work, a context ontology is a part of application-dependent middleware and service compositions are tightly coupled with applications. We focus on a general context-aware middleware for dynamic service composition applications, where services are combined at process run-time. The requirements attached to composition and execution of services at process run-time are the main concern. Compared to previous work, the proposed context model is not tightly coupled with individual Web service applications. Our approach is a general context model that addresses domain specific aspects as well. The service providers can use the context model for developing context-aware services, which can also be organized in service communities proposed in [Medjahed 2007, Mrissa 2008].

4.2.2 Dynamic service context

A notion of context is a major challenge, requiring greater expressiveness, reasoning capabilities, and architectural components than are provided by the current widely accepted building blocks of the Web service stack. However, there is no widely accepted definition for context in information science. Context is defined and used in various applications with their own perspectives [Martin 2006, Euzenat 2008, Dey 2000], such as to define locative and temporal aspects in mobile [Sheshagiri 2004] and ubiquitous [Hong 2008, Lee 2007] applications. The service composition can be static or dynamic composition. In static composition, services to be composed are selected at process design-time. In dynamic service composition, services to be composed are selected at process run-time [Dustdar 2008]. In previous works on service composition, context has been explored for various service discovery and selection phases at process design-time [Medjahed 2007]. However, there is still a gap where dynamic service context and context operationalization are needed at service process run-time in order to validate dynamic requirements. Therefore, the notion of context needs to be rich enough to illustrate dynamic aspects relevant to composition and execution aspects of Web services and processes. In order to fulfill the above mentioned needs, we define dynamic service context.

Dynamic service context is client, provider, or service related information, which enables or enhances effective composition and collaboration between them.

Consequently, the explicit formalisation of dynamic aspects relevant to composition and execution of Web services into a processable model (context model) is the central objective, which we address in this chapter. The context manipulation and operationalization to address dynamic requirements is needed, which is our secondary objective that we address in chapters 5, 6 and 7.

4.2.3 Taxonomy development methodology

A more general complete context taxonomy is important for the development of context-aware Web service applications. First, we discuss the taxonomy development methods, which includes empirical experiments before detailing context model ontology development.

These context aspects need to cover **effective composition and collaboration**. The **effective composition and collaboration** involves service composition and collaboration at Web service process run-time. Therefore, dynamic aspects relevant to composition and execution phases of Web services need to be considered. Here are some motivating examples for identifying dynamic aspects of dynamic service context,

- Service response time is a constraint - the exact service response time can not be pre-defined. It changes over executions. Thus, service response time is a dynamic aspect, which is needed for effective composition and collaboration at process run-time.

- Cost of a service process is a constraint - cost of constituent services and processes can be changed based on currency exchange rates. Thus, cost of a service is a dynamic aspect, which supports effective composition and collaboration at process run-time.

- A service can be executed on selected devices - service execution can depend on device features; thus, device context is needed for effective composition and collaboration of Web services.

- A service needs high bandwidth - A service needs high bandwidth for its execution; thus, connectivity context is needed for effective composition and collaboration of such services.

These are all requirements, but requirements related to the service execution contexts. For example, functionality and quality of service provided at provider-end, and the platform and domain aspects (environmental aspects at execution), such as execution engine, network / platform services, domain ontologies, etc.

Our context taxonomy development methodology has two steps where we define all the possible context categories and organise context categories in a more general taxonomy. **Step 1** involves two parts. They are analysis of context classifications in literature and analysis of real world application scenarios for capturing dynamic service context, which we define in section 4.2.2. In **Step 2**, we describe context orientation in which first we use two general perspectives and then detail the orientation of context categories based on various criteria.

Step 1. This step involves two parts where we focus on capturing all the possible context categories relevant to dynamic service context:

- In the first part, we consider domain specific context taxonomies, comprehensive business services, and process context models, particularly described in [Rosemann 2008, Medjahed 2005, Medjahed 2007, Heravizadeh 2008, O'Sullivan 2002, Roy 2010, Wang 2004]. We capture dynamic aspects, having the perspective of Web services in general focusing on service composition and execution at service process run-time. Most of the previous work is domain specific, such as [Wang 2004], [Roy 2010], [Chen 2004], etc. However, the community structure proposed in [Medjahed 2005] is more general than other approaches and we adopt some aspects such as run-time attributes, business attributes, and security attributes from it.

- In the second part, we use an empirical analysis of application scenarios in the classical business domain. Scenarios from up to date commercial applications with different system architectures are considered. We explore dynamic aspects of constituent services relevant to application scenarios focusing on service composition and execution at service process run-time. The respective results are detailed in Appendix A.7.

Step 2. The orientation of context attributes in a general context taxonomy is an important contribution to the literature towards context-aware Web services.

- We analyse the identified context categories in terms of **inward** and **outward** perspectives on Web services. In the inward perspective, dynamic aspects relevant to service interfaces and quality of service properties are explored. In the outward perspective, dynamic aspects relevant to process execution environment are explored.

- We further classify context categories and sub categories having different criteria until the taxonomy becomes a general classification. This detailed classification is also supported by the literature related to various non-functional and context classifications, such as [Chung 2009], [Medjahed 2005], [Medjahed 2007], [Wang 2004], etc.

Step 1 and Step 2 are iteratively followed until the context taxonomy becomes stable.

4.2.4 Context model taxonomy definition

Our context taxonomy focuses on dynamic aspects to manage dynamic requirements in Web service processes at process run-time. This context taxonomy is located in the immediate layer of the abstract context taxonomy in [Rosemann 2008], i.e. has a technical focus. This taxonomy forms the core of our context model ontology that enhances the taxonomy by further specifications and reasoning features [Bandara 2009, Pahl 2010].

We define inward and outward perspectives in section 4.2.3 (Step 2) focusing on organising dynamic service context in a general context taxonomy.

- The inward perspective describes how a service execution interacts with its environment. The *functional context* captures the functional capabilities from an input / output and pre-condition / post-condition perspective and the *quality of service context* captures non-functional aspects of services.
- The outward perspective describes how the client (or user) and deployment environment impact on service execution. The *domain context* captures dynamic requirements stemming from the application domain of the service and the *platform context* captures dynamic requirements stemming from the technological environment of the service.

As defined in section 4.2.3 (Step 2), the detailed classification of context categories can be exemplified as follows. The quality of service classification towards behavior, dependency, and determination criteria can be described as in table 4.1. However, this classification does not distinguish QoS properties of services towards major demands in the industry today. For example,

Categories	Behavior [Static / Dynamic]	Dependency [Primary / Dependent]	Determination
Input	Static	Primary	Extracted from specifications
Performance	Dynamic	Primary	Execution monitoring
Reliability	Dynamic	Primary	Execution monitoring
Availability	Dynamic	Primary	Execution monitoring
Cost	Static	Primary	Extracted from specifications
Reputation	Dynamic	Primary	Extracted from specifications
Regulatory	Static	Primary	Extracted from specifications
Security	Dynamic	Dependent	Execution monitoring
Trust	Dynamic	Dependent	Execution monitoring
Connectivity	Dynamic	Dependent	Execution monitoring

Table 4.1: QoS classification for behavior, dependency and collection

- Services with high performance are important for communication processes
- Services with less cost are important for cost-effective processes
- Services with high security are important in payment processes

- Services with high trust are important for on-line buying process

Having various classifications, we observe four main categories of quality of service properties: run-time context, financial and business context, security context, and trust context as demanding a general direction of classification. The quality of service properties are a subset of non-functional properties.

We define a comprehensive context model classification with four core dimensions (functional context, quality of service context, domain context, platform context) and detail each dimension with sub-categories of context aspects. These *four core dimensions* represent fundamental dimensions of dynamic service context relevant to Web service compositions and executions. The comprehensive context model classification needs to be completed, which we evaluate in chapter 8. However, this classification can be expanded without breaking its generality so that it provides a flexible context infrastructure for further improving service-based architectures and applications in the future. The context categories in the context model taxonomy have hierarchical relationships. The ontology-based representation of the context model taxonomy reinforces the expressiveness of context relationships further including non-taxonomic relationships and context reasoning. This is meant to be a flexible and evolvable context model that provides a vocabulary of context aspects, their properties, and their explicit and implicit relationships. We define context model as a DL specification, $Context\ Model = \langle \Sigma, \Phi \rangle$ with

- a signature $\Sigma = \langle C, R \rangle$ consisting of **concepts** C and **roles** R to define context aspects and their attributes (section 4.3.3).
- **context descriptions** $\phi \in \Phi$ based on Σ. $\Phi = \langle C \leftrightarrow R \rangle$ defines properties in terms of concepts and roles as DL formulas.

The mechanisms for evolution and other forms of modifying context descriptions are an important part of the overall framework, which we detail in chapter 5. The context model taxonomy is described in figure 4.1.

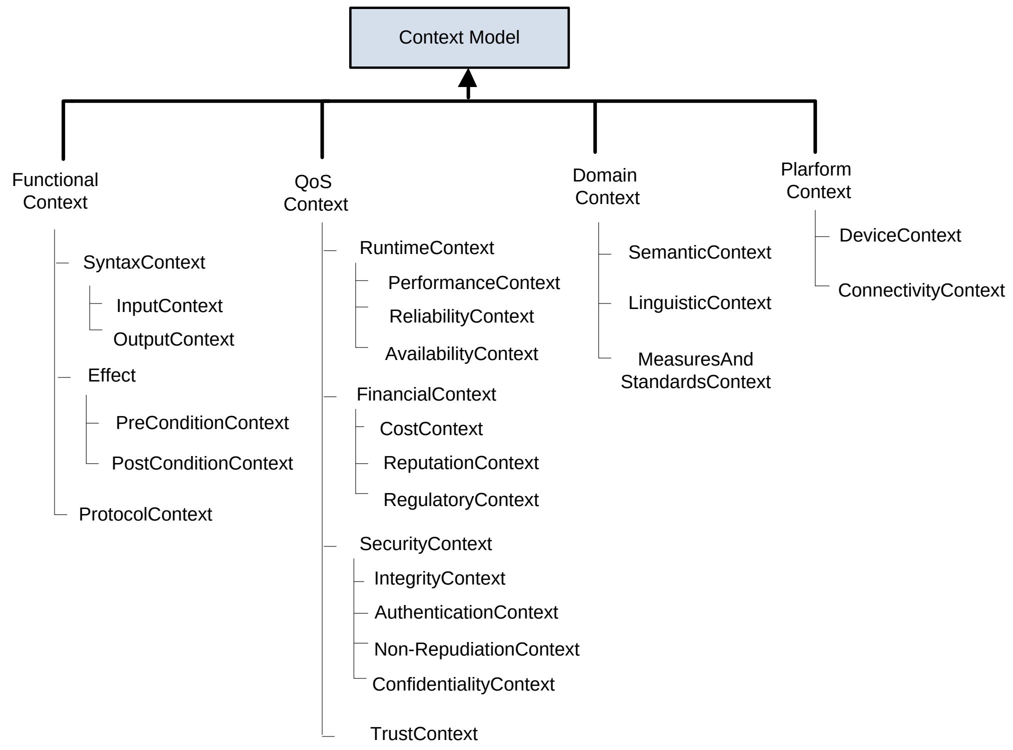


Figure 4.1: Taxonomy of the context model ontology

Functional context (FunctionalContext)

The functionality of a service is defined in the service interface. At the moment, service interfaces are implemented in the Web service description language (WSDL), which is the W3C standard. The context of functionality, which we call FunctionalContext is still missing in parts of the service interface. FunctionalContext describes the operational features of services and their operations. However, FunctionalContext does not work alone in context-aware services and links to non-functional context categories, such as quality of service context - section 4.2.4, domain context - section 4.2.4, and platform context - section 4.2.4 in our context model.

SyntaxContext. Syntax refers to the ways symbols can be combined to create well-formed sentences (or programs, models) in a language. Syntax defines the formal relations between the constituents of a language, thereby providing a structural description of the various expressions that make up legal strings in the language [Slonneger 1994]. The inputs and outputs are major constituents of a program or model in general. Inputs to outputs of a program need to be defined using the syntax of a language (programming or modeling language). The inputs and outputs are major constituents of Web service operations. The inputs and outputs of operations, and their basic data type descriptions are defined in the service interface. The service interface does not describe semantics of input/output parameters, which is a massive limitation in service composition. The semantic descriptions of input and output parameters are needed. Hence, we focus on defining semantics of input and output parameters using the notion of context. We define SyntaxContext as a list of input/output parameters that define messages of operations and the semantics of these parameters for invoking a service.

EffectContext. The service operations can have some conditions to be satisfied before and after operation execution. The justification of the need for addressing pre/post-conditions is described relevant to Semantic Web Services Framework (SWSF) in [Farrell 2007] and ubiquitous computing environments in [Urbieta 2008]. The features of services are usually described in a syntactic way based on their inputs and outputs, but global operational conditions are missing. For example, a service composition/execution can be affected by factors in the execution environment and some quality of service features. Therefore, the functionalities offered by services cannot be adequately used by users, clients or devices that populate the environment [Urbieta 2008]. Some conditions can be mandatory and some of them can be optional. The pre/post-conditions do not describe facts local to the service operation, but global to the process operation. We define EffectContext as pre-conditions and post-conditions of service operations, i.e. the operational effect of an operation execution. The EffectContext needs to be attached to the service interface to reinforce the service composition.

ProtocolContext. The functionalities offered by services and their dynamic conditions are not properly described so far [Urbieta 2008]. The conditions relevant to a service operation are described in the EffectContext, but their protocol of execution can be changed dynamically. We focus on describing the protocol of executing pre/post-conditions relevant to a service operation using the notion of context. For example, the composition structure of pre/post-conditions and context of their data flow can be defined in ProtocolContext. For simplicity, we define ProtocolContext as [status - $cs1, cs2, ..., cs_n$] where status can be "pre" or "post" attached to pre-conditions and post-conditions of a service, also $cs1, cs2, ..., cs_n$ are pre/post condition constraints in their validation order.

Other concerns. In context-aware services, we ignore data types of parameters, such as integer, float, boolean, etc. and assume all of them are of the type string to reduce the complexity. We define each input/output parameter with two fields - data and context.

We have observed dependencies between functional context categories and other context categories in the context taxonomy, which is discussed in detail with examples below. For example,

- A financial service may have an input parameter *transactionAmount* that has

$$transactionAmount : data = 8000$$
$$transactionAmount : context = USD : CurrencyContext$$

- A pervasive service may have an input parameter *distance* that has

$$distance : data = 200$$
$$distance : context = yards : MeasuresContext$$

- A language translation service may have an input parameter *message* that has

$$message : data = \text{"}Bill\text{-}Bezahlung\text{"}$$

$$message : context = German : LanguageContext$$

Each parameter of a context-aware service has both data and context information. One can argue that the parameter *transactionAmount* of the financial service can be of data type currency, but the currency instance (e.g. USD) is still missing. Similarly, distance can be argued to be of type float, but the measurement unit is missing. In context-aware services, we use both data and context of each parameter, which resolves this limitation.

The pre- and post-conditions of a service can be used to validate dynamic requirements relevant to a service,

- Service impact validation, for example, security of a service needs to be validated before invoking a service, while response time of a service needs to be validated after invoking a service.

- Environmental impact validation, for example, DeviceContext needs to be validated before invoking a service.

The ProtocolContext of a service operation can represent the control structure of pre/post-conditions. For example, the ProtocolContext relevant to a subset of pre-conditions of a constituent service operation, such as

- user name, address, and account number verification are needed in that order before invoking a banking service operation.

Quality of service context (QoSContext)

The number of service providers and number of services available for a single task are increasing. The difference among services with equal functionality is the different level of quality of service.

RuntimeContext:

The RuntimeContext is relevant for the measurement of properties that are related to the execution of a service. The RuntimeContext has some common features such as dynamic behavior, primary dependency, and collection in execution monitoring, with relation to the previous classification in table 4.1. Moreover, the quality criteria for elementary services proposed in [Zeng 2004, Medjahed 2005], and the ISO9126 standard [9126-1 2001] are useful in detailing RuntimeContext categories.

PerformanceContext. The measurement of the time behavior of services in terms of response time, throughput, etc. is called performance. The performance of a service is the expected response time in milliseconds between the time a request is sent to a service and the time the results are received. That is, performance of a service is the sum of service execution time and message transmission time. The service execution time depends on the complexity of the operational task and hardware/software facilities available at the service provider end. The message transmission time depends on the network bandwidth available at the time of service request. Therefore, the performance of a service is a dynamic aspect, which changes at each operation execution. If the performance context is response time then $PerformanceContext(S_i, OP_j) > TExecution(S_i, OP_j)$ where $TExecution(S_i, OP_j)$ is execution time of operation j in service i.

ReliabilityContext. The ability of a service to be executed within the maximum expected time frame is defined in ReliabilityContext. We define reliability of a service S as the probability that a request is successfully responded to within the maximum expected time frame defined in the Web service description. The reliability is a measure relevant to hardware and/or software features of Web services, and strength of network connections between the requester and provider. We define the value of ReliabilityContext based on the past invocations using the expression $ReliabilityContext = \frac{N_S}{M}$; where N_S is the number of times that the service S has been successfully delivered within the maximum expected time frame, and M is the total number of invocations of service S (1 is perfect, closer to 0 is less reliable).

AvailabilityContext. The probability that a service is accessible is defined in the AvailabilityContext. We define availability of a service S as the probability that the service is accessible. The service providers define the availability of a service as a percentage based on their criteria. For example, $AvailabilityContext(S^j)$ is $UpTime^{S^j} / TotalTime^{S^j}$ where $UpTime^{S^j}$ is the length of time that the service is accessible, and $TotalTime^{S^j}$ is the length of time that the service is available.

FinancialContext:

These contexts allow the assessment of a service from a financial and business perspective. We observe cost, reputation, and regulatory factors as key important attributes of a service.

CostContext. The amount of money required for provision and execution. The CostContext of a service is the fee that a client has to pay for invoking that service. Given an operation $\mathtt{op}$ of a service $\mathtt{s}$, the CostContext of an operation can be defined as $Cost(\mathtt{s}, \mathtt{op})$. At the moment, service providers advertise prices of service operations or provide facilities for potential requesters to inquire, as a separate task, which is not integrated with the service descriptions. Cost can be defined in various currencies, hence selecting the best service needs a currency conversion service. Moreover, we can observe that the taxonomic relationships are not sufficient to define relationships between CostContext and CurrencyContext.

ReputationContext. This measures the service's trustworthiness. The reputation of a service depends on end users' experiences using the service. Different end users can have different values for one service. The value of reputation is the average ranking given to a service by the end users [Zeng 2004]. The ReputationContext can be defined as, $\frac{\sum_{i=1}^{n} R_i}{n}$, where R_i is the end users' ranking of a service's reputation and n is the number of times a service has been graded. The end users are given a range of values by the service providers so that they can select a ranking. For example, on amazon.com the given range is [0,5] and the given positive feedback percentage on ebay.co.uk. The ReputationContext of a service needs to be dynamically updated and easily accessible through the service description.

RegulatoryComplianceContext. This is a measure of how well a service is aligned with government, organizational, and international regulations. The regulatory compliance could be a value within a range, such as between 1 and 10. The lowest value refers to an operation, which is highly compliant with regulations. Unlike reputation, regulatory compliance is not frequently changing, but can be changed from time to time. If there is more than one service available for a single task, then it is important to define and access regulatory compliance information of a service at service composition and execution of service-based applications development.

SecurityContext:

The focus is on data protection for Web service messages at XML message level. SecurityContext states whether a Web service is compliant with security requirements. Security requirements can be of the types integrity, authentication, non-repudiation and confidentiality, which we call context categories of SecurityContext. For example, a banking service needs authentication checks. Security enforcement meta-data can be defined in four context categories. These context categories describe how a service operation is compliant with security requirements. Service providers collect, store, process, and share information relevant to millions of users who have different security requirements regarding their information. We define SecurityContext based on integrity, authentication, non-repudiation and confidentiality, which are the known standards to define security. Security aspects validation can be integrated with a service operation as pre-condition / post-condition checks of the operation. We define these validations as constraints validation. Different operations of a service may need different security settings. For example, Operation1 may need to validate all four aspects as pre-condition checks, Operation2 may need integrity and authentication as pre-condition checks, and Operation3 may need only integrity as a post-condition check. The order of pre-condition and post-condition checks relevant to each operation of a service can be defined in ProtocolContext, in our context framework - section 4.2.4. Moreover, we observe that the taxonomic relationships are not sufficient to define all these relationships. In order to reduce the complexity of using SecurityContext, we represent them with boolean values either 0 and 1. We assume that the security processing and logging systems are maintained at both the service provider end and the client end, which justify boolean results. For example, if a service operation has an IntegrityContext 1, that means integrity checks are involved with that operation execution. Similarly, it works for other security aspects. We define the usage of integrity, authentication, non-repudiation and confidentiality in Web services domain as follows.

IntegrityContext. This ensures the protection of information from being deleted or altered in any way without the permission of the owner of that information. Integrity checks are used to check whether information are deleted or altered in any way by intruders. Encryption techniques are used to protect information from being deleted or altered. We define integrity of an operation OP_k in a service S^j based on attached encryption capabilities, so that relevant decryption techniques can be used to decrypt messages used in the operation - $Integrity(S^j_{OP_k}) = (EnDecrypt_k)$, where $(EnDecrypt_k)$ is the encryption / decryption technique. However, we define IntegrityContext of a service as a boolean value 1 or 0 to indicate that a message encryption feature is available or not with a particular service operation - $IntegrityContext(S^j_{OP_k})^{Available} = (1|0)$.

AuthenticationContext. This ensures that both consumers and providers are identified and verified. Authentication ensures that a service user and provider are authenticated, for example through passwords. We define authentication of an operation OP_k in a service S^j as $Authentication(S^j_{OP_k}) = (PW_k)$. PW_k can be user id and password, a certificate encoded in a HTTP basic authentication header, or WS-Security header information [Erradi 2005]. If authentication is successfully validated, then the service operation is executed and the result is routed to the message queue associated with the destination end point, otherwise a SOAP fault is returned. Here, we define AuthenticationContext of a service operation as a boolean value 1 or 0 to indicate that a authentication feature is available or not with a particular service operation - $AuthenticationContext(S^j_{OP_k})^{Available} = (1|0)$.

Non-RepudiationContext. The ability of the receiver to prove that the sender really did send a message. Non-repudiation checks meta-data relevant to the digital signature of a user, which is used to request a service operation and read operation output. The service operation is executed only for valid digital signatures, and the operation result can only be read using a valid signature. A digital signature can only be created by one person. The non-repudiation of an operation OP_k in a service S^j is defined as $Non\text{-}repudiation(S^j_{OP_k}) = (DS_m)$, where DS_m is digital signature of user m. We define the Non-RepudiationContext of a service operation as a boolean value 1 or 0 to indicate that a non-repudiation feature is available or not with a particular service operation - $Non\text{-}RepudiationContext(S^j_{OP_k})^{Available} = (1|0)$.

ConfidentialityContext. This ensures the protection of information from being read or copied by anyone who has not been explicitly authorized by the owner of that information. Confidentiality checks whether some operations of a service are explicitly authorized by the service provider. That is authorisation validation is needed to invoke some operations. We define confidentiality of an operation OP_k of a service j as $Confidentiality(S^j_{OP_k}) = (E\text{-}Client_1...E\text{-}Client_n)$. $E\text{-}Client_i$, $i = 1...n$ are authorised end users. We define the ConfidentialityContext of a service operation as a boolean value 1 or 0 to indicate a confidentiality feature is available or not with a particular service operation - $ConfidentialityContext(S^j_{OP_k})^{Available} = (1|0)$.

Security settings are determined based on 1 or 0 of each sub context category of *SecurityContext*. There are 2^4 security settings relevant to Web services.

Integrity	Authentication	Non-Repudiation	Confidentiality
1	1	1	1
1	1	1	0
1	1	0	0
...	...	...	...
0	0	0	0

The security setting 1111 implies all the security aspects are involved. The security setting 0000 implies none of these security aspects are involved. These security settings can be default settings for services and can also be dynamically changed, for example, ignoring non-repudiation means dynamically setting 0 for the Non-RepudiationContext.

TrustContext:

Trust is a key prerequisite for the wide adaptation of Web services in service-based applications. Most of us purchasing items via the Internet feel reluctant about transactions at some point, when providing credit card details or receiving unexpected items. We need guarantees that the other party does not misuse confidential information, and merchants need guarantees that they will receive payments for the goods delivered [Atif 2002]. The trust relationships of a service is an important aspect in service composition. An attribute that refers to the establishment of trust relationships between client and provider - which is a combination of technical assertions (measurable and verifiable quality) and relationship-based factors (reputation, history of cooperation). This work does not focus on detailing trust relationships, which can be found in [Yahyaoui 2010, Spanoudakis 2009, Spanoudakis 2007, Hasselbring 2006]. We assume Web services have trust certificates issued by third party organizations to reduce the complexity of trust relationships. These third party organizations are responsible for maintaining trust relationships with Web service providers, so that they issue trust certificates for individual Web services. The trust certificates of services need to be validated at process run-time. TrustContext of operation OP_k in a service S^j is defined as $TrustContext(S^j_{OP_k}) = (TCert_k)$, where $TCert_k$ is the trust certificate of operation k.

We highlight two prominent features of the quality of service context in this paragraph. Quality of service context are features about Web services, some of which can be defined at the service provider end, such as cost, regulatory etc. Some of them can be accurately measured only at service run-time, such as response time, availability etc. Two examples to clarify selected QoSContext are:
A service $S1$, which has a cost,

$$(S1 : Service) \overset{hasCostContext}{\longrightarrow} (0.1\ \text{USD})$$

and a service $S1$, which has response time estimation,

$$(S1 : Service) \overset{hasResponseTimeContext}{\longrightarrow} (< 200\ ms).$$

Domain context (DomainContext)

DomainContext defines the non-technical aspects of the environment, where the services are composed, deployed, and executed. Each application domain has its own requirements for interacting with Web services in that domain. Multi-application context modeling is being addressed by the pervasive computing research community [Euzenat 2008, Roy 2010, Wang 2004]. Due to the evolving nature of context-aware computing, completely detailing, categorising, and formalising all context information relevant to all the application domains is likely to be an in-surmountable task [Wang 2004]. We define a framework of DomainContext that defines high level contexts relevant to domain aspects, which are common, abstract, and influence service-based applications in general. High level context categories capture domain specific general features, which are reusable. A high level context category can have sub-categories of context devised in a top-down way, which facilitates extensibility in a hierarchical manner.

SemanticContext. This refers to semantic frameworks (i.e. concepts and their properties) in terms of vocabularies, taxonomies or ontologies. Our context classification is towards a more general context framework for Web services. SemanticContext refers to semantic frameworks relevant to domain specific services, such as a context framework for an air conditioning monitoring service in [Euzenat 2008]. Some service-based applications use domain specific tightly bound SemanticContext frameworks, such as CONON in [Wang 2004], which provides a context-enabled deployment environment for Web services. We do not detail the SemanticContext framework because its elaboration depends on the application domain such as CONON in [Wang 2004], SOUPA in [Chen 2004]. We define SemanticContext in an abstract view, as a domain-dependent semantic framework, which facilitates contextual support in the deployment and execution environments for service-based applications.

LinguisticContext. The language used to express queries, functionality, and responses. Creating a comprehensive ontology that can be useful to the future linguistic community of practice is a daunting task, and the project GOLD, which is an ongoing work at the moment is a stepping stone. Linguistically related concepts are organized in four major domains - expressions, grammar, data constructs, and meta concepts in [Farrar 2003]. More details about the GOLD ontology can be found in [Farrar 2003, Farrar 2010]. Moreover, social networks are increasingly popular at the moment, and new services attached with linguistic features are needed. As a complete framework, LinguisticContext is complex. LinguisticContext supports context-aware Web services. In the initial phase of linguistic support for context-aware Web services, we define LinguisticContext as the language used to express queries, functionality, and responses. Further explorations focusing on a complete LinguisticContext framework for context-aware Web services is needed, but it is out of our scope in this thesis.

MeasuresContext. Measures and Standards refers to locally used standards for measurements, currencies, etc., which are defined in MeasuresContext. The Measures and Standards context refers to domain-dependent measures and standards relevant to Web services. One of the prominent attempts is the QUOMOS (Quantities and Units of Measure Ontology Standard) project from OASIS (Advanced Open Standards for the Information Society). The preliminary work on QUOMOS came out of the Ontology Summit in 2009. Their technical committee is now working on developing an ontology to specify the basic concepts of quantities, systems of quantities, systems of measurement units, scales, various base dimensions, metric prefixes (nano, micro, milli, etc.), rules for constructing various derived units (joules, watts, etc.) for use across multiple industries. We have emphasized the importance of such an ontology in [Bandara 2009] towards context-aware Web service applications.

DomainContext categories are necessary to be used with other context categories in our context model. DomainContext mainly facilitates context for the deployment and execution environment. However, some of them can be attached to FunctionalContext of services, such as CurrencyContext and LanguageContext, which can be contexts of input parameters of a service.

$$(para1 : InputContext) \overset{hasLanguageContext}{\longrightarrow} (German : LanguageContext).$$

Some SemanticContext can also support domain-dependent services. For example, a domain-dependent food service uses a pizza ontological framework as in [Sheshagiri 2004].

Platform context (PlatformContext)

PlatformContext defines the technical aspects of the environment, where the services are composed, deployed and executed. If we use a mobile device to deploy and run a service process, then the mobile device has many limitations compared to a usual PC, such as lack of processing power, lack of memory, etc. The connectivity bandwidth can also impact on process execution, for example, low bandwidth can cause security problems, service time outs etc. We define two high level context categories under PlatformContext.

DeviceContext. This refers to the computer/hardware platform on which the service is provided. DeviceContext plays a major role in pervasive and ubiquitous systems, and we have observed the emergence of new device related constraints in service-based applications. The need for modeling device capabilities and status is obvious since service deployment decisions are based on this information [Chrysoulas 2007]. Moreover, we have observed that output data processing is also affected by device features, for example, some mobile phones do not support MMS messaging. An OWL-based ontology has been proposed in [Bandara 2004], with the aim of providing a formal framework to describe devices and their services to support effective service discovery. The usability and appropriateness of this ontology should be further investigated and refined accordingly [Bandara 2004]. Some services are specifically supported and offered by devices. The capabilities and characteristics of a device may play an integral role for a service. Moreover, some external information about the device may also be important, for example, when selecting a fax service using dynamic service composition, the location context of the fax machine can be a dynamic requirement. A more complete DeviceContext framework is needed to facilitate emerging future trends, but it is not easy and a pluggable DeviceContext framework could be a feasible solution. In this thesis, we do not focus on providing a complete DeviceContext framework, and we define DeviceContext as hardware and platform features, their relationships and environmental aspects such as location. For example, $(N95 : DeviceContext) \overset{hasFeature}{\longrightarrow} (MMS : MobileMessageContext)$.

ConnectivityContext. This refers to the network infrastructure used by the service to communicate. Ontology-based network management has recently evolved from a theoretical proposal to a more mature technology [Vergara 2009]. Ontology-based manipulation of network infrastructure such as connections, bandwidths, connection features, and their relationships are important in currently emerging service-based applications. We define ConnectivityContext as network infrastructures, their features, and their implicit relationships used by the Web services to communicate. For example, $(WifiConnection : ConnectionContext) \overset{hasBandwith}{\longrightarrow}$ $(10\ \texttt{Mbps})$.

During the execution of a Web service $S(i_1, ..., i_n)$, S takes some information as input and outputs some information. One can argue these outputs update the *information space* of the requester of a service S, defined in [Keller 2006]. Information space itself is not enough to address the dynamic nature of a Web service process, hence we define the *context space* of a service process, which is initially null and then enriched from context instances at service composition, deployment, and executions. Context instances are provided by functional and non-functional features of constituent services, and by deployment environment-based factors such as DomainContext and PlatformContext. The context space evolves during the execution of each service. Where possible, these context categories can be aligned with standardised or widely-used vocabularies, such as software quality standards (ISO 9126) or business directory information (UDDI) for the quality of service context. Overall, this is a flexible and evolvable context taxonomy that provides a vocabulary of context, properties, and explicit relations.

4.2.5 Non-taxonomic relationships

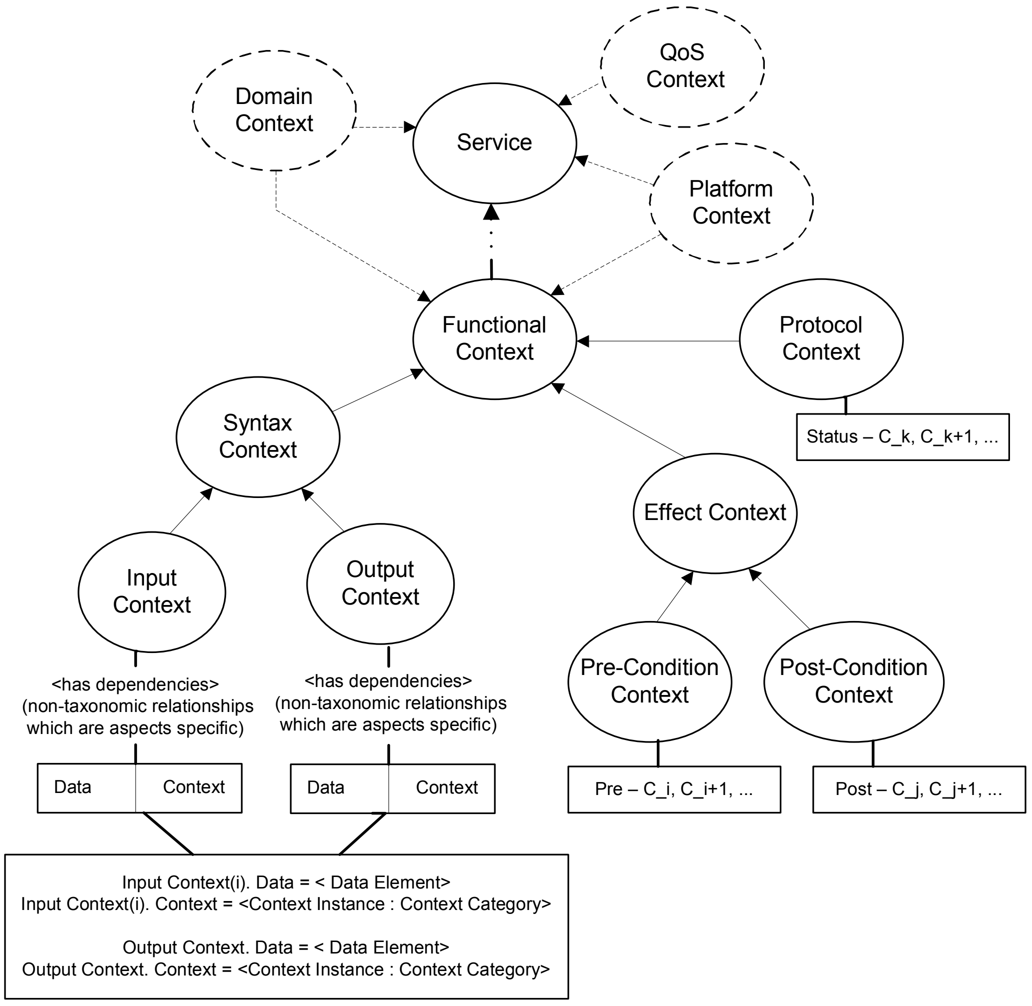
Although we have developed a flexible taxonomy to align all context categories, there are many types of non-taxonomic relationships in existence between context categories. One context category can depend on different context categories in different cases creating non-taxonomic relationships. Taxonomic relationships are defined in subsumption relationships. Non-taxonomic relations are mostly aspect-specific, i.e. local or non-local in terms of the hierarchy of the context model. Here are some examples to illustrate local and non-local relationships, which are difficult to define using taxonomic relations.

- Local : *SecurityContext* is the integration of Integrity, Authentication, Non-repudiation and Confidentiality contexts. Different levels of each factor can bring different levels of security. All the constituent context categories are local to SecurityContext.

- Non-local : Non-local relationships can be changed case by case. *TrustContext* is a combination of technical assertions (measurable and verifiable quality) and relationship-based factors (reputation, history of cooperation). The constituent context categories of TrustContext are distributed in the context taxonomy and not local in the TrustContext. We have observed that some of the non-local relationships have dependencies. For example, the TrustContext has relationships with measurable and verifiable aspects.

Trust can be defined in various ways in different cases. For example, - a requester and provider interact through an exchange of encrypted and signed messages accompanied by additional trust information to establish identity and trust context of each participant. A Web service, which is guaranteed as a secure and reputed service from a reputed organisation can be considered as a trusted service. Moreover, integrity (data cannot be modified undetectably) is violated when a message is actively modified in a transit. Message integrity can be implemented in case by case, such as signing a message, adding a check digit to the message, XML signature and XML encryption used in WS-Security for SOAP messages, Temporal Key Integrity Protocol (TKIP) used in wireless networks, etc. It is not our intention to define all these cases, which is impossible. However, software architects can use the proposed techniques to develop their specific cases. Our context model provides an abstract terminological framework, which needs to be customised in concrete situations. The abstract view of dependencies of context categories can be modeled as in figures 4.2, 4.3, 4.4 and 4.5.

Functional context:



QoS Context
Domain Context
Service
Platform Context
Functional Context
Protocol Context
Status – C_k, C_k+1, ...
Syntax Context
Effect Context
Input Context
Output Context
Pre-Condition Context
Post-Condition Context
<has dependencies> (non-taxonomic relationships which are aspects specific)
<has dependencies> (non-taxonomic relationships which are aspects specific)
Data
Context
Data
Context
Pre – C_i, C_i+1, ...
Post – C_j, C_j+1, ...
Input Context(i). Data = < Data Element>
Input Context(i). Context = <Context Instance : Context Category>
Output Context. Data = < Data Element>
Output Context. Context = <Context Instance : Context Category>

Figure 4.2: Dependencies in functional context

The abstract view of dependencies of functional context categories can be briefly illustrated as in figure 4.2. Each Input and Output parameter has data and context elements. Pre-conditions, Post-conditions and protocol context can be defined using condition identifiers.

Quality of service context: The abstract view of dependencies of quality of ser-

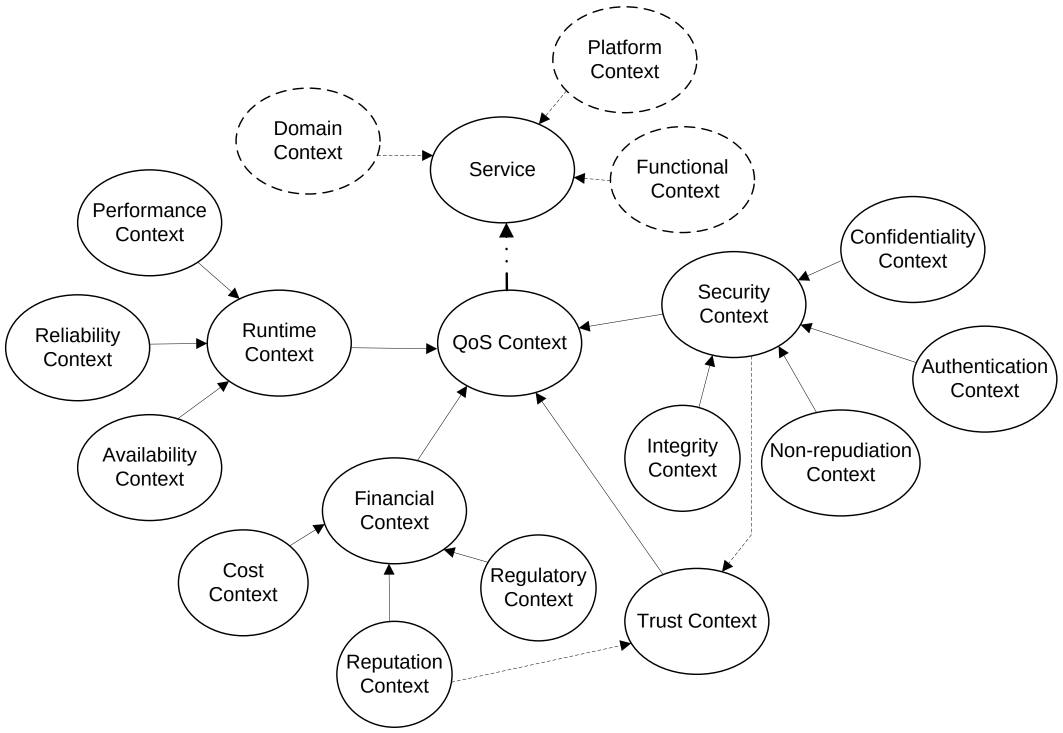


Figure 4.3: Dependencies in QoS context

vice context categories can be briefly illustrated as in figure 4.3. For example, security setting is combination of integrity, authentication, non-repudiation and confidentiality, trust depends on security and reputation, etc. Another QoS aspect that can be defined through non-taxonomic relationships is software dependability often defined as a combination of reliability and availability aspects [Avizienis 2004], but also sometimes in varity of other criteria decided by software architects [Vladimir 2011]. This adds to our point that not all context aspects can and should be fixed in one context model.

Domain context: The abstract view of dependencies of domain context categories

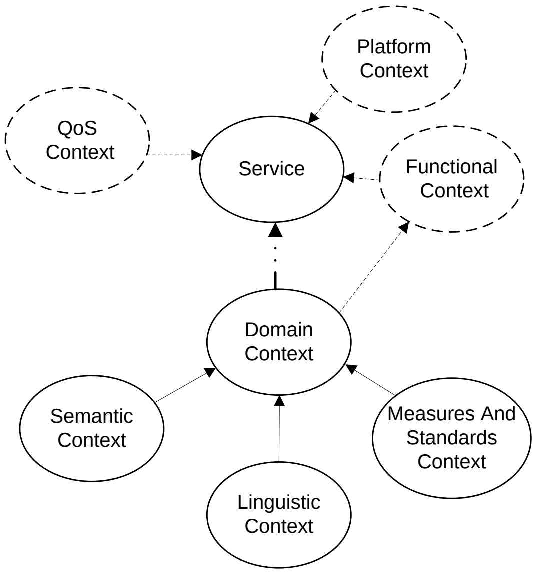


Figure 4.4: Dependencies in domain context

can be briefly illustrated as in figure 4.4. For example, domain contexts provide inputs for functional context, semantic context can be a conceptual framework used for domain specific services, etc.

Platform context: The abstract view of dependencies of platform context cate-

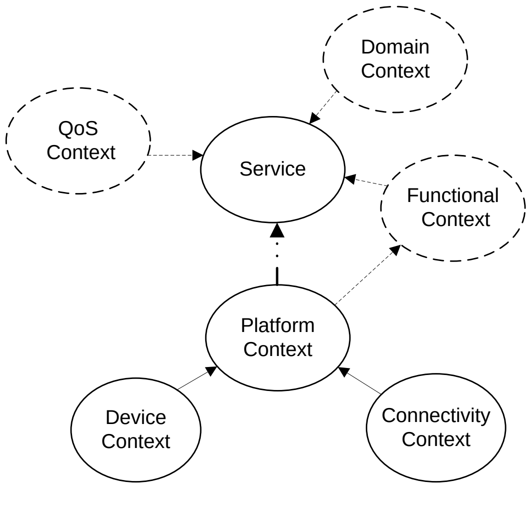


Figure 4.5: Dependencies in platform context

gories can be briefly illustrated as in figure 4.5. For example, platform contexts provide inputs for functional context, device context can be a simple context category to complex framework depending on the devices and applications, etc.

Though there are several context models defined in mobile and pervasive application domains, they do not sufficiently discuss non-taxonomic relationships. Taxonomic relationships can be modeled in UML following object-oriented concepts [Priestley 2003], but that is not strong enough to define all non-taxonomic relationships, such as implicit dependencies between context categories. An interesting modelling approach, which is a keystone in resolving this inconsistency is ontology-based conceptualization, in which we define taxonomic and non-taxonomic relationships in a context model ontology. We define non-taxonomic relationships within the ontology using ontological roles and SWRL rules, which we describe in detail in section 4.3.

4.3 Context modeling

We realise dynamic service context modeling as ontology-based context representation and reasoning focusing on dynamic aspects in Web service processes. A modeling language, which facilitates formalisation support for both taxonomic and non-taxonomic relationships between context categories / instances, reasoning support for context instances, and composition support for context categories / instances is needed to model dynamic service context. In existing context-aware systems, XML, XML-based CC / PP [Doulkeridis 2006], UML [Kapitsaki 2009], Topic Maps [Goslar 2004], RDF [Medjahed 2007], and OWL [Wang 2004] are widely used for context modeling and implementation in various domains. We select ontology-based modeling and OWL-based representation. The background and relative merits for selecting OWL are described in section 4.3.1. This section further describes building blocks used for OWL-based context modeling. The OWL-based context model ontology implementation is illustrated in appendix A. While description logic (DL) is used to formalise aspects of the context model, OWL will be further used later on. OWL is necessary to operationalise (make the context model processable) the context model.

4.3.1 Ontology-based context modeling

Context modeling describes how dynamic aspects of Web services are aligned and maintained. A context model supports efficient context management. An ontology consists of entities, relations, functions, axioms, and instances. Ontologies have many merits like information sharing, re-usability, extensibility, programmability and reasoning support. The choice of OWL over other approaches for this research is motivated by several reasons - (1) OWL supports XML-based interoperability. (2) OWL ontology provides a logical language support for composition and reasoning (OWL-DL). (3) OWL ontology supports SWRL (Semantic Web Rule Language), which supports rule-based context derivation and reasoning. (4) OWL facilitates shared conceptualization, which is important for cross-organizational service compositions. (5) The underlying logical language supports context composition. (6) The underlying logical language supports context constraints enhancements. These properties satisfy best the requirements identified in section 2.5. For example, logical reasoning can reduce the complexity of some dynamic tasks (device settings for payment confirmation in figure 1.1), shared conceptualization of dynamic aspects of services as necessary, etc. We select the ontology web language (OWL) as the context formalisation language.

4.3.2 Description logic - $\mathcal{SHOIN}(\mathcal{D})$

We introduce the core elements of the description logic language used with OWL. The Attribute Concept Language with Complements ($\mathcal{ALC}$) is the basis of many more expressive description logic languages. The OWL-DL, the description logic variant of OWL is $\mathcal{SHOIN}(\mathcal{D})$, a description logic language [Baader 2003]. Each letter describes a particular feature.

S . An abbreviation for $\mathcal{ALC}$ with transitive roles.

H . Role hierarchy (subproperties - rdfs : subPropertyOf).

O . Nominals. (Enumerated classes of object value restrictions - owl : oneOf, owl : hasValue).

I . Inverse properties.

N . Cardinality restrictions (owl : Cardinality, owl : MaxCardinality).

(D) . Optional inclusion of concrete data types.

In order to encode dynamic aspects in $\mathcal{SHOIN}(\mathcal{D})$, and eventually in OWL-DL, an understanding of the allowed constructors for $\mathcal{SHOIN}(\mathcal{D})$ is necessary, and constructors are illustrated in table 4.2 [Farrar 2010]. C denotes concepts and R denotes role relationships.

Constructor	$\mathcal{SHOIN}(\mathcal{D})$	OWL-DL
conjunction	C1⊓C2	intersectionOf(C1,C2)
disjunction	C1⊔C2	unionOf(C1,C2)
negation	¬C1	complementOf(C1)
oneOf	$o_1...o_n$	oneOf($o_1...o_n$)
exists restriction	∃R.C	someValuesFrom(C)on(R)
value restriction	∀R.C	allValuesFrom(C)on(R)
atleast restriction	≥nR	minCardinality(n)on(R)
atmost restriction	≤nR	maxCardinality(n)on(R)
datatype exists	∃R.D	someValuesFrom(D)on(R)
datatype value	∀R.D	allValuesFrom(D)on(R)
datatype atleast	≥nR	minCardinality(n)on(R)
datatype atmost	≤nR	maxCardinality(n)on(R)
datatype oneOf	$v_1...v_n$	oneOf($v_1...v_n$)

Table 4.2: Table: $\mathcal{SHOIN}(\mathcal{D})$ notations

A DL knowledge base can be constructed as a set of description logic axioms. The basic constructors of $\mathcal{SHOIN}(\mathcal{D})$ can be used with either the $\sqsubseteq$ or $\equiv$ symbol to create logical statements of various kinds. The resulting logical statements are the axioms of a DL. DL axioms can be Terminological axioms (TBox) or Assertional axioms (ABox). Terminological axioms (statements about entities such as concepts and roles, but not individuals) can be inclusion or equality axioms ($\sqsubseteq$ or $\equiv$). A Terminological axiom is denoted by T. Assertional axioms (pertaining only to individuals) can be concept assertion or role assertion axioms [Farrar 2010]. An Assertional axiom is denoted by A.

- Inclusion axiom (T). An axiom that gives necessary conditions for some concept to be included (subclassed) in another. A $\sqsubseteq$ B , where A, B are concepts.
- Equality Axiom (T). An axiom of the general form $A \equiv B$, where A, B are concepts.
- Concept Assertion Axiom (A). An axiom is of the form C(`i`), where C is some concept from TBox and `i` is an individual.
- Role Assertion Axiom (A). An axiom is of the form R(`a`,`b`), where R is some role from the TBox and `a` and `b` are individuals.

4.3.3 Ontology-based service context formalisation

Our context model ontology consists of concepts (or classes in OWL terminology), its properties are in the form of roles and individuals. The description logic constructors and axioms can be used to formalize the context model ontology. These logical relations further support context compositions and rule-based context instance derivations.

- Concepts (OWL classes) are interpreted as sets that contain individuals.
- Roles (OWL properties) are binary relations on individuals linking two individuals.
- Individuals represent context instances of concepts and roles. OWL does not use the Unique Name Assumption (UNA), i.e. two different names could actually refer to the same individual. For example, `SecOfS1` and `SecurityOfServiceOne` refer to the same individual.

Subsumption expresses whether or not a contextual concept/role is a subconcept/role of another concept/role. Subconcepts specialise (are subsumed by) their superconcepts. It uses context instance subsumption based on the hierarchical relationships of context in the context model taxonomy, i.e.

$$\mathrm{C} \sqsubseteq \mathrm{D} \text{ iff } C^I \subseteq D^I$$

where C and D are concepts or roles and C^I and D^I their respective interpretations. Classes can be organised into a concept hierarchy that formalises the taxonomy described earlier. For example, `FunctionalContext` subsumes `InputContext`.

$$FunctionalContext \sqsupseteq InputContext$$

Subsumption can be used to match user context requirements against provider context (later called service context profiles) and to determine configurables (service selection and process composition), i.e. comparing user requirements (stemming from SLAs) against actual or declared provider properties (through satisfaction and matching). Constraints compare actual and required context properties. Both structural (subconcept) and logical (implication) subsumption relationships can be determined automatically. Now we will look into concepts, roles, and the formalisation of their relationships in detail to specify further characteristics of the context model. Please note the sub sections in 4.3.3, we illustrate DL constructs by concrete context model examples. These examples are part of our context model that further axiomatise taxonomic and non-taxonomic aspects. However, most of these axioms might be necessary in concrete situations, which cannot be provided in a generic context model as discussed above. Thus, we provide here a list of modeling features for a software architect, illustrated by concrete context model examples.

Concept description

The building blocks of an OWL ontology are classes that represent concepts (DL term). $\mathcal{SHOIN}(\mathcal{D})$ axioms can be used to specify complex class descriptions – classes can be a subclass or disjoint with other classes:

- **Subclasses** represent hierarchical relationships between classes. For example, Security subsumes Integrity.

$$Security \sqsupseteq Integrity$$

- **Disjointness** means that individual components are different. For example, high security and high performance together is hard to achieve. Extra processing is required for highly secured processes.

$$Security \sqcap Performance \equiv \bot \text{ (owl:Nothing)}$$

- **Completeness** means that a context is built only from pre-specified contexts. For example, security is an integration of integrity, authentication, non-repudiation and confidentiality.

$$Security \equiv Integrity \sqcap Authentication \sqcap Non\text{-}repudiation \sqcap Confidentiality$$

- **Composed class descriptions:** The composition of more than one context category can be described in complex class descriptions. For example, an EffectContext can have either a Pre-ConditionContext or a Post-ConditionContext or both.

$$EffectContext \sqsupseteq Pre\text{-}ConditionContext \sqcup Post\text{-}ConditionContext$$

  The PlatformContext has both DeviceContext and ConnectivityContext.

$$PlatformContext \sqsupseteq DeviceContext \sqcap ConnectivityContext$$

These descriptions can be used to further constrain the subsumption relationships between EffectContext and PlatformContext and their respective subclasses.

Role description

Context in the taxonomy can have relationships, which can be formalized within the context model ontology as roles (DL term). Roles represent relationships between individuals or an individual and data literals. In this work, individuals are context instances. Generally, a role could be an object role, datatype role or annotation role based on how they are used within the structural elements of ontologies [Horridge 2004].

- **Object roles** link an individual to an individual. For example, a service instance has security instance 1111,

$$hasSecurity\ \ (Service(s), Security(1111)\ )$$

- **Datatype roles** link an individual to an XML schema datatype value or an RDF literal. For instance, for device d,

$$hasDisplaySettings\ \ (Device(d), "6 \times 8(in)"\ )$$

- **Annotation roles** can be used to add information (metadata) to contextual concepts, individuals and object/datatype roles.

  An annotation role of a concept,

$$hasType\ \ (Device, "Mobile")$$

  An annotation role of an individual,

$$hasproducer\ \ (Device(d), "Bell")$$

An annotation role of an object role,

$$hasSameMeaning \ \ (hasPart, hasComponent)$$

and an annotation role of datatype role,

$$means \ \ (hasDisplaySettings, "defines\ device\ display\ settings")$$

Roles can be specifically used to express context constraints of concrete services that needs to be monitored and validated at run-time, such as the required dimensions of a particular device.

Note that roles can also be categorised in terms of their roles within the context model vocabulary. **Generic** (core) roles are hard-coded into the context model taxonomy, such as *hasPart*, e.g., $hasPart\,(\ Security, Integrity\ )$. Some roles are **aspect-specific**, such as *hasSecurity* of a service, e.g., $hasSecurity\,(\ Service(s), Security(1111)\ )$. The second category is introduced to further qualify context. Note, that they could have been done as *hasPart* or subconcept roles with typed instances, but in this way they become part of the vocabulary. The introduced roles have one (or more) of the following functional, transitive or symmetric properties [Horridge 2004].

- **Functional.** A role R is functional, if for a given individual there is at most one individual that is related to the individual via the role. For example, if a service has at most one cost then *hasCost* is functional.

$$Service(s).\ \exists hasCost\ \leq\ 1\ (Cost)$$

- **Inversible.** A role R is inversible, if the role links individual a to individual b then its inverse will link b to a. For example,

$$\text{If } hasPart(Security(s), Integrity(i))\ \sqcap\ isPartOf(Integrity(i), Security(s))$$
$$\text{Then } hasPart\ \equiv\ (isPartOf)^{-}.$$

- **Transitive.** A role `R` is transitive, if the role relates individual `a` to individual `b` and individual `b` to individual `c` then individual `a` is related to individual `c` via `R`. For example,

$$\text{If } hasPart(QoS(q), Security(s)) \sqcap hasPart(Security(s), Integrity(i)) \text{ Then } hasPart(QoS(q), Integrity(i)).$$

$$\Rightarrow Tr(hasPart)$$

- **Symmetric.** A role `R` is symmetric, if the role relates individual `a` to individual `b` then individual `b` is also related to individual `a` via `R`. For example, integrity and authentication are parts of security, That is

$$hasSibling(Integrity(i), Authentication(a)) \sqcap hasSibling(Authentication(a), Integrity(i))$$

$$\Rightarrow hasSibling \equiv (hasSibling)^{-}$$

Knowledge about functional, transitive or symmetric roles can be used to determine the validity of context constraints by matching observed (monitored) actual conditions with the requirement context constraints using OWL-DL-based inference.

Rule-based context

OWL-DL can be extended with rules. OWL-DL is expressive, a decidable fragment of first-order logic, and thus cannot express arbitrary axioms. The only axioms it can express are of a certain tree structure [Grosof 2003]. In contrast, decidable rule-based formalism such as Horn-like rules do not share this restriction, but lack some of the expressive power of OWL-DL, such as restrictions on universal quantification and lack of negation in their basic form [Motik 2005]. In order to overcome the limitations of both approaches, OWL-DL can be extended with rules. However, this extension is undecidable, which loses any form of tree structure properties. Rules can also be used with OWL-DL to a certain extent, where decidability is obtained by restricting the rules to so-called DL-safe rules [Motik 2005].

The implicit context can be derived from the explicit context in the context model ontology based on rules in the form,

$$Antecedent \rightarrow Consequent;\ Antecedent\ \textbf{implies}\ Consequent.$$

Antecedent and consequent consist of one or more context concepts and role descriptions. For example, if in a client context, a mobile device is indicated in the respective context aspect, the output message display should be matched with the display settings of the device,

$$\text{hasMessage (client,message)} \wedge \text{hasDevice (client,mobile)}$$
$$\rightarrow \text{hasDisplaySetting (message,3x5inches).}$$

These rules can be implemented for instance as SWRL (Semantic Web Rule Language) rules. SWRL is intended to be the rule language of the semantic Web, and all rules are expressed in terms of OWL classes (concepts in our case), properties (roles in our case) and individuals. SWRL is a combination of variable free OWL-DL (section 4.3.2) and variable supported RuleML (Rule Markup Language) [Boley 2010]. SWRL extends the set of OWL axioms to include Horn-like rules. It thus enables Horn-like rules to be combined with an OWL knowledge base [Horrocks 2004a]. OWL knowledge bases consist of OWL-DL and OWL axioms, described in section 4.3.2.

4.4 Case study - Context model ontology integration

The context model ontology includes contextual concepts, individuals, roles and role characteristics. These can be integrated with Web services and processes. This integration can formalise dynamic service context, generic roles, aspect-specific roles, reasoning aspects, etc. The following case study illustrates dynamic service context integration with a Web service.

4.4.1 Tool support

We implement the context model ontology and ontology integration with Web services using the Protege 4.0.2 ontology development environment, which is built on top of the OWL API that facilitates more flexible development and a straightforward migration path for OWL-based applications. This is a design-time tool. This ontology development environment is also integrated with reasoners. This is compatible with both Java 5 and Java 6 versions.

4.4.2 Case study

In this section, we describe the OWL formalised context integration with a Web service (HSBC Banking Service - Banking Service in figure 1.1) and illustrate the detailed implementation in appendix A. In most cases, object roles and data type roles are used to establish context integration relationships in a Web service.

Functional context integration

The FunctionalContext formalised in OWL can be integrated with a Web service. For example, we consider a HSBC banking service to illustrate FunctionalContext integration. We assume this banking service has only one operation. The syntax, effect, and protocol context of the service imply FunctionalContext of the operation.

1. SyntaxContext: We consider two input parameters, one with context (`transactionAmountIns`), and one without context (`accountNumberIns`) to illustrate input context integration. We define each parameter to be attached with its data and context. The ontology-based roles are used to define context integration.

`HSBCBankingService` has an input parameter `transactionAmountIns`. That is, `HSBCBankingService` has an object role *hasInputContext* connecting to an instance of `transactionAmountIns`. The instance of `transactionAmount-`

`Ins` is of the type *InputContext*. The `transactionAmountIns` has

- an object role *hasContext* connected to an instance `USD`. The instance `USD` is of the type CurrencyContext.
- a data type role *hasData* connected to a string. We assume all the data is of the type String.

That is,

$$hasInputContext\,(\ Service(HSBCBankingService), InputContext(transAmountIns)\ )$$
$$\sqcap\ hasContext\,(\ InputContext(transAmountIns), CurrencyContext(USD)\ )\ \sqcap$$
$$hasData\,(\ InputContext(transAmountIns), (\text{"100"}{:}String)\ )$$

Some parameters may not have context, but have only a data element, such as Account number. That is,

$$hasInputContext\,(\ Service(HSBCBankingService), InputContext(accNumberIns)\ )$$
$$\sqcap\ hasData\,(\ InputContext(accNumberIns), (\text{"100"}{:}String)\ )$$

Similar to input parameters, some output parameters can have context, some may not. This can be integrated in the same way as inputs. An OWL code excerpt of the InputContext integration to the `HSBCBankingService` can be illustrated as follows,

```
<Service rdf:about="#HSBCBankingService">
    <rdf:type rdf:resource="&owl;Thing"/>
    <hasInputContext rdf:resource="#accountNumberIns"/>
    <hasInputContext rdf:resource="#transactionAmountIns"/>
</Service>

<InputContext rdf:about="#accountNumberIns">
    <rdf:type rdf:resource="&owl;Thing"/>
    <hasData rdf:datatype="&xsd;string">ISS1234</hasData>
</InputContext>

<owl:Thing rdf:about="#transactionAmountIns">
    <rdf:type rdf:resource="#InputContext"/>
```

```
14      <hasData rdf:datatype="&xsd;string">3000</hasData>
15      <hasContext rdf:resource="#USD"/>
16  </owl:Thing>
17
18  <CurrencyContext rdf:about="#USD">
19      <rdf:type rdf:resource="&owl;Thing"/>
20  </CurrencyContext>
```

Lines 1-5 specify that the `HSBCBankingService` service has two input parameters `accountNumberIns` and `transactionAmountIns`. Lines 7-10 specify the parameter `accountNumberIns` and its data. Lines 12-20 specify the parameter `transactionAmountIns`, its data and context.

2. EffectContext: `HSBCBankingService` has an input parameter (*accountNumberIns:InputContext*), that needs to be verified before invoking the service. That is, (*UserAccountVerification:Pre-ConditionContext*) is a pre-condition of the service. Similarly, the HSBC service has CurrencyContext verification of (*transactionAmountIns:InputContext*), and encryption verification of all inputs as pre-conditions.

$hasPreConditionContext(\ Service(HSBCBankingService), Pre\text{-}ConditionContext(UserAccountVerification)\ ) \sqcap hasPreConditionContext(\ Service(HSBCBankingService), Pre\text{-}ConditionContext(CurrencyTypeVerification)\ ) \sqcap hasPreConditionContext(\ Service(HSBCBankingService), Pre\text{-}ConditionContext(EncryptionVerification)\ ).$

Post-ConditionContext can also be integrated with services, similar to Pre-ConditionContext. An OWL code excerpt of the EffectContext integration to the `HSBCBankingService` can be illustrated as follows,

```
<Service rdf:about="#HSBCBankingService">
    <rdf:type rdf:resource="&owl;Thing"/>
    <hasPreConditionContext rdf:resource="#CurrencyTypeVerification"/>
    <hasPreConditionContext rdf:resource="#EncryptionVerification"/>
    <hasPreConditionContext rdf:resource="#UserAccountVerification"/>
</Service>

```

```
 8 <PreConditionContext rdf:about="#EncryptionVerification">
 9     <rdf:type rdf:resource="&owl;Thing"/>
10 </PreConditionContext>
11
12 <PreConditionContext rdf:about="#UserAccountVerification">
13     <rdf:type rdf:resource="&owl;Thing"/>
14 </PreConditionContext>
15
16 <owl:Thing rdf:about="#CurrencyTypeVerification">
17     <rdf:type rdf:resource="#PreConditionContext"/>
18 </owl:Thing>
19
20 <owl:Class rdf:about="#PreConditionContext">
21     <rdfs:subClassOf rdf:resource="#EffectContext"/>
22 </owl:Class>
```

Lines 1-6 specify that the service `HSBCBankingService` has three pre-conditions `CurrencyTypeVerification`, `EncryptionVerification`, and `UserAccountVerification`. Lines 8-10 specify that the `EncryptionVerification` is an instance of Pre-ConditionContext. Lines 12-14 specify that the `UserAccountVerification` is an instance of Pre-ConditionContext. Lines 16-18 specify that the `CurrencyTypeVerification` is an instance of Pre-ConditionContext. Lines 20-22 specify that the relationship between Pre-ConditionContext and EffectContext, which is a subsumption relationship.

3. ProtocolContext: In the *HSBCBankingService* example, the order of executing pre-conditions is defined in the ProtocolContext. This is an object role relationship.

$$hasProtocolContext(\ Service(HSBCBankingService), ProtocolContext(Pro1\text{-}HSBCBankingService)\ )$$

In order to define the ProtocolContext instance, we use a data type role relationship.

$$hasDefinition(\ ProtocolContext(Pro1\text{-}HSBCBankingService),\ (Pre\text{-}EncryptionVerification, UserAccountVerification, CurrencyTypeVerification\ :$$

$$String)\ )$$

An OWL code excerpt of the ProtocolContext integration to the *HSBCBankingservice* can be illustrated as follows,

```
1  <Service rdf:about="#HSBCBankingService">
2          <rdf:type rdf:resource="&owl;Thing"/>
3          <hasProtocolContext rdf:resource="#Pro1—HSBCBankingService"/>
4  </Service>
5
6  <ProtocolContext rdf:about="#Pro1—HSBCBankingService">
7      <rdf:type rdf:resource="&owl;Thing"/>
8      <hasDefinition rdf:datatype="&xsd;string">
9      Pre —
10         EncryptionVerification,
11         UserAccountVerification,
12         CurrencyTypeVerification.
13     </hasDefinition>
14 </ProtocolContext>
```

Lines 1-4 specify that the service `HSBCBankingService` has a ProtocolContext instance called `Pro1-HSBCBankingService`. Lines 6-14 specify the ProtocolContext instance. It is the execution sequence of pre-conditions in this example.

Quality of service context integration

Here, we use the `HSBCBankingService` and its quality of service context (QoSContext) integration. The Web service (`HSBCBankingService`) has an object role *hasCostContext* connecting to `0.1`, which is an instance of CostContext. The instance `0.1` has an object role *hasCurrencyContext* connecting to `GBP`, which is an instance of CurrencyContext. This can be formalised as,

$$hasCostContext(\ Service(HSBCBankingService), CostContext(0.1)\ ) \sqcap$$
$$hasCurrencyContext(\ CostContext(0.1), CurrencyContext(GBP)\ )$$

The `HSBCBankingService` service also has reputation, security, trust and performance contexts.

$hasReputationContext(\ Service(HSBCBankingService), ReputationContext(1)\ )$, $hasSecurityContext(\ Service(HSBCBankingService), SecurityContext(1111)\ )$, $hasTrustContext(\ Service(HSBCBankingService), TrustContext(TCert[HSBCBank\text{-}ingService])\ )$, $hasPerformanceContext(\ Service(HSBCBankingService), Performa\text{-}nceContext(perf\text{-}HSBCBankingService)\ ) \sqcap hasResponseTime(\ PerformanceContext\ (perf\text{-}HSBCBankingService), \text{"} < 600\ ms\text{"}\ )$.

An OWL code excerpt of the QoSContext integration to the `HSBCBankingService` can be illustrated as follows,

```
<Service rdf:about="#HSBCBankingService">
    <rdf:type rdf:resource="&owl;Thing"/>
    <hasCostContext rdf:resource="#0.1"/>
    <hasReputiationContext rdf:resource="#1"/>
    <hasSecurity rdf:resource="#1111"/>
    <hasTrustContext rdf:resource="#TCert[HSBCBankingService]"/>
    <hasPerformanceContext rdf:resource="#perf-HSBCBankingService"/>
</Service>

<owl:Thing rdf:about="#0.1">
    <rdf:type rdf:resource="#CostContext"/>
    <hasCurrencyContext rdf:resource="#GBP"/>
</owl:Thing>

<owl:Thing rdf:about="#1">
    <rdf:type rdf:resource="#ReputationContext"/>
</owl:Thing>

<SecurityContext rdf:about="#1111">
    <rdf:type rdf:resource="&owl;Thing"/>
</SecurityContext>

<SecurityContext rdf:about="#1111">
<rdf:type rdf:resource="&owl;Thing"/>
    <hasNon-RepudiationContext rdf:resource="#1"/>
    <hasConfidentialityContext rdf:resource="#1"/>
    <hasAuthenticationContext rdf:resource="#1"/>
```

```
28      <hasIntegrityContext rdf:resource="#1"/>
29  </SecurityContext>
30
31  //1 is an instance of all four categories.
32  <AuthenticationContext rdf:about="#1">
33      <rdf:type rdf:resource="#ConfidentialityContext"/>
34      <rdf:type rdf:resource="#IntegrityContext"/>
35      <rdf:type rdf:resource="#NonRepudiationContext"/>
36  </AuthenticationContext>
37
38  //All four categories are parts of SecurityContext
39  <owl:ObjectProperty rdf:about="#isPartOf">
40      <rdfs:domain rdf:resource="#AuthenticationContext"/>
41      <rdfs:domain rdf:resource="#ConfidentialityContext"/>
42      <rdfs:domain rdf:resource="#IntegrityContext"/>
43      <rdfs:domain rdf:resource="#NonRepudiationContext"/>
44      <rdfs:range rdf:resource="#SecurityContext"/>
45      <owl:inverseOf rdf:resource="#hasPart"/>
46  </owl:ObjectProperty>
47
48  <owl:Thing rdf:about="#TCert[HSBCBankingService]">
49      <rdf:type rdf:resource="#TrustContext"/>
50  </owl:Thing>
51
52  <owl:Thing rdf:about="#perf-HSBCBankingService">
53      <rdf:type rdf:resource="#PerformanceContext"/>
54      <hasResponseTime rdf:datatype="&xsd;string">&lt; 600 ms</hasResponseTime>
55  </owl:Thing>
```

Lines 1-8 specify that the service `HSBCBankingService` has cost, reputation, security, trust and performance contexts. Lines 10-13 specify the CostContext of the service. Lines 15-17 specify the ReputationContext of the service. Lines 19-46 specify the SecurityContext of the service including authentication, non-repudiation, confidentiality, and integrity contexts. Lines 48-50 specify the TrustContext of the service. $TCert[HSBCBankingService]$ is the trust certificate of HSBCBankingService. Lines 52-55 specify the PerformanceContext of the service.

Domain context and Platform context integration

DomainContext and PlatformContext bring environmental aspects into Web services. In the previous example,

$$hasCurrencyContext(\ CostContext(0.1), CurrencyContext(GBP)\ )$$

describes the usage of CurrencyContext relevant to *HSBCBankingService.* If this service is used somewhere in Germany, then the CurrencyContext needs to be changed accordingly. That is, CurrencyContext of the service changes based on the domain. Moreover, PlatformContext brings heterogeneous platform aspects into service processes. For example, *HSBCBankingService* needs connectivity bandwidth greater than 5 Mbps.

$needConnectivityContext(\ Service(HSBCBankingService), ConnectivityContext(connectivity)\ ) \sqcap hasConnectivityStrength(\ ConnectivityContext(connectivity),\ "> 5"\ )$

An OWL code excerpt of above context integration with the *HSBCBankingService* can be illustrated as follows,

```
<Service rdf:about="#HSBCBankingService">
    <hasCostContext rdf:resource="#0.1"/>
    <needConnectivityContext rdf:resource="#connectivity"/>
</Service>

// service cost has domain context
<owl:Thing rdf:about="#0.1">
    <rdf:type rdf:resource="#CostContext"/>
    <hasCurrencyContext rdf:resource="#GBP"/>
</owl:Thing>
<owl:Thing rdf:about="#GBP">
    <rdf:type rdf:resource="#CurrencyContext"/>
</owl:Thing>
<owl:Class rdf:about="#CurrencyContext">
    <rdfs:subClassOf rdf:resource="#MeasuresContext"/>
</owl:Class>
<owl:Class rdf:about="#MeasuresContext">
    <rdfs:subClassOf rdf:resource="#DomainContext"/>
</owl:Class>

```

```
21 //service connectivity has platform context
22 <owl:ObjectProperty rdf:about="#needConnectivityContext">
23     <rdfs:range rdf:resource="#ConnectivityContext"/>
24     <rdfs:domain rdf:resource="#Service"/>
25 </owl:ObjectProperty>
26
27 <owl:Thing rdf:about="#connectivity">
28     <rdf:type rdf:resource="#ConnectivityContext"/>
29     <hasConnectivityStrength rdf:datatype="&xsd;string">
30         &gt; 5 Mbps</hasConnectivityStrength>
31 </owl:Thing>
32
33 <owl:DatatypeProperty rdf:about="#hasConnectivityStrength">
34     <rdfs:domain rdf:resource="#ConnectivityContext"/>
35 </owl:DatatypeProperty>
36
37 <owl:Class rdf:about="#ConnectivityContext">
38     <rdfs:subClassOf rdf:resource="#PlatformContext"/>
39 </owl:Class>
```

Lines 1-4 specify that the *HSBCBankingService* has CostContext and Connectivity-Context. With regard to the CostContext, the focus is on CurrencyContext change in various domains. Lines 7-19 specify the CostContext, which is attached to CurrencyContext. The CurrencyContext is a DomainContext. Lines 22-39 specify the connection bandwidth, which is a PlatformContext.

4.5 Chapter summary

In this chapter, first we defined dynamic service context, which is one of the main challenges identified in section 2.5. Then, we defined a context model ontology to describe dynamic aspects of Web service processes (the core challenge identified in section 2.5). In defining a context model, we captured dynamic service context attached to dynamic requirements at service process run-time using an empirical analysis of application scenarios from the classical business domain and the state of the art review of previous classifications. We also considered the context orientation as an important aspect in defining a general context model. We considered inward and outward perspectives and the related previous work. This context model is at the core a classification and formalisation of dynamic aspects. It works as a shared conceptualization framework for Web service processes. This context model is embedded into a rich conceptual modeling technique, including language and elements to specify and reason about dynamic service context in a context model ontology. We formalised the context taxonomy as an OWL-based context model ontology to gain benefits outlined in section 4.3.1. The proposed context model ontology, which is located at the immediate context in [Rosemann 2008], can serve as a part of middleware component for managing dynamic aspects at service process run-time in dynamic service applications.

The proposed context model ontology can also be adapted to fill the gap between application-dependent context-aware middleware and general context-aware service community frameworks proposed in [Medjahed 2007, Mrissa 2008]. At the moment, WSDL does not directly support context integration, hence context embedding architecture for WSDL could be a useful future direction of research in the context-aware Web services domain. The related work on semantic annotation for WSDL (SAWSDL) can possibly be improved to embed context information towards context annotated WSDL. We detail this idea in section 9.3.2 as future work.

Chapter 5

Context manipulation and reasoning

Contents

5.1 Introduction

A context specification needs to be adapted for further processing in several contexts. From a process perspective, the context specifications of constituent services need further processing to derive and adapt dynamic service context. For example, several services in a process need to be combined creating the need for the corresponding dynamic service contexts to be combined. There are some situations, which cannot be addressed by available OWL-DL operators (e.g., the security of a process is the weakest security of all individual services, the security of a service is restricted to its integrity context, etc.), and some context specifications need to be adapted at process run-time. We address these limitations using context manipuation, context composition, and context reasoning. We propose an operator calculus for context specifications to facilitate manipulation and composition, and techniques for context reasoning. The context specifications attached to services are provided by service providers. The dynamic aspects in these specifications need manipulation, composition, and reasoning to support dynamic requirements (context constraints in chapter 6) attached to composition and execution of services at process run-time within the semantic client (or broker).

The context model ontology described in chapter 4 is used at the semantic client and the provider ends. Operator calculus and reasoning can be combined with OWL-DL constructs and implemented at the semantic client (or broker) to support dynamic requirements, which are defined in terms of dynamic service context. OWL-DL is a description logic, which provides maximum expressiveness while maintaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). OWL-DL includes all OWL language constructs, but they can be used only under certain restrictions (for example, while a class may be a subclass of many classes, a class cannot be an instance of another class)[McGuinness 2004]. We use both description logic and OWL terminologies through out this chapter. We use description logic-based context reasoning for capturing subsumption relationships of contextual concepts, checking the consistency of available context information, and deriving implicit context from explicitly defined context instances. Context reasoning supports requirements validation of services and composed service processes at process run-time. Context instances are generated for each Web service process based on their constituent services and requirements. Suppose a service process needs to replace a service **Sa** with a service **Sb**, which has a given security context. If the service context profile of **Sb** provides integrity, authentication, non-repudiation and confidentiality, the security context needs to be derived from the given context. In order to send a mobile message (Pay Confirmation Service in figure 1.1), display settings and the messaging context (TXT/ MMS support) of the client's device need to be derived from the device context, that is, the implicit context needs to be derived from the explicit context. Context categories have relationships between them. A change in one context instance may have an affect on other context instances.

In section 5.2, we address context model specifications and service context profiles in detail. We describe context manipulation operators including service-level context manipulation and process-level context manipulation in section 5.3. We describe context composition in section 5.4. In section 5.5, we describe context reasoning in detail and provide a case study analysis in section 5.6. Finally in section 5.7, we summarise the chapter proposing future work.

5.2 Context model specification and service context profiles

The manipulation and composition of dynamic aspects can be precisely described at DL-level. We introduced description logic formalisms in section 4.3. Before addressing the manipulation of context, the notion of a context specification and its semantics need to be made precise.

We assume context model to be a DL specification, $Context\ Model = \langle \Sigma, \Phi \rangle$, see section 4.2.4. For instance, for the *SecurityContext*, we define Σ and Φ as follows.

$\Sigma = \langle \{IntegrityContext, AuthenticationContext, ...\} ; \{hasPart, isPartOf\} \rangle$

$$\phi = \left\{ IntegrityContext \overset{isPartOf}{\rightarrow} SecurityContext \right\}$$

We assume in general the following signature inclusion $T \subset \Sigma$ for all signatures Σ where T is the **context taxonomy** signature, as defined in section 4.2.4.

$$\Sigma \; = < \{FunctionalContext, QoSContext, DomainContext, PlatformContext, ...\} \; ; \\ \{hasPart, isPartOf, hasCost, hasCurrency, hasSecurity, hasIntegrity, ...\} \; >$$

$$T \; = < \{FunctionalContext, QoSContext, DomainContext, PlatformContext, ...\} \; ; \\ \{hasPart, isPartOf, ...\} \; >$$

If the taxonomy is not adhered to or other changes or extensions take place, context modeling might require syntactical elements to be renamed.

The $Context\ Model = \langle \Sigma, \Phi \rangle$ can be interpreted by a set of models M. The model notion [Kozen 1990] refers to algebraic structures that satisfy all context descriptions ϕ in Φ. The set M contains algebraic structures $m \in M$ with

- instances C^I for each contextual concept (class) C,
- roles $R^I \subseteq C_i^I \times C_j^I$ for all context roles $R : C_i \rightarrow C_j$

such that m satisfies the context description. We define the satisfaction relation over the connectors of the description logic $\mathcal{SHOIN}(\mathcal{D})$.

We define the **Context specification**, which is application-specific and have instances and instance-level axioms as,

$$(Context\ specification \in Context\ Model), \text{ where } Context\ Model = \langle \Sigma, \Phi \rangle.$$

The **consistency** of a **context specification** ensures that a **context model** does not contain any contradictory facts. A **context specification** is consistent, if there are models that satisfy the specification. For example, based on the descriptions of a contextual concept, a reasoner can check whether or not it is possible for a concept to have any instances. The concept is deemed to be inconsistent if it cannot have any instances - section 5.5.2.

We use the notion of a **service context profile** (SCP) to extend the **context specification** notion towards a Web service. A **service context profile** captures **context model** instances of individual services, i.e. adding instance-level axioms to the

context specification. The **context model** provides a contextualization framework (Chapter 4), in which service-related context aspects are captured. The SCP is maintained at the service provider end. A service context profile is represented as an association of values (instances) to context model aspects

$$\begin{aligned} SCP = [&\{F(1)...F(n_F)\}, \\ &\{Q(1)...Q(n_Q)\}, \\ &\{D(1)...D(n_D)\}, \\ &\{P(1)...P(n_P)\}] \end{aligned}$$

where $\{F(1)...F(n_F)\}$ are functional context instances, $\{Q(1)...Q(n_Q)\}$ are quality of service context instances, $\{D(1)...D(n_D)\}$ are domain-based context instances, and $\{P(1)...P(n_P)\}$ are platform-based context instances. Each of the instance elements is typed by the respective context model aspect, see section 4.4.2. We use the following example to illustrate a SCP with a selected set of functional and QoS contexts.

```
<owl:Thing rdf:about="#HSBCBankingService">
    <rdf:type rdf:resource="#Service"/>

    Functional Context

        <hasProtocolContext rdf:resource="#Pro1-HSBCBankingService"/>
        <!-- http://.../ServiceContextOntology.owl#Pro1-HSBCBankingService -->
        <ProtocolContext rdf:about="#Pro1-HSBCBankingService">
                <rdf:type rdf:resource="&owl;Thing"/>
                <hasDefinition rdf:datatype="&xsd;string">
                    Pre-
                        EncryptionVerification ,
                        UserAccountVerification,
                        CurrencyTypeVerification.
                </hasDefinition>
        </ProtocolContext>

        <hasPreConditions rdf:resource="#UserAccountVerification"/>
        <!-- http://.../ServiceContextOntology.owl#UserAccountVerification -->
        <owl:Thing rdf:about="#UserAccountVerification">
                <rdf:type rdf:resource="#EffectContext"/>
        </owl:Thing>

        <hasInputContext rdf:resource="#transactionAmount"/>
        <!-- http://.../ServiceContextOntology.owl#transactionAmount -->
        <owl:Thing rdf:about="#transactionAmount">
            <rdf:type rdf:resource="#InputContext"/>
            <hasData rdf:datatype="&xsd;string">3000</hasData>
        <hasContext rdf:resource="#USD"/>
        </owl:Thing>

    QoS Context

        <hasCostContext rdf:resource="#0.1"/>
        <!-- http://.../ServiceContextOntology.owl#0.1 -->
        <owl:Thing rdf:about="#0.1">
            <rdf:type rdf:resource="#CostContext"/>
            <hasCurrencyContext rdf:resource="#GBP"/>
        </owl:Thing>

  </owl:Thing>
```

SCPs are used as inputs at validation monitoring, figure 7.1.

5.3 Context manipulation operators

We introduce context manipulation operators, before addressing composition operators that preserve the internal composition structure (i.e. are reversible). The consistency of context specifications is a concern. We have two perspectives of context manipulation. They are service-level and process-level perspectives. At service-level, we discuss context aspects relevant to individual services, e.g., manipulating different context aspects of single service. At process-level, we discuss context aspects relevant to contextualized service processes, e.g., manipulating single context aspect relevant to different services in a process. We define three fundamental context manipulation operators for service-level context manipulation. They are **Renaming**, **Restriction**, and **Refinement** operators. We also define two operators, **Union** and **Intersection** for process-level context manipulation. We discuss the consistency preservation of context specifications by the operators. We use DL-level formalisms described in section 4.3 to define context manipulation operators.

5.3.1 Service-level context manipulation

Renaming. If the taxonomy is not adhered to or other changes or extensions take place, context modeling might require syntactical elements to be renamed. A **Renaming** operator can be defined element-wise for a given signature Σ. By providing mappings for the elements that need to be modified, a new signature Σ' is defined,

$$\Sigma' \overset{\text{def}}{=} \Sigma\,[n_1 \mapsto n'_1,\ \ldots\ , n_m \mapsto n'_m]$$

for all concepts or roles $n_i(i = 1, \ldots, m)$ of Σ that need to be modified. For example, concepts `OSContext` is used instead of `PlatformContext` and roles `hasOperatingSystem` is used instead of `hasPlatform`.

$$\Sigma' = \Sigma\ [\ \{PlatformContext \mapsto OSContext\ \};$$
$$\{hasPlatform \mapsto hasOperatingSystem\ \}\ ]$$

Restriction. While context aspects are often used as is, it is sometimes desirable to focus on specific parts. Restriction is an operator that allows context combinations to be customised and undesired elements (and their roles) to be removed, A restriction can be expressed using the **Restriction** operator $\langle \Sigma, \Phi \rangle_{|\Sigma'}$ for a context specification, defined by

$$\langle \Sigma, \Phi \rangle_{|\Sigma'} \overset{\text{def}}{=} \langle \Sigma \cap \Sigma', \{\phi \in \Phi \mid rls(\phi) \in rls(\Sigma \cap \Sigma')\ \wedge cpts(\phi) \in cpts(\Sigma \cap \Sigma')\}\rangle$$

with the usual definition of role and concept projections $rls(\Sigma) = R$ and $cpts(\Sigma) = C$ on a signature $\Sigma = \langle C, R \rangle$. For example, if an integrity context of a service is a concern instead of the complete security context, then the results can be viewed as,

$$rls(\phi) = \{hasIntegrity\} \text{ and } cpts(\phi) = \{IntegrityContext\}$$

Restriction preserves consistency as constraints are, if necessary, removed. **Restriction** can be applied in combination with any context combinator such as **In-**

tersection, **Union** or **Refinement**.

Refinement. Consistency is a requirement that should apply to all combinations of ontologies. A typical situation is the derivation of a new context from an existing one [Baresi 2004]. We introduce a constructive operator **Refinement**, which is a consistent (i.e. consistency-preserving) extension in terms of contextual concepts and roles. The **Refinement** can be linked to the subsumption relation and semantically constrained by an inclusion of interpretations (models that interpret a context). The **Refinement** preserves existing roles, e.g. the satisfiability of the original context specification. As the original contextual concept and role types cannot be further constrained, the extension is consistent. Our consistency-preserving **Refinement** operator provides a constructive subsumption variant that allows,

- New subconcepts and new subroles to be added, and
- New constraints to be added, if these apply consistently to the new elements.

Assume a context specification $C = \langle \Sigma, \Phi \rangle$. For any specification $\langle \Sigma', \Phi' \rangle$ with $\Sigma \cap \Sigma' = \emptyset$, we define a **Refinement** of C by $\langle \Sigma', \Phi' \rangle$ through

$$C \oplus \langle \Sigma', \Phi' \rangle \overset{\text{def}}{=} \langle \Sigma + \Sigma', \Phi + \Phi' \rangle$$

The pre-condition $\Sigma \cap \Sigma' = \emptyset$ implies $\Phi \sqcap \Phi' = \bot$, i.e. consistency is preserved, which is an important property for dynamic, automated environments. In this situation, existing roles of $C = \langle \Sigma, \Phi \rangle$ are inherited by $C \oplus \langle \Sigma', \Phi' \rangle$. Existing roles can be refined as long as consistency is maintained, which might require manual proof in specific situations that go beyond the operator-based application. A **Refinement** operator can be used to adapt provider context to a context signature Σ' and a context description Φ', e.g, to add device aspects to the context specification $\langle \Sigma', \Phi' \rangle$ if the user's device context supports a given feature (example in figure 1.1),

$$\langle \Sigma', \Phi' \rangle \oplus \langle \{DeviceContext, FeatureContext\}, \{hasDevice, hasFeature\} \rangle$$

5.3.2 Process-level context manipulation

Adding a context specification to another specification (or removing specific context roles from a context specification) is often required, particularly if service contexts are combined within a process. The operators **Union** and **Intersection** deal with these situations, respectively. Two context specification $C_1 = \langle \Sigma_1, \Phi_1 \rangle$ and $C_2 = \langle \Sigma_2, \Phi_2 \rangle$ can be considered (generally associated to two different services) in a process.

- The **Intersection** of C_1 and C_2, expressed by $C_1 \cap C_2$, is defined by

$$C_1 \cap C_2 \quad \overset{\text{def}}{=} \quad \langle \Sigma_1 \cap \Sigma_2, (\Phi_1 \cup^+ \Phi_2)|_{\Sigma_1 \cap \Sigma_2} \rangle$$

  We describe the $\cup^+$ operator for context specification later in this section, which is defined on a case by case basis for different context aspects. The **Intersection** is semantically defined based on an intersection of context interpretations, achieved through projection onto common signature elements.

- The **Union** of C_1 and C_2, expressed by $C_1 \cup C_2$, is defined by

$$C_1 \cup C_2 \quad \overset{\text{def}}{=} \quad \langle \Sigma_1 \cup \Sigma_2, (\Phi_1 \cup^+ \Phi_2)|_{\Sigma_1 \cup \Sigma_2} \rangle$$

  Union is semantically defined based on a union of context interpretations.

Both **Union** and **Intersection** operations can result in consistency conflicts, but the combination of two context specifications of two services should be conflict-free, i.e. semantically, no contradictions should occur. A consistency condition can be verified by ensuring that the set-theoretic interpretations of two contexts C_1 and C_2 are not disjoint, $C_1^I \cap C_2^I \neq \emptyset$, i.e. their combination is satisfiable and no contradictions occur.

We describe the operator $\cup^{+}$ in terms of the types of context aspects involved. The combination mechanism, which is the functionality of the $\cup^{+}$ operator, differs between context aspects. We investigated all context aspects in our context model ontology to define a complete list of $\cup^{+}$ operators. $C(i)$ refers to the context aspect value of service i, e.g., $C(i)$ for the service i can equal to 600(ms) for the response time aspect.

- The **Lowest Common Denominator** ($\cup^{+}_{LCD}$)
  $\cup^{+}_{LCD} \overset{def}{\rightarrow} Min_{i=1}^{n} C(i)$ for all $C(i)$ in the ϕ.
  Example : for a security aspect, the overall security of a process is determined by the weakest security setting of all individual services.

- The **Least Common Subsumer** [Cohen 1992] ($\cup^{+}_{LCS}$)
  $\cup^{+}_{LCS} \overset{def}{\rightarrow} \bigcap_{i=1}^{n} C(i)$ for all $C(i)$ in the ϕ
  Example: for a language aspect, the least common subsumer of all individually used languages are the language(s) common to all (intersection).

- The **Logical OR** ($\cup^{+}_{OR}$)
  $\cup^{+}_{OR} \overset{def}{\rightarrow} \mathtt{OR}_{i=1}^{n} C(i)$ for all $C(i)$ in the ϕ
  Example: for the deployment environment, the service deployment environment needs secure internet connection or connection bandwidth greater than 10Mbps.

- The **Accumulation** ($\cup^{+}_{ACC}$)
  $\cup^{+}_{ACC} \overset{def}{\rightarrow} \Sigma_{i=1}^{n} C(i)$ for all $C(i)$ in the ϕ
  Example: The cost of a process is an accumulation through summation of the cost of each service.

- The **Logical AND** ($\cup^{+}_{AND}$)
  $\cup^{+}_{AND} \overset{def}{\rightarrow} \mathtt{AND}_{i=1}^{n} C(i)$ for all $C(i)$ in the ϕ
  Example: for the deployment environment, the service deployment environ-

ment needs Windows operating system and connection bandwidth greater than k Mbps.

- The **Mediation** $(\cup^{+}_{MED})$
  $\cup^{+}_{MED} \overset{def}{\rightarrow} \mathtt{MED}^{n}_{i=1} C(i)$ for all $C(i)$ in the ϕ
  Example: in service composition, if an output context (boolean, true or false) of a service S_j is composed with an input context (integer, 0 or 1) of a service S_{j+1} then a mediation is needed. Mediations are represented as mappings.

In order to illustrate this for a service process P, we assume P has two services S_i and S_j and corresponding context specifications SCP_i and SCP_j. Both specifications are characterised in terms of the context aspects in-parameter, out-parameter, response time, security and language.

$$SCP_i = [int, bool, 1ms, 1111, EN] \text{ and } SCP_j = [int \times int, int, 10ms, 1001, FR]$$

The aim is to combine the SCPs to process-level contexts,

- *in, out* - sequential composition, which is a causal structural composition (mediation). Correctness of this composition is a concern. We address this type of composition further in Section 5.4.
- *cost, performance* - numerical composition through addition (accumulative).
- *security* - the lowest common denominator, which is a kind of intersection for security settings.
- *language* - intersection as the composition principle.

The results of the combination can be illustrated as follows:
For a service process P,

$$P = \{S_i, S_j\}$$
$$\langle \Sigma, \Phi \rangle_p = \langle \Sigma, \Phi \rangle_{S_i} + \langle \Sigma, \Phi \rangle_{S_j}$$

$$[\text{int, bool, 1ms, 1111, EN}] + [\text{ int}\times\text{int, int, 10ms, 1001, FR }]$$

The composition can be illustrated as,

$$[\text{bool} \cup^{+}_{MED} \text{int} \times \text{int}], [\text{1ms} \cup^{+}_{ACC} \text{10ms}], [1111 \cup^{+}_{LCD} 1001], [\text{EN} \cup^{+}_{LCS} \text{FR}].$$

The results of the individual aspect combinations can be illustrated as,

$$[\text{1ms} \cup^{+}_{ACC} \text{10ms}] = \text{11 ms}, [1111 \cup^{+}_{LCD} 1001] = 1001.$$

5.4 Context composition

The explicit support for composition is important for service context profiles. Composition is also central for service context aspects. As an extension to the context manipulation operators, we introduce two types of composite elements for context specifications. In contrast to **Union** and **Intersection**, composition retains subcomponents as identifiable parts of the result and, therefore, makes composition reversible.

The subsumption is the central relationship in ontology languages, allowing context taxonomies to be defined in terms of subtype relationships [Baader 2003]. The composition is a fundamental relationship that describes the part-whole relationship between concepts or instances (individuals) [Priestley 2003]. Composition is less often used in ontological modeling languages. The notion of composition shall be applied for context in two different ways:

- *Structural (service-level) composition*. The structural hierarchies define an important aspect of context [Daconta 2003]. The structural composition can be applied for instance for input/output or for security with its sub-aspects confidentiality or availability. In the latter case, composition is more adequate than seeing these as subtypes if their later implementation through different system components is considered.

- *Sequential (process-level) composition*. Dynamic elements (services) can be composed to represent sequential process behavior. While context does not directly represent behavior, we have already seen that context models need to be aggregated along with the behavioral composition of services in a process.

We use the symbol "$\triangleright$" to express composition. The composition is syntactically used in the same way as subsumption "$\sqsubseteq$" to relate context descriptions.

- *Context composition hierarchies* can consist of unordered subcomponents, expressed using the component composition operator "$\triangleright$". An example is $Security \triangleright Confidentiality$, meaning that a *Security* aspect consists of *Confidentiality* as a part as in figure 4.1. The components can be interpreted by unordered multi-sets. The structural composition $C \triangleright \{D_1, \ldots, D_n\}$ is defined by $C \triangleright \{D_1\} \sqcap \ldots \sqcap C \triangleright \{D_n\}$ where $C \triangleright \{D\}$ means that C is structurally composed of D if C and D are context specifications. The parts $D_i, i = (1, .., n)$ are not assumed to be ordered. The structurally composed concepts are interpreted as multi-sets.

- *Service processes* can be *sequences* that consist of ordered process elements, again expressed using the composition operator "$\triangleright$". An example is $Process \triangleright Service$, meaning that *Process* is actually a composite service, which contains for instance a *Service* element. We see composite process implementations as being interpreted as ordered tuples providing a notion of sequence. More complex behavioral compositions are not covered here. The sequential composition $C \triangleright [D_1, \ldots, D_n]$ is defined by $C \triangleright [D_1] \sqcap \ldots \sqcap C \triangleright [D_n]$ where $C \triangleright [D]$ means that C is sequentially composed of D if C and D are services. The sequentially composed concepts are interpreted as tuples. The parts D_i with $i = (1, .., n)$ are assumed to be ordered with $D_1 \leq \ldots \leq D_i \leq \ldots \leq D_n$ describing an execution ordering $\leq$ on the D_i.

Note, that the composition operators are specific to the respective element types, whereas subsumption is generic. We allow the composition type delimiters

$\{\ldots\}$ and $[\ldots]$ to be omitted if the type of the part-element D is clear from the context.

While subsumption as a relationship is defined through subset inclusion, the composition relationships are defined through membership in collections (multisets for structural composition and tuples for behavioral composition).

5.5 Context reasoning

Context manipulation and composition can be further supported by context reasoning, which we detail in this section. Context reasoning is implemented as reasoning services, which support context manipulation operators and context composition described in section 5.3 and section 5.4. The context reasoning approaches used in most of the context-aware applications are ontology reasoning and rule based reasoning [Bikakis 2008]. Ontology reasoning approaches integrate well with the ontology model. Ontology reasoning takes advantage of concept relationships and role relationships (Object roles, Data type roles). The underlying description logic of the ontology models supports subsumption and consistency checking of available context information. Rule-based reasoning basically focuses on if-then-else type rule statements. Rule languages provide a formal model for context reasoning. They are easy to understand and can be reused. There are many systems that integrate rules with ontology models [Bikakis 2008]. The semantic Web rule language (SWRL) aims to be the standard rule language of the Semantic Web [Horrocks 2004a]. SWRL can be used for rule-based reasoning, since it is built on with OWL-DL (variable free and corresponding to $\mathcal{SHOIN}(\mathcal{D})$) and RuleML (variables are used) support. All rules can be expressed in terms of OWL concepts (classes, properties and individuals). SWRL includes a high-level abstract syntax for Horn-like rules [Horrocks 2004a]. SWRL-based rules can be saved as a part of the context model ontology, and reasoning can be supported by the standard tools - such as Pellet, which is the first sound and complete OWL-DL reasoner with extensive support for reasoning. Pellet also supports OWL/Rule hybrid reasoning [Sirin 2007].

We use description logic (ABox and TBox reasoning in chapter 4). There are three major constituents in our context reasoning. They are subsumption, consistency checking, and context derivation. However, a deeper investigation of the algorithms used in reasoning, and the trade-off between the complexity of the reasoning problems and the expressive power of the DL are out of our scope [Küsters 2000]. The context specifications described in section 5.2, context manipulations described in section 5.3, and context compositions described in section 5.4 are further supported by context reasoning.

5.5.1 Subsumption

Subsumption can be used to check whether or not a contextual concept is a subconcept of another concept. In other words, to check context instances evolution based on the hierarchical relationships of context in the context model ontology. Subsumption relationships can make some implicit context explicit.
C subsumes D :

$$\text{C} \sqsubseteq \text{D iff } C^I \subseteq D^I \text{ for all interpretations } I;$$
$$\text{C and D are concepts.}$$

In the context model ontology,

$$\text{QoSContext} \sqsupseteq \text{RuntimeContext} \sqsupseteq \text{PerformanceContext};$$

For example, we can take `t1` as an instance of the Performance concept (direct type). Then `t1` becomes indirectly typed with RuntimeContext and QoSContext. A service with good run-time context can infer that the service has a good performance based on $\cup^{+}_{LCD}$.

Subsumption is needed for context specifications in section 5.2, context manipulations in section 5.3 and context compositions in section 5.4. For example, subsumption is needed for the **Refinement** operator, mediation in the $\cup^{+}$ operator, structural composition, etc. Subsumption is also a major constituent in consistency checking. Subsumption plays a major role in context matching scenarios [Pahl 2011] where the service is required to be better than service user requirements, i.e. a service needs to be subsumed by the requirements of the service user. For instance, QoS values or functional types (in/out) of services need to be better than service user requirements. In practical terms, **Refinement** (in section 5.3.1) is a constructive operator that implies subsumption.

5.5.2 Consistency checking

Consistency checking ensures that an ontology does not contain any contradictory facts. Based on the descriptions of a contextual concept, the reasoner can check whether or not it is possible for a concept to have any instances. A concept is deemed to be inconsistent if it cannot possibly have any instances.

$$\exists a \in (C \sqcap D) \text{ iff } \neg(C \sqcap \neg D); \text{ a is an instance, C, D are concepts.}$$

For example,

$$(\text{ DeviceContext} \sqsubseteq \text{PDA}) \sqcap (\text{ DeviceContext} \sqsubseteq \text{Phone}) \sqcap \text{dis(PDA,Phone);}$$

dis() defines disjointness relation, section 4.3.3.1.

A device N97 cannot be an instance of both PDA and Phone concepts. In a contextualized service process, the context specifications (SCPs in section 5.2) need to be checked for the consistency with the context model ontology. The consistency checking is also needed for context manipulation (in section 5.3) and composition (in section 5.4). For example, if a contextual concept in a context specification is renamed, then the renamed contextual concept needs to be checked for consistency with the context model ontology (**Renaming** operator in section 5.3.1).

5.5.3 Context derivation

Context derivation is needed for context specifications (for example, SCPs in section 5.2) and dynamically generated context instances. **Context derivation** implies deriving implicit context from the explicit context in the context model ontology. These can be implemented as SWRL rules [Horrocks 2004a]. We use the rule-based context defined in section 4.3.3.3 for context derivation.

$$\text{Antecedent} \rightarrow \text{Consequent}$$

Antecedent and Consequent consist of one or more contextual concepts and their property descriptions.

The derived context affects other context and constraints, which we define using security context and response time context. Reasoning about SecurityContext (derived context) based on given explicit context of Integrity, Authentication, Non-Repudiation and Confidentiality is illustrated in table 5.1. Whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold.

Input	Rules	Service(?s)∧objectRoleHasIntegrity(?s, ?x) ∧objectRoleHasAuthentication(?s, ?x) ∧objectRoleHasConfidentiality(?s, ?x) ∧objectRoleHasNonRepudiation(?s, ?x) ∧swrlb:stringEqualIgnoreCase(?x, ”1”) →objectRoleHasSecurity(?s, ”1111”)
	Explicit Context	<Service rdf:ID="ServiceS1"> <objectRoleHasAuthentication rdf:resource="1"/> <objectRoleHasIntegrity rdf:resource="1"/> <objectRoleHasConfidentiality rdf:resource="1"/> <objectRoleHasNonRepudiation rdf:resource="1"/> </Service>
Output	Derived Context	<Service rdf:ID="ServiceS1"> <objectRoleHasSecurity rdf:resource="1111"/> </Service>

Table 5.1: Security context reasoning

The rule defined in table 5.1 can be read as, if a service has its integrity, authentication, confidentiality and non-repudiation equal to `1`, then the service has security setting `1111`. More details of the implementation can be found in appendix A.5.

In order to achieve a security setting `1111`, an extra processing time is needed for a service. This can be specified as a rule, if the security setting is `1111`, then the response time needs to be greater than 100 ms.

$$\text{Service(?s)} \wedge \text{ObjectRoleHasSecurity}(?s, ?x) \wedge$$
$$\text{swrlb:stringEqualIgnore-Case}(?x,\text{"1111"}) \wedge$$
$$\text{DataTypeRoleHasResponseTime}(?s, ?y) \rightarrow \text{swrlb:GreaterThan}(?y,\text{100ms}).$$

A service with security setting `1111` and very low response time (such as less than 10ms) is hard to achieve. These rules can also be seen as context constraints, which define requirements.

We can conclude that in a contextualized Web service process, if a context instance is generated at process run-time then

- This needs to be reasoned for consistency aspects based on the underlying description logic of the ontology in order to check the validity of the context instance.
- Context instances need to be checked with derived context rules. They are SWRL-based rules implemented within the ontology in order to broaden context information implicitly by introducing derived context deduced from the generated context instances.
- Derived context needs further reasoning for consistency depending on the requirements of the application.

These steps can be iteratively used at process run-time.

5.6 Case study

We consider some aspects from the example in figure 1.1 and illustrate different aspects associated to operator calculus, composition and reasoning techniques.

We assume the *Pay Confirmation Service* invokes a *Mobile Service* or *Email Service* based on the device context of the user.

Renaming operator. The post-condition attached to the context specification of the *Pay Confirmation Service* needs to be renamed to make the specification consistent with the context model ontology used at the Broker.

$\langle \Sigma, \Phi \rangle|_{\Sigma'} = \{hasConditionPost \rightarrow hasPostCondition\}$

Restriction operator. The Security context of the *Pay Confirmation Service* is restricted to the Integrity context.

$\langle \Sigma, \Phi \rangle|_{\Sigma'} = \{SecurityContext \rightarrow IntegrityContext\}$

Refinement operator. We assume that the *Pay Confirmation Service* invokes a *Mobile Service* or *Email Service* based on the device context of the user. We assume the pay confirmation output is a MMS message and a user profile is defined in the domain context. Here, the *Pay Confirmation Service* is using domain context where services are composed, deployed and executed. A post-condition of the *Pay Confirmation Service* needs to check the user device (defined in the user profile) for the MMS feature. If the user device supports the MMS feature, then the *Mobile Serivce* can be invoked. In here, the **Refinement operator** needs to process relevant $\langle \Sigma', \Phi' \rangle$ defined in the domain context specification and mobile device context specification while keeping the consistency to derive the MMS feature (for example, if device is N95 which has the MMS feature). The explicit context is user device N95, which can be found in the UserContext in the domain context and the implicit context is MMS, which needs to be derived - described in section 5.5.

$\Sigma' = \langle \{UserContext, DeviceContext, FeatureContext\}\ ; \{hasDevice, hasFeature\} \rangle$

$\Phi' = \left\{ UserContext \overset{hasDevice}{\rightarrow} DeviceContext, DeviceContext \overset{hasFeature}{\rightarrow} FeatureContext \right\}$

Intersection operator. The overall security of a process is the weakest security of all constituent services. This can be achieved by applying the Intersection operator with $\cup^{+}_{LCD}$ for context specifications attached to the three services. If we assume

the security of *Billing Service*, *Banking Service* and *Pay Confirmation Service* as 1100, 1111 and 1001 respectively,
The overall security of the process is [1100 $\cup^{+}_{LCD}$1111 $\cup^{+}_{LCD}$ 1001] = [1000].

Union operator. If the costs of *Billing Service*, *Banking Service* and *Pay Confirmation Service* are given in the context specifications $SCP_1.Cost$, $SCP_2.Cost$ and $SCP_3.Cost$ attached to each service, then the process cost can be calculated by applying the Union operator with $\cup^{+}_{ACC}$ for context specifications. If $SCP_1.Cost$=0.1, $SCP_2.Cost$=0.2 and $SCP_3.Cost$=0.1, then the cost of the process is [0.1 $\cup^{+}_{ACC}$ 0.2 $\cup^{+}_{ACC}$ 0.1] = [0.4].

Structural composition. The security context of the *Banking Service* is the structural composition of Integrity context, Authentication context, Confidentiality context and Non-repudiation context as in section 4.2.4.

$$\text{Security "}\triangleright\text{" } \{ \text{ Integrity, Authentication, Confidentiality, Non-repudiation } \}$$

Sequential composition. Composing context specifications attached to *Billing Service*, *Banking Service* can be helpful in dynamically composing a *Pay Confirmation Service*, which does not violate the required process cost defined by the Broker.

Context derivation. We assume that the *Pay Confirmation Service* invokes a *Mobile Service* or *Email Service* based on the device context of the user (if the user device supports the MMS feature). That is, based on a user device (e.g., N95), the device context (MMS) needs to be derived from the device ontology.

Consistency checking. The derived context from the device ontology (e.g., MMS feature) needs to be consistent with the context model specification used at the Broker, otherwise manipulation operators such as Renaming need to be applied. Consistency checking is necessary at the semantic client (or broker), where the service process is composed, deployed and executed.

5.7 Chapter summary

The context model ontology itself is not enough for context-aware Web service applications and it is necessary to have context manipulation, context composition and context reasoning for context specifications, because context specifications need to be statically or dynamically adapted for dynamic requirements by the software architects. This chapter makes precise the notion of context model specifications and their semantics. We defined the service context profile, which is an instance level context specification for a Web service. The context manipulation operators are introduced for context specifications having service-level and process-level perspectives. The dynamic service composition needs service-level and process-level context compositions in a contextualized service process. We detailed the context reasoning attached to contextualized service processes. Context manipulation, composition and reasoning about context specifications support dynamic requirements validation monitoring described in chapter 6 and 7.

Chapter 6

Context constraints

Contents

6.1 Introduction

We modeled dynamic service context in a context model ontology in chapter 4. We now generate context constraints in terms of dynamic service context to represent dynamic requirements attached to composition and execution of services at process run-time. A requirement relevant to one dynamic aspect or combination of dynamic aspects is called a dynamic requirement. Dynamic requirements are attached to service process run-time. We assume dynamic requirements are captured from SLAs (Service-Level Agreements). A service-level agreement is a formal definition of an agreement that exists between a service provider and a customer [Verma 2004]. A service process has services combined from heterogenous domains. A dynamic service process is always assumed to be a dynamically composed service process in which services are combined at process run-time. The context (defined environment) imposes constraints on the subsystem's behavior [Cheung 1996, Graf 1991]. The authors in [Cheung 1996, Graf 1991] define context constraints as restrictions imposed by the environment on subsystem behavior. In this thesis, context constraints (CC) are specified as assertions, which are restrictions imposed by the dynamic service context for dynamic requirements.

The context constraints generation from dynamic requirements is illustrated in figure 6.1. WSDL defines an interface of a constituent service. The context con-

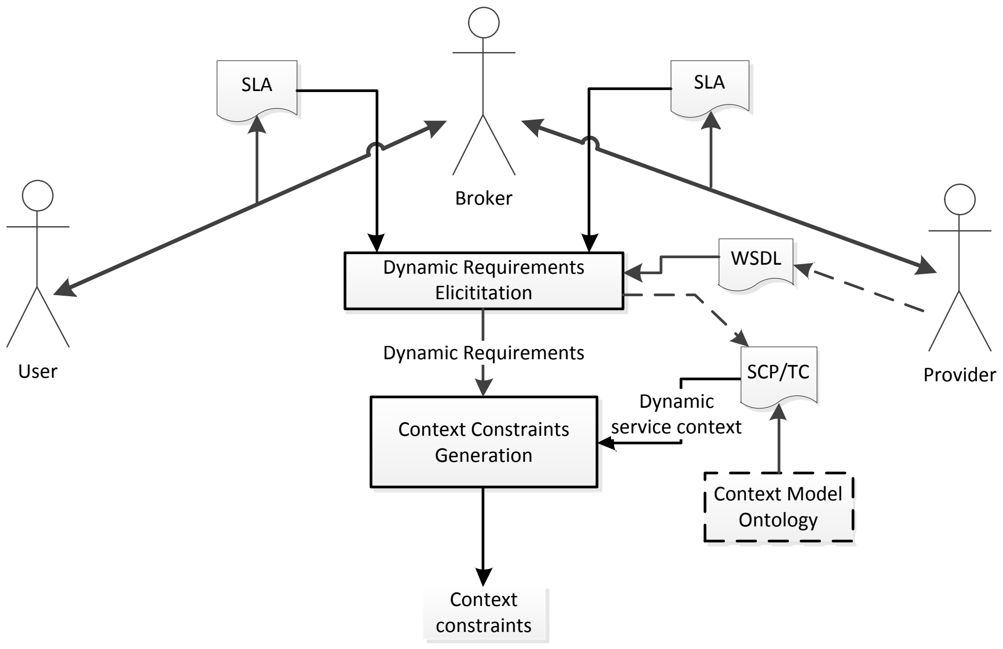


Figure 6.1: Dynamic requirements to context constraints

straints generation process has two inputs; dynamic requirements and dynamic service context. Dynamic service contexts are specified in service context profiles (SCP), which are specified by the providers or transient context specifications (TC). The derived context and monitored context are specified in TCs. The operator calculus and context reasoning techniques described in chapter 5 can be used to derive context aspects, which can be specified in TCs. SCP and TC are instance-level specifications of the context model ontology. We assume, service-level agreements (SLAs) are defined between user and broker, and between broker and provider. Dynamic requirements are captured from SLAs at the broker by the dynamic requirement elicitation process. The dynamic requirement elicitation process is assumed to be an automated process, which reads SLAs and outputs dynamic requirements. Context constraints generation process outputs context constraints (CC).

Constraint representation modules can define constraints within ontologies, such as in [Aggarwal 2004, Boukadi 2008]. However, defining dynamic requirements as context constraints within an ontology has major drawbacks as outlined in section 2.4.1. Context reasoning is more costly than constraints validation against context specifications, which are small in sizes. In our context constraint generation process, the context constraints are separated from the context model ontology, while preserving the required ontology supported capabilities. Consequently, context constraints need to be modeled preserving the useful features of the context model ontology, a generation process is needed to generate context constraints, the generation process needs to be defined in an algorithm and a constraints implementation language is needed.

We explore context constraints having the objective of dynamic service context validation at Web service process run-time. We describe context constraints modeling in section 6.2. We describe context model utilisation and constraints modeling using a class model in figure 6.3. In section 6.3, we detail context constraints generation including a generation process and an algorithm. The tool support for constraint generation is also described in this section. In section 6.4, we use a real world case study scenario to illustrate the applicability of context constraints. Finally, we conclude the chapter in section 6.5, also presenting future work.

6.2 Context constraints modeling

In this section, we describe context model utilisation and a class model for context constraints modeling. Context constraints are generated from dynamic requirements. The authors in [Verma 2004] define what the customer can expect from the provider, the obligations, and functional and quality of service properties in SLAs. Our usage of SLAs is more specific. We assume dynamic requirements are captured from SLAs in terms of dynamic service context to define context constraints, see figure 6.1.

6.2.1 Context model utilisation for constraints

We describe dynamic service context and the relation of dynamic service context to client (for example, client specification) and provider (for example, service specification) in figure 6.2.

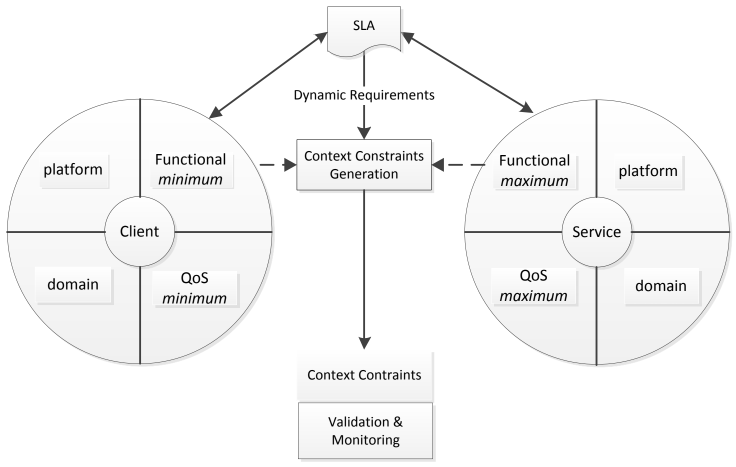


Figure 6.2: Context model utilisation for constraints generation

The context model ontology in chapter 4 provides conceptualized dynamic aspects, which can be used to define dynamically changing requirements in service processes. Consequently, the context model specifications, their applications in the form of service context profiles and operators on context specifications (in chapter 5) need to be supported at run-time through an integration with a service process execution. Context specifications can be attached to client and service provider sides. Service context profiles on the server-side define the provider specifications about services. The service context profiles can be frequently updated. The client-side specifications define the minimum requirements, whereas the provider-side specifications define the maximum range of capabilities. The server and client-side specifications are attached to service-level agreements (SLAs). Dynamic requirements can be captured from SLAs. Dynamically, the client requirements (minimum expectations) need to be validated against profile and transient context, which are instance-level specifications of context model ontology in chapter 4. The corresponding constraints are generated, which we detail in this chapter.

6.2.2 Class model for context constraints modeling

The context model utilisation overview in figure 6.2 is detailed towards context constraints modeling. The main concern is describing major constituents and their relationships attached to context constraints. A class model for Web service context constraints modeling is illustrated in figure 6.3.

Dynamic requirements attached to dynamic service context are the main concerns for constraints modeling. For example, requirements defined in SLAs (server and client-side specifications define service-level agreements (SLA)) in figure 6.2, such as response time of a service needs to be less than 100 milliseconds, cost of a service needs to be less than 0.2 USD, etc. Each service has a service context profile, for example, provider-side **service specifications** in figure 6.2. A service context profile characterises dynamic service contexts, which are formalised in a context model ontology in chapter 4. Context constraints (CCs) are associ-

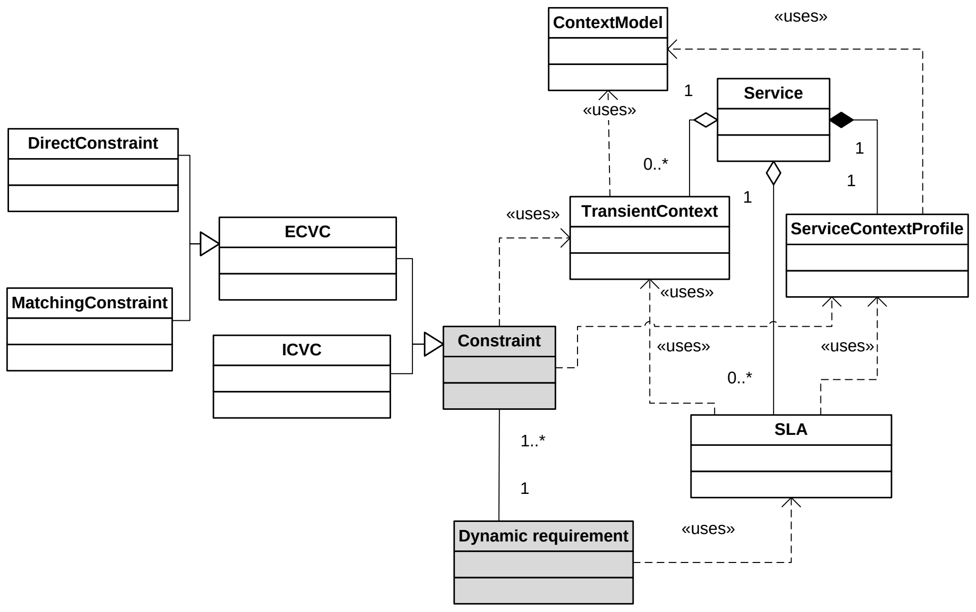


Figure 6.3: A class model for context constraints modeling

ated with dynamic service contexts, which can be transient contexts or contexts extracted from SCPs. A context constraint can be an explicit context validation constraint (ECVC) or implicit context validation constraint (ICVC). The explicit context validation constraints are further categorised into direct validation constraints and context matching constraints. We define and detail these constraints in section 6.2.3.

6.2.3 Context constraints (CC)

Context constraints define dynamic requirements attached to service process execution time. Context categories can have implicit and explicit relationships, which can be modelled in a service context model, chapter 4. A context category, which is availabe in a SCP is called an explicit context category. A context category which is implicitly connected to an explicit context category is called an implicit context category. The explicit context can be validated using context specifications, but implicit context validation needs context reasoning, such as device context and device feature context in figure 1.1. We define two types of context constraints to validate explicit and implicit contexts using different validation techniques.

- Explicit context validation constraints (ECVC)
- Implicit context validation constraints (ICVC)

ECVCs are defined to validate the explicit context in context specifications (e.g., SCP, TC). ICVCs are defined to validate the implicit context attached to context specifications (e.g., TC). However, most of the dynamic requirements are attached to explicit context.

Explicit Context validation constraints (ECVC)

The explicit context validation constraints work as assertions, which provide true or false results at validation. A ECVC can be a direct validation constraint or a context matching constraint. In a Web service process execution, an ECVC can be defined as direct context instances validation against a pre-defined constraint value (defined in SLAs) or context instances matching in two services. In both cases, the constraint returns true or false. They are assertions (boolean expressions).

Direct validation constraints. Direct validation constraints validate context instances against pre-defined constraints agreed by service-level agreements (SLAs).

If a ECVC is of the form `predicate`, context, expression;

(`Predicate`, Context, ϕ)

then the *ECVC* is called a direct validation constraint. Here, the expression ϕ is a pre-defined constraint value or a context instance. The predicate can be equal, less, greater, same, lessOrEqual, greaterOrEqual, or notEqual as in section 6.3.1.

The pre-defined constraint values are defined in SLAs. For example, we define a service cost based constraint for the illustration purpose. The cost of a service can be in different currencies and different values. If cost of a service in GBP then it should be less than or equal to 0.2, or if cost of a service in Euro then it should be less than or equal to 0.3, or if cost of a service in USD then it should be less than or equal to 0.4. The direct validation constraint can be defined as follows.

Functions: FCurrency(Service,CostContext) returns currency type of cost context, FCost(x) returns cost of service x. Then the cost context constraint for a Web service,

$\forall\ x : Service.[$

$(\texttt{equal}, FCurrency(x, CostContext), "GBP") \Rightarrow (\texttt{lessOrEqual}, FCost(x), 0.2)$

$\vee$

$(\texttt{equal}, FCurrency(x, CostContext), "Euro") \Rightarrow (\texttt{lessOrEqual}, FCost(x), 0.3)$

$\vee$

$(\texttt{equal}, FCurrency(x, CostContext), "USD") \Rightarrow (\texttt{lessOrEqual}, FCost(x), 0.4)\ ]$

This constraint becomes true or false depending on the cost context of the constituent services. That is, these constraints work as assertions at validation.

Context matching constraints. These constraints match context instances of two services.

If a ECVC is of the form `equal`, context, context;

(`equal`, Service[i].Context[j], Service[i+1].Context[j])

then the *ECVC* is called a context matching constraint. Here, j and i define context category and service identification, respectively.

Context matching constraints match context instances of one context category but belong to two services. For example, an output context of one service may be needed to match with an input context of another service. Currency context of the output parameter of service x_i needs to be matched with that of the input parameter k of service x_{i+1}.

Functions: FCurrencyContext(z) returns the currency context of parameter z, FOutput(x) returns the output parameter of service x, FInput(x_i^k) returns the input parameter k of service x_i.

The Input and Output context matching constraint,

$\forall\ x_i\ , x_{i+1} : Service.$

$[\texttt{equal}, FCurrencyContext(FOutput(x_i)), FCurrencyContext(FInput(x_{i+1}^k))]$

This constraint matches context instances and outputs true or false at validation. When a constraint fails at validation, a failure report needs to be generated.

Implicit Context Validation Constraints (ICVC)

The context model ontology formalises both explicit and implicit relationships between context categories. The implicit contexts can be derived based on these relationships. In some situations, these implicit contexts attached to some requirements need to be validated at process runtime. These we define as implicit context validation constraints.

ICVCs are bundled as reasoning services. For example, if an ICVC needs to validate an implicit context from an explicit context, then a reasoning service, which can derive implicit context instances, is needed. An ICVC can be bundled with more than one reasoning service. For example, an ICVC is bundled with a device feature reasoning service and a device connection reasoning service, appendix C. The device feature reasoning service is used to validate any given feature (e.g. MMS) in a given device (e.g. Mobile device NX). The device connection reasoning service is used to validate any given feature (e.g. MMS) in a given connection (Mobile connection). A context category can have implicit relationships with other context categories and these implicit relationships need to be implemented in reasoning services. That is an ICVC attached to a context category is bundled with a set of reasoning services attached to implicit relationships. We assume the required reasoning services are maintained at the broker. The reasoning services can be developed as assertions (e.g., as in section 6.4.2). In this work, we develop a collection of reasoning services to illustrate the needs of context reasoning for real world case study scenarios, section 6.4.2.

6.3 Context constraints generation

The main focus of this section is defining a process for context constraints generation and proposing a language for defining context constraints. The context constraints need to be defined in terms of dynamic service context aspects, which are defined in the context model ontology so that they can be validated with ontology-based context specifications and transient contexts within a service process. A process (**ECVCGProcess**) and an algorithm (**ECVCGAlgorithm**) are proposed to describe ECVC generation. ICVC generation is described using reasoning services.

6.3.1 Tool support

An expressive constraint specification language is needed with low validation and analysis cost, particularly a large constraint set demands formal analysis for consistency checking to resolve contradicting constraints. The complexity of a specification language must not become an obstacle for constraints specification and validation of processes against constraints [Ly 2008]. The core technologies used in Web service frameworks, such as Web Service Description Language (WSDL), Universal Description Discovery and Integration (UDDI), Business Process Execution Language (BPEL) [Jordan 2007], and Simple Object Access Protocol (SOAP) are XML-based, hence XML-based constraints can bring consistency for service-based applications. We use the Message Automation Workbench [1] as the ECVC generation tool to generate CLiX-based (Constraint Language in XML) context validation constraints [2] [Ahmed 2009]. OWL is the W3C standard for developing ontologies and OWL-API is used to implement reasoning services, which validate implicit context.

[1] http://www.messageautomation.com/

[2] http://clixml.sourceforge.net/

Constraint language in XML (CLiX)

XML (Extensible Markup Language) is used to structure, store and transport information [Bray 2008]. CLiX (Constraint Language in XML) is an XML-based constraint language with a high expressive power. CLiX is implementable, validatable and compatible with other Web service technologies [Jungo 2007, Nentwich 2002]. CLiX combines XML, XPath and first order logic. CLiX has language elements such as quantifiers (forall, exists), operators (and, or, not, iff, implies) and predicates (equal,less, greater, same, lessOrEqual, greaterOrEqual, notEqual) [Jungo 2007, Nentwich 2005, Dui 2003]. We implement ECVCs using these quantifiers, operators and predicates in sections 6.3.3 and 6.4.2. For example, the cost of a service needs to be less than or equal to 0.2 GBP can be defined in CLiX as follows,

```
<?xml version="1.0" encoding="UTF-8"?>
<rules xsi:schemaLocation="http://www.clixml.org/clix/1.0 clix.xsd"
     version="1.1" xmlns="http://www.clixml.org/clix/1.0"
     xmlns:macro="http://www.clixml.org/clix/1.0/Macro"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
     <rule id="rule-1">
          <forall var="service" in="/Context/Service">
               <and>
                    <equal op1="$service/Cost/@currency" op2="GBP"/>
                    <lessOrEqual op1="$service/Cost" op2="0.2"/>
               </and>
          </forall>
     </rule>
</rules>
```

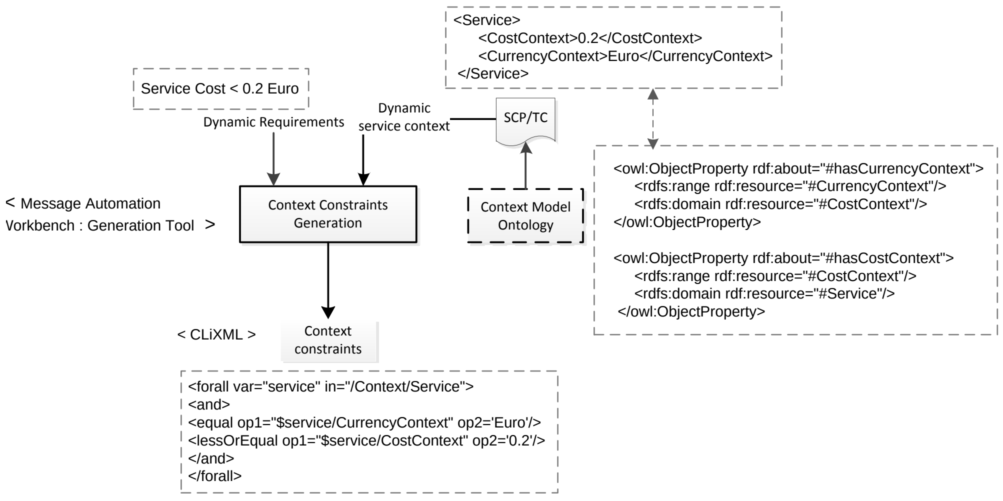
The currency context of cost context is defined as "*/Cost/@currency*" and the cost context of a service is defined as "*service/Cost*".

Message Automation Workbench

The Message Automation Workbench can be used to define complex CLiX constraints. The Message Automation Workbench supports for creating, editing, and validating constraints for data defined in XML. This workbench has features, such as a graphical rule editor, an XPath editor with a recoding mode to automate rule writing, a rule set editor to pick and mix rules, etc.

6.3.2 ECVC generation process (ECVCGProcess)

The ECVCGProcess in figure 6.4 describes ECVCs generation for dynamic requirements. ECVCs can be attached to more than one context category. In this



```
<Service>
    <CostContext>0.2</CostContext>
    <CurrencyContext>Euro</CurrencyContext>
</Service>
```

```
<owl:ObjectProperty rdf:about="#hasCurrencyContext">
    <rdfs:range rdf:resource="#CurrencyContext"/>
    <rdfs:domain rdf:resource="#CostContext"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasCostContext">
    <rdfs:range rdf:resource="#CostContext"/>
    <rdfs:domain rdf:resource="#Service"/>
 </owl:ObjectProperty>
```

```
<forall var="service" in="/Context/Service">
<and>
<equal op1="$service/CurrencyContext" op2='Euro'/>
<lessOrEqual op1="$service/CostContext" op2='0.2'/>
</and>
</forall>
```

Figure 6.4: Explicit Context validation constraints generation process (ECVCGProcess)

constraint generation, dynamic service contexts and dynamic requirements defined in SLAs are inputs. Dynamic service contexts are captured from an SCP or transient contexts (includes derived context), which are attached to the context model ontology. The constraint generation process generates and outputs context constraints in CLiX. We formalised the ECVC generation process in an algorithm, see Algorithm 6.1. We use the Message Automation Workbench as the ECVC generation tool.

6.3.3 ECVC generation algorithm (ECVCGAlgorithm)

We define an algorithm to describe the ECVC generation process, algorithm 6.1.

Algorithm 6.1 : Explicit context validation constraints generation algorithm (ECVCGAlgorithm)

```
Input : $Process(P)$,$ContextOntology(url)$
Output : $ExplicitContextValidationConstraint(ECVC)$
1:  CMO ← getCMO($url$)
2:  SLA[]← getSLA($P$)
3:  SCP[] ← getSCP($P, CMO$)
4:  DR[] ← elicit-DR($SLA[]$)
5:  for (each $DR$) do
6:      scope ← getScope($DR$)
7:      if $(scope = SL)$ then
8:          context[] ← getContext($DR, SCP[]$)
9:          predicate[] ← getPredicate($DR$)
10:         constraintValue[] ← getConstraintValue($DR$)
11:         ECVC ← createContextConstraint($predicate[], context[], constraintValue[]$)
12:     else if $(scope = PL)$ then
13:         context ← getContext($DR, SCP[]$)
14:         predicate ← getPredicate($DR$)
15:         constraintValue ← getConstraintValue($DR$)
16:         ECVC ← createContextConstraint($predicate, context, constraintValue$)
17:     end if
18: end for
```

We treat service-level constraints and process-level constraints separately in the algorithm. The abbreviations used in the algorithm are SLA: service-level agreement, DR: dynamic requirement, P: service process, *url*: web link to the context model ontology, CMO: context model ontology, SL: service-level, PL: process-level, and SCP: service context profile. The method *getCMO()* returns an instance of context model ontology, *getSLA()* returns SLAs attached to a process, *elicit-DR()* returns DRs attached to SLAs. *getScope()* returns scope of a requirement, *getSCP()* returns SCPs of a process, *getContext()* returns contexts attached to a requirement, *getPredicate()* returns predicates attached to a requirement, *getConstraintValue()* returns constraint values defined in a requirement. The *createContextConstraint()* returns validatable context constraints, which can be direct constraints generated from *createDirectConstraint()* and matching constraints generated from *createMatchingConstraint()*. The methods *createDirectConstraint()* and *createMatchingConstraint()* are built-in methods within *createContextConstraint()*.

The behavior of *createDirectConstraint()* can be illustrated as pair-wise combinations,

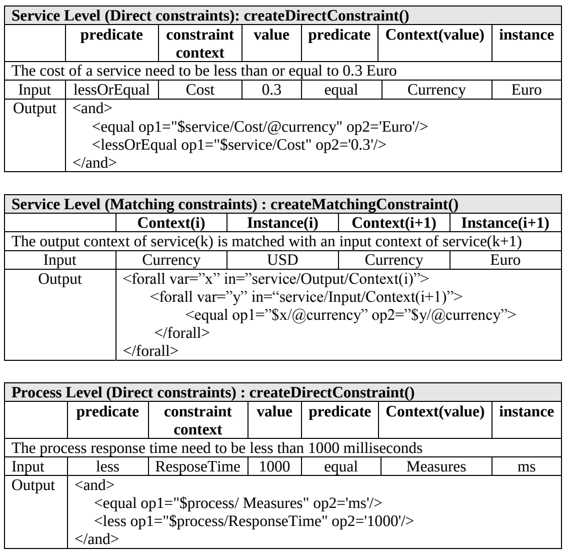
```
< $\otimes$ >
    < predicate , Context, $\phi$ >
    < predicate , Context , $\phi$ >
< $\otimes$ >
```

The expression ϕ is a pre-defined constraint value or a context instance, and the $\otimes$ is an operator such as AND, OR and NOT as in section 6.3.1. The predicates, which are used in the constraint language are detailed in section 6.3.1.

The behavior of *createMatchingConstraint()* can be illustrated as,

```
for all Context(i)
    for all Context(j)
        < equal, Instance(Context(i)), Instance(Context(j))>
```

The "equal" is a predicate defined in CLiX, section 6.3.1. Context(i) returns a contextual concept (called context) defined in the context model ontology. Instance(Context) returns an instance of the context. A working example for these two methods is illustrated in figure 6.5.



Service Level (Direct constraints): createDirectConstraint()						
	predicate	**constraint context**	**value**	**predicate**	**Context(value)**	**instance**
The cost of a service need to be less than or equal to 0.3 Euro						
Input	lessOrEqual	Cost	0.3	equal	Currency	Euro
Output	<and> <equal op1="$service/Cost/@currency" op2='Euro'/> <lessOrEqual op1="$service/Cost" op2='0.3'/> </and>					

Service Level (Matching constraints) : createMatchingConstraint()				
	Context(i)	**Instance(i)**	**Context(i+1)**	**Instance(i+1)**
The output context of service(k) is matched with an input context of service(k+1)				
Input	Currency	USD	Currency	Euro
Output	<forall var="x" in="service/Output/Context(i)"> <forall var="y" in="service/Input/Context(i+1)"> <equal op1="$x/@currency" op2="$y/@currency"> </forall> </forall>			

Process Level (Direct constraints) : createDirectConstraint()						
	predicate	**constraint context**	**value**	**predicate**	**Context(value)**	**instance**
The process response time need to be less than 1000 milliseconds						
Input	less	ResposeTime	1000	equal	Measures	ms
Output	<and> <equal op1="$process/ Measures" op2='ms'/> <less op1="$process/ResponseTime" op2='1000'/> </and>					

Figure 6.5: ECVC generation examples

The most important factor affecting the performance of an algorithm is normally the size of the input [Shaffer 1998]. For a given input size n (number of dynamic requirements), the time T of the algorithm is a function of n, written as $T(n)$. For each requirement, this algorithm executes once. We can define execution time, $T(n) = kn$ where k is a real number and its value depends on the constraint type. The time complexity of the algorithm is of order n, $O(n)$, which is acceptable.

6.3.4 ICVCs generation

In this thesis, we assume ICVCs use reasoning services, which can be dynamically invoked within a context-aware service process at run-time when required. The ontology web language (OWL) is used to conceptualise context aspects in a context model ontology, chapter 4. The OWL-API alone with Web services technologies (section 6.3.1) are used to implement reasoning services. ICVCs do not use the message automation workbench or CLiX, because those technologies are not strong enough for implicit context validation.

6.4 Case study - Context constraints generation

In this section, we use case study scenarios to exemplify the ECVC generation process and describe the ICVC implementation as context reasoning services.

6.4.1 Tool support

We implement the context model ontology using the protege 4.0.2 ontology editor. We use OWL formalisations and the OWL API to implement reasoning services in the NetBeans IDE 6.9.1 environment. Context validation constraints are generated using CLiX syntax, section 6.3.1. We use the Message Automation Workbench as the generation tool, section 6.3.1. At this stage, we manually generate context constraints using above languages and tools.

6.4.2 Case study

We consider a scenario where a broker (semantic client) has requirements such as,

- Example 1: constituent services of a process need to satisfy service cost requirements
- Example 2: context matching for input and output parameters of constituent services
- Example 3: the service process dynamically decides sending results as a mobile message (MMS message) or as an email. In the case of a mobile message, it is necessary to check whether the user's device supports the MMS feature (context in outwards perspective in chapter 4)

These requirements need to be validated and monitored in a service process instance at run-time. Consequently, the required CVCs need to be generated and the reasoning services need to be implemented.

CLiX rule definition

We implement ECVCs as CLiX rules. We define CLiX rules in the following structure, including constraint failure report definition and constraint definition.

```
<rule id = "r1">
    <failure report definition>
    <constraint definition>
</rule>
```

Direct validation constraint

Example 1. Service cost requirement validation. The cost of a service can be defined in a SCP as in section 5.2. The dynamic service context aspects, for example cost context, are defined in a context model ontology as in section 4.3. A context constraint needs to be generated for a service cost requirement in terms of dynamic service context.

$\forall\ x : Service.$

$[(\texttt{equal}, FCurrency(x, Cost), "GBP") \Rightarrow (\texttt{lessOrEqual}, FCost(x), 0.2)$

$\vee$

$(\texttt{equal}, FCurrency(x, Cost), "Euro") \Rightarrow (\texttt{lessOrEqual}, FCost(x), 0.3)\ ]$

The inputs and outputs of the constraint generation process can be described as follows. The input is a dynamic requirement captured from an SLA that defines a service cost requirement. The output is a CLiX-based constraint, which is defined in terms of dynamic service context.

Input:

```
<Service>
     <Cost>
         <Currency type="GBP">
              <Predicate> lessThanOrEqual </Predicate>
              <ConstraintValue>  0.2 </ConstraintValue>
         </Currency>
     </Cost>
     <Cost>
         <Currency type="Euro">
              <Predicate> lessThanOrEqual </Predicate>
              <ConstraintValue>  0.3 </ConstraintValue>
         </Currency>
     </Cost>
</Service>
```

Output: (CLiX-based constraint) The direct validation constraint is generated as,

```
<?xml version="1.0" encoding="UTF-8"?>
<rules xsi:schemaLocation="http://www.clixml.org/clix/1.0 clix.xsd"
    version="1.1" xmlns="http://www.clixml.org/clix/1.0"
    xmlns:macro="http://www.clixml.org/clix/1.0/Macro"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <rule id="rule-1">

        <report>%@=$service/Cost%
                %@=$service/Cost/@currency%
        </report>

        <forall var="service" in="/Context/Service">
            <or>
                <and>
                    <equal op1="$service/Cost/@currency" op2="'GBP'"/>
                    <lessOrEqual op1="$service/Cost" op2="0.2"/>
                </and>
                <and>
                    <equal op1="$service/Cost/@currency" op2="'Euro'"/>
                    <lessOrEqual op1="$service" op2="0.3"/>
                </and>
            </or>
        </forall>
    </rule>

</rules>
```

Context matching constraint

Example 2. Currency context matching of the output parameter of S_i with an input parameter of S_{i+1}.

$\forall\ x_i, x_{i+1} : Service.$

$[\texttt{equal}, FCurrencyContext(FOutput(x_i)), FCurrencyContext(FInput(x^k_{i+1}))\ ]$

The inputs and outputs of the constraints generation process can be described as follows.

Input:

```
<Service>
  <ID id=S(i)>
      <Output>
        <Context>Currency</Context>
        <Instance>USD</Instance>
    </Output>
  </ID>
  <ID id=S(i+1)>
      <Input>
        <Context>Currency</Context>
        <Instance>GBP</Instance>
    </Input>
  </ID>
</Service>
```

Output: (CLiX-based constraint) The context matching constraint is generated as,

```
<?xml version="1.0" encoding="UTF-8"?>
<rules xsi:schemaLocation="http://www.clixml.org/clix/1.0 clix.xsd"
    version="1.1" xmlns="http://www.clixml.org/clix/1.0"
    xmlns:macro="http://www.clixml.org/clix/1.0/Macro"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <rule id="MatchingRule-1">

        <report>
            <Invoke>
                <CurrencyConversion>
                    <From> @=$x/@currency </From>
                    <To> @=$y/@currency </To>
                </CurrencyConversion>
            </Invoke>
        </report>

        <forall var="x" in="/Context/Service/Output">
            <forall var="y" in="/Context/Service/Input">
                <equal op1="$x/@currency" op2="$y/@currency"/>
            </forall>
        </forall>

    </rule>

</rules>
```

Implicit context validation constraints

Implicit context validation constraints are implemented as a collection of context reasoning services and can be available through a reasoning service registry. In this work, each context reasoning service performs reasoning in a controlled environment. For example, a reasoning service reasons for a given feature of a given device, using a given ontology.

Example 3. A service process at a broker needs to send a process response as a

MMS message to a user. A user has agreed with a broker (SLA) to use a N95 mobile device for his communication. Then, an ICVC to validate whether the user device supports the feature MMS is needed at the broker. The device context needs to be determined to derive such an implicit relation. We assume such reasoning services are implemented and registered in a reasoning service registry. The required reasoning service need to be discovered from reasoning service registry at the broker.

Input (request): A service for reasoning and validate a device feature.

Output (response): End point of a reasoning service for the request.

A code excerpt of a reasoning service available in the reasoning service registry for the above example can be illustrated as follows,

```
// Create the manager that we will use to load ontologies.
OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
// load an ontology
File file = new File("Uri-Ontology");
OWLOntology ontology = manager.loadOntologyFromOntologyDocument(file);

Set classes = ontology.getClassesInSignature();
//select device class defined in dClass [e.g. dClass = "MobileDevice"]
for(Iterator itins = classes.iterator(); itins.hasNext();){
 c = (OWLClass)itins.next();
 if( c.getNNF().toString().contains(dClass)){
   Set<OWLIndividual> individuals = c.getIndividuals(ontology);
   //select device instance defined in dInstance [e.g. dInstance = "N95"]
   for(Iterator it2 = individuals.iterator(); it2.hasNext();){
     ins = (OWLNamedIndividual)it2.next();
     if(ins.toString().contains(dInstance)){
       Set<OWLObjectPropertyExpression> props =
                                ins.getObjectPropertyValues(ontology).keySet();
       //select required role defined in dRole [e.g. dRole = "hasFacilities"]
       for(Iterator it3 = props.iterator(); it3.hasNext();){
         Oprop = (OWLObjectProperty)it3.next();
         if(Oprop.toString().contains("hasFacilities")){
           Set <OWLIndividual> ins2 =
                                    ins.getObjectPropertyValues(Oprop, ontology);
               //check for required feature defined in dFeature
                                 //[e.g. dFeature = "MMS"]
               for(Iterator it4= ins2.iterator(); it4.hasNext();){
                 if(it4.next().toString().contains(dFeature)){
                   ret = "true";
                  }else{
                   ret = "false";
                  }
                }//for Feature
              }//if object property
            }//for object property
         }//if individual
       }//for select individual
     }//if MobileDevice
   }//for select class
```

This code excerpt illustrates reasoning a given feature (e.g., MMS) of a given device (e.g., N95) from the context model ontology (e.g., Uri−Ontology). If the device has the MMS feature, then this reasoning service returns true. This reasoning service is implemented as an assertion to validate an implicit context.

6.5 Chapter summary

Dynamic requirements need to be defined in terms of dynamic service context so that they can be dynamically instrumented and validated within a service process at process run-time. We define dynamic requirements as context constraints. Our concern is to separate out context constraints from the context model ontology to provide flexibility and acceptable performance for instrumentation and validation monitoring. The two types of constraints, explicit context validation constraints (ECVCs) and implicit context validation constraints (ICVCs) preserve the required features of the context model ontology, such as shared conceptualization, context reasoning, etc. We propose a process and an algorithm for ECVCs generation, and a language for defining ECVCs. The constraint generation process (**ECVCGProcess**) is defined in a ECVC generation algorithm (**ECVCGAlgorithm**). As a proof of concept for the proposed process, algorithm and language, we use the Message Automation Workbench to implement CLiX-based constraints. CLiX is an XML-based constraint language, which is expressive, implementable, validatable and compatible with other Web service technologies [Jungo 2007, Nentwich 2002]. We implement ICVCs as reasoning services using the OWL-API and propose a reasoning service registry as a possible solution to select reasoning services.

Chapter 7

Instrumentation and validation monitoring

Contents

7.1 Introduction

7.1.1 Overview

In this chapter, we present how a deployed Web service process is dynamically instrumented with context constraints (CCs) to perform constraints validation monitoring at process run-time. **Dynamic validation monitoring** *is defined as validation* *monitoring at process run-time.* **Dynamic instrumentation** *is defined as instrumentation at process run-time*

Service-based systems and requirements can frequently change at run-time [Baresi 2010b], for example, service applications can bind to different services according to the context in which they are executed and providers can modify or improve their services. These aspects can hamper the correctness and quality levels of service applications. The pre-deployment validations are inadequate for service-oriented applications [Baresi 2005, Zhao 2007, Baresi 2010b, Baresi 2011] as discussed in section 3.2.4. There are challenges to realise SOA to its full potential, such as to ensure that the behavior of Web services is consistent with requirements [Wang 2009c, Baresi 2011]. In [Moser 2008b, Moser 2008a], the authors propose a tool for monitoring and dynamically adapting BPEL processes. Their monitoring focuses on computing quality of service (QoS) data for various service selection strategies to support dynamic service adaptation. However, these approaches do not sufficiently address dynamic instrumentation with requirements for validation monitoring at process run-time.

Dynamic requirements for a service process instance need to be monitored at process runtime to identify violations or do further processing. For example, - Performance, reliability and connectivity based requirements or settings frequently change at process runtime. - Cost of a service can be in different currencies and currency rates can change frequently. - The same process instance can be used for different clients but with different sets of dynamic requirements (DRs). - An execution time of a constituent service can determine the selection of the next constituent service for a service process at process run-time, etc. An architecture and techniques are necessary for dynamic requirements instrumentation and validation monitoring at process run-time. **Business process flow** defines the execution sequence of services, which run across heterogeneous systems. On the other hand, the **constraint flow** defines the validation sequence of dynamic requirements attached to a service process. At the moment, service process execution engines do not provide any other types of execution attached to a service process, such as dynamic requirements validation. In this work, we propose a collaboration of two independent engines for service process execution and dynamic requirements validation, see figure 7.1.

In this chapter, we detail the overall architecture for context constraints instrumentation and validation monitoring in section 7.1.2. In section 7.2, we describe the configurator generation attached to context constraints. Then, we detail the process instrumentation and validation monitoring in section 7.3. The use of data collectors for validation monitoring is also detailed. Finally, we describe a case study scenario in section 7.4, to illustrate the applicability of the proposed approaches and techniques. The chapter is summarised at the end.

7.1.2 Overall architecture description

This section presents the overall architecture description for the management of context constraints to facilitate dynamic instrumentation and validation monitoring of context constraints attached to Web service processes. At the moment, the most widely used Web service process definition language is BPEL (Business Process Execution Language)[Alves 2006], which is an OASIS standard. However, BPEL provides limited support for dynamic instrumentation. The main supported mechanism is via assigning endpoints at run-time using dynamic partner link constructs. The endpoints of member services need to be statically defined at design-time and the service requester has to know the exact endpoint address of a partner service. This does not conform to the basic principles of service orientation [Erradi 2005]. The concept of dynamic instrumentation is not new. The dynamic binding is applied for middleware technologies using WSFL (Web Service Flow Language) [Karastoyanova 2005]. However, the corresponding support in Web service processes, in particular in BPEL, is not sufficiently addressed

[Karastoyanova 2005, Mosincat 2009].

The core idea behind the proposed approach is somewhat near to the dynamic execution of monitoring rules proposed in [Baresi 2005] and a proxy-based approach for dynamic execution of services proposed in [Canfora 2008], but different in realization. We propose the overall architecture in figure 7.1 to address the challenges described in section 7.1.1. The IVS (Instrumentation and validation

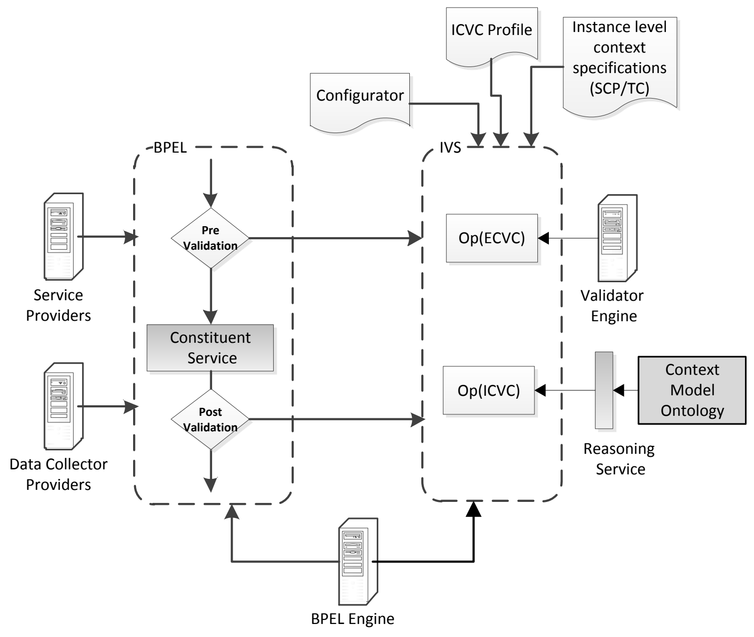
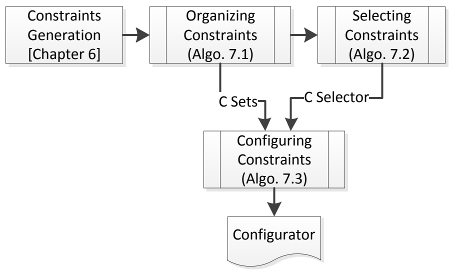


Figure 7.1: Dynamic instrumentation and validation monitoring architecture

service) is a process that is made available as a service, which uses a validator engine to validate ECVC and the context model ontology to validate ICVC using reasoning services. The BPEL engine is used to execute BPEL application processes including IVS.

The IVS uses input from the constraints generation process. A configurator generation from context constraints can be illustrated as in figure 7.2.

Definition (Constraint set) $Cset^{vp} = \{CC_1^{vp}, ..., CC_n^{vp}\}$, where CC_1 to CC_n are context constraints and vp is a validation point of a constraint set. Context constraints and constraint sets need to be generated and attached as pre-conditions


Constraints Generation [Chapter 6]
Organizing Constraints (Algo. 7.1)
Selecting Constraints (Algo. 7.2)
C Sets
C Selector
Configuring Constraints (Algo. 7.3)
Configurator
Figure 7.2: Context constraints to configurators

or post-conditions of services, which we call validation points. For example, vp_1= Service$X$-pre, vp_2= Service$X$-post, etc.

Definition (Constraint selector) $Cselector = \{Cset_1, ..., Cset_n\}$, where $Cset_1$ to $Cset_n$ are constraint sets. A number of different constraint sets need to be selected for a validation point by a software architect depending on requirements. For example, due to performance considerations, not all constraints might be monitored at all times.

Definition (Configurator) $Configurator = \{CSelector, \{Cset_1, ..., Cset_n\}, ConfigInfo\}$, where $CSelector$ is a constraint selector and $Cset_1$ to $Cset_n$ are the constraint sets that refers to the selector. $ConfigInfo$ is information required to configure the validation engine. A configurator provides information about constraint sets, a selector and $ConfigInfo$ to the validator engine. Moreover, configurators support to configure the validator engine with different configuration settings at different validation points, such as schema validation ON and OFF at different validation points [Nentwich 2005].

Our proposed approach can dynamically manage context constraints independently from the service process execution. In other words, service process execution and context constraints validation are two independent parallel processes. We described constraint generation in detail in chapter 6. The instrumentation and validation service (IVS) is used to implement the instrumentation functionality for an instrumented application process. IVS has two operations Op(ECVC) and Op(ICVC) to support explicit context validation constraints (ECVCs) and implicit context validation constraints (ICVCs). The IVS has a configurator, an ICVC profile, and instance level context specifications, such as SCP, TC and monitored context (chapter 5 and 6) as inputs, which enable instrumentation and validation of ECVCs and ICVCs. We describe the configurator in detail in section 7.2. The ICVC profile contains information about which reasoning function needs to be invoked, for example, the end point information and soap message information of a reasoning service.

We focus on dynamically instrumenting context constraints to a deployed service process. Context constraints need to be managed based on dynamic requirements to dynamically link them to a service process instance at validation points. The validation points of context constraints are determined based on the dynamic requirements. The proposed configurator, which is near to the idea in [Spertus 2010], can be dynamically generated as described in figure 7.4. The constraint sets and selectors can be modified at process run-time to add or remove requirements for validation. The configurators are passed to IVSs (Instrumentation and Validation Service) as inputs at process run-time, which we detail in section 7.3. This configurator-based approach can be applied for ECVCs. The ICVCs are instrumented to a service process using an operation called Op(ICVC) of IVC. The Op(ICVC) enables dynamically invoking ICVCs, which are provided as reasoning services. Most of the dynamic requirements can be defined as ECVCs, which are XML-based constraints.

7.2 Context constraint configurator generation for ECVCs

Most of the context constraints are based on explicit context categories in context specifications. Configurators can be used to combine ECVC sets and selectors at process runtime. Constraints, constraint sets, constraint selectors, constraint configurators and their relationships can be illustrated as in figure 7.3. A constraint

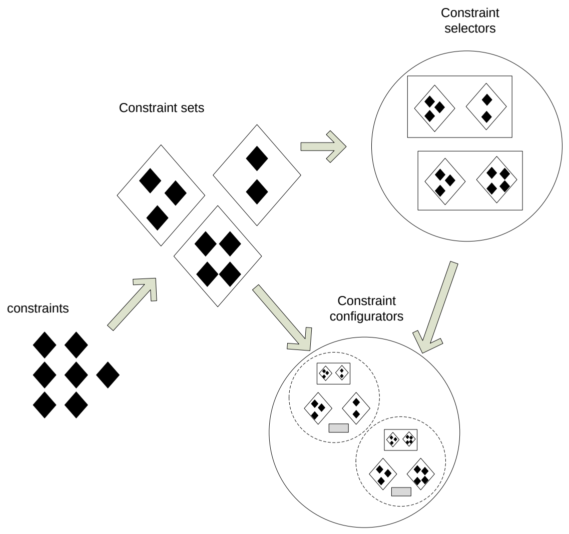


Figure 7.3: Constraint sets, selectors and configurators

set contains ECVCs. A constraint selector contains constraint sets in validation order assigned by an architect. A configurator contains a selector, constraint sets and configuration information for the validation engine.

The proposed configurator generation is a three step process, which is illustrated in figure 7.4. This can be considered as a platform independent software

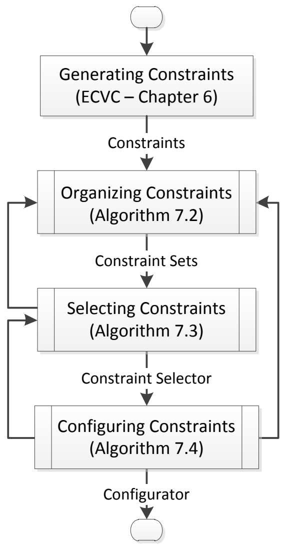

Generating Constraints (ECVC – Chapter 6)
Constraints
Organizing Constraints (Algorithm 7.2)
Constraint Sets
Selecting Constraints (Algorithm 7.3)
Constraint Selector
Configuring Constraints (Algorithm 7.4)
Configurator

Figure 7.4: Configurator generation process

component for dynamic instrumentation and validation monitoring of context constraints.

The ECVCs are generated at the semantic client (or broker), see Chapter 6. Firstly, constraint sets are generated from ECVCs based on a criteria decided by a software architect, such as based on major context categories, validation points of services or combination of both. For example, gather quality of service constraints attached to one service and one validation point in one set. One service can have two validation points for its pre and post conditions. Secondly, a constraint selector is generated to select different constraint sets attached to one validation point of a service process. Thirdly, we generate configurators. Two configurators are needed for each service in a deployed service process, to support pre-condition validation and post-condition validation. A configurator has a constraint selector, constraint sets and configuration information for the validator engine.

7.2.1 Generating constraint sets

Dynamic requirements in a service process need to be validated before invoking constituent services and after invoking constituent services at process run-time. That is, the constituent services in a service process need pre-condition validation before invoking services and post-condition validation after invoking services at process run-time. Dynamic requirements need to be defined and attached to pre-conditions or post-conditions of services, which we call validation points of constituent services. Each constituent service can have more than one pre-conditions and (or) post-conditions attached to dynamic requirements, for example, requirements related to financial aspects of services, requirements related to security aspects of services, requirements related to syntax aspects of services, requirements related to domain aspects of services, etc. Therefore, constraints need to be arranged in constraint sets for each service in a process instance. If there are pre and post-conditions available for a constituent service, then there should be at least two constraint sets available. Moreover, dynamic requirements can be changed at process run-time so that constraints in constraint sets need to be updated accordingly at process run-time.

Constraints are categorized as **logical** constraints, **if-then** constraints and **control** constraints focusing on service selection in [Karakoc 2009]. Logical constraints are any logical expression on numerical or string values (e.g. a user can define a constraint that the total cost of a complete travel plan should not be more than 2500 Euro), if-then constraints define the structure of a business rule (e.g. if Hotel.star $\leq$ 4 then Hotel.cost $\leq$ 100 Euro), and control constraints are the type of constraints that are directly specified on services (e.g. if a Web service interface WT1 belongs to a Web service W1, then the Web service interface WT2 must belong to the same / different Web service - special constructs, such as same / different are used in control constraints). In [Kim 2004], constraints are classified in detail to simplify complex constraints found in business process specification schemas. However our focus is on categorising constraints relevant to the constraint validation points of a process instance. Which constraint to validate at which validation point of a process instance needs to determine constraint sets, that is determined based on the dynamic requirement at the semantic client (or broker). For example, the cost of a service needs to be less than 0.2 USD, which needs to be applied as a pre-condition for all constituent services.

Constraint sets combine multiple constraints for validation monitoring. We propose an algorithm to formalise the constraint set generation process, Algorithm 7.2. This constraint sets generation is based on context constraints and their validation points. The input of the algorithm is context constraints attached to a process instance and output is constraint sets attached to pre-conditions and post-conditions of constituent services of the process instance. That is, it does separation of context constraints into pre-condition sets and post-condition sets of constituent services.

Algorithm 7.2 Constraint sets generation

```
   $Input : CC[\,]$
   $Output : Cset[\,]$
1: for (each $CC$) do
2:     scope ← getScope($CC$)
3:     if $(scope = SL)$ then
4:         $SID \leftarrow getSID(CC)$
5:         $CON \leftarrow getCondition(SID, CC)$
6:         $Cset \leftarrow CreateAmendConstraintSet(SID, CON)$
7:         {
8:         if ( $Cset_{SID}^{CON}$ = NULL ) then
9:             initialise ($Cset_{SID}^{CON}$)
10:            $Cset_{SID}^{CON}$ = $Cset_{SID}^{CON}$ $\wedge$ CC
11:        else
12:            $Cset_{SID}^{CON}$ = $Cset_{SID}^{CON}$ $\wedge$ CC
13:        end if
14:        }
15:    else if $(scope = PL)$ then
16:        $VP \leftarrow getVP(CC)$
17:        $Cset \leftarrow CreateAmendConstraintSet(VP)$
18:        {
19:        if ( $Cset^{VP}$ = NULL ) then
20:            initialise ($Cset^{VP}$)
21:            $Cset^{VP}$ = $Cset^{VP}$ $\wedge$ CC
22:        else
23:            $Cset^{VP}$ = $Cset^{VP}$ $\wedge$ CC
24:        end if
25:        }
26:    end if
27: end for
```

We use the terms CC for ECVCs, SID for service identification, VP for constraint validation point, CON for pre or post-conditions, SL for service-level, PL for process-level, and Cset for constraint set. Each context constraint has validation point information about the constituent service that it is attached to either as a pre-condition or post-condition. The method `getScope(`CC`)` returns the scope (SL or PL) of the constraint, `getSID(CC)` returns the service identification of the constraint, `getVP(CC)` returns the validation point of the constraint, `getCondition()` returns the pre or post-condition status of a context constraint relevant to a service and `CreateAmendConstraintSet()` adds a new constraint either to create a new constraint set or amend an available constraint set. The method `initialise()` is used to initialise a constraint set and set the validation point. The priorities of constraints within a constraint set are not considered in this work. This algorithm will not proceed if there are no context constraints. This algorithm executes on a per constraint basis and n times for n context constraints. Therefore, the time complexity of the algorithm is of order n, O(n).

We implement constraint sets in XML, so that we can change them dynamically and they are compatible with any platform. A constraint set only has a set of links to constraints, so that new constraints can easily be included and unnecessary constraints can easily be identified and removed from a constraint set. The software architect is responsible for identifying and removing constraints based on requirements. A constraint set can have only necessary constraints.

7.2.2 Selecting constraints (Constraint selector)

A constraint selector contains information about which constraint sets to validate at which validation point. This is similar to the idea of a rule selector in [Lunteren 2004]. A Web service application needs to have the flexibility to dynamically change the instrumentation of requirements. Consequently, context constraints need to be selected at process run-time based on dynamic requirements. Our concern of constraint selection is selecting sets of constraints based on their validation points focusing constraints validation monitoring at process run-time.

We described constraint set generation in section 7.2.1. Context constraints can have service-level scope and process-level scope, therefore we define them as service-level constraints and process-level constraints - local constraints and global constraints in [Rosenberg 2010]. Service-level constraints can be pre-condition focused constraints or post-condition focused constraints. A service can have more than one pre-condition focused or post-condition focused constraint sets. Some constraint sets may not need to be validated in some circumstances, such as high security may be not needed all the time. We generate constraint selectors for each pre and post-condition relevant to constituent services and process-level validation points in a deployed service process. We define the process of constraint selectors generation in algorithm 7.3. The input of the algorithm is constraint sets attached to a process instance and the output is a set of selectors attached to various validation points of the process instance.

Algorithm 7.3 Constraint selector generation

```
  $Input$ : $Cset[]$
  $Output$ : $Cselector[\ ]$
 1: initialise $\{Cselector\}$
 2: for (each $Cset$) do
 3:     scope ← getScope($Cset$)
 4:     if (scope = SL) then
 5:         SID ← getConstituentService($Cset$)
 6:         CON ← getConditionStatus($Cset$)
 7:         Cselector ← createAmendSelector($Cset$,$SID$,$CON$)
 8:         {
 9:         if ( $Cselector^{CON}_{SID}$ = NULL ) then
10:             edit $Cselector$ to $Cselector^{CON}_{SID}$
11:             $Cselector^{CON}_{SID}$ = $Cselector^{CON}_{SID}$ $\wedge$ Cset
12:         else
13:             $Cselector^{CON}_{SID}$ = $Cselector^{CON}_{SID}$ $\wedge$ Cset
14:         end if
15:         }
16:     else if (scope = PL) then
17:         $VP \leftarrow getVP(Cset)$
18:         Cselector ← createAmendSelector($VP$)
19:         {
20:         if ( $Cselector^{VP}$ = NULL ) then
21:             edit $Cselector$ to $Cselector^{VP}$
22:             $Cselector^{VP}$ = $Cselector^{VP}$ $\wedge$ Cset
23:         else
24:             $Cselector^{VP}$ = $Cselector^{VP}$ $\wedge$ Cset
25:         end if
26:         }
27:     end if
28: end for
```

The `getConstituentService(`$Cset$`)` method returns a constituent service id related to the Cset. The `getConditionStatus(`$Cset$`)` method returns a pre-condition or post-condition status of a Cset. The methods `createAmendSelect-or(`$Cset, SID, CON$`)`, and `createAmendSelector(`VP`)` either creates or amends a constraint selector. The operator $\wedge$ is used to add constraint sets to a Cselector. The algorithm of constraint selector generation will not proceed if there are no constraint sets. This algorithm executes once for each constraint set and n times for n constraint sets, therefore, the time complexity of the algorithm is of order n, O(n).

The constraint selector has information about which constraint to be selected for validation. The constraint selector is independent from the Web service process instance and enables dynamic constraint selection at process run-time. We define constraint selectors in XML as before.

7.2.3 Configuring constraints (Configurator)

A configurator contains information about which constraint selector is used at which validation point, information about the constituent constraint sets attached to each selector and configuration information ($ConfigInfo$) as information to the validator engine, see section 7.1.2. It is necessary for a configurator to combine a selector, constraint sets and configuration information so that dynamically selected constraints can be validated using a validator at a validation point. The configurator needs to be dynamically determined based on dynamic requirements at service process run-time. A service process can be executed on heterogeneous platforms. Therefore, the implementation of a configurator needs to be platform independent. The configurator generation process for each constraint selector is defined in algorithm 7.4. The input of this algorithm is a constraint selector and

Algorithm 7.4 The configurator generation

```
  $Input : CSelector$
  $Output : Configurator$
1: **if** ($CSelector$!= NULL) **then**
2:     $ConfigInfo \leftarrow$ getConfigInfo($CSelector$)
3:     initiate $Configurator$ with $ConfigInfo$
4:     Cset[] $\leftarrow$ readConstraintSets($CSelector$)
5:     $Configurator \leftarrow$ addSelector($Configurator, CSelector$)
6:     $Configurator \leftarrow$ addConstraintSet($Configurator, Cset[]$)
7: **end if**
```

the output is a configurator consisting of a reference to the selector, constraint sets selected by the selector and configuration information. We assume our configurator has a default template, which needs to be initialised with configuration information ($ConfigInfo$). The method `getConfigInfo(`$CSelector$`)` returns $ConfigInfo$ required for the validator engine to validate constraints selected by the given $CSelector$. The method `readConstraintSets(`$CSelector$`)` reads a selector and outputs an array of constraint sets. The method `addSelector(`$Configurator, CSelector$`)` adds the selector to the configurator and `addConstraintSet(`$Configurator,\ Cset[]$`)` adds each constraint set to the configurator.

7.3 Process instrumentation and validation monitoring

Instrumentation is the most widely used monitoring mechanism [Wang 2009c]. The authors in [Wang 2009c] introduce an online monitoring approach for Web service requirements, where monitoring code is embedded inside the target code. Process instrumentation with monitoring rules before deployment is proposed through source code weaving in [Baresi 2004, Baresi 2010b], in which a change in a monitoring code needs re-deploying the whole process. An aspect-oriented extension for monitoring a BPEL process according to certain QoS criteria to replace existing partner services is proposed in [Moser 2008a]. The Kieker framework monitors control flows of applications and response times of method executions [Hoorn 2009, Ehlers 2011]. However, more work is still necesary to address dynamic instrumentation of context constraints to a deployed service process instance for validation monitoring of dynamic requirements at process run-time. The dynamic instrumentation and validation monitoring of ECVCs and ICVCs is the concern.

7.3.1 Tool support

We use WS-BPEL (Web Service Business Process Execution Language) [Alves 2006] to define Web service processes, the xlinkit validator engine for constraints validation [Nentwich 2002], and Java Web service technology for data collectors, reasoning services implementation using OWL API[1] and their instrumentation. WS-BPEL [Alves 2006] is the de-facto standard for service process modeling. WS-BPEL provides limited support for run-time binding via assigning endpoints at run-time using dynamic partner link constructs. A dynamic partner link needs a re-assigning service of the same interface. For simplicity, sometimes we use the term BPEL for WS-BPEL. The core of the xlinkit validator functionality is quite simple. The xlinkit is given a set of XML resources and a set of rules in CLiX that relate the content of those XML resources for validation. The rules express consistency constraints across the XML resource elements [Nentwich 2002, Dui 2003, Nentwich 2005]. The CLiXML constraints, constraint sets, constraint selectors and constraint configurators can be implemented using the Message Automation Workbench, see chapter 6.

7.3.2 Instrumentation and validation monitoring

The context constraints instrumentation and validation monitoring architecture in figure 7.1 facilitates dynamic instrumentation of context constraints using the IVS(Instrumentation and Validation Service). In this architecture, for ECVCs, the **configurators** are dynamically bound to explicit context validation constraints and passed to the IVS for run-time validation. The configurators (e.g., config.xml) and dynamic service context (e.g., context instances) are provided as inputs for the IVS. The IVS passes these configurator and dynamic service contexts to a validation engine for constraint validation and outputs validation results accordingly. A service process is executed in a process engine, for example, a BPEL engine, and constraints are validated in a validator engine, for example, the xlinkit validator engine[2]. The validation results are recorded for monitoring at the semantic client (or broker). For ICVCs, a ICVC profile, which has end-points of reasoning services is passed to the IVS for deriving implicit context or validating implicit

[1]http://owlapi.sourceforge.net/index.html
[2]http://www.messageautomation.com/

context based on explicit context using context model ontology reasoning. The validation reports are recorded in a logging repository, which supports the validation monitoring process.

7.3.3 Instrumentation and validation service (IVS)

The IVS is used to provide the instrumentation for a service process with context constraints (CC). The IVC is bound to the pre and post conditions of constituent services. Our IVS has two operations, `OperationValidation` and `OperationReasoningInvoke`. `OperationValidation` is used for the instrumentation of ECVCs and `OperationReasoningInvoke` is used for the instrumentation of ICVCs for validation monitoring. The dynamic binding of the IVS for the pre and post conditions is decided based on validation requirements of constraints.

Process instrumentation with ECVCs (`OperationValidation`)

Here we describe our techniques for instrumenting a deployed service process with ECVCs using the `OperationValidation` of an IVS. Each constituent service can be attached to a pre and post IVS as shown in figure 7.1. An abstract view of a constraints instrumented service process is illustrated in figure 7.5. Constraint

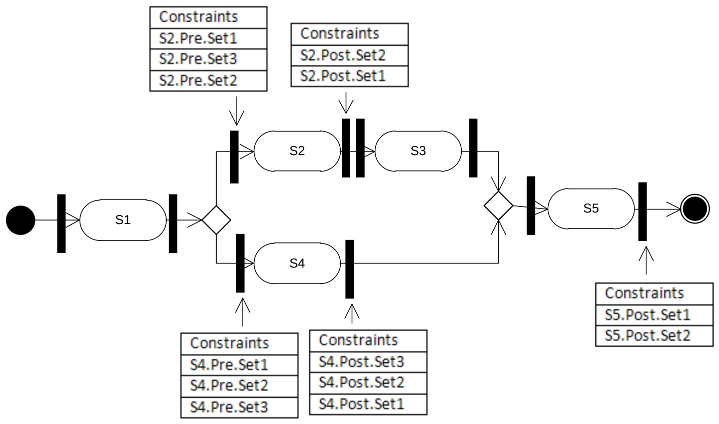


Figure 7.5: An abstract view of a constraints instrumented process

sets are described in constraint tables, selected by a constraint selector.

`OperationValidation` has two inputs, such as the configurator, which is the bridge between service process and constraints as described in section 7.2.3 and the dynamic service context. The configurator is dynamically attached to constraints and constraint selectors, facilitating dynamic binding of ECVCs. The constraints selector enables dynamic selection of ECVCs for validation at process run-time. `OperationValidation` of the IVS is responsible for validating ECVCs as shown in Appendix C. The interface definition of the operation `OperationValidation` is,

```
Inputs:
    Dynamic service context
    Configurator

Task:
    1. Initialise the validator engine
    2. Read constraints
    3. Read dynamic service context
    4. Validate constraints against dynamic service context

Output:
    Boolean [TRUE/FALSE]
```

This approach enriches dynamic instrumentation of requirements with a deployed process instance. It is a flexible approach for instrumentation and validation monitoring of dynamic requirements within a service process at process run-time.

Process instrumentation with ICVCs (`OperationReasoningInvoke`)

`OperationReasoningInvoke` is used for instrumenting and invoking reasoning services. We implemented this operation as shown in Appendix C. The interface definition of the operation `OperationReasoningInvoke` for reasoning about a specific property is,

```
Inputs:
    Parameter Name
    Parameter Value
    Property
    Uri_Ontology

Tasks:
1. Read a ICVC profile relevant to the Parameter Name and Property
2. Select a related reasoning service from the ICVC profile
3. Create a SOAP message including the reasoning feature and Uri_Ontology.
4. Invoke the reasoning service operation, which is defined in the ICVC profile

Output:
    Boolean [TRUE/FALSE]
```

We implemented this operation in a pre-defined setting. In this setting, each parameter name (e.g. MobileDevice, Connection) and each property (e.g. Message, Data) has a specific profile (e.g. Mobile device and Message has one profile, Connection and Data has one profile). The required reasoning service information is defined in a profile, such as the endpoint, service name, port name and soap message details. In this approach, various profiles enable instrumenting various reasoning services through the operation `OperationReasoningInvoke` of the IVS. The operation `OperationReasoningInvoke` of IVS works as an adapter to connect reasoning services. This is a complete service-based approach to dynamically invoke ICVCs.

7.3.4 Data collectors

Some context constraints need the support of data collectors for their validation. In our dynamic instrumentation and validation approach, context instances can be generated and derived at process run-time, which we call transient context instances (TCI). For example, new context instances can be derived at context reasoning-time, run-time context instances (such as response time, availability) can only be measured at process run-time, etc. These TCIs can be necessary components for ECVCs and ICVCs, section 6.1. In this approach, we use data collectors to facilitate TCIs for ECVCs and ICVCs. In our approach, the data collector directives (meters) are provided for metering in a service process. Data collectors (for metering) are defined and provided as services as shown in Appendix C.

An abstract service process is instrumented with data collector directives, for example, meters to record the time when a service starts execution and when a service outputs a response, and data collectors as shown in figure 7.6. If a service

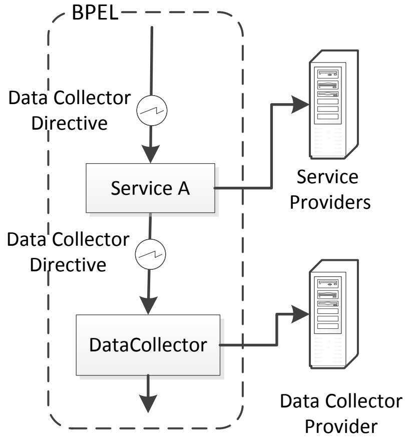


Figure 7.6: Data collector directives and data collectors

fails at process run-time, a new service can be dynamically combined. However, the data collector directives will not change and they record information relevant to the new service, and data collectors can be used to collect recorded data. The data collector directives and data collectors can be typed by context category, for example, data collector directives and a data collector for the response time context.

However, the instrumentation of data collectors has a limitation, which is that data collector directives and data collectors are not instrumented dynamically and only instrumented with an abstract service process before deployment. The authors in [Baresi 2005] propose attaching monitoring rules with a service process before deployment. The run-time instrumentation of data collector directives and data collectors is necessary for service-based applications [Baresi 2010a, Baresi 2011].

7.3.5 Discussion

Software monitoring involves obtaining information relating to the state, behavior, and environment of a software system at run-time, so as to deal with potential deviations of system behavior from requirements at the earliest possible time. Monitoring is usually carried out in parallel with the system's execution, without interrupting its operation [Wang 2009c]. The main concern of our monitoring process is collect and monitor context instances and constraint validations at process run-time. We propose a data collector directives based approach to collect transient context instances as shown in section 7.3.4, and a logging-based approach, which generates records about successes and failures of constraint validations to support validation monitoring.

In the proposed approach, the constraint sets, constraint selectors and configurators are XML-based. The priorities of constraint sets within selectors are not considered in this approach, which is a possible extension. For simplicity, ECVCs, configurators and selectors are held in file folders at the semantic client (or broker). Databases can also be used for a constraint repository as in [Medjahed 2007]. This monitoring process ensures real time context instances, reports on constraint violations and ensures backward compatibility. We equiped IVSs with logs generation to support constraints validation monitoring. These logs can be used for administrative purposes. However, developing a complete monitoring framework is out of our scope in this thesis.

7.4 Case study

In this section, we describe an implementation of a case study scenario to illustrate the definitions introduced in section 7.2, and the process instrumentation and validation monitoring introduced in section 7.3. The tools described in section 7.3.1 are used for the implementation. The complete implementation of the case study is described in Appendix C.

Case study. We consider a simple business process, which has a `Payment service` and a `Payment confirmation service`. The `Payment confirmation service` can be a `Mobile service` or an `Email service`. We assume that there is more than one service available for a single task and the required services can be combined based on requirements at process run-time. The requirements for a `Payment service` are that the cost of the service needs to be less than 0.3 Euro, the connectivity strength of the service needs to be greater than 2 Mbps and the response time of the service needs to be less than 200 ms. At process run-time, these requirements or settings can change (for example, currency rates can change, connectivity strength changes over time, etc.) and need to be monitored. The payment confirmation service can be a `Mobile service` or an `Email service`, decided based on the response time of the `Payment service` (if response time is less than 100 ms, then an `Email service` is composed, if not, a `Mobile service` is composed). In this scenario, a mobile message is a MMS message, so that it is necessary to check whether the user's device supports MMS messaging. If the user's device does not support MMS messaging then an `Email service` is used. The abstract view of the case study design can be viewed in figure 7.7.

Constraint sets (Rule sets).

The ECVCs need to be generated and organised in constraint sets, and implemented as rule sets. The rule sets are collections of rules, defined in an XML

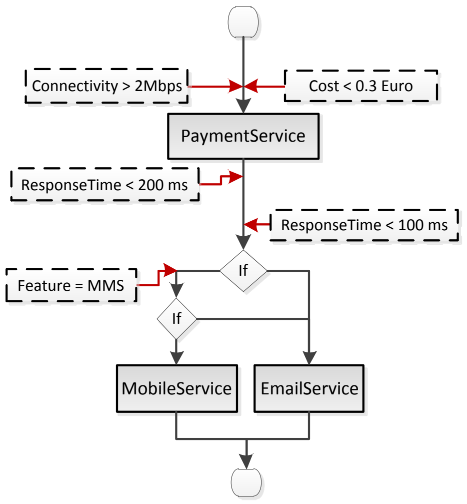


Figure 7.7: Abstract view of payment confirmation case study design

template with the extension ruleset. The rule sets have advantages, such as combining multiple rule files for validation without having to physically aggregate them, composing big sets of rules by combining other rule sets in a hierarchical manner, etc. We implement constraints as rules in CLiXML syntax.

We implement the cost constraint in costContext.clix, connectivity constraint in connectivityContext.clix, and response time constraint in responseTimeContext.clix. The cost and connectivity of the `Payment service` are pre-validation constraints and the response time of the `Payment service` is a post-validation constraint.

We implement a constraint set for pre-conditions attached to the `Payment service` using the Message Automation Workbench as PreCon-PaymentService.ruleset,

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet
     id="PreCon-PaymentService" >

     <RuleFile href="costContext.clix"/>
     <RuleFile href="connectivityContext.clix"/>

</RuleSet>
```

We implement a constraint set for post-conditions attached to the `Payment service` using the Message Automation Workbench as PostCon-PaymentService.ruleset,

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet
    id="PostCon-PaymentService" >

    <RuleFile href="responseTimeContext.clix"/>

</RuleSet>
```

Constraint selector (Rule selector).

We implement a constraint selector as a rule selector that contains information about which rules to select at validation. The rule selector generation is independent from the Web service process. The Payment service needs two selectors for pre-condition and post-condition validations.

We implement a selector for pre-condition validation of the `Payment service` as PreCon-PaymentService.mavselector using the Message Automation Workbench,

```
<?xml version="1.0" encoding="UTF-8"?>
<selector:Selector>

    <selector:RuleSet id="PreCon-PaymentService"/>

</selector:Selector>
```

We implement a selector for post-condition validation of `Payment service` as PostCon-PaymentService.mavselector using the Message Automation Workbench,

```
<?xml version="1.0" encoding="UTF-8"?>
<selector:Selector>

    <selector:RuleSet id="PostCon-PaymentService"/>

</selector:Selector>
```

Configurator.

We implement a configurator to comply with the Message Automation API. The configurator is implemented in a configuration file, which is an XML template that contains information about how the Message Automation API will react when validating context constraints. The rule validation can be set ON or OFF. The rule selectors and rule sets are attached to a configuration file. In this case study, we implement two configuration files for the `Payment service` for its pre-condition and post-condition validations.

We implement a configurator for pre-conditions validation attached to the `Payment service` using the Message Automation Workbench as PreCon-PaymentService. mavconfig,

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <rules>
        <ruleValidation>ON</ruleValidation>
        <selector>PreCon-PaymentService.mavselector</selector>
        <ruleSet>PreCon-PaymentService.ruleset</ruleSet>
    </rules>
</config>
```

We implement a configurator for post-conditions validation attached to the `Payment service` using the Message Automation Workbench as PostCon-PaymentService. mavconfig,

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <rules>
        <ruleValidation>on</ruleValidation>
        <selector>PostCon-PaymentService.mavselector</selector>
        <ruleSet>PostCon-PaymentService.ruleset</ruleSet>
    </rules>
</config>
```

Data collector.

The data collectors collect transient context to enrich the context space of the service process at process run-time. We implement data collectors as services, which collect data based on data collector directives. In this case study, data collector directives are attached before and after the invocation of `Payment service` to collect service response time. Data collector invokes after the `Payment service` but before executing the post-conditions of the `Payment service`. The data collector provides the response time in ms to the context space of the service process as,

```
<Context>
    <RuntimeContext>
        <ResponseTime> 120 </ResponseTime>
    </RuntimeContext>
</Context>
```

The complete implementation of the data collector service is shown in Appendix C.

IVS for ICVCs.

In this case study, we need dynamic instrumentation of a context reasoning service to reason about whether the user device supports MMS messaging. We describe a reasoning service, which is an implicit context validation constraint in chapter 6 that needs to be dynamically invoked through an IVS attached as a pre-condition to the `Mobile service`. The reasoning service is invoked through the `operationReasoningInvoke` of the IVS. The complete implementation is shown in Appendix C.

IVS for ECVCs.

In this case study, two configuration files are needed for pre-condition and post-condition validation of the `Payment service`. The `Payment service` is attached to one IVS for pre-condition validation and post-condition validation having different inputs. One IVS with different inputs can be used for all ECVC validations. The `OperationValidation` of IVS is used for ECVC validation. The implementation is shown in Appendix C.

7.5 Chapter summary

Dynamic instrumentation of requirements to a deployed service process for their run-time validation monitoring is still not appropriately addressed in the literature. In this chapter, we proposed a service-based approach for context constraint instrumentation and validation monitoring at process run-time. We identified context constraints as ECVCs and ICVCs and used the IVS to dynamically instrument them with a deployed process instance. We illustrated an architecture to dynamically instrument ECVCs and ICVCs using the IVS. The ECVCs are organised in constraint sets, selectors, and configurators. We introduced algorithms to define the generation process of constraint sets, selectors and configurators. A configurator is attached to a selector and constraint sets and the IVS is used for validation monitoring of dynamic requirements. This approach is a flexible approach for instrumentation and validation monitoring of dynamic requirements within a service process at process run-time. We implemented ICVCs as reasoning services and managed dynamic instrumentation of ICVCs in a pre-defined setting. This is a complete service-based approach to dynamically invoke context reasoning services. We instrument data collector directives and data collectors in the abstract process-level, but do not address dynamic instrumentation of them in this thesis. Finally, we described a case study scenario to illustrate the applicability and implementation of all these introduced concepts. The complete implementation is illustrated in Appendix C.

Chapter 8

Evaluation

Contents

8.1 Introduction

8.1.1 Aims

We illustrate the overview of evaluation aspects and strategies used to evaluate our contributions in figure 8.1.

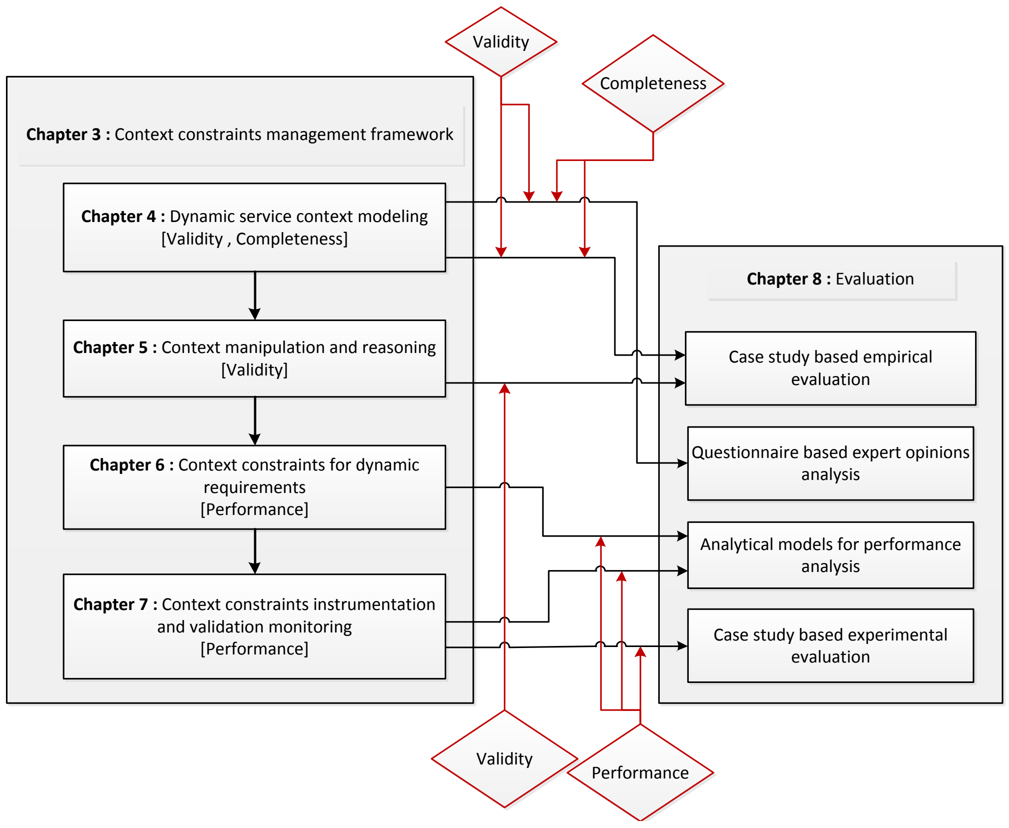

Validity
Completeness
Chapter 3 : Context constraints management framework
Chapter 4 : Dynamic service context modeling [Validity , Completeness]
Chapter 5 : Context manipulation and reasoning [Validity]
Chapter 6 : Context constraints for dynamic requirements [Performance]
Chapter 7 : Context constraints instrumentation and validation monitoring [Performance]
Chapter 8 : Evaluation
Case study based empirical evaluation
Questionnaire based expert opinions analysis
Analytical models for performance analysis
Case study based experimental evaluation
Validity
Performance

Figure 8.1: Evaluation overview

We evaluated our contributions focusing on the following aspects.

- Validity - Context categories defined in the context model ontology represent the needs of dynamic requirements (requirements at process run-time) management, and the context manipulation operators and context compositions process context for dynamic requirements.
- Completeness - All of the required dynamic requirements need to be covered by the context aspects in the proposed context model ontology. This

complements the validity requirements.

- Performance - The performance is measured based on process execution-time, which includes dynamic instrumentation of context constraints and their validation.

8.1.2 Evaluation strategy

In this section, we describe the evaluation strategies used to evaluate our contributions. Empirical studies and experiments can be used to confirm or reject the effectiveness of some methods, techniques, or tools [Easterbrook 2008]. An empirical method is a set of organising principles around which empirical data is collected and analyzed [Easterbrook 2008]. We used both quantitative and qualitative research methods and analysis throughout our evaluation. The overall evaluation strategy adopted in our work involves following research methods, which we selected based on our analysis and experiences in the literature.

- Case studies
- Expert opinions analysis
- Analytical models
- Controlled experiments

We explored and applied these research methods in the following sections to evaluate our contributions in this thesis. In section 8.2, we detail a confirmatory case study analysis and questionnaire-based expert opinions analysis to evaluate the proposed context aspects and context model for validity and completeness. In section 8.3, we detail a case study based evaluation to evaluate the validity of context manipulation operators and context compositions. We analytically assess the time complexity of the context constraint generation process by defining an analytical model in section 8.4. Finally in section 8.5, we evaluate the experimental aspects such as the constraints instrumentation and validation monitoring for performance using both analytical models and controlled experiments using a prototype.

8.2 Context model ontology

8.2.1 Overview

In this section, we describe a detailed evaluation of the proposed context model ontology (CMO). Firstly, we evaluated the validity of context aspects using context aspects in confirmatory case study scenarios from the content-oriented domain and emerging convenience services domain, section 8.2.2. The completeness of CMO is evaluated by comparing context aspects in confirmatory case study scenarios and the proposed context model, section 8.2.2. Secondly, we evaluated the validity of context aspects and completeness of CMO using an expert opinion analysis. We conducted a survey to collect expert opinions in the industry, which we describe in Appendix D. Eighteen experts were selected from different countries, covering different cultures with different expertise on software development, quality assurance, project management, consultancy, and computing research. The experts range from software engineers to software architects in the development track, quality assurance engineers to quality assurance leads in the quality assurance track, junior project managers to delivery managers in the management track, junior consultants to senior consultants in the consultancy track, and junior research engineers to senior research scientists in the research track covering most of the roles in the software industry. The experts were sourced from a cross section of small to large in size software companies and research labs.

8.2.2 Case study based evaluation

Exploratory case studies are used as initial investigations of some phenomena to derive new hypotheses and build theories, and confirmatory case studies are used to test them [Easterbrook 2008]. In our research, a confirmatory case study scenario based method has been used with test cases to assess the proposed context model ontology with regard to its validity and completeness. We considered validity and completeness as technical aspects. We used exploratory case study scenarios from a classical business domain at the development phase of the context model. The use of scenarios from multiple complementary domains for the evaluation of context categories increases the accuracy of them. The selection of case study domains is a crucial step. The experts in e-learning application domain and convenience services domain participated in developing scenarios. We followed empirical evaluation on confirmatory case study scenarios from content oriented domain and convenience services domain, which are two complementary domains, particularly looking into context categories in them. This evaluation was done manually by analysing each scenario in detail and capturing their requirements at service process run-time. However, the case study based research has much less control over the variables than that of experiment-based research [Rowley 2002]. Here, analysis of case study scenarios has two defined objectives that is to evaluate validity and completeness of context categories.

Validity perspective of context categories

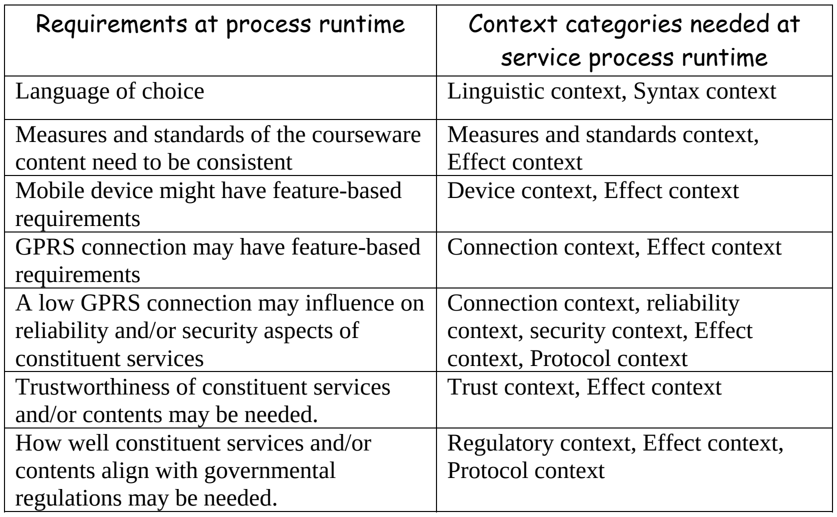
Analysis of application scenarios from two complementary domains for the validity perspective of context categories in the context model ontology is the concern. We analysed two scenarios from the content oriented domain and the emerging convenience services domain for dynamic requirements at application run-time. We checked whether context categories represent the needs of dynamic requirements at process run-time to test the hypothesis that *context categories can represent the needs of dynamic requirements management*.

A scenario from the content-oriented service-based application domain :

Service-based applications are widely used in the content-oriented application domain. We explored an e-learning courseware generation sample scenario and analysed it for the use of context categories in defining dynamic requirements.

A courseware is generated for a mobile device with a low speed GPRS connection on a *train*.

In this scenario, a user may need a courseware in a language of his choice, a mobile device may need feature-based requirements, a GPRS connection may need feature-based requirements, the measures and standards of the courseware content may have inconsistencies, a low GPRS connection may influence reliability and/or security aspects of constituent services, etc. In this scenario, we assume Web services are context-aware (service context profiles are attached to services) and they are dynamically composed and deployed to satisfy the client's request. The identification of requirements at process run-time and analysis of context categories on them relevant to this scenario is illustrated in figure 8.2.

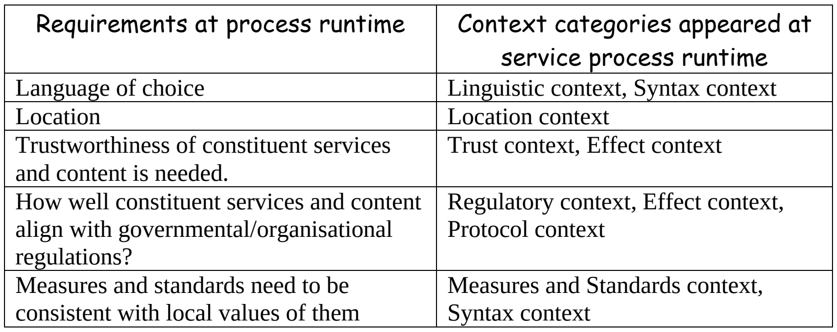


Requirements at process runtime	Context categories needed at service process runtime
Language of choice	Linguistic context, Syntax context
Measures and standards of the courseware content need to be consistent	Measures and standards context, Effect context
Mobile device might have feature-based requirements	Device context, Effect context
GPRS connection may have feature-based requirements	Connection context, Effect context
A low GPRS connection may influence on reliability and/or security aspects of constituent services	Connection context, reliability context, security context, Effect context, Protocol context
Trustworthiness of constituent services and/or contents may be needed.	Trust context, Effect context
How well constituent services and/or contents align with governmental regulations may be needed.	Regulatory context, Effect context, Protocol context

Figure 8.2: Context aspects in an e-learning courseware generation scenario

A scenario from convenience services application domain :

There is a trend towards convenience services based application developments, such as http://www.moneysupermarket.com/. We explored a multilingual convenience services sample scenario from the convenience services application domain and analysed it for the use of context categories in defining dynamic requirements. *A user wants to get a suitable broadband provider and also all information in a language of his choice.* Here, we assume that the convenience service process architecture combines required services and outputs information regarding broadband services available based on user requirements. We assume the constituent services are context-aware services. This sample scenario outputs a broadband service information available in a user selected area in a user requested language. The trustworthiness, measures and standards, and governmental regulations attached to constituent services are also assumed as requirements. The identification of dynamic requirements and analysis of context categories on them relevant to this scenario is illustrated in figure 8.3.



Requirements at process runtime	Context categories appeared at service process runtime
Language of choice	Linguistic context, Syntax context
Location	Location context
Trustworthiness of constituent services and content is needed.	Trust context, Effect context
How well constituent services and content align with governmental/organisational regulations?	Regulatory context, Effect context, Protocol context
Measures and standards need to be consistent with local values of them	Measures and Standards context, Syntax context

Figure 8.3: Context aspects in a multilingual convenience services scenario

Completeness perspective of context categories

Analysis of application scenarios from two complementary domains for the completeness perspective of context categories compared to the proposed context model ontology is the concern. We observed whether all required dynamic requirements can be covered by context categories in the context model to test the hypothesis that *context categories can cover all required dynamic requirements.* This complements the validity, which we evaluated in section 8.2.2. The completeness is generally a relative property of a data model [Böhlen 1995]. As in the validity perspective in section 8.2.2, we explored the e-learning courseware generation scenario in figure 8.2 and the multilingual convenience services scenario in figure 8.3, and compared their dynamic requirements with the context aspects in the proposed context model in chapter 4 to evaluate its completeness. That is, to check that all dynamic requirements in the scenarios can be defined in context categories. The results show that dynamic requirements can be defined in context categories and context categories can be attached to functional context, quality of service context, domain context and platform context of the context model. Dynamic requirements in the scenarios of the two complementary domains can be defined in context categories in the proposed context model. Based on the analysis, we can conclude that the proposed context model is complete.

8.2.3 Discussion : Case study based evaluation

We used a case study-based empirical evaluation to analyse empirical validity and completeness of context categories of the proposed context model ontology. In a case study-based design and evaluation, case studies have to be selected carefully and the lack of rigor is a more general weakness of case study-based research [Yin 2009]. We used exploratory case studies from a classical business domain for the development phase of the context model and used confirmatory case studies from two complementary domains (content oriented domain and convenience services domain) for the evaluation of context aspects. The case studies were designed and analysed with the support of domain experts to mitigate potential weaknesses in the case study based design and evaluation approaches.

We analysed the confirmatory case study scenarios to check context categories attached to dynamic requirements in each scenario. We observed that the context categories can represent the needs of dynamic requirements, which we call the validity of context aspects. We compared the context categories attached to dynamic requirements in the confirmatory case study scenarios with the proposed context model in chapter 4 to evaluate the completeness of context aspects in the context model. Based on the results we observed that the context aspects in the proposed context model are valid, and the proposed context model is complete. In order to further clarify validity and completeness of context aspects in the proposed context model, we used a questionnaire-based experts analysis method.

8.2.4 Analysis of expert opinions : Questions and answers

Nine out of ten social surveys use a questionnaire of some kind [Moser 1986]. A key element in the achievement of reliable and valid information in survey research is the construction of well-written manageable questionnaires [Nardi 2003]. We used domain experts, and questions and answers based qualitative analysis methods to clarify the validity and completeness of our context model. We analysed expert opinions regarding dynamic requirements relevant to composition and execution phases of service-based applications. Experts were selected based on their expert knowledge and experiences on various software application domains.

Questionnaire design and analysis

An important consideration throughout questionnaire construction is the impact of own bias [Kitchenham 2002]. In order to avoid bias in this questionnaire construction, we followed some techniques [Kitchenham 2002],

- developed natural questions so that wording does not influence the way the respondent thinks about the problem.
- developed enough questions from each section to adequately cover the topic.

- arranged the order of questions so that the answer to one does not influence the response to the next.
- provided exhaustive, unbiased and mutually exclusive response categories.
- made clear unbiased instructions.

We defined questionnaire focusing context categories, their validity and completeness aspects for dynamic requirements attached to composition and execution phases of Web service applications. We defined the validity scale as, (1) - Strongly Agree, (2) - Agree, (3) - No Opinion, (4) - Disagree, and (5) - Strongly Disagree.

We also have provided separate questions to evaluate the completeness of our context model, such as questions Q1(b), Q2, Q3(b), Q4(b), Q5(b), and Q6(b). The questionnaire was developed with the support of professionals experienced in SOA application domain and domain experts. The questionnaire can be viewed in Appendix D.

This survey is about dynamic requirements attached to dynamically composed service applications, but it is difficult to find many experts in the industry, who are working on advanced dynamically composed service applications. Therefore, firstly we did a pilot survey to detect any difficulties that were not anticipated at the survey proposal stage. In particular, to get a feeling about how people in the industry would understand the questionnaire. We used five participants in the software industry (Tech leads, Software architects, Project managers) for the pilot survey. The pilot survey resulted in some changes to the questionnaire. In particular, we simplified the terminology used in the questionnaire. After the pilot survey, we assumed the questionnaire to be stable and made the survey available online. We used participants from the software companies distributed in various locations, for example participants from Sri Lanka, Iran, Ireland, Canada, and USA. The participants are from organisations, such as SAP research (FL-USA), Ericsson (Ireland), Enovation solutions (Ireland), Lero (Ireland), SSE Group@DCU (Ireland), Teksystems (Canada), Simon Fraser University (Canada), Pegah aftab (Iran), ASER group @SBU (Iran), CDConsulting (Sri Lanka), Renaissance solutions (Sri Lanka), hsenid (Sri Lanka), Mubasher (Sri Lanka), etc. The participants, who are willing to be acknowledged are Maurice O'Connor, Bardia Mohabbati, Balasundaram Shankarganesh, Pubudu Rathnayake, Hamidreza Sarabadani, Mingxue Wang, Amir Mahjorian, Veronica Gacitua, Mark Melia, Pooyan Jamshidi, Vipul Kuruppu, Mahshid Marbouti, Reza Teimourzadegan, and Soodeh Farokhi. We assumed this variation, that is participants from various locations, in various cultures, and expertises in various domains of software developments can result in an un-biased practical result.

8.2.5 Analysis of expert opinions : Results and discussion

Our survey starts with an introduction page, which describes our definition of dynamic service context and fundamental information required through out the survey. Two types of questions (open and closed questions) were presented to find the validity and completeness of dynamic service context definition and context categories. The survey was made available online and sent to ninety experts. From the participants, eighteen have completed the survey. We analysed the results to evaluate the validity and completeness of dynamic service context definition and context categories.

Validity - definition of dynamic service context and context categories

The validity of context categories was surveyed through the expert opinions using closed questions. The expert opinions were collected, analysed and illustrated to check the validity of the definition of dynamic service context and context categories.

Validity of the definition of dynamic service context was analysed from the answers given in Q6(a) and illustrated in figure 8.4.

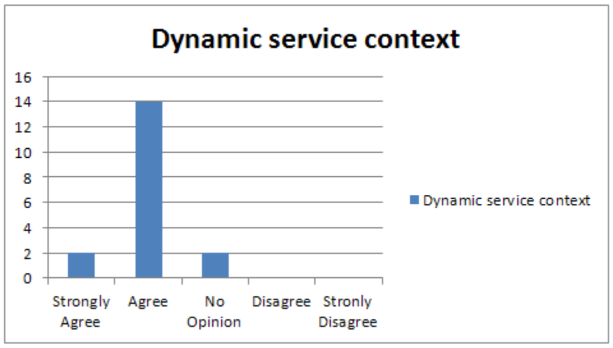


Figure 8.4: Validity - definition of dynamic service context

The results show that most of the participants agree with the definition of dynamic service context and its content. That is, we can clarify that our definition of dynamic service context is valid.

From Q1(a), we surveyed the expert opinions with regard to quality of service context categories. The survey results have been illustrated in figure 8.5 to check the validity of QoS context categories, that is context categories can be requirements in a service process at process run-time.

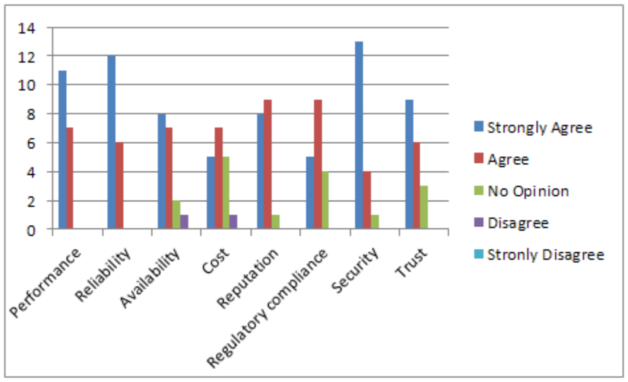


Figure 8.5: Quality of service context categories and their validity

Moreover, each quality of service context category along with expert opinions were analysed and illustrated in figure 8.6. We can observe that most of the participants are either strongly agree or agree with the claim.

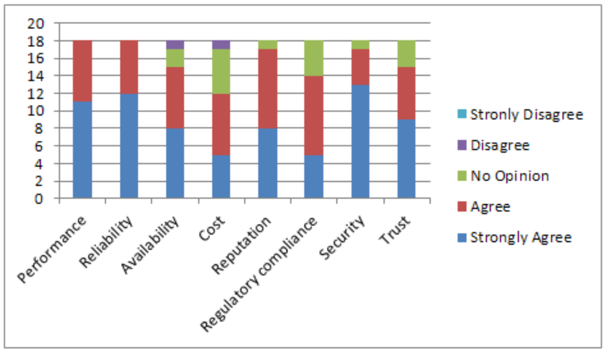


Figure 8.6: Quality of service context categories and expert opinions analysis

In line with the observations, we can conclude the validity of quality of service context categories, that is QoS context categories can be requirements in a service process at process run-time.

From Q3(a), we surveyed the experts opinions with regard to domain context categories. The survey results have been illustrated in figure 8.7 to observe the validity of domain context categories.

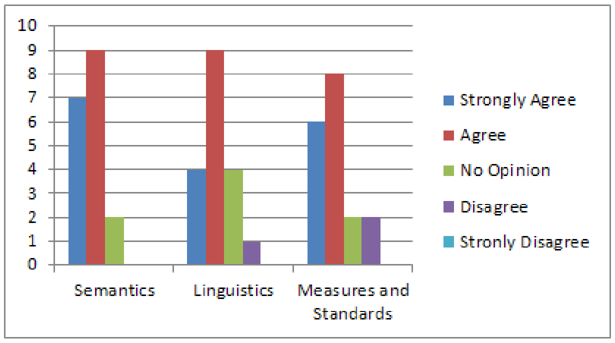


Figure 8.7: Domain context categories and their validity

Each domain context category along with expert opinions were analysed and illustrated in figure 8.8. We can observe that most of the participants are either strongly agree or agree with the claim.

In line with the observations, we can conclude the validity of domain context cat-

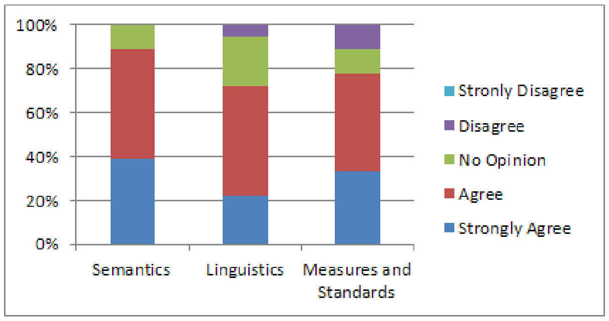


Figure 8.8: Domain context categories and expert opinions analysis

egories, that is domain context categories can be requirements in a service process at process run-time.

From Q4(a), we surveyed the expert opinions with regard to platform context categories. The survey results have been illustrated in figure 8.9 to observe the validity of platform context categories. In here, experts have mixed opinions about operating system context.

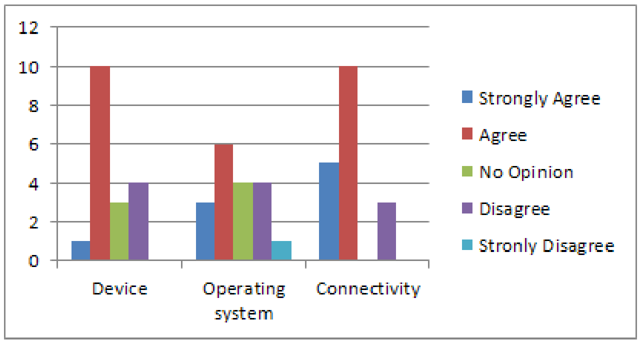


Figure 8.9: Platform context categories and their validity

The each platform context category along with expert opinions were analysed and illustrated in figure 8.10. We can observe most of the participants are either strongly agree or agree with the claim.

In line with the observations, we can conclude the validity of platform context categories, that is platform context categories can be requirements in a service process at process run-time.

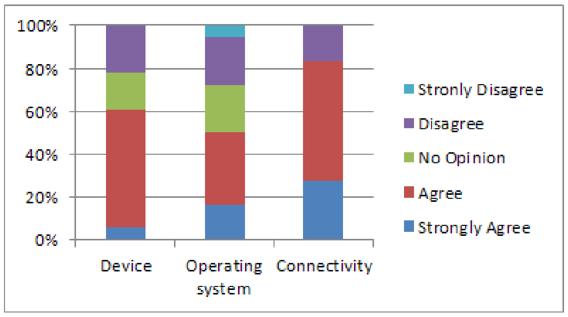


Figure 8.10: Platform context categories and expert opinions analysis

From Q5(a), we surveyed the expert opinions with regard to functional context categories. The survey results have been illustrated in figure 8.11 to observe the validity of functional context categories.

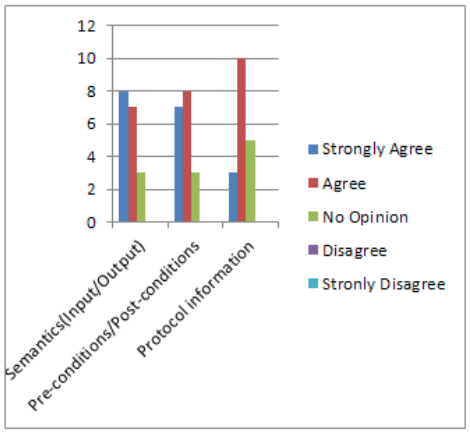


Figure 8.11: Functional context categories and their validity

The each functional context category along with expert opinions were analysed and illustrated in figure 8.12. We can observe most of the participants are either strongly agree or agree with the claim.

In line with the observations, we can conclude the validity of functional context categories, that is functional context categories can be requirements in a service process at process run-time.

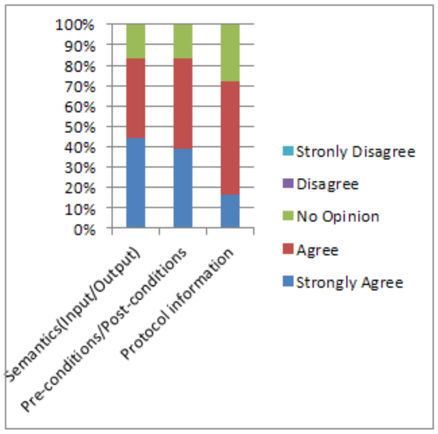


Figure 8.12: Functional context categories and expert opinions analysis

Completeness - definition of dynamic service context and context taxonomy

The concern is survey open questions to evaluate the completeness of context categories and the definition of dynamic service context.

The completeness of QoS context. We raised Q1(b) specifically focusing on completeness of aspects defined in Q1(a). Most of the participants agree about the comprehensiveness of defined context categories. However, some participants proposed some features, which we find not sufficiently related to the definition of dynamic service context since dynamic service context addresses requirements attached to composition and execution of services at process run-time. For example, maintainability, scalability and granularity are not process run-time related aspects.

The completeness of QoS context compared to ISO/IEC 9126. We raised Q2 to check whether the context categorised defined in Q1 cover the scope of dynamic service context attached to ISO / IEC 9126 software quality standard. Most of the participants agree that the QoS context is complete in comparison to ISO / IEC 9126 software quality standard in the scope of dynamic service context. The results are illustrated in figure 8.13.

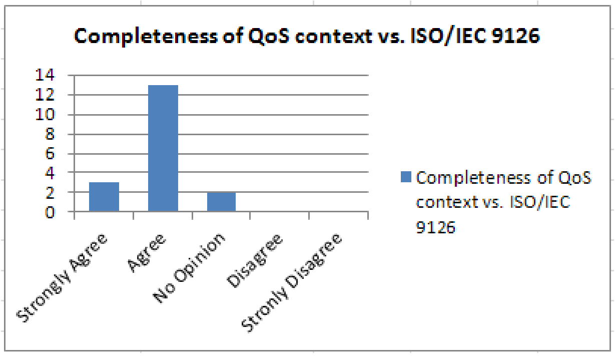


Figure 8.13: Completeness of QoS context vs. ISO / IEC 9126

The completeness of domain context. We raised Q3(b) specifically focusing on completeness of aspects defined in Q3(a). Most of the participants agree with the claim that the defined aspects are complete. One participant mentioned that legislative requirements could be integrated. However, in our context taxonomy the legislative requirements attached to services can be defined in regulatory compliance context attached to QoS context.

The completeness of platform context. We raised Q4(b) specifically focusing completeness of aspects defined in Q4(a). Most of the participants agree that these aspects are complete, but one participant says device / hardware is not considered in Web service technology stack. However, according to our understanding, smart services are tightly integrated with devices and hardware therefore device / hardware context are necessary.

The completeness of functional context. We raised Q5(b) specifically focusing completeness of aspects defined in Q5(a). Most of the participants' opinion is that the defined aspects are complete. One participant, however says that the co-requisites property (two services need to communicate with another service at the same time) is missing. However, in our categorisation, this kind of property can be defined in protocol context.

The completeness of dynamic service context definition. We asked Q6(a) as the last question after providing them with a clear description about the quality model of ISO / IEC 9126, all the context categories, and our definition for dynamic service context. We gathered their views and illustrated them in figure 8.14.

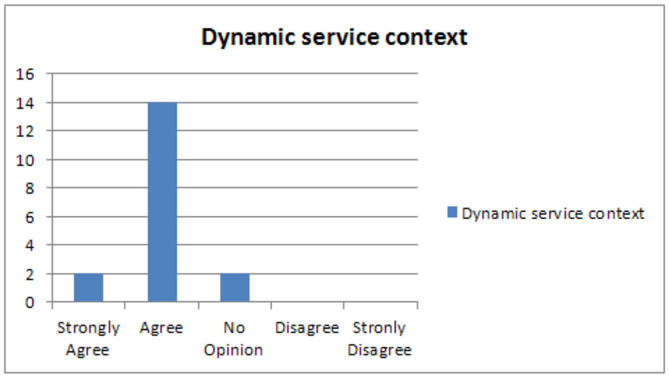


Figure 8.14: Completeness - definition of dynamic service context

This illustration clearly shows that most of the people in the industry agree with the definition. In Q6(a), we asked from participants whether our definition of dynamic service context is informative enough. Most of the participants agree with our definition, and they think it is informative enough. This clarifies that our definition is complete.

8.2.6 Summary and Discussion

The dynamic service context definition and context categories in CMO were evaluated for validity and completeness aspects. Firstly, we evaluated the context categories for validity using case study scenarios from two complementary domains, which are content-oriented domain and convenience services domain. Then, these two scenarios were explored to evaluate the completeness of context categories compared to the proposed CMO. According to the results, we can conclude that our context categories are valid and the context model is complete. Secondly, we evaluated the dynamic service context definition and context categories in CMO using expert opinions via a survey. The experts were selected from different countries of different cultures, but working in the software industry. The survey results clarified the validity and completeness aspects of dynamic service context definition and context categories defined in the context model ontology. In figure 8.9, experts have mixed opinions about operating system context; however in emerging service applications, various devices support various operating systems, such as ubiquitous service applications, smart devices, etc. Moreover, additional aspects suggested by experts, such as legislative requirements can easily be integrated by software architects, if they find them necessary for different applications using operators proposed in chapter 5, such as the refinement operator.

8.3 Context manipulation and composition

8.3.1 Overview

The context manipulation operators and context compositions can process context for dynamic requirements. These operators and compositions can be evaluated using examples from case study scenarios. That is to check whether manipulation operators and compositions are valid in processing dynamic changes on context specifications, derived context and monitored context for dynamic requirements; while preserving semantics of the context model.

8.3.2 Validity : Case study based empirical evaluation

We illustrate the validity of manipulation operators and compositions using illustrative common examples attached to case study scenarios. We detail examples, their context aspects, and the use of valid operators and compositions. This complements case study examples used in chapters 4, 5, 6 and 7.

Manipulation operators

The context manipulation operators operate on various context aspects attached to context specifications, derived context and monitored context.

- **Renaming.** A SCP, which is a context specification of a service uses `ExecutionTimeContext` instead of `ResponseTimeContext`, which is in the context ontology. The operator **Renaming** can be used to map this specification element to the ontology element.

- **Restriction.** We considered a situation where a service process needs to be restricted on security context of constituent services. The operator **Restriction** can be used to restrict the SCPs of services for the security context.

- **Refinement.** We considered a situation where the service process dynamically decides the output service. For example, the output service could be a Fax service, Email service or Mobile service. The device context and mobile connection context of a user is needed to select a Mobile service. For example, if the output message is of type MMS, the user device and mobile connection need to support MMS messaging. The operator **Refinement** can be used to consistently extend the context specification of the process with device context and connection context.

- **Lowest Common Denominator ($\cup^{+}_{LCD}$).** The security of a service process or composite service can be an important aspect. The operator $\cup^{+}_{LCD}$ can be used to determine the overall security of a service process by the weakest security of individual services. The operator $\cup^{+}_{LCD}$ can be applied to security context attached to context specifications of the service process or composite service. If a service fails at process run-time, a new service will be replaced and the operator $\cup^{+}_{LCD}$ can dynamically determine the security context of the process.

- **Least Common Subsumer ($\cup^{+}_{LCS}$).** In a service process at run-time, a service fails and a new service needs to be replaced (dynamic service composition). For the language aspect, the language common to all constituent services is needed at dynamic service composition. That is, the operator $\cup^{+}_{LCS}$ can be applied to language context attached to context specifications.

- **Logical OR ($\cup^{+}_{OR}$).** This operator can be used on context specifications attached to a service process, for e.g. the deployment environment needs a secure internet connection or a connection bandwidth greater than 10Mbps.

- **Accumulation ($\cup^{+}_{ACC}$).** The response time of a service can only be measured at service run-time. The response time of a process is the sum of response times of individual services, which can be determined by this operator.

- **Logical AND ($\cup^{+}_{AND}$).** A service with cost < 0.3 USD and high in availability is needed at dynamic service composition, then this operator can be used on SCPs of possible services to determine the cost and availability.

- **Mediation ($\cup^{+}_{MED}$).** The output context of a service needs to be compatible with the input context of the next service, at service composition. For e.g. if a service has an output of currency type USD, then the next service needs to have an input of the same currency type. This operator can be used for such context mediation at dynamic service composition.

Context composition

The concern is to check the validity of the context composition for context model specifications, monitored context and derived context.

- **Structural composition.** The security context of a service is the sum of Integrity, Authentication, Non-repudiation and Confidentiality contexts of the service.

- **Sequential composition.** This supports behavioral composition of services in a service process, which requires sequential composition of context specifications, monitored context, and derived context of the sequence of services. For example, the security context of a process is the composition of security contexts of the constituent services in the sequence.

8.3.3 Discussion

We used examples from case study scenarios to evaluate the validity of context manipulation operators and context composition. This is a qualitative evaluation. We illustrated the validity of manipulation operators and compositions using examples. The results indicated that our manipulation operators and compositions on context specifications, monitored context and derived context are valid.

8.4 Context constraints generation

In our approach, dynamic requirements are generated as context constraints in terms of conceptualized dynamic service context and context constraints are dynamically instrumented and validated at process run-time. The performance of the dynamic instrumentation and validation is important, which we address in section 8.5.In this section, we analyse the time complexity of the proposed context constraint generation algorithm.

8.4.1 Overview

We evaluated the context constraints generation process for performance. The concern was an analytical evaluation, so that an analytical model with variables and parameters was defined.

8.4.2 Performance : Analytical evaluation

We defined an analytical model and analysed the performance of the CVCs generation algorithm. The symbols used to represent variables and performance measurement parameters of the analytical model are defined in table 8.1.

Table 8.1: **Algorithm 6.1**- Performance evaluation variables and parameters

Variables	
N_R	Number of requirements
N_{SCP}	Number of service context profiles
$N_{Context}$	Number of context categories
$N_{Operator}$	Number of context operators
$N_{constraintValue}$	Number of constraint values
Performance measurement parameters	
T_{CMO}	Time to fetch the context model ontology
T_{scope}	Time to get the scope of a requirement
$T_{constraintValue}$	Time to get the constraint value of a requirement
$T_{Operator}$	Time to get an operator of a requirement
T_{SCP}	Time to fetch a service context profile
$T_{Context}$	Time to fetch a context category
T^S_{dCVC}	Time to generate direct validation constraint of service-level
T_{mCVC}	Time to generate context matching constraint
T^P_{dCVC}	Time to generate direct validation constraint of process-level
T_λ	Variable time taken to build a constraint logic and generate the CLiX syntax

As we have discussed in chapter 6, CVCs are of two types, direct validation constraints and context matching constraints. The direct validation constraints can be of service-level or process-level. The time needed for generating a direct validation constraint of service-level is,

$$T^S_{dCVC} = T_{CMO} + T_{scope} + (T_{Context} \text{ x } N_{Context}) + (T_{constraintValue} \text{ x } N_{constraintValue}) + (T_{Operator} \text{ x } N_{Operator}) + T_\lambda$$

The time needed for generating a context matching constraint is,

$$T_{mCVC} = T_{CMO} + T_{scope} + (T_{SCP} \text{ x } N_{SCP}) + (T_{Context} \text{ x } N_{Context}) + (T_{Operator} \text{ x } N_{Operator}) + T_\lambda$$

The time needed for generating a process-level CVC is,

$$T^P_{dCVC} = T_{CMO} + T_{scope} + T_{Context} + T_{constraintValue} + T_{Operator} + T_\lambda$$

For each requirement, this algorithm executes once. That is, for n requirements this algorithm executes n times. We can define time complexity, $T(n) = kn$ where k is a real number, and its value depends on the constraint type. The time complexity of the algorithm is of order n, $O(n)$.

8.4.3 Discussion

We proposed a context constraints generation algorithm and evaluated the algorithm analytically for performance. We observed that the performance of the algorithm is of order n, $O(n)$. We implemented context constraints generation using the Message Automation Workbench [1], which facilitates creating first order logic based, validation support context constraints in CLiX [2]. The core technologies used in Web service frameworks, such as Web service description language (WSDL), universal description discovery and integration (UDDI), business process execution language (BPEL), and simple object access protocol (SOAP) are XML-based, hence XML-based constraints bring consistency for service-based applications. The input and output models in our constraints generation process are XML compatible. Therefore, a model transformation techniques can be used to extend context constraints generation process.

8.5 Constraints instrumentation and validation monitoring

8.5.1 Overview

We used both analytical evaluation and experimental evaluation to evaluate the constraints instrumentation and validation monitoring for performance. We used an analytical evaluation to evaluate the algorithms of constraint set generation, constraint selection, and configurator generation for performance. We imple-

[1]http://www.messageautomation.com/
[2]http://clixml.sourceforge.net/

mented a prototype to experimentally evaluate constraints instrumentation and validation monitoring at process run-time for performance.

8.5.2 Performance : Analytical evaluation

We analytically evaluated the algorithms of constraint sets generation, constraints selection, and configurators generation for performance. The variables and performance measurement parameters of the analytical models have been defined accordingly.

Evaluation : Constraint sets generation algorithm

The symbols used to represent variables and performance measurement parameters of the analytical model have been defined in table 8.2.

Table 8.2: **Algorithm 7.2** - Performance evaluation variables and parameters

Variables	
N_{CC}^{SL}	Number of service-level context constraints
N_{CC}^{PL}	Number of process-level context constraints
N_{Cset}^{SL}	Number of service-level constraint sets
N_{Cset}^{PL}	Number of process-level constraint sets
Performance measurement parameters	
T_{Cset}	Time to create and initialise a constraint set
T_{AddCC}	Time to Add a CC to a constraint set
T_{scope}	Time to get the scope of a CC
T_{SID}	Time to fetch a service ID attached to a CC
T_{CON}	Time to get the condition status of a CC
T_{VP}	Time to fetch the validation point of a CC
T_{IDCset}	Time to load a constraint set of given ID

The time needed to generate constraint sets from a given set of CCs at service-level is,

$$T_{SL} = (T_{Cset} \text{ x } N_{Cset}^{SL}) + (\ T_{scope} + T_{SID} + T_{CON} + T_{IDCset} + T_{AddCC}) \text{ x } (N_{CC}^{SL})$$

The average time needed for service-level constraint sets generation is,

$$T_{SL}^{AVG} = T_{SL}\ /\ N_{Cset}^{SL}$$

The time needed to generate constraint sets from a given set of CCs at process-level is,

$$T_{PL} = (T_{Cset} \text{ x } N^{PL}_{Cset}) + (T_{scope} + T_{VP} + T_{IDCset} + T_{AddCC}) \text{ x } (N^{PL}_{Cset})$$

The average time needed for process-level constraint sets generation is,

$$T^{AVG}_{PL} = T_{PL} \ / \ N^{PL}_{Cset}$$

This algorithm will not proceed if there are no context constraints, and there are no loops within the algorithm, therefore infinite loops are not possible. This algorithm executes once for one context constraint and n times for n context constraints. Therefore, the time complexity of the algorithm is of order n, O(n).

Evaluation : Constraints selection algorithm

The symbols used to represent variables and performance measurement parameters of the analytical model have been defined in table 8.3.

Table 8.3: **Algorithm 7.3** - Performance evaluation variables and parameters

Variables	
$N^{SL}_{CSelector}$	Number of service-level constraint selectors
$N^{PL}_{CSelector}$	Number of process-level constraint selectors
N^{SL}_{Cset}	Number of service-level constraint sets
N^{PL}_{Cset}	Number of process-level constraint sets
Performance measurement parameters	
$T_{CSelector}$	Time to create and initialise a constraint selector
$T_{AddCset}$	Time to Add a Cset to a constraint selector based on priority
T_{scope}	Time to get the scope of a constraint set
T_{SID}	Time to fetch a service ID attached to a constraint set
T_{CON}	Time to get the condition status of a constraint set
T_{VP}	Time to fetch the validation point of a constraint set
$T_{IDCselector}$	Time to load a constraint selector of given ID
$T_{PrioCset}$	Time to get the priority of a constraint set

The time needed to generate constraint selectors from a given set of constraint sets at service-level is,

$$T_{SL} = (T_{CSelector} \text{ x } N^{SL}_{CSelector}) + (T_{scope} + T_{SID} + T_{CON} + T_{IDCselector} + T_{PrioCset} + T_{AddCset}) \text{ x } (N^{SL}_{CSelector})$$

The average time needed for service-level constraint selectors generation is,

$$T_{SL}^{AVG} = T_{SL} \ / \ N_{CSelector}^{SL}$$

The time needed to generate constraint selectors from a given set of constraint sets at process-level is,

$$T_{PL} = (T_{CSelector} \text{ x } N_{CSelector}^{PL}) + (T_{scope} + T_{VP} + T_{IDCselector} + T_{PrioCset} + T_{AddCset}) \text{ x } (N_{CSelector}^{PL})$$

The average time needed for process-level constraint selectors generation is,

$$T_{PL}^{AVG} = T_{PL} \ / \ N_{CSelector}^{PL}$$

The constraint selector generation algorithm will not proceed if there are no constraint sets, and also there are no loops within the body of the algorithm, therefore infinite loops are not possible. This algorithm executes once for one constraint set and n times for n constraint sets. Therefore, the time complexity of the algorithm is of order n, O(n).

Evaluation : configurator generation algorithm

The symbols used to represent variables and performance measurement parameters of the analytical model have been defined in table 8.4.

Table 8.4: **Algorithm 7.4 -** Performance evaluation variables and parameters

Variables	
$N_{Configurator}$	Number of service-level constraint selectors
N_{Cset}	Number of constraint sets
Performance measurement parameters	
$T_{Configurator}$	Time to create and initialise a configurator
T_{CSets}	Time to read constraint sets of a selector into an array
$T_{AddSelector}$	Time to add a selector to a configurator
$T_{AddCset}$	Time to Add a Cset to a configurator
$T_{CHKCselector}$	Time to check whether selector is null

The time needed to generate a configurator for a given collector is,

$$T_{Configurator} = T_{CHKCselector} + T_{Configurator} + T_{CSets} + T_{AddSelector} + (T_{AddCset} \text{ x } N_{Cset})$$

For a given process, there is more than one configurator. Therefore, the complete configurator generation time for a service process is,

$$T^{P}_{ConfigGeneration} = T_{Configurator} \text{ x } N_{Configurator}$$

The configurator generation algorithm will not proceed, if there are no constraint selectors, and there is only one loop, which works for a number of constraint sets. If there are no constraint sets, then this loop will not proceed. There is no possibility for infinite loops. This loop proceeds once for one constraint set and n times for n constraint sets. Therefore, the time complexity of the algorithm is of order n, O(n).

8.5.3 Performance : Experimental evaluation

We experimentally evaluated the performance of the proposed approach of constraints instrumentation and validation monitoring. The performance is evaluated based on the process execution time including instrumentation and validation monitoring. We looked at performance in execution time over various constraint settings, for various invocations of instrumented service processes. We used three services processes, which have 3, 5 and 7 constituent services to test our approach for a practical result. We assume, Web service applications often have constituent services around these numbers. We measured performance by implementing test cases with different number of constraints, and testing for number of successful instrumentation and validations.

Methods and experimental setup

We defined an experimental method based on case study scenarios. Case study scenarios utilise various types of context from our context model. At the core of our case study scenario, we have the user, the enterprise client (E-client), and service providers. The E-client uses services provided by external service providers, distributed around the world. We used free services provided by www.xmethods.net to make the evaluation method more practical. However; these services do not have the appropriate relevance compared to the constituent services in real world applications. Our interest is not about the relevance of these constituent services, but about their dynamic features, which we defined using the notion of context. We defined service context profiles for each constituent service based on our context model. Service context profiles cover all major context categories defined in the context model to make the evaluation process more practical.

In order to perform the evaluation for performance, we compared process execution / validation time of constraints instrumented processes and un-instrumented processes (base processes). The correctness of successful validations was monitored using log files facilitated by our experimental setup.

Base process:

In the first phase, we defined 3 service processes with 3, 5 and 7 constituent services and called them base processes. We defined a service context profile for each constituent service. Constituent services were composed in a BPEL process called a BPEL module. The BPEL module is then taken into a deployable and testable application called a composite application. Then, we deployed the composite application, created a test case, executed the test case 600 times, and collected process execution times at each execution. All the constituent services were invoked through the internet and the network traffic can influence on the process execution time. In order to minimise the error resulting from the variation of network traffic, we executed all process instances within a fixed time interval and each process instance executed 600 times. We developed a service process, which calculates the process execution time and logs each process execution time into a text file. Service execution statuses, which are success and failure information relevant to services invocation were recorded in a log file. In order to execute each test case 600 times, we defined *concurrent.properties* of each test case as follows in NetBeans IDE 6.5 environment.

```
concurrent.properties


calculatethroughput=false
#comparisontype’s possible values: identical|binary|equals
comparisontype=identical
concurrentthreads=10
description=testcase TestCase1
destination=http://localhost:${HttpDefaultPort}/RequestReplyService/RequestReplyPort
#featurestatus’s possible values: progress|done
featurestatus=done
inputfile=Input.xml
invokesperthread=60
outputfile=Output.xml
testtimeout=600
```

The test drivers use the concurrent.properties files as their configuration for the testing process. Concurrent threads - each thread can invoke a test case independently. In here, ten concurrent threads were created for this test case. Invokes per thread - this defines the number of times each thread invokes the test case. Test timeout (sec) - this defines how long each thread has to finish and if it does not finish in the allocated time, then an exception is thrown. The number of executions of a test case is calculated as the number of concurrent threads times the number of invocations per thread, that is (10 * 60 = 600). The computer platform, which we used to run this prototype has a 32-bit Windows vista home premium operating system, a Intel(R) Core(TM)2 Duo 2.00 GHz processor and 3.00 GB RAM.

Constraints instrumented process:

In the second phase, we defined the constraint instrumented service processes, which has 3, 5 and 7 constituent services. Each service process is instrumented with various types of context constraints. We defined context constraints, so that they cover all the major context categories defined in the context model. Each constituent service was instrumented with pre and post validation services, and they were dynamically bound to different number of context constraints. Context constraints validate service context profiles (SCPs) and transient context instances (TCIs) relevant to constituent services and service processes. We defined test cases and concurrent.properties files as in the base process. The process execution time including constraint instrumentations and validations were collected. The validation logs were analysed for validations / violations of constraints and service invocations / violations.

Experimental evaluation and results

Our experimental evaluation focuses on evaluating the constraints instrumentation and validation for performance. We implemented test cases of various constraint settings. In these test cases, we addressed context aspects relevant to both pre-defined context and transient context. The pre-defined context are cost context of a service, trust context of a service, etc. The transient context are response time context of a service, availability context of a service, etc.

Service process of three constituent services (P1):

A service process with three constituent services was dynamically instrumented with different numbers of constraints for constraints validation at process runtime. The run-time instrumentation and validation time for test cases for the number of executions have been illustrated in figure 8.15. We developed 9 test cases (Series 1 to 9) relevant to various constraint settings. Series 1 is an uninstrumented process and series 2 to 9 are constraints instrumented processes, which have 6, 12, 18, 24, 30, 36, 42, 48 constraints respectively.

We used T-test statistical analysis techniques to analyse variations of process

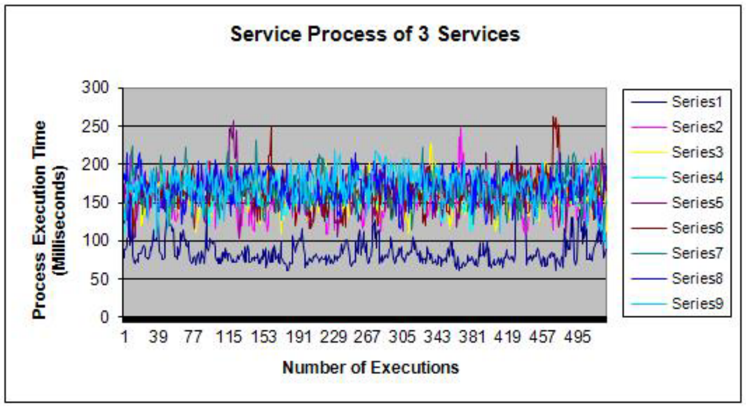
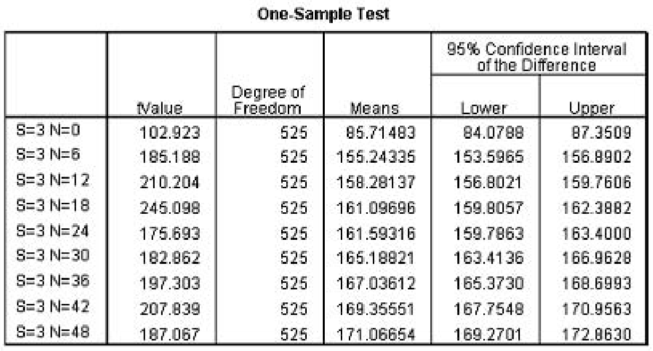


Figure 8.15: P1 with various constraint settings

execution times with and without constraint validations. The independent one-sample T-test analysis has been illustrated in figure 8.16. tValue is the value of t defined in one-sample T-test.



One-Sample Test

	tValue	Degree of Freedom	Means	95% Confidence Interval of the Difference	
				Lower	Upper
S=3 N=0	102.923	525	85.71483	84.0788	87.3509
S=3 N=6	185.188	525	155.24335	153.5965	156.8902
S=3 N=12	210.204	525	158.28137	156.8021	159.7606
S=3 N=18	245.098	525	161.09696	159.8057	162.3882
S=3 N=24	175.693	525	161.59316	159.7863	163.4000
S=3 N=30	182.862	525	165.18821	163.4136	166.9628
S=3 N=36	197.303	525	167.03612	165.3730	168.6993
S=3 N=42	207.839	525	169.35551	167.7548	170.9563
S=3 N=48	187.067	525	171.06654	169.2701	172.8630

Figure 8.16: P1 T-test analysis

The degree of freedom is (sample size - 1). However, our interest is on the 95% confidence interval of the difference.

Figure 8.17 illustrates the variation of average process execution time for different constraint settings in test cases. We can observe that the increase in execution times of instrumented processes (N=6, N=12, N=18, N=24, N=30, N=36, N=42, N=48) is very small. The only significant increase is at N=0 to N=12, but it is less than twice the mean value of the uninstrumented process.

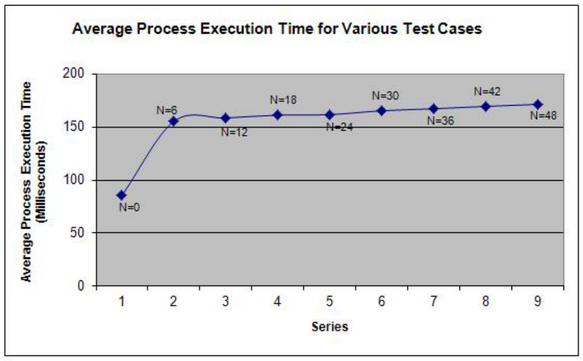


Figure 8.17: P1 average execution time

Service process of five constituent services (P2):

A service process with five constituent services was dynamically instrumented with different numbers of constraints for constraints validation at process run-time. The run-time instrumentation and validation time for test cases for the number of executions have been illustrated in figure 8.18. We developed 9 test

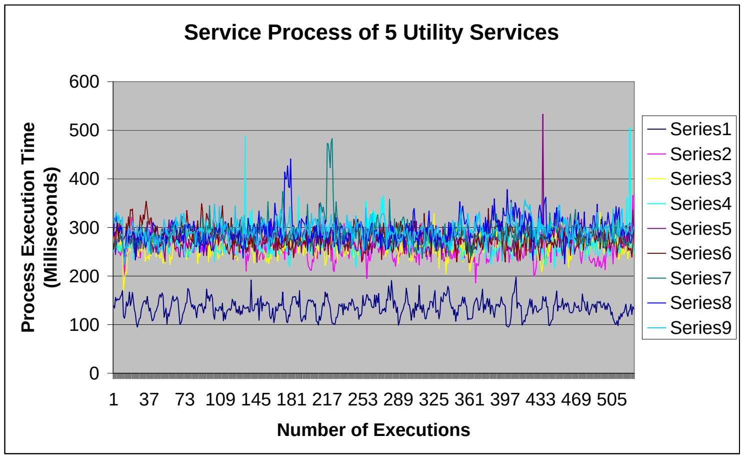
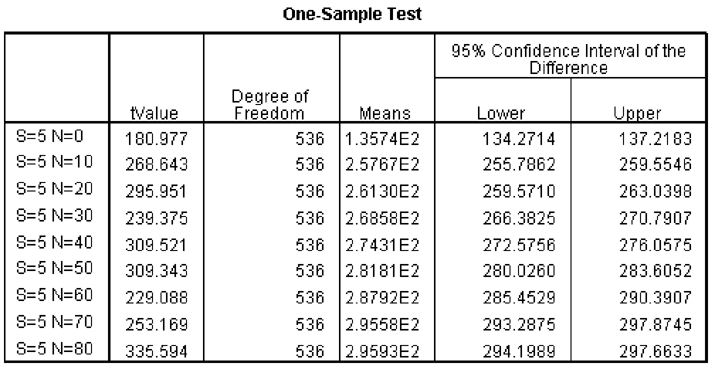


Figure 8.18: P2 with various constraint settings

cases (Series 1 to 9) relevant to various constraint settings. Series 1 is an uninstrumented process and series 2 to 9 are constraints instrumented processes, which have 10, 20, 30, 40, 50, 60, 70, 80 constraints respectively.

We used T-test statistical analysis techniques to analyse variations of process execution times with and without constraint validations. The independent one-sample T-test analysis has been illustrated in figure 8.19.



One-Sample Test

	tValue	Degree of Freedom	Means	95% Confidence Interval of the Difference	
				Lower	Upper
S=5 N=0	180.977	536	1.3574E2	134.2714	137.2183
S=5 N=10	268.643	536	2.5767E2	255.7862	259.5546
S=5 N=20	295.951	536	2.6130E2	259.5710	263.0398
S=5 N=30	239.375	536	2.6858E2	266.3825	270.7907
S=5 N=40	309.521	536	2.7431E2	272.5756	276.0575
S=5 N=50	309.343	536	2.8181E2	280.0260	283.6052
S=5 N=60	229.088	536	2.8792E2	285.4529	290.3907
S=5 N=70	253.169	536	2.9558E2	293.2875	297.8745
S=5 N=80	335.594	536	2.9593E2	294.1989	297.6633

Figure 8.19: P2 T-test analysis

The degree of freedom is (sample size - 1). However, our interest is on the 95% confidence interval of the difference.

Figure 8.20 illustrates the variation of average process execution time for different constraint settings in test cases. We can observe that the increase in execution

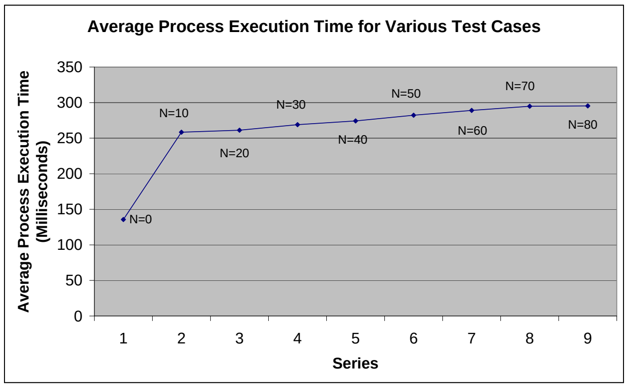


Figure 8.20: P2 average execution time

times of instrumented processes (N=10, N=20, N=30, N=40, N=50, N=60, N=70, N=80) is very small. The only significant increase is at N=0 to N=10, but it is less than twice the mean value of the uninstrumented process.

Service process of seven constituent services (P3):

A service process with seven constituent services was dynamically instrumented with different numbers of constraints for constraints validation at process run-time. The run-time instrumentation and validation time for test cases for the number of executions have been illustrated in figure 8.21. We have 9 test cases (Se-

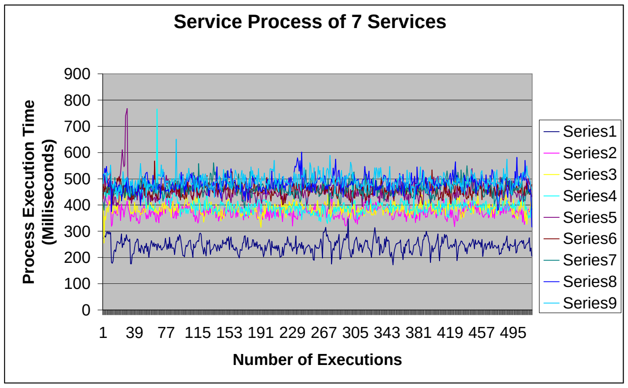
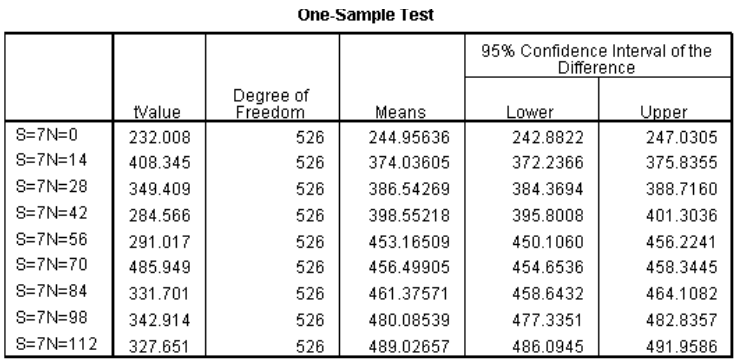


Figure 8.21: P3 with various constraint settings

ries 1 to 9) relevant to various constraint settings. Series 1 is an uninstrumented process, and series 2 to 9 are constraints instrumented processes, which have 14, 28, 42, 56, 70, 84, 98, 112 constraints respectively.

We used T-test statistical analysis techniques to analyse variations of process execution times with and without constraint validations. The independent one-sample T-test analysis has been illustrated in figure 8.22. The degree of freedom



One-Sample Test

	tValue	Degree of Freedom	Means	95% Confidence Interval of the Difference	
				Lower	Upper
S=7N=0	232.008	526	244.95636	242.8822	247.0305
S=7N=14	408.345	526	374.03605	372.2366	375.8355
S=7N=28	349.409	526	386.54269	384.3694	388.7160
S=7N=42	284.566	526	398.55218	395.8008	401.3036
S=7N=56	291.017	526	453.16509	450.1060	456.2241
S=7N=70	485.949	526	456.49905	454.6536	458.3445
S=7N=84	331.701	526	461.37571	458.6432	464.1082
S=7N=98	342.914	526	480.08539	477.3351	482.8357
S=7N=112	327.651	526	489.02657	486.0945	491.9586

Figure 8.22: P3 T-test analysis

is (sample size - 1). However, our interest is on the 95% confidence interval of the difference.

Figure 8.23 illustrates the variation of average process execution time for different constraint settings in test cases. We can observe that the increase in execution

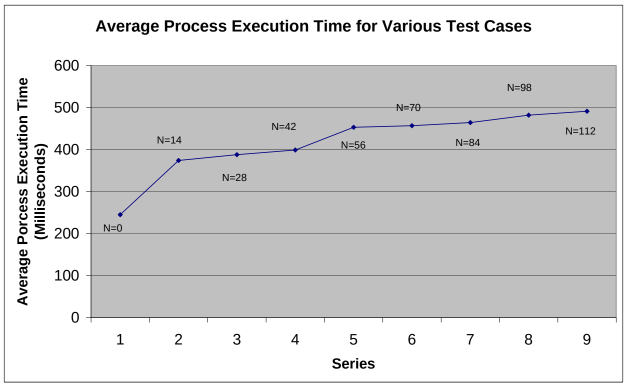


Figure 8.23: P3 average execution time

times of instrumented processes (N=14, N=28, N=42, N=56, N=70, N=84, N=98, N=112) is very small. The only significant increase is at N=0 to N=14, but it is less than twice the mean value of the uninstrumented process.

8.5.4 Tool support and discussion

Firstly, we analytically evaluated constraint sets generation, constraint selector generation, and configurator generation algorithms for performance and observed their execution time complexity is of order n. Secondly, we used the NetBeans IDE 6.5 environment for implementing the experiment setup. We detailed all the technical tools used for instrumentation and validation monitoring in chapter 7. We used SPSS 16.0, which is a statistical analysis tool to perform statistical tests, such as the T-test. The T-test descriptive table displays the tValue, degree of freedom, mean and confidence interval of the difference. We considered a 95% confidence interval for the T-test analysis. All the used services are external services (service providers in the USA), which involve network traffic. However, we tested all the invocations continuously in a fixed time interval to minimise the error resulting from the variation of network traffic. The average process execution times of test cases were illustrated. The results show that the execution time overhead for constraint instrumentation and validation at process run-time is less than twice the mean value of the un-instrumented process. We interpret this as acceptable for most situations, because the number of constraints taken into account is quite high and all these response times are still in a milliseconds range.

8.6 Threats to validity

8.6.1 Empirical methods

We used expert's opinion analysis and case study analysis for evaluating core research. We used expert's opinion analysis for the evaluation of the context model for validity and completeness. Though experts were selected based on a range of criteria, there is always a possibility an expert not participating in this survey could have added different opinions. There is also a possibility that expert's opinions can change over time with the technology evolution. However, the survey does confirm a reasonable validity and completeness but also that differing opinions need to be accounted for, as done by providing a customisable framework. The case study analysis was used to evaluate the context model and operator calculus. While we have aimed to cover a range of domains, technology is rapidly evolving and new application domains could emerge. The use of the notion of context for further dynamic aspects could be necessary. The operator calculus may also need improvements accordingly in that case.

8.6.2 Analytic methods

We used analytical models to evaluate time complexity of the proposed algorithms and the prototype-based experimental evaluation to estimate and categorise process execution time for instrumentation and validation monitoring. Analytical methods are not sufficient in some situations such as performance evaluation for instrumentation and validation monitoring of context constraints. In particular, it is hard to take into account the workload of constraints, the workload of services and the TCP/IP (Transmission Control Protocol/Internet Protocol) connection-based response times from constituent services. Furthermore, in the proposed prototype-based experimental evaluation, we did not use services with the required relevance for case study scenarios because the required services with appropriate relevance were not available for research purpose and only test configurations were used.

Chapter 9

Conclusions

Contents

9.1 Overview

In service-based application developments, architects initially focused on process design-time aspects to validate and test applications at design-time. However, in emerging service computing applications, dynamic requirements attached to service composition and execution need to be addressed, for example, SLA monitoring at process run-time. We particularly focused on this direction in this thesis. In our approach, we defined dynamic service context and modeled dynamic service context in a context model ontology. We proposed context constraints to define dynamic requirements. Finally, context constraints were instrumented with a service process instance at run-time for dynamic requirements validation monitoring.

In section 9.2, we describe a summary of contributions attached to the objectives, hypotheses, and practical implementations of our work. In section 9.3.1, we discuss assumptions, contributions, and evaluation results. Finally, in section

9.3.2, we detail proposed future work including extensions of this work to other domains.

9.2 Summary of contributions

This thesis contributes a dynamic service context model and its utilisation in Web service applications at process run-time focusing on instrumentation and validation monitoring of their dynamic requirements. Consequently, the context manipulation, composition, reasoning, context constraints generation, run-time instrumentation and validation monitoring based on the context model are introduced. We can summarise our contributions as,

- Dynamic service context definition to define dynamic aspects attached to composition and execution of Web services at process run-time.
- Conceptualize dynamic service context into a processable context model ontology.
- Operator-based context manipulation, context composition, and context reasoning.
- XML-based explicit context validation constraints and service-based implicit context validation constraints, which define dynamic requirements for service processes.
- An architecture and implementation for dynamic instrumentation and validation monitoring of context constraints.

9.3 Discussion and future work

9.3.1 Discussion

There is no common definition for context in information science. Context has been defined in various service-based application domains having a process design perspective [Boukadi 2008, Medjahed 2007, Dey 2000]. These definitions do not sufficiently address process run-time aspects. We identified defining dynamic service context aspects and organising them in a general context space as a challenge. We defined dynamic service context with a focus on composition and execution of services and processes at run-time. We evaluated the definition of dynamic service context based on expert opinions, and results show that our definition is valid and complete.

There are several context taxonomies in the literature, which are defined with a process design time perspective, and a domain specific perspective [Medjahed 2007, Wang 2004, Chen 2004]. We analysed previous context classifications and models, standards such as [9126-1 2001], and scenarios from business application domains for developing a general context taxonomy. Then, we modeled this taxonomy as a context model ontology, which can enrich dynamic service applications with features such as shared conceptualisation, context reasoning, etc. We evaluated the context aspects in our context model ontology for validity and completeness. This evaluation is a two phase evaluation. In the first phase, we used scenarios from complementary service domains. In the second phase, we used expert opinions analysis through a survey. The use of complementary service domains further reinforces that this context model ontology can be used for other service domains towards dynamic service applications. The context specifications and transient context information need to be manipulated, composed and reasoned about in dynamic service applications. We proposed an operator-based manipulation and composition technique and evaluated the operators for correctness using case study examples.

In this thesis, we assume dynamic requirements are captured from SLAs. The client-side specifications define the minimum requirements, whereas the provider-side specifications define the maximum range of capabilities. The context constraints are generated to define dynamic requirements. However, exploring SLAs is out of our scope. An XML-based context constraints specification provides technological compatibility for service-based applications. We proposed a process for the generation of XML-based context validation constraints and an algorithm to define the generation process. The algorithm has been evaluated for performance and it is of order n. We developed context reasoning constraints as reasoning services. We assume a semantic client (or broker) maintains a registry of context reasoning services. The proposed context constraints can be used in any service-based application domain. The context constraints can be dynamically instrumented with a service process and validated at process run-time.

We proposed a framework for the instrumentation and validation monitoring of dynamic requirements at process run-time. We implemented a prototype and case study scenarios to evaluate the dynamic instrumentation and validation monitoring of context constraints. All the services used in scenarios are remotely hosted services. We experimentally evaluated the instrumentation and validation monitoring in a fixed time interval to minimise the effect caused by the network traffic. We assumed the network traffic within that time interval is more or less the same. We also used constraints from all the main context dimensions (functional, QoS, domain, and platform). The data collector directives and data collectors were also used for transient context. The results show that the performance of the instrumentation and validation monitoring for successful validations is a practical value. The proposed dynamic instrumentation and validation monitoring of context validation constraints is a general approach, and can be applied in any service-based application for requirements (or policy) validations. However, the proposed instrumentation and validation monitoring approach for context reasoning services is in a controlled environment. Our framework provides a validation logging mechanism, which generates logs about successes and failures. The validation failures may need service re-composition at run-time, which is beyond our scope.

9.3.2 Future work

We describe some directions for future work in this section as,

- Context annotation for WSDL
- Intelligent decision making for requirements elicitation and improved automated generation of constraints
- Constraints validation failure handling needs service replace and re-composition

Context annotation for WSDL. Semantic annotation for WSDL (SAWSDL) is a W3C specification for building semantics on top of WSDL [Farrell 2007]. In this work, we defined a service context profile (SCP), which has context information attached to a service. This SCP can be improved using model references and schema mappings, so that WSDL description can be attached with the context model ontology. The authors in [Ibáñez 2010] describe how an RDF vocabulary is used to annotate functional and non-functional properties of basic activities of XML-based BPEL processes. A lightweight ontology for semantic Web services is proposed in [Kopecky 2008] called WSMO-Lite, which fills in SAWSDL annotations with concrete semantic constructs. The work described in [Kopecky 2008, Ibáñez 2010] can be a possible starting point for context annotated WSDL.

Intelligent decision making for requirements elicitation and improved automated generation of constraints. Intelligent decision making techniques for dynamic requirements elicitation and automatically generating relevant context constraints at process run-time can be useful research for dynamic service applications. For example, multi-tenant process governance at run-time [Wang 2010]. In the proposed ECVC generation, input models and output models are XML-based models. Therefore, model transformation tools, such as ATL [1], QVT[2], etc. can be used to implement the ECVC generation algorithm, so that ECVCs can be generated at process run-time. This is a model transformation solution for ECVC gen-

[1]http://www.eclipse.org/atl/
[2]http://www.omg.org/spec/QVT/

eration, and that helps to generate new ECVCs for validation monitoring at run-time. This generation process can be integrated with intelligent decision making techniques for dynamic requirements elicitation. Some of the dynamic aspects of a service process can implicitly affect other dynamic aspects. These implicit complexities can be resolved using reasoning techniques on conceptualized dynamic aspects. The proposed context reasoning can reduce the complexity of some aspects, such as deriving implicit context from the explicit context in Web service applications. In this work, we have implemented ICVCs as resoning services, but this kind of an approach can only be used in a pre-defined environment with a set of constraints. An approach and techniques for automatically generating ICVCs and dynamically integrating them with a deployed service process at run-time are needed. These approaches and techniques need to be integrated with intelligent decision making techniques to elicitate relevant dynamic requirements.

Constraints validation failure handling needs service replace and re-composition. The context constraints validation failure (violation) may need service replace and re-composition at process run-time. This type of replace and re-composition can reinforce the capabilities of dynamic service composition. Our proposed framework provides a logging mechanism. Dynamically monitoring logging details and applying self-healing techniques [Baresi 2004, Wang 2009b, Wang 2009a] can be a possible approach to strengthen the service replace and re-composition capability at run-time.

Bibliography

[9126-1 2001] ISO / IEC 9126-1. *Software engineering, Product quality, Part 1: Quality model*, 2001. http: / / www.iso.org / iso / iso-catalogue / catalogue-tc / .

[Agarwal 2008] V. Agarwal, G. Chafle, S. Mittal and B. Srivastava. *Understanding Approaches for Web Service Composition and Execution*. In Proceedings of the 1st Bangalore Annual Compute Conference, COMPUTE '08. ACM, 2008.

[Aggarwal 2004] R. Aggarwal, K. Verma, J. Miller and W. Milnor. *Constraint Driven Web Service Composition in METEOR-S*. In Proceedings of the IEEE International Conference on Services Computing. IEEE Computer Society, 2004.

[Ahmed 2009] Z. Ahmed. Dynamic monitoring and constraint validation framework for autonomic web services. Master's thesis, Dublin City University, Ireland, 2009.

[Alrifai 2009] M. Alrifai and T. Risse. *Combining Global Optimization with Local Selection for Efficient QoS-Aware Service Composition*. In Proceedings of the 18th International Conference on World Wide Web. ACM, 2009.

[Alves 2006] A. Alves, A. Arkin, S. Askary, B. Bloch, F. Curbera, Y. Goland, N. Kartha, Sterling, D. König, V. Mehta, S. Thatte, D. van der Rijn, P. Yendluri and A. Yiu. *Web Services Business Process Execution Language Version 2.0*. OASIS Committee, May 2006.

[Ardagna 2011] D. Ardagna, L. Baresi, S. Comai, M. Comuzzi and B. Pernici. *A Service-Based Framework for Flexible Business Processes*. Journal of IEEE Software, vol. 28, pages 61–67, 2011.

[Atif 2002] Y. Atif. *Building Trust in E-Commerce*. Journal of IEEE Internet Computing, vol. 6, pages 18–24, 2002. IEEE Computer Society.

[Aulbach 2008] S. Aulbach, T. Grust, D. Jacobs, A. Kemper and J. Rittinger. *Multi-tenant Databases for Software as a Service: Schema-Mapping Techniques*. In Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data. ACM, 2008.

[Avizienis 2004] A. Avizienis, J.C. Laprie, B. Randell and C. Landwehr. *Basic Concepts and Taxonomy of Dependable and Secure Computing*. Journal of IEEE Transactions on Dependable and Secure Computing, vol. 1, pages 11–33, 2004.

[Baader 2003] F. Baader, D. Calvanese, D. McGuinness, D. Nardi and P. Patel-Schneider. *The Description Logic Handbook - Theory, Implementation and Applications*. Cambridge University Press, 2003.

[Baader 2007] F. Baader, I. Horrocks and U. Sattler. *Description Logics*. In F.V Harmelen, V. Lifschitz and B. Porter, editors, Handbook of Knowledge Representation, volume 1, pages 135–179. Elsevier, 2007.

[Bandara 2004] A. Bandara, T.R. Payne, D. Roure and G. Clemo. *An Ontological Framework for Semantic Description of Devices*. In International Semantic Web Conference (ISWC), 2004.

[Bandara 2009] K.Y. Bandara, M.X. Wang and Pahl C. *Dynamic Integration of Context Model Constraints in Web Service Processes*. In International Conference on Software Engineering. IASTED, 2009.

[Barbon 2006] F. Barbon, P. Traverso, M. Pistore and M. Trainotti. *Run-Time Monitoring of Instances and Classes of Web Service Compositions*. In Proceedings of the IEEE International Conference on Web Services. IEEE Computer Society, 2006.

[Baresi 2004] L. Baresi, C. Ghezzi and S. Guinea. *Towards Self-Healing Service Compositions*. In Proceedings of the 1st Conference on the Principles of Software Engineering, 2004.

[Baresi 2005] L. Baresi and S. Guinea. *Towards Dynamic Monitoring of WS-BPEL Processes*. volume 3826 of *Lecture Notes in Computer Science*, pages 269–282. Springer, 2005.

[Baresi 2010a] L. Baresi and C. Ghezzi. *The Disappearing Boundary Between Development-time and Run-time*. In Proceedings of the FSE / SDP Workshop on Future of Software Engineering Research, FoSER '10. ACM, 2010.

[Baresi 2010b] L. Baresi, S. Guinea, O. Nano and G. Spanoudakis. *Comprehensive Monitoring of BPEL Processes*. Journal of IEEE Internet Computing, vol. 14, pages 50–57, 2010. IEEE Educational Activities Department.

[Baresi 2011] L. Baresi and S. Guinea. *Self-Supervising BPEL Processes*. Journal of IEEE Transactions on Software Engineering, vol. 37, pages 247–263, 2011.

[Bastida 2008] L. Bastida. *A Methodology for Dynamic Service Composition*. In Proceedings of the Seventh International Conference on Composition-Based Software Systems. IEEE, 2008.

[Battle 2005] S. Battle, A. Bernstein, H. Boley, B. Grosof, M. Gruninger, R. Hull, M. Kifer, D. Martin, S. Mcllraith, D. McGuinness, J. Su and S. Tabet. *Semantic Web Services Framework (SWSF) Overview*, 2005. http://www.w3.org/Submission/SWSF/.

[Bettini 2010] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan and D. Riboni. *A Survey of Context Modelling and Reasoning Techniques*. Journal of Pervasive and Mobile Computing, vol. 6, pages 161–180, 2010. Elsevier.

[Bianculli 2008] D. Bianculli, C. Ghezzi, P. Spoletini, L. Baresi and S. Guinea. *A Guided Tour through SAVVY-WS: A Methodology for Specifying and Validating Web Service Compositions*. In E. Börger and A. Cisternino, editors, Advances in Software Engineering, pages 131–160. Springer, 2008.

[Bikakis 2008] A. Bikakis, T. Patkos, G. Antoniou and D. Plexousakis. *A Survey of Semantics-based Approaches for Context Reasoning in Ambient Intelligence*. In Proceedings of the Artificial Intelligence Methods for Ambient Intelligence at the European Conference on Ambient Intelligence. Springer, 2008.

[Böhlen 1995] M.H. Böhlen, C. S. Jensen and R.T. Snodgrass. *Evaluating the Completeness of TSQL2*. In Proceedings of the International Workshop on Temporal Databases: Recent Advances in Temporal Databases, pages 153–172. Springer, 1995.

[Boley 2010] H. Boley, A. Paschke, S. Tabet, B. Grosof, N. Bassiliades, G. Governatori, D. Hirtle and O. Shafiq. *Schema Specification of RuleML 1.0*, 2010. http://ruleml.org/1.0/.

[Boukadi 2008] K. Boukadi, C. Ghedira, S. Chaari, L. Vincent and E. Bataineh. *How to Employ Context, Web Service, and Community in Enterprise Collaboration*. In Proceedings of the 8th International Conference on New Technologies in Distributed Systems. ACM, 2008.

[Boukadi 2009] K. Boukadi, C. Ghedira, Z. Maamar, D. Benslimane and L. Vincent. *Context-aware Data and IT Services Collaboration in e-Business*. In Transactions on Large-Scale Data- and Knowledge-Centered Systems I, pages 91–115. Springer, 2009.

[Bray 2008] T. Bray, J. Paoli, E. Maler and F. Yergeau. *Extensible Markup Language (XML)*, 2008. http://www.w3.org/TR/2008/REC-xml-20081126/.

[Brickley 2004] D. Brickley and R.V. Guha. *RDF Vocabulary Description Language 1.0: RDF Schema*, 2004. http://www.w3.org/TR/rdf-schema/.

[Cacciagrano 2006] D. Cacciagrano, F. Corradini, R. Culmone and L. Vito. *Dynamic Constraint-Based Invocation of Web Services*. In M. Bravetti, M. Núñez and G. Zavattaro, editors, Web Services and Formal Methods, volume 4184 of *Lecture Notes in Computer Science*, pages 138–147. Springer, 2006.

[Cakmakci 2002] O. Cakmakci, J. Coutaz, K.V. Laerhoven and H.W. Gellersen. *Context Awareness in Systems with Limited Resources*. In Proceedings of the 3rd Workshop on Artificial Intelligence in Mobile Systems (AIMS), ECAI 2002, 2002.

[Cámara 2008] J. Cámara, C. Canal and G. Salaün. *Multiple Concern Adaptation for Runtime Composition in Context-Aware Systems*. Journal of Electronic Notes in Theoretical Computer Science, vol. 215, pages 111–130, 2008. Elsevier.

[Canfora 2005] G. Canfora, M. Di Penta, R. Esposito and M. L. Villani. *An Approach for QoS-aware Service Composition Based on Genetic Algorithms*. In Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, GECCO '05. ACM, 2005.

[Canfora 2008] G. Canfora, M. Di Penta, R. Esposito and M. L. Villani. *A Framework for QoS-aware Binding and Re-binding of Composite Web Services*. The Journal of Systems and Software, vol. 81, pages 1754–1769, 2008. Elsevier.

[Cardoso 2004] J. Cardoso, J. Miller, A. Sheth and J. Arnold. *Quality of Service for Workflows and Web Service Processes*. Journal of Web Semantics, vol. 1, pages 281–308, 2004. Elsevier.

[Channa 2005] N. Channa, S. Li, A.W. Shaikh and X. Fu. *Constraint Satisfaction in Dynamic Web Service Composition*. In Proceedings of the 16th International Workshop on Database and Expert Systems Applications, 2005.

[Chen 2003] H. Chen, T. Finin and A. Joshi. *An Ontology for Context-Aware Pervasive Computing Environments*. Journal of The Knowledge Engineering Review, vol. 18, pages 197–207, 2003. Cambridge University Press.

[Chen 2004] H. Chen, F. Perich, T. Finin and A. Joshi. *SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications*. In Proceedings of the International Conference on Mobile and Ubiquitous Systems: Networking and Services, 2004.

[Chen 2006] I.Y.L. Chen, S.J.H. Yang and J. Zhang. *Ubiquitous Provision of Context Aware Web Services*. In Proceedings of the IEEE International Conference on Services Computing, pages 60 –68. IEEE, 2006.

[Cheung 1996] S.C. Cheung and J. Kramer. *Context Constraints for Compositional Reachability Analysis.* Journal of ACM Transactions on Software Engineering and Methodology, vol. 5, pages 334–377, 1996. ACM.

[Chrysoulas 2007] C. Chrysoulas, G. Koumoutsos, S. Denazis, K. Thramboulidis and O. Koufopavlou. *Dynamic Service Deployment Using an Ontology Based Description of Devices and Services.* In Proceedings of the 3rd International Conference on Networking and Services. IEEE Computer Society, 2007.

[Chung 2009] L. Chung, L. Prado and C. Julio. *On Non-Functional Requirements in Software Engineering.* In Alexander T. Borgida, Vinay K. Chaudhri, Paolo Giorgini and Eric S. Yu, editors, Conceptual Modeling: Foundations and Applications, pages 363–379. Springer, 2009.

[Cohen 1992] W. W. Cohen, A. Borgida and H. Hirsh. *Computing Least Common Subsumers in Description Logics.* In Proceedings of the 10th National Conference on Artificial Intelligence. AAAI Press, 1992.

[Coutaz 2005] J. Coutaz, J.L. Crowley, S. Dobson and D. Garlan. *Context is Key.* Journal of Communications of the ACM, vol. 48, pages 49–53, 2005. ACM.

[Daconta 2003] M.C. Daconta, L.J. Obrst and K.T. Smith. *The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management.* Wiley, 2003.

[Dey 2000] A.K. Dey. *Providing Architectural Support for Building Context-Aware Applications.* PhD thesis, Georgia Institute of Technology, 2000.

[Doulkeridis 2006] C. Doulkeridis, N. Loutas and M. Vazirgiannis. *A System Architecture for Context Aware Service Discovery.* Journal of Electronic Notes in Theoretical Computer Science, pages 101–116, 2006. Elsevier.

[Dui 2003] D. Dui, W. Emmerich, C. Nentwich and B. Thal. *Consistency Checking of Financial Derivatives Transactions.* In Proceedings of the Revised Papers from the International Conference NetObjectDays on Objects, Components, Architectures, Services, and Applications for a Networked World, NODe ’02, pages 166–183. Springer, 2003.

[Dustdar 2008] S. Dustdar and M.P. Papazoglou. *Services and Service Composition - An Introduction.* Journal of Information Technology, vol. 50, pages 86–92, 2008. Palgrave Macmillan.

[Easterbrook 2008] S. Easterbrook, J. Singer, M. A. Storey and D. Damian. *Selecting Empirical Methods for Software Engineering Research.* In Guide to Advanced Empirical Software Engineering, pages 285–311. SpringerLink, 2008.

[Ehlers 2011] J. Ehlers and W. Hasselbring. *A self-adaptive monitoring framework for component-based software systems.* In Proceedings of the 5th European conference on Software architecture. Springer-Verlag, 2011.

[Erradi 2005] A. Erradi and P. Maheshwari. *QoS-Aware Middleware for Reliable Web Services Interactions.* In Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service on e-Technology, e-Commerce and e-Service. IEEE Computer Society, 2005.

[Euzenat 2008] J. Euzenat, J. Pierson and F. Ramparany. *Dynamic Context Management for Pervasive Applications.* Journal of The Knowledge Engineering Review, vol. 23, pages 21–49, 2008. Cambridge University Press.

[Farrar 2003] S. Farrar and T. Langendoen. *A Linguistic Ontology for the Semantic Web*. Journal of GLOT International, vol. 7, pages 97–100, 2003. GOLT.

[Farrar 2010] S. Farrar and D. T. Langendoen. *An OWL-DL Implementation of Gold- An Ontology for the Semantic Web*. Journal of Linguistic Modeling of Information and Markup Languages, vol. 40, pages 45–66, 2010. Springer.

[Farrell 2007] J. Farrell and H. Lausen. *Semantic Annotations for WSDL and XML Schema*, 2007. http://www.w3.org/TR/sawsdl/.

[Fitzgerald 2006] B. Fitzgerald and M. Olsson. *The Software and Services Challenge*. Software, Grids, Security and Dependability of the 7th Framework Programme, 2006.

[Foster 2006] H. Foster. *A Rigourous Approach to Engineering Web Service Compositions*. PhD thesis, Imperial College London, 2006.

[Frølund 1998a] S. Frølund and J. Koistinen. *QML: A Language for Quality of Service Specification*. HP Laboratories Technical Report, 1998. Hewlett-Packard Laboratories. Available from: http://www.hpl.hp.com/techreports/98/HPL-98-10.pdf.

[Frølund 1998b] S. Frølund and J. Koistinen. *Quality of services specification in distributed object systems design*. In Proceedings of the 4th USENIX Conference on Object-Oriented Technologies and Systems, volume 4. USENIX Association, 1998.

[Fujii 2009] K. Fujii and T. Suda. *Semantics-based Context-aware Dynamic Service Composition*. Journal of ACM Transactions on Autonomous and Adaptive Systems, vol. 4, pages 1–31, 2009. ACM.

[Goslar 2004] K. Goslar and A. Schill. *Modeling Contextual Information Using Active Data Structures*. In Proceedings of the EDBT Workshops, volume 3268 of *Lecture Notes in Computer Science*. Springer, 2004.

[Graf 1991] S. Graf and B. Steffen. *Compositional Minimization of Finite State Systems*. In Proceedings of the 2nd International Workshop on Computer Aided Verification. Springer, 1991.

[Grosof 2003] B. N. Grosof, I. Horrocks, R. Volz and S. Decker. *Description Logic Programs: Combining Logic Programs with Description Logic*. In Proceedings of the 12th International Conference on World Wide Web. ACM, 2003.

[Gu 2004] T. Gu, X.H. Wang, H.K. Pung and D.Q. Zhang. *An Ontology-based Context Model in Intelligent Environments*. In Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference, pages 270–275, 2004.

[Guermouche 2008] N. Guermouche, O. Perrin and C. Ringeissen. *Timed Specification For Web Services Compatibility Analysis*. Journal of Electronic Notes in Theoretical Computer Science, vol. 200, pages 155–170, 2008.

[Haarslev 2001] V. Haarslev and R. Möller. *RACER System Description*. In Proceedings of the 1st International Joint Conference on Automated Reasoning, The International Joint Conference on Automated Reasoning (IJCAR '01), pages 701–706. Springer, 2001.

[Harry 2004] C. Harry and Anupam J. *The SOUPA Ontology for Pervasive Computing*. In The Volume on Ontologies for Agent Systems, pages 233–258. Birkhauser Publishing Ltd., April 2004.

[Hasselbring 2006] W. Hasselbring and R. Reussner. *Toward Trustworthy Software Systems*. Journal of IEEE Computer, vol. 39, pages 91–92, 2006. IEEE.

[Henricksen 2002] K. Henricksen, J. Indulska and A. Rakotonirainy. *Modeling Context Information in Pervasive Computing Systems*. In Proceedings of the 1st International Conference on Pervasive Computing. Springer, 2002.

[Heravizadeh 2008] M. Heravizadeh, J. Mendling and M. Rosemann. *Dimensions of Business Process Quality*. In Proceedings of the 6th International Conference on Business Process Management Workshop, pages 80–91. Springer, 2008.

[Hervás 2010] R. Hervás, J. Bravo and J. Fontecha. *A Context Model based on Ontological Languages: a Proposal for Information Visualization*. Journal of Universal Computer Science, vol. 16, pages 1539–1555, 2010. Springer.

[Hong 2008] M. Hong and D. Cho. *Ontology Context Model for Context Aware Learning Service in Ubiquitous Learning Environments*. International Journal of Computers, pages 193–200, 2008. NAUN.

[Hoorn 2009] A.V. Hoorn, M. Rohr, W. Hasselbring, J. Waller, J. Ehlers, S. Frey and D. Kieselhorst. *Continuous Monitoring of Software Services: Design and Application of the Kieker Framework*. http://www.informatik.uni-kiel.de - Technical Report, 2009.

[Horridge 2004] M. Horridge, H. Knublauch, A. Rector, R. Stevens and C. Wroe. *A Practical guide to building OWL ontologies using the Protege-OWL plugin and CO-ODE tools edition 1.0*. World, 2004. The University Of Manchester.

[Horrocks 2003] I. Horrocks and F. Patel-Schneider. *Reducing OWL Entailment to Description Logic Satisfiability*. The Semantic Web - ISWC 2003, Lecture Notes in Computer Science, vol. 2870, pages 17–29, 2003. Springer.

[Horrocks 2004a] I. Horrocks, P. Patel-Schneider, H. Boley, S. Tabet, B. Grosof and M. Dean. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. W3C Member Submission, 2004. W3C.

[Horrocks 2004b] I. Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof and M. Dean. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*, 2004. http://www.w3.org/Submission/SWRL/.

[Ibáñez 2010] M. J. Ibáñez, G. Vulcu, J. Ezpeleta and S. Bhiri. *Semantically Enabled Business Process Discovery*. In Proceedings of the 2010 ACM Symposium on Applied Computing. ACM, 2010.

[Jordan 2007] D. Jordan and J. Evdemon. *Web Services Business Process Execution Language*, 2007. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[Jungo 2007] D. Jungo. *An Open Source CLiXML Schema Validator*, 2007. http://clixml.sourceforge.net/.

[Kaiya 2005] H. Kaiya and M. Motoshi Saeki. *Ontology Based Requirements Analysis: Lightweight Semantic Processing Approach*. In Proceedings of the 5th International Conference on Quality Software. IEEE Computer Society, 2005.

[Kallel 2009] S. Kallel, A. Charfi, T. Dinkelaker, M. Mezini and M. Jmaiel. *Specifying and Monitoring Temporal Properties in Web Services Compositions*. In Proceedings of the Seventh IEEE European Conference on Web Services. IEEE Computer Society, 2009.

[Kapitsaki 2009] G. Kapitsaki, D. Kateros, G. Prezerakos and I. Venierris. *Model-driven Development of Composite Context-aware Web Applications*. Journal of Information and Software Technology, vol. 51, pages 1244–1260, 2009. Butterworth-Heinemann.

[Karakoc 2009] E. Karakoc and P. Senkul. *Composing Semantic Web services under Constraints*. Journal of Expert Systems with Applications, vol. 36, pages 11021–11029, 2009. Pergamon Press, Inc.

[Karastoyanova 2005] D. Karastoyanova, A. Houspanossian, M. Cilia, F. Leymann and A. Buchmann. *Extending BPEL for Run Time Adaptability*. Proceedings of the IEEE International Enterprise Distributed Object Computing Conference, pages 15–26, 2005. IEEE Computer Society.

[Karvounarakis 2002] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis and M. Scholl. *RQL: A Declarative Query Language for RDF*. In Proceedings of the 11th International Conference on World Wide Web, pages 592–603. ACM Press, 2002.

[Kazhamiakin 2006] Raman Kazhamiakin, P. Pandya and M. Pistore. *Representation, Verification, and Computation of Timed Properties in Web*. In Proceedings of the IEEE International Conference on Web Services. IEEE Computer Society, 2006.

[Keller 2006] U. Keller and H. Lausen. *Functional Description of Web Services*, 2006. http://www.wsmo.org/TR/d28/d28.1/v0.1/.

[Khedr 2004] M. Khedr and A. Karmouch. *Negotiating Context Information in Context-aware Systems*. Journal of IEEE Intelligent Systems, vol. 19, pages 21–29, 2004. IEEE.

[Kim 2004] J.W. Kim and H.D. Kim. *Semantic Constraint Specification and Verification of ebXML Business Process Specifications*. Journal of Expert Systems with Applications, vol. 27, pages 571 – 584, 2004. Elsevier.

[Kitchenham 2002] B. Kitchenham and S. L. Pfleeger. *Principles of Survey Research Part 4: Questionnaire Evaluation*. SIGSOFT Software Engineering Notes, vol. 27, pages 20–23, 2002. ACM.

[Kona 2009] S. Kona, A. Bansal, L. Simon, A. Mallya, G. Gupta and T.D. Hite. *USDL: A Service-Semantics Description Language for Automatic Service Discovery and Composition*. International Journal of Web Services Research, pages 20–48, 2009.

[Kopecky 2008] J. Kopecky and T. Vitvar. *WSMO-Lite: Lowering the Semantic Web Services Barrier with Modular and Light-Weight Annotations*. In Proceedings of the IEEE International Conference on Semantic Computing, 2008.

[Kozen 1990] D. Kozen and J. Tiuryn. *Logics of Programs*. In Handbook of Theoretical Computer Science, volume B: Formal Models and Sematics, pages 789–840. 1990.

[Küsters 2000] R. Küsters. Non-standard inferences in description logics. Springer-Verlag, 2000.

[Lausen 2005] H. Lausen, A. Polleres and D. Roman. *Web Service Modeling Ontology (WSMO)*, 2005. http://www.w3.org/Submission/WSMO/.

[Lee 2007] K. C. Lee, J. Kim, J. Lee and K. M. Lee. *Implementation of Ontology based Context Aware Framework for Ubiquitous Environments*. In proceedings of the International Conference on Multimedia and Ubiquitous Engineering, pages 278–282, 2007.

[Leitner 2010] P. Leitner, A. Michlmayr, F. Rosenberg and S. Dustdar. *Monitoring, Prediction and Prevention of SLA Violations in Composite Services*. In Proceedings of the IEEE International Conference on Web Services, 2010.

[Li 2009] W. Li and L. Ping. *Trust Model to Enhance Security and Interoperability of Cloud Environment*. In Proceedings of the 1st International Conference on Cloud Computing, CloudCom '09. Springer-Verlag, 2009.

[Lunteren 2004] J. V. Lunteren, T. Engbersen, J. Bostian, B. Carey and C. Larsson. *XML Accelerator Engine*. In Processing of the 1st International Workshop on High Performance XML. ACM, 2004.

[Ly 2008] L.T. Ly, K. Goser, S. Rinderle-Ma and P. Dadam. *Compliance of Semantic Constraints - A Requirements Analysis for Process Management Systems*. In Proceedings of the 1st International Workshop on Governance, Risk and Compliance - Applications in Information Systems, 2008.

[Maamar 2006] Z. Maamar, D. Benslimane and N.C. Narendra. *What Can Context do for Web Services?* Journal of Communications of the ACM, vol. 49, pages 98–103, 2006. ACM.

[Mahbub 2005] K. Mahbub and G. Spanoudakis. *Run-time Monitoring of Requirements for Systems Composed of Web-Services: Initial Implementation and Evaluation Experience*. In Proceedings of the IEEE International Conference on Web Services. IEEE Computer Society, 2005.

[Martin 2004] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan and K. Sycara. *OWL-S:Semantic Markup for Web Services*, 2004. http://www.w3.org/Submission/OWL-S/.

[Martin 2006] D. Martin. *Putting Web Services in Context*. Journal of Electronic Notes in Theoretical Computer Science, vol. 146, pages 3–16, 2006. Elsevier.

[McGuinness 2004] D.L. McGuinness and F. Harmelen. *OWL Web Ontology Language Overview*, 2004. http://www.w3.org/TR/owl-features/.

[Medjahed 2005] B. Medjahed and A. Bouguettaya. *A Dynamic Foundation Architecture for Semantic Web Services*. Journal of Distributed and Parallel Databases, vol. 17, pages 179–206, 2005. Kluwer Academic Publishers.

[Medjahed 2007] B. Medjahed and Y. Atif. *Context-based Matching for Web Service Composition*. Journal of Distributed and Parallel Databases, vol. 21, pages 5–37, 2007. Kluwer Academic Publishers.

[Milanovic 2004] N. Milanovic and M. Malek. *Current Solutions for Web Service Composition*. Jounal of IEEE Internet Computing, vol. 8, pages 51–59, 2004. IEEE.

[Moser 1986] S.C. Moser and G. Kalton. *Survey Methods in Social Investigation*. Gower Publishing Company, 2 édition, 1986.

[Moser 2008a] O. Moser, F. Rosenberg and S. Dustdar. *Non-intrusive Monitoring and Service Adaptation for WS-BPEL*. In Proceeding of the 17th International Conference on World Wide Web, WWW '08. ACM, 2008.

[Moser 2008b] O. Moser, F. Rosenberg and S. Dustdar. *VieDAME - Flexible and Robust BPEL Processes through Monitoring and Adaptation*. In Proceedings of the Companion of the 30th International Conference on Software Engineering, ICSE Companion '08. ACM, 2008.

[Mosincat 2009] A. Mosincat and W. Binder. *Enhancing BPEL Processes with Self-tuning Behavior*. In Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications (SOCA-2009), 2009.

[Mostéfaoui 2003] S.K. Mostéfaoui and B. Hirsbrunner. *Towards a Context-Based Service Composition Framework*. In Proceedings of the 1st International Conference on Web Services, pages 42–45, 2003.

[Motik 2005] B. Motik, U. Sattler and R. Studer. *Query Answering for OWL-DL with Rules*. Journal of Web Semantics: Science, Services and Agents on the World Wide Web, vol. 3, pages 41–60, 2005. Elsevier.

[Mrissa 2006a] M. Mrissa, C. Ghedira, D. Benslimane and Z. Maamar. *Context and Semantic Composition of Web Services*. Lecture Notes in Computer Science, pages 266–275. Springer, 2006.

[Mrissa 2006b] M. Mrissa, C. Ghedira, D. Benslimane and Z. Maamar. *A Context Model for Semantic Mediation in Web Services Composition*. In Proceedings of the 25th International Conference on Conceptual Modeling, pages 12–25. Springer, 2006.

[Mrissa 2007] M. Mrissa, C. Ghedira, D. Benslimane, Z. Maamar, F. Rosenberg and S. Dustdar. *A Context-based Mediation Approach to Compose Semantic Web Services*. Journal of ACM Transactions on Internet Technology (TOIT), vol. 8, 2007. ACM.

[Mrissa 2008] M. Mrissa, P. Thiran, C. Ghedira, D. Benslimane and Z. Maamar. *Using Context to Enable Semantic Mediation in Web Service Communities*. In Proceedings of the 2008 International Workshop on Context Enabled Source and Service Selection, Integration and Adaptation. ACM, 2008.

[Nardi 2003] P.M. Nardi. *Doing Survey Research - A Guide to Quantitative Methods*. Pearson Education, Inc, 2003.

[Nentwich 2002] C. Nentwich, L. Capra, W. Emmerich and A. Finkelstein. *xlinkit: A Consistency Checking and Smart Link Generation Service*. Journal of ACM Transactions on Internet Technology, vol. 2, pages 151–185, 2002. ACM.

[Nentwich 2005] C. Nentwich. *CLiX - A Validation Rule Language for XML*. In Proceedings of the W3C Workshop on Rule Languages for Interoperability. W3C, 2005.

[Ngo 2004] H.Q. Ngo, A. Shehzad, K.A. Pham, M. Riaz, S. Liaquat and S.Y. Lee. *Developing Context-Aware Ubiquitous Computing Systems with a Unified Middleware Framework*. In Proceedings of Embedded and Ubiquitous Computing (EUC 2004). Springer, 2004.

[OCL 2010] OCL. *Object Constraint Language (OCL)*, 2010. http://www.omg.org/spec/OCL/.

[O'Connor 2005a] M. O'Connor, H. Knublauch, S. Tu, B. Grosof, M. Dean, W. Grosso and M. Musen. *Supporting Rule System Interoperability on the Semantic Web with SWRL*. In Proceedings of the Semantic Web Ű The International Semantic Web Conference 2005, volume 3729 of *Lecture Notes in Computer Science*, pages 974–986. Springer, 2005.

[O'Connor 2005b] M. O'Connor, H. Knublauch, S. Tu and M. Musen. *Writing Rules for the Semantic Web Using SWRL and Jess*. In Proceedings of the 8th International Protege Conference, Protege with Rules Workshop (2005), 2005.

[ODM 2009] ODM. *Ontology Definition Metamodel (ODM)*, 2009. http://www.omg.org/spec/ODM/1.0/.

[O'Sullivan 2002] J. O'Sullivan, D. Edmond and A. Hofstede. *What's in a Service?* Journal of Distributed and Parallel Databases, vol. 12, pages 117–133, 2002. Springer.

[O'Sullivan 2006] J. O'Sullivan. *Towards a precise understanding of service properties*. PhD thesis, Queensland University of Technology, 2006.

[Pahl 2007] C. Pahl, S. Giesecke and W. Hasselbring. *An Ontology-Based Approach for Modelling Architectural Styles*. In F. Oquendo, editor, Software Architecture, volume 4758 of *Lecture Notes in Computer Science*, pages 60–75. Springer, 2007.

[Pahl 2008] C. Pahl, Bandara K.Y. and Wang M.X. *Case Study*. CASCAR Project Deliverable, vol. D1, 2008. http://www.computing.dcu.ie/ cpahl/project-CASCAR.htm.

[Pahl 2009] C. Pahl, S. Giesecke and W. Hasselbring. *Ontology-based Modelling of Architectural Styles*. Journal of Information and Software Technology, vol. 51, pages 1739–1749, 2009. Elsevier.

[Pahl 2010] C. Pahl, K. Y. Bandara and M. X. Wang. *Context Constraint Integration and Validation*. In Q. Z. Sheng, J. Yu and S. Dustdar, editors, Enabling Context-Aware Web Services: Methods, Architectures, and Technologies, pages 81–105. Chapman & Hall/CRC, 1st édition, 2010.

[Pahl 2011] C. Pahl, V. Gacitua-Decar, M.X. Wang and K.Y. Bandara. *Ontology-Based Composition and Matching for Dynamic Service Coordination*. In Proceedings of the CAiSE 2011 International Workshops. Springer, 2011.

[Park 2005] S. Park, L. Liu, C. Pu, M. Srivatsa and J. Zhang. *Resilient Trust Management for Web Service Integration*. In Proceedings of the IEEE International Conference on Web Services, ICWS '05. IEEE Computer Society, 2005.

[Pessoa 2007] R.M. Pessoa, C.Z. Calvi, J.G.P. Filho, C.R.G. de Farias and R. Neisse. *Semantic Context Reasoning using Ontology based Models*. In Proceedings of the 13th Open European Summer School and IFIP TC6.6 Conference on Dependable and Adaptable Networks and Services, pages 44–51. Springer, 2007.

[Priestley 2003] M. Priestley. *Practical Object-oriented Design with UML*. McGraw Hill Higher Education, 2003.

[Raimondi 2008] F. Raimondi, J. Skene and W. Emmerich. *Efficient Online Monitoring of Web Service SLAs*. In Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering. ACM, 2008.

[Rajasekaran 2004] P. Rajasekaran, J. Miller, K. Verma and A. Sheth. *Enhancing Web Services Description and Discovery to Facilitate Composition*. In Proceedings of the 1st International Workshop on Semantic Web Services and Web Process Composition, pages 34–47, 2004.

[Ranganathan 2002] A. Ranganathan, R.H. Campbell, A. Ravi and A. Mahajan. *ConChat: A Context-Aware Chat Program*. Journal of IEEE Pervasive Computing, vol. 1, pages 51–57, 2002. IEEE Educational Activities Department.

[Rao 2004] J. Rao and X. Su. *A Survey of Automated Web Service Composition Methods*. In Proceedings of the 1st International Workshop on Semantic Web Services and Web Process Composition, pages 43–54, 2004.

[Rosemann 2006] M. Rosemann and J. C. Recker. *Context-aware Process Design: Exploring the Extrinsic Drivers for Process Flexibility*. In Proceedings of the 7th Workshop on Business Process Modeling, Development, and Support (BPMDS'06), 2006.

[Rosemann 2008] M. Rosemann, J. C. Recker and C. Flender. *Contextualisation of Business Processes*. Journal of Business Process Integration and Management, vol. 3, pages 47–60, 2008. Inderscience Enterprises Ltd.

[Rosenberg 2010] F. Rosenberg, M.B. Müller, P. Leitner, A. Michlmayr, A. Bouguettaya and S. Dustdar. *Metaheuristic Optimization of Large-Scale QoS-aware Service Compositions.* In Proceedings of the IEEE International Conference on Services Computing (SCC), 2010.

[Rowley 2002] J. Rowley. *Using Case Studies in Research.* Journal of Management Research News, vol. 25, pages 16–27, 2002. Emerald.

[Roy 2010] N. Roy, T. Gu and S.K. Das. *Supporting Pervasive Computing Applications with Active Context Fusion and Semantic Context Delivery.* Journal of Pervasive and Mobile Computing, vol. 6, pages 21–42, 2010. Elsevier.

[Sattanathan 2006] S. Sattanathan, N. C. Narendra and Z. Maamar. *Ontologies for Specifying and Reconciling Contexts of Web Services.* Journal of Electronic Notes in Theoretical Computer Science, 2006. Elsevier.

[Seaborne 2004] A. Seaborne. *RDQL - A Query Language for RDF*, 2004. http://www.w3.org/Submission/RDQL/.

[Serrano 2007] M. Serrano, J. Serrat and J. Strassner. *Ontology Based Reasoning for Supporting Context Aware Services on Autonomic Networks.* In Proceedings of the IEEE International Conference on Communication, 2007.

[Shaffer 1998] C.A. Shaffer. *A Practical Introduction to Data Structures and Algorithm Analysis.* Prentice Hall, java édition, 1998.

[Shanahan 1999] M. Shanahan. *The Event Calculus Explained.* In Artificial Intelligence Today, pages 409–430. 1999.

[Sheshagiri 2004] M. Sheshagiri, N. Sadeh and F. Gandon. *Using Semantic Web Services for Context Aware Mobile Applications.* Proceedings of the MobiSys 2004 Workshop on Context-Awareness, 2004.

[Sintek 2002] M. Sintek and S. Decker. *TRIPLE - A Query, Inference, and Transformation Language for the Semantic Web.* In Proceedings of the 1st International Semantic Web Conference on The Semantic Web, ISWC 2002, pages 364–378. Springer, 2002.

[Sirin 2007] E. Sirin, B. Parsia, B.C. Grau, A. Kalyanpur and Y. Katz. *Pellet: A practical OWL-DL reasoner.* Journal of Web Semantics: Science, Services and Agents on the World Wide Web, vol. 5, pages 51 – 53, 2007. Elsevier.

[Slonneger 1994] K. Slonneger and B.L. Kurtz. *Formal Syntax and Semantics of Programming Languages: A Laboratory Based Approach.* Addison Wesley Longman, Reading, Massachusetts, 1st edition édition, 1994.

[SoaML 2009] SoaML. *Service oriented architecture Modeling Language (SoaML)*, 2009. http://www.omg.org/spec/SoaML/.

[Soylu 2009a] A. Soylu, P.D. Causmaecker and P. Desmet. *Context and Adaptivity in Context-Aware Pervasive Computing Environments.* In Proceedings of the 2009 Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing. IEEE Computer Society, 2009.

[Soylu 2009b] A. Soylu, P.D. Causmaecker, P. Desmet and K.U. Leuven. *Context and Adaptivity in Pervasive Computing Environments: Links with Software Engineering and Ontological Engineering.* Journal of Software, vol. 4, 2009. ACADEMY Publisher.

[Spanoudakis 2007] G. Spanoudakis. *Dynamic Trust Assessment of Software Services.* In Proceedings of the 2nd International Workshop on Service Oriented Software Engineering: in Conjunction with the 6th ESEC/FSE Joint Meeting, IW-SOSWE '07, pages 36–40. ACM, 2007.

[Spanoudakis 2009] G. Spanoudakis and S. LoPresti. *Web Service Trust: Towards a Dynamic Assessment Framework.* In Proceedings of the International Conference on Availability, Reliability and Security. IEEE Computer Society, 2009.

[Spertus 2010] M.P. Spertus, C.D. Metcalf and G. Wolfman. *Pre-Computed Dynamic Instrumentation, US Patent 7,805,717 B1*, 2010.

[Srivastava 2003] B. Srivastava and J. Koehler. *Web Service Composition - Current Solutions and Open Problems.* In Proceedings of the ICAPS Workshop on Planning for Web Services, 2003.

[Strang 2004] T. Strang and C. Linnhoff-Popien. *A Context Modeling Survey.* In Proceedings of the International Workshop on Advanced Context Modelling, Reasoning and Management - The Sixth International Conference on Ubiquitous Computing, 2004.

[Strembeck 2004] M. Strembeck and G. Neumann. *An Integrated Approach to Engineer and Enforce Context Constraints in RBAC Environments.* Journal of ACM Transactions on Information and System Security, vol. 7, pages 392–427, 2004. ACM.

[Tsarkov 2007] D. Tsarkov and I. Horrocks. *FaCT++ Website*, 2007. http://owl.man.ac.uk/factplusplus/.

[Urbieta 2008] A. Urbieta, E. Azketa, I. Gomez, J. Parra and N. Arana. *Analysis of Effects and Preconditions-based Service Representation in Ubiquitous Computing Environments.* Proceedings of the International Conference on Semantic Computing, pages 378–385, 2008. IEEE Computer Society.

[USDL 2011] USDL. *Unified Service Description Language (USDL)*, 2011. http://www.w3.org/2005/Incubator/usdl/.

[Vergara 2009] J.E. Vergara, A. Guerrero, V.A. Villagrá and J. Berrocal. *Ontology-based Network Management: Study Cases and Lessons Learned.* Journal of Network and Systems Management, vol. 17, September 2009. Plenum Press.

[Verma 2004] D.C. Verma. *Service Level Agreements on IP Networks.* Proceedings of the IEEE, vol. 92, pages 1382 – 1388, 2004. IEEE.

[Vladimir 2011] S. Vladimir and M. Miroslaw. *Addressing Dependability throughout the SOA Life Cycle.* Journal IEEE Transactions on Services Computing, vol. 4, pages 85–95, 2011.

[Wang 2004] X.H. Wang, D. Q. Zhang, T. Gu and H.K. Pung. *Ontology Based Context Modeling and Reasoning using OWL.* In Proceedings of the 2nd Annual Conference on Pervasive Computing and Communications Workshops. IEEE, 2004.

[Wang 2009a] M.X. Wang, K.Y. Bandara and C. Pahl. *Constraint Integration and Violation Handling for BPEL Processes.* In Proceedings of the 2009 Fourth International Conference on Internet and Web Applications and Services. IEEE Computer Society, 2009.

[Wang 2009b] M.X. Wang, K.Y. Bandara and C. Pahl. *Integrated Constraint Violation Handling for Dynamic Service Composition.* In Proceedings of the 2009 IEEE International Conference on Services Computing, SCC '09. IEEE Computer Society, 2009.

[Wang 2009c] Q. Wang, J. Shao, F. Deng, Y. Liu, M. Li, J. Han and H. Mei. *An Online Monitoring Approach for Web Service Requirements*. Journal of IEEE Transactions on Services Computing, vol. 2, pages 338–351, 2009. IEEE.

[Wang 2010] M. Wang, K.Y. Bandara and C. Pahl. *Process as a Service Distributed Multi-tenant Policy-Based Process Runtime Governance*. In Proceedings of the 2010 IEEE International Conference on Services Computing. IEEE, 2010.

[Yahyaoui 2010] H. Yahyaoui. *Trust Assessment for Web Services Collaboration*. In Proceedings of the 2010 IEEE International Conference on Web Services, pages 315–320. IEEE Computer Society, 2010.

[Ye 2007] J. Ye, L. Coyle, S. Dobson and P. Nixon. *Ontology-based Models in Pervasive Computing Systems*. The Knowledge Engineering Review, vol. 22, pages 315–347, 2007. Cambridge University Press.

[Yin 2009] R.K. Yin. *Case Study Research Design and Methods*. volume 5. SAGE Inc., 4th edition édition, 2009.

[Yu 2007] T. Yu, Y. Zhang and K.J. Lin. *Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints*. Journal of ACM Transactions on the Web (TWEB), vol. 1, 2007. ACM.

[Zeng 2004] L. Zeng, B. Benatallah, A.H.H. Ngu, M. Dumas, J. Kalagnanam and H. Chang. *QoS-aware Middleware for Web Services Composition*. Journal of IEEE Transactions on Software Engineering, vol. 30, pages 311–327, 2004. IEEE.

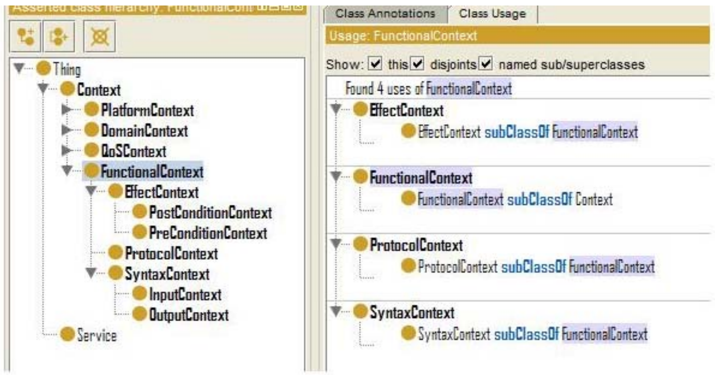
[Zhao 2007] H. Zhao and H. Tong. *A Dynamic Service Composition Model Based on Constraints*. Proceedings of the International Conference on Grid and Cloud Computing, pages 659–662, 2007. IEEE Computer Society.

Appendix A

OWL-based implementation of the context model ontology

We implemented the main constituents of the OWL-based context model ontology using the Protege 4.0.2 editor. Firstly, we present Protege screen shots of the main constituents of our context model ontology implementation from section A.1 to section A.5. Secondly, we illustrate the OWL implementation of main constituents of our context model ontology in section A.6. Thirdly, in section A.7, we describe the analysis of scenarios used from the classical business domain for the development of context model ontology.

A.1 Functional context



Found 4 uses of FunctionalContext

EffectContext
- EffectContext subClassOf FunctionalContext

FunctionalContext
- FunctionalContext subClassOf Context

ProtocolContext
- ProtocolContext subClassOf FunctionalContext

SyntaxContext
- SyntaxContext subClassOf FunctionalContext

Figure A.1: Functional context

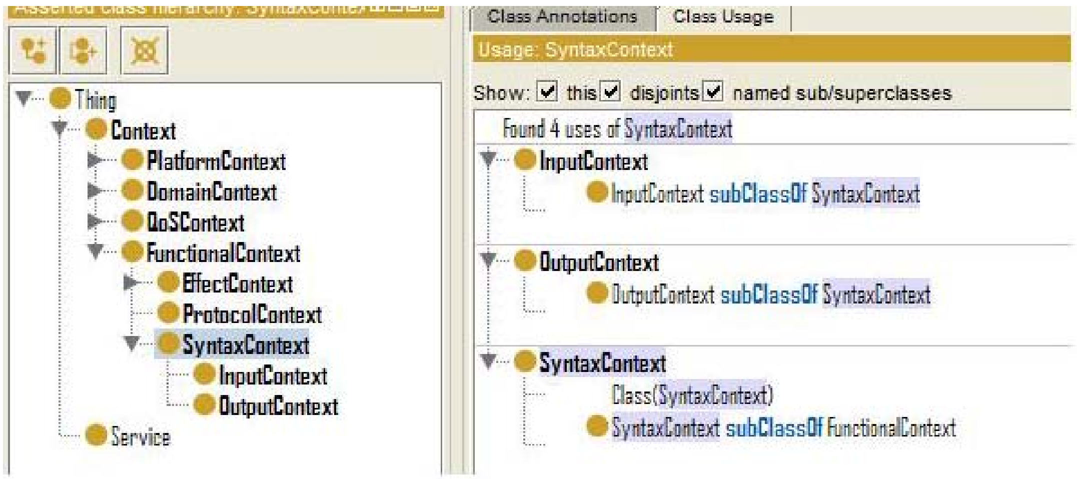
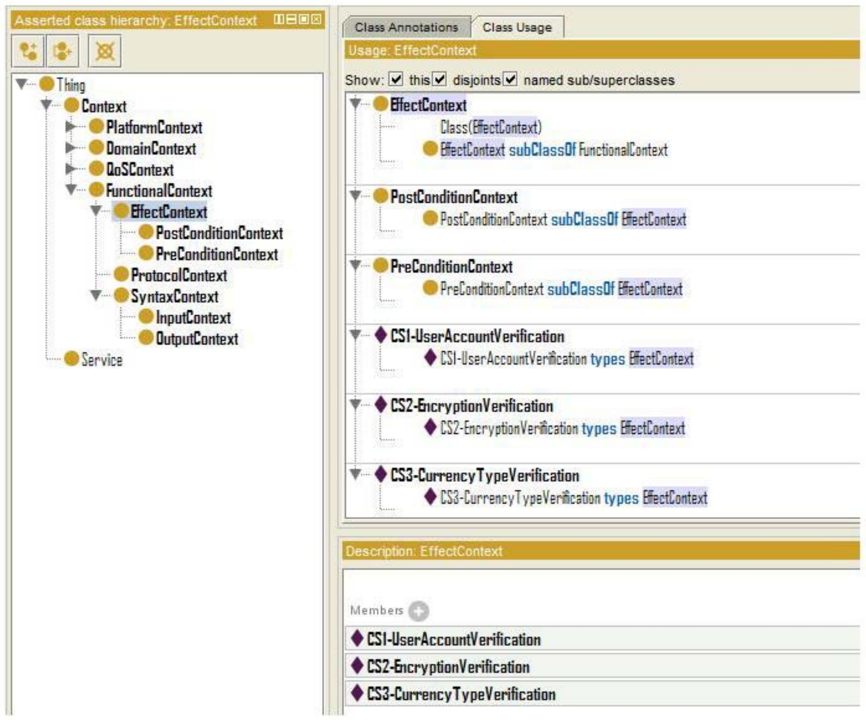
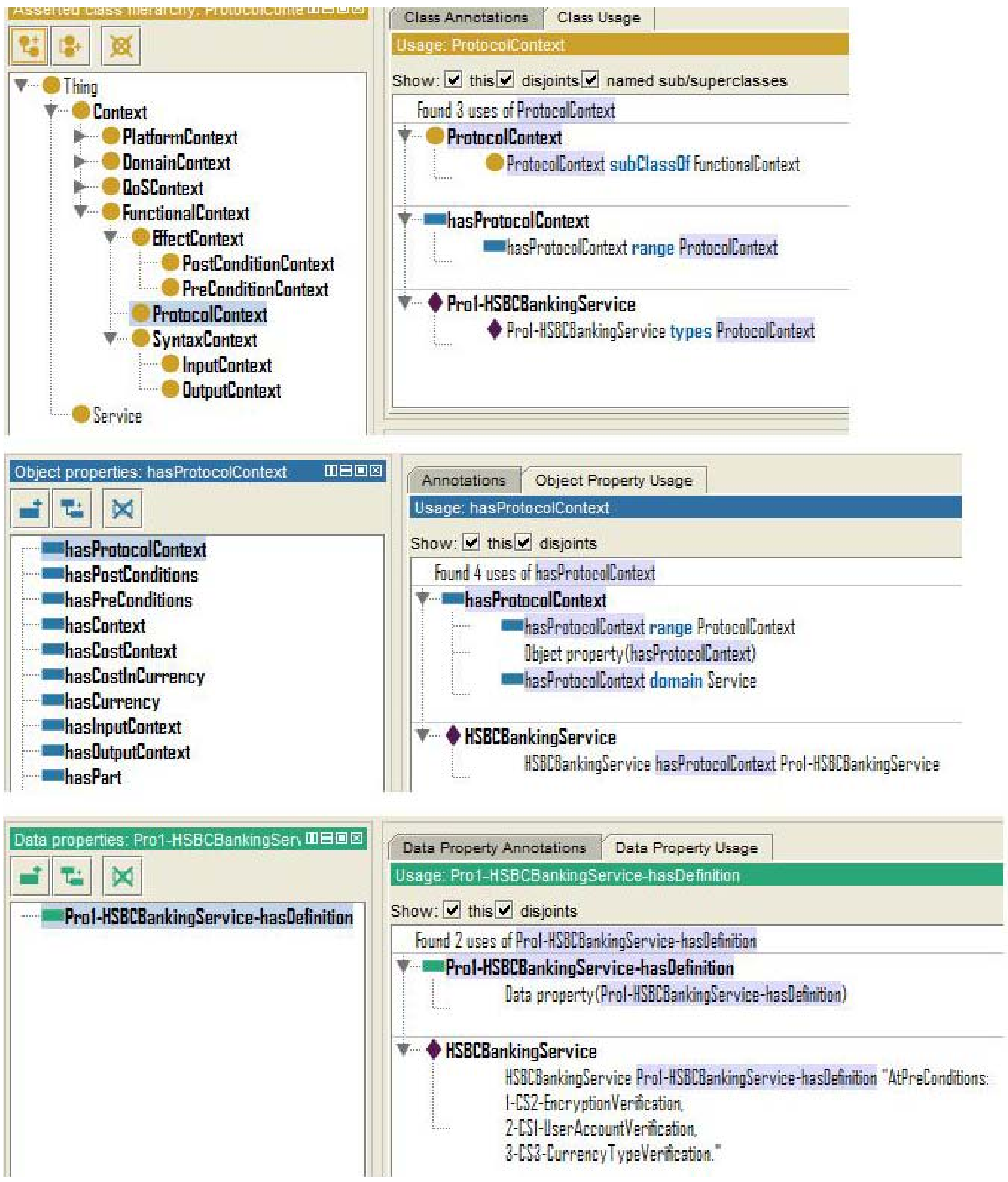

```
Asserted class hierarchy: SyntaxContext
Thing
  Context
    PlatformContext
    DomainContext
    QoSContext
    FunctionalContext
      EffectContext
      ProtocolContext
      SyntaxContext
        InputContext
        OutputContext
  Service
```
```
Class Annotations | Class Usage
Usage: SyntaxContext
Show: ☑ this ☑ disjoints ☑ named sub/superclasses
Found 4 uses of SyntaxContext
InputContext
    InputContext subClassOf SyntaxContext
OutputContext
    OutputContext subClassOf SyntaxContext
SyntaxContext
    Class(SyntaxContext)
    SyntaxContext subClassOf FunctionalContext
```

Figure A.2: Syntax context


```
Asserted class hierarchy: EffectContext
Thing
  Context
    PlatformContext
    DomainContext
    QoSContext
    FunctionalContext
      EffectContext
        PostConditionContext
        PreConditionContext
      ProtocolContext
      SyntaxContext
        InputContext
        OutputContext
  Service
```
```
Class Annotations | Class Usage
Usage: EffectContext
Show: ☑ this ☑ disjoints ☑ named sub/superclasses
EffectContext
    Class(EffectContext)
    EffectContext subClassOf FunctionalContext
PostConditionContext
    PostConditionContext subClassOf EffectContext
PreConditionContext
    PreConditionContext subClassOf EffectContext
CS1-UserAccountVerification
    CS1-UserAccountVerification types EffectContext
CS2-EncryptionVerification
    CS2-EncryptionVerification types EffectContext
CS3-CurrencyTypeVerification
    CS3-CurrencyTypeVerification types EffectContext

Description: EffectContext
Members
CS1-UserAccountVerification
CS2-EncryptionVerification
CS3-CurrencyTypeVerification
```

Figure A.3: Effect context


Asserted class hierarchy: ProtocolConte
Class Annotations
Class Usage
Usage: ProtocolContext
Show: this disjoints named sub/superclasses
Object properties: hasProtocolContext
Annotations
Object Property Usage
Usage: hasProtocolContext
Show: this disjoints
Data properties: Pro1-HSBCBankingServ
Data Property Annotations
Data Property Usage
Usage: Pro1-HSBCBankingService-hasDefinition
Show: this disjoints

```
Thing
  Context
    PlatformContext
    DomainContext
    QoSContext
    FunctionalContext
      EffectContext
        PostConditionContext
        PreConditionContext
      ProtocolContext
      SyntaxContext
        InputContext
        OutputContext
  Service
```

Found 3 uses of ProtocolContext

```
ProtocolContext
    ProtocolContext subClassOf FunctionalContext
hasProtocolContext
    hasProtocolContext range ProtocolContext
Pro1-HSBCBankingService
    Pro1-HSBCBankingService types ProtocolContext
```

```
hasProtocolContext
hasPostConditions
hasPreConditions
hasContext
hasCostContext
hasCostInCurrency
hasCurrency
hasInputContext
hasOutputContext
hasPart
```

Found 4 uses of hasProtocolContext

```
hasProtocolContext
    hasProtocolContext range ProtocolContext
    Object property(hasProtocolContext)
    hasProtocolContext domain Service
HSBCBankingService
    HSBCBankingService hasProtocolContext Pro1-HSBCBankingService
```

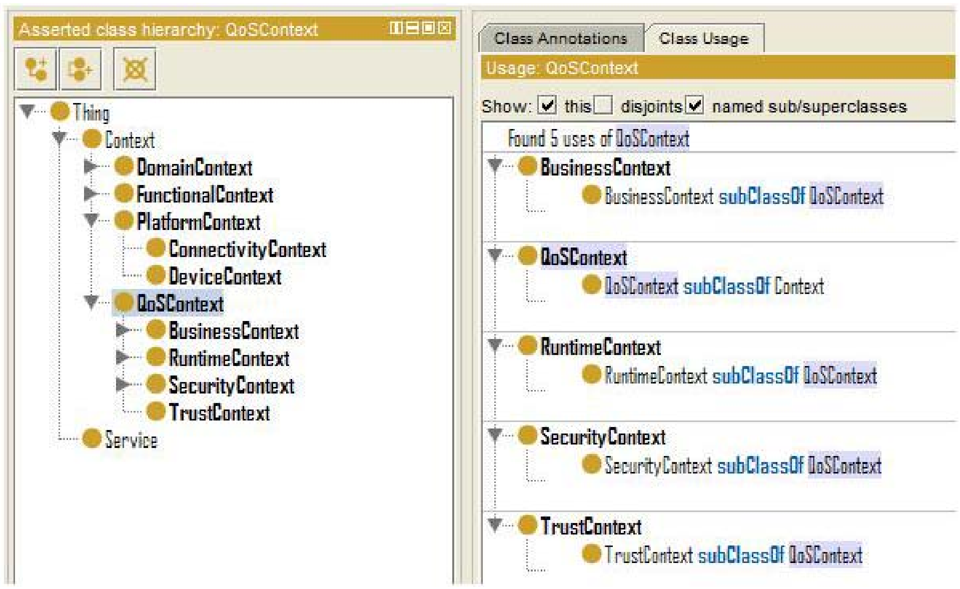
```
Pro1-HSBCBankingService-hasDefinition
```

Found 2 uses of Pro1-HSBCBankingService-hasDefinition

```
Pro1-HSBCBankingService-hasDefinition
    Data property(Pro1-HSBCBankingService-hasDefinition)
HSBCBankingService
    HSBCBankingService Pro1-HSBCBankingService-hasDefinition "AtPreConditions:
    1-CS2-EncryptionVerification,
    2-CS1-UserAccountVerification,
    3-CS3-CurrencyTypeVerification."
```

Figure A.4: Protocol context

A.2 Quality of service context

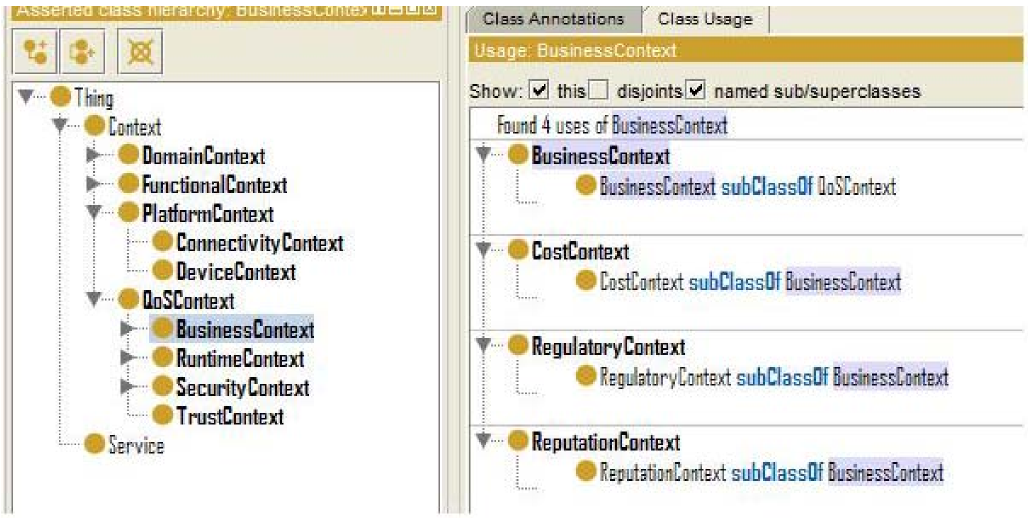


```
Asserted class hierarchy: QoSContext

Thing
  Context
    DomainContext
    FunctionalContext
    PlatformContext
      ConnectivityContext
      DeviceContext
    QoSContext
      BusinessContext
      RuntimeContext
      SecurityContext
      TrustContext
  Service
```

```
Class Annotations | Class Usage
Usage: QoSContext
Show: ☑ this ☐ disjoints ☑ named sub/superclasses
Found 5 uses of QoSContext
BusinessContext
    BusinessContext subClassOf QoSContext
QoSContext
    QoSContext subClassOf Context
RuntimeContext
    RuntimeContext subClassOf QoSContext
SecurityContext
    SecurityContext subClassOf QoSContext
TrustContext
    TrustContext subClassOf QoSContext
```

Figure A.5: Quality of service context

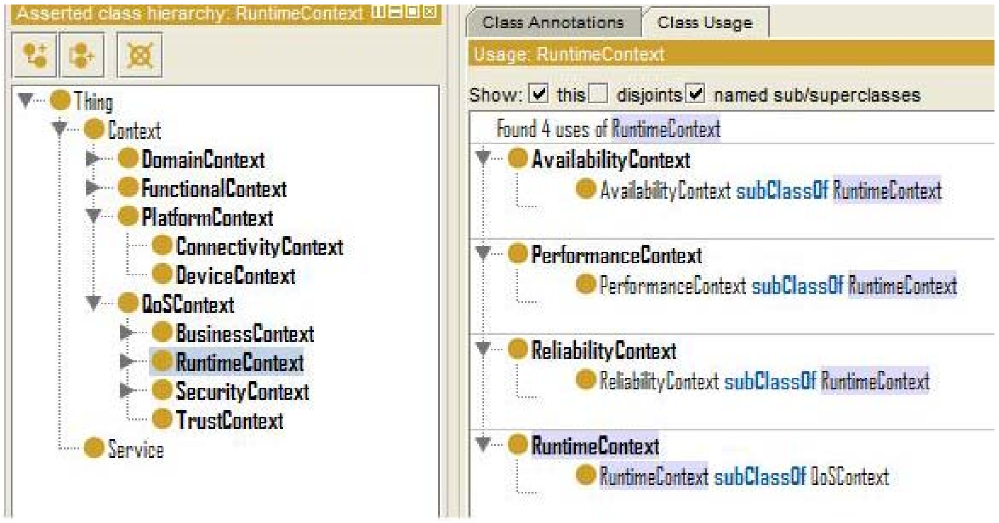


```
Asserted class hierarchy: BusinessContext

Thing
  Context
    DomainContext
    FunctionalContext
    PlatformContext
      ConnectivityContext
      DeviceContext
    QoSContext
      BusinessContext
      RuntimeContext
      SecurityContext
      TrustContext
  Service
```

```
Class Annotations | Class Usage
Usage: BusinessContext
Show: ☑ this ☐ disjoints ☑ named sub/superclasses
Found 4 uses of BusinessContext
BusinessContext
    BusinessContext subClassOf QoSContext
CostContext
    CostContext subClassOf BusinessContext
RegulatoryContext
    RegulatoryContext subClassOf BusinessContext
ReputationContext
    ReputationContext subClassOf BusinessContext
```

Figure A.6: Business context


Asserted class hierarchy: RuntimeContext
Class Annotations
Class Usage
Usage: RuntimeContext
Show: this disjoints named sub/superclasses

Thing
- Context
  - DomainContext
  - FunctionalContext
  - PlatformContext
    - ConnectivityContext
    - DeviceContext
  - QoSContext
    - BusinessContext
    - RuntimeContext
    - SecurityContext
    - TrustContext
- Service

Found 4 uses of RuntimeContext

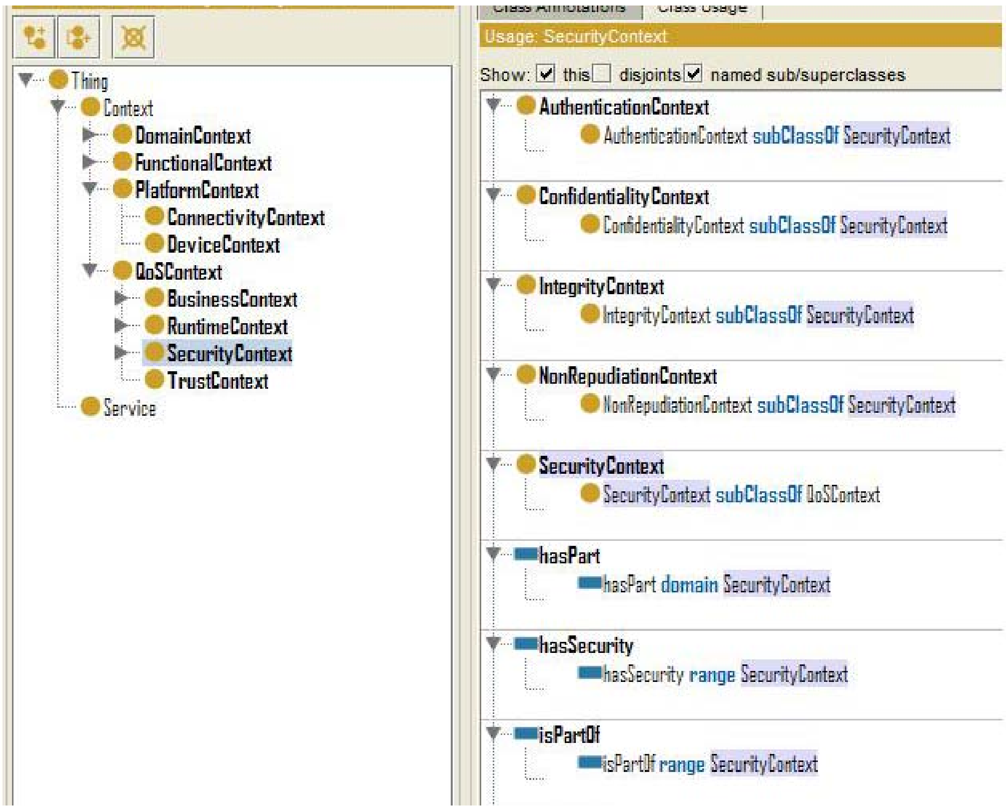
AvailabilityContext
- AvailabilityContext subClassOf RuntimeContext

PerformanceContext
- PerformanceContext subClassOf RuntimeContext

ReliabilityContext
- ReliabilityContext subClassOf RuntimeContext

RuntimeContext
- RuntimeContext subClassOf QoSContext

Figure A.7: Runtime context



Thing
- Context
  - DomainContext
  - FunctionalContext
  - PlatformContext
    - ConnectivityContext
    - DeviceContext
  - QoSContext
    - BusinessContext
    - RuntimeContext
    - SecurityContext
    - TrustContext
- Service

AuthenticationContext
- AuthenticationContext subClassOf SecurityContext

ConfidentialityContext
- ConfidentialityContext subClassOf SecurityContext

IntegrityContext
- IntegrityContext subClassOf SecurityContext

NonRepudiationContext
- NonRepudiationContext subClassOf SecurityContext

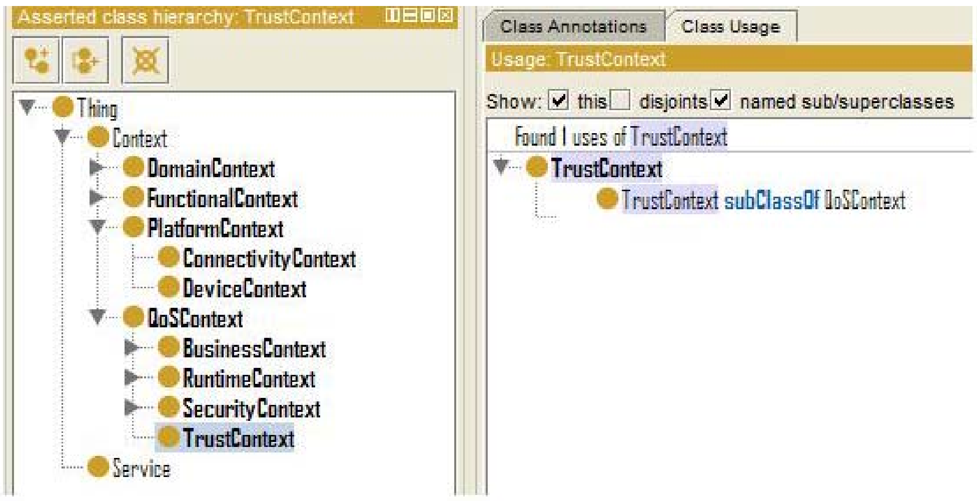
SecurityContext
- SecurityContext subClassOf QoSContext

hasPart
- hasPart domain SecurityContext

hasSecurity
- hasSecurity range SecurityContext

isPartOf
- isPartOf range SecurityContext

Figure A.8: Security context



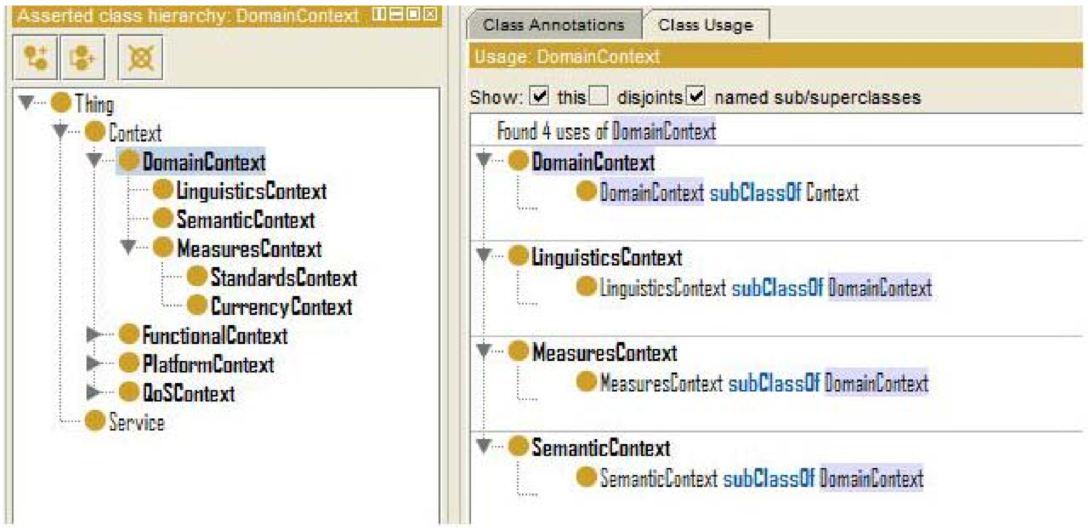
```
Thing
  Context
    DomainContext
    FunctionalContext
    PlatformContext
      ConnectivityContext
      DeviceContext
    QoSContext
      BusinessContext
      RuntimeContext
      SecurityContext
      TrustContext
  Service
```

Show: [✓ this] disjoints [✓ named sub/superclasses]

```
Found 1 uses of TrustContext
TrustContext
    TrustContext subClassOf QoSContext
```

Figure A.9: Trust context

A.3 Domain context



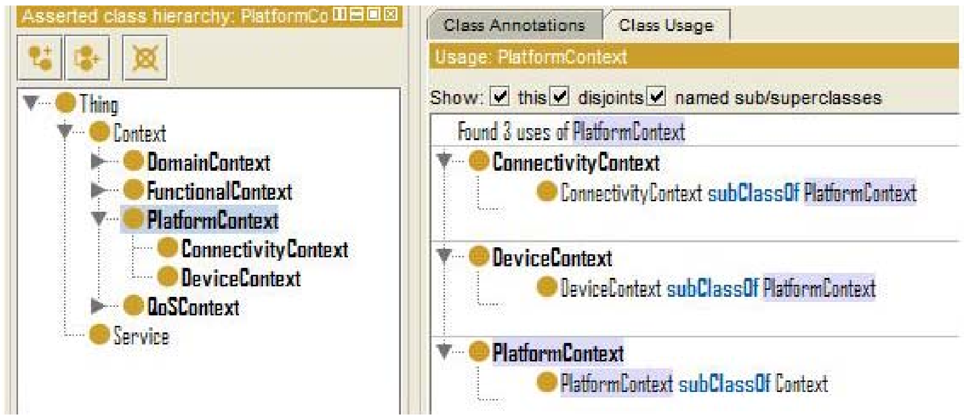
```
Thing
  Context
    DomainContext
      LinguisticsContext
      SemanticContext
      MeasuresContext
        StandardsContext
        CurrencyContext
    FunctionalContext
    PlatformContext
    QoSContext
  Service
```

Show: [✓ this] disjoints [✓ named sub/superclasses]

```
Found 4 uses of DomainContext
DomainContext
    DomainContext subClassOf Context
LinguisticsContext
    LinguisticsContext subClassOf DomainContext
MeasuresContext
    MeasuresContext subClassOf DomainContext
SemanticContext
    SemanticContext subClassOf DomainContext
```

Figure A.10: Domain context

A.4 Platform context



Found 3 uses of PlatformContext

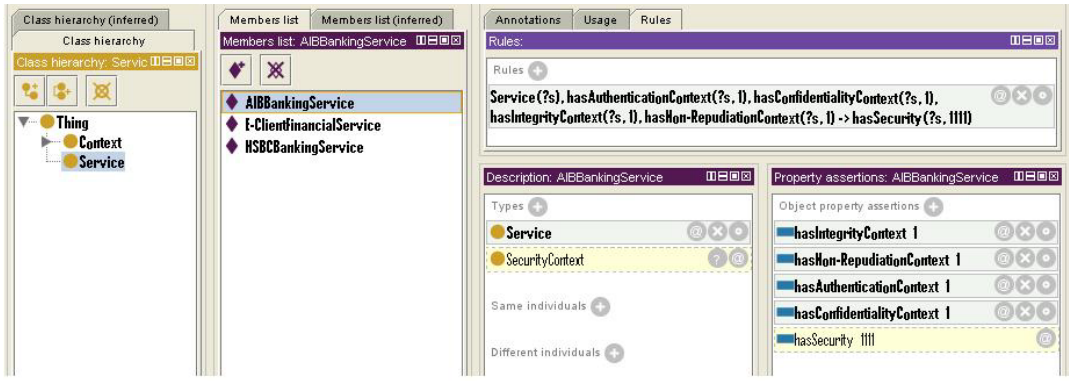
ConnectivityContext
ConnectivityContext subClassOf PlatformContext

DeviceContext
DeviceContext subClassOf PlatformContext

PlatformContext
PlatformContext subClassOf Context

Figure A.11: Platform context

A.5 Context derivation - SWRL/OWL rule



```
Service(?s), hasAuthenticationContext(?s, 1), hasConfidentialityContext(?s, 1),
hasIntegrityContext(?s, 1), hasNon-RepudiationContext(?s, 1) -> hasSecurity(?s, 1111)
```

hasIntegrityContext 1
hasNon-RepudiationContext 1
hasAuthenticationContext 1
hasConfidentialityContext 1
hasSecurity 1111

Figure A.12: SWRL rule for security context derivation

A rule for the security context of a Web service can be implemented as in figure A.12. This rule was reasoned about using the Pellet reasoner. The resulting derived context can be seen highlighted in yellow colour.

A.6 OWL-based implementation of context model ontology

The sections A.1 to A.5 describe the Protege screen shots of the implementation aspects. In this section, we present the OWL implementation of the major constituents of the context model ontology and context information integration for a HSBC banking service.

```
<?xml version="1.0"?>


<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <!ENTITY ServiceContextOntology "http://www.semanticweb.org/
     ontologies/2010/3/28/ServiceContextOntology.owl#" >
]>


<rdf:RDF xmlns="http://www.semanticweb.org/ontologies/2010/3/28/ServiceContextOntology.owl#"
     xml:base="http://www.semanticweb.org/ontologies/2010/3/28/ServiceContextOntology.owl"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:ServiceContextOntology="http://www.semanticweb.org/ontologies/2010/3/28/
      ServiceContextOntology.owl#">
    <owl:Ontology rdf:about=""/>


    <!--
    ///////////////////////////////////////////////////////////////////////////
    //
    // Object Properties
    //
    ///////////////////////////////////////////////////////////////////////////
     -->



    <!-- http://www.semanticweb.org/ontologies/2010/3/28/
     ServiceContextOntology.owl#hasAuthenticationContext -->

    <owl:ObjectProperty rdf:about="#hasAuthenticationContext">
        <rdfs:range rdf:resource="#AuthenticationContext"/>
        <rdfs:domain rdf:resource="#SecurityContext"/>
        <rdfs:domain rdf:resource="#Service"/>
    </owl:ObjectProperty>



    <!-- http://www.semanticweb.org/ontologies/2010/3/28/
     ServiceContextOntology.owl#hasConfidentialityContext -->
```

```
<owl:ObjectProperty rdf:about="#hasConfidentialityContext">
    <rdfs:range rdf:resource="#ConfidentialityContext"/>
    <rdfs:domain rdf:resource="#SecurityContext"/>
    <rdfs:domain rdf:resource="#Service"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#hasContext -->

<owl:ObjectProperty rdf:about="#hasContext">
    <rdfs:range rdf:resource="#Context"/>
    <rdfs:domain rdf:resource="#InputContext"/>
    <rdfs:domain rdf:resource="#OutputContext"/>
    <rdfs:domain rdf:resource="#Service"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#hasCostContext -->

<owl:ObjectProperty rdf:about="#hasCostContext">
    <rdfs:range rdf:resource="#CostContext"/>
    <rdfs:domain rdf:resource="#Service"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#hasCurrencyContext -->

<owl:ObjectProperty rdf:about="#hasCurrencyContext">
    <rdfs:domain rdf:resource="#CostContext"/>
    <rdfs:range rdf:resource="#CurrencyContext"/>
    <rdfs:domain rdf:resource="#InputContext"/>
    <rdfs:domain rdf:resource="#OutputContext"/>
    <rdfs:domain rdf:resource="#Service"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#hasInputContext -->

<owl:ObjectProperty rdf:about="#hasInputContext">
    <rdfs:range rdf:resource="#InputContext"/>
    <rdfs:domain rdf:resource="#Service"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#hasIntegrityContext -->

<owl:ObjectProperty rdf:about="#hasIntegrityContext">
    <rdfs:range rdf:resource="#IntegrityContext"/>
    <rdfs:domain rdf:resource="#SecurityContext"/>
    <rdfs:domain rdf:resource="#Service"/>
</owl:ObjectProperty>
```

```
<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#hasNon-RepudiationContext -->

<owl:ObjectProperty rdf:about="#hasNon-RepudiationContext">
    <rdfs:range rdf:resource="#NonRepudiationContext"/>
    <rdfs:domain rdf:resource="#SecurityContext"/>
    <rdfs:domain rdf:resource="#Service"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#hasOutputContext -->

<owl:ObjectProperty rdf:about="#hasOutputContext">
    <rdfs:range rdf:resource="#OutputContext"/>
    <rdfs:domain rdf:resource="#Service"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#hasPart -->

<owl:ObjectProperty rdf:about="#hasPart">
    <rdfs:range rdf:resource="#AuthenticationContext"/>
    <rdfs:range rdf:resource="#ConfidentialityContext"/>
    <rdfs:range rdf:resource="#IntegrityContext"/>
    <rdfs:range rdf:resource="#NonRepudiationContext"/>
    <rdfs:domain rdf:resource="#SecurityContext"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#hasPerformanceContext -->

<owl:ObjectProperty rdf:about="#hasPerformanceContext">
    <rdfs:range rdf:resource="#PerformanceContext"/>
    <rdfs:domain rdf:resource="#Service"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#hasPostConditionContext -->

<owl:ObjectProperty rdf:about="#hasPostConditionContext"/>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#hasPreConditionContext -->

<owl:ObjectProperty rdf:about="#hasPreConditionContext"/>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#hasProtocolContext -->

<owl:ObjectProperty rdf:about="#hasProtocolContext">
    <rdfs:range rdf:resource="#ProtocolContext"/>
    <rdfs:domain rdf:resource="#Service"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#hasReputiationContext -->

<owl:ObjectProperty rdf:about="#hasReputiationContext">
    <rdfs:range rdf:resource="#ReputationContext"/>
    <rdfs:domain rdf:resource="#Service"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#hasSecurity -->

<owl:ObjectProperty rdf:about="#hasSecurity">
    <rdfs:range rdf:resource="#SecurityContext"/>
    <rdfs:domain rdf:resource="#Service"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#hasTrustContext -->

<owl:ObjectProperty rdf:about="#hasTrustContext">
    <rdfs:domain rdf:resource="#Service"/>
    <rdfs:range rdf:resource="#TrustContext"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#isPartOf -->

<owl:ObjectProperty rdf:about="#isPartOf">
    <rdfs:domain rdf:resource="#AuthenticationContext"/>
    <rdfs:domain rdf:resource="#ConfidentialityContext"/>
    <rdfs:domain rdf:resource="#IntegrityContext"/>
    <rdfs:domain rdf:resource="#NonRepudiationContext"/>
    <rdfs:range rdf:resource="#SecurityContext"/>
    <owl:inverseOf rdf:resource="#hasPart"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#needConnectivityContext -->

<owl:ObjectProperty rdf:about="#needConnectivityContext">
    <rdfs:range rdf:resource="#ConnectivityContext"/>
    <rdfs:domain rdf:resource="#Service"/>
</owl:ObjectProperty>


<!--
////////////////////////////////////////////////////////////////////////////////
//
// Data properties
//
////////////////////////////////////////////////////////////////////////////////
```

```
 -->


<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#hasConnectivityStrength -->

<owl:DatatypeProperty rdf:about="#hasConnectivityStrength">
    <rdfs:domain rdf:resource="#ConnectivityContext"/>
</owl:DatatypeProperty>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#hasData -->

<owl:DatatypeProperty rdf:about="#hasData">
    <rdfs:domain rdf:resource="#InputContext"/>
    <rdfs:domain rdf:resource="#OutputContext"/>
</owl:DatatypeProperty>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#hasDefinition -->

<owl:DatatypeProperty rdf:about="#hasDefinition">
    <rdfs:domain rdf:resource="#ProtocolContext"/>
</owl:DatatypeProperty>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#hasResponseTime -->

<owl:DatatypeProperty rdf:about="#hasResponseTime">
    <rdfs:domain rdf:resource="#PerformanceContext"/>
</owl:DatatypeProperty>


<!--
///////////////////////////////////////////////////////////////////////////////
//
// Classes
//
///////////////////////////////////////////////////////////////////////////////
 -->


<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#AuthenticationContext -->

<owl:Class rdf:about="#AuthenticationContext">
    <rdfs:subClassOf rdf:resource="#SecurityContext"/>
    <owl:disjointWith rdf:resource="#ConfidentialityContext"/>
    <owl:disjointWith rdf:resource="#IntegrityContext"/>
    <owl:disjointWith rdf:resource="#NonRepudiationContext"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#AvailabilityContext -->

<owl:Class rdf:about="#AvailabilityContext">
```

```
    <rdfs:subClassOf rdf:resource="#RuntimeContext"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#BusinessContext -->

<owl:Class rdf:about="#BusinessContext">
    <rdfs:subClassOf rdf:resource="#QoSContext"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#ConfidentialityContext -->

<owl:Class rdf:about="#ConfidentialityContext">
    <rdfs:subClassOf rdf:resource="#SecurityContext"/>
    <owl:disjointWith rdf:resource="#IntegrityContext"/>
    <owl:disjointWith rdf:resource="#NonRepudiationContext"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#ConnectivityContext -->

<owl:Class rdf:about="#ConnectivityContext">
    <rdfs:subClassOf rdf:resource="#PlatformContext"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#Context -->

<owl:Class rdf:about="#Context"/>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#CostContext -->

<owl:Class rdf:about="#CostContext">
    <rdfs:subClassOf rdf:resource="#BusinessContext"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#CurrencyContext -->

<owl:Class rdf:about="#CurrencyContext">
    <rdfs:subClassOf rdf:resource="#MeasuresContext"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#DeviceContext -->

<owl:Class rdf:about="#DeviceContext">
    <rdfs:subClassOf rdf:resource="#PlatformContext"/>
```

```
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#DomainContext -->

<owl:Class rdf:about="#DomainContext">
    <rdfs:subClassOf rdf:resource="#Context"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#EffectContext -->

<owl:Class rdf:about="#EffectContext">
    <rdfs:subClassOf rdf:resource="#FunctionalContext"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#FunctionalContext -->

<owl:Class rdf:about="#FunctionalContext">
    <rdfs:subClassOf rdf:resource="#Context"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#InputContext -->

<owl:Class rdf:about="#InputContext">
    <rdfs:subClassOf rdf:resource="#SyntaxContext"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#IntegrityContext -->

<owl:Class rdf:about="#IntegrityContext">
    <rdfs:subClassOf rdf:resource="#SecurityContext"/>
    <owl:disjointWith rdf:resource="#NonRepudiationContext"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#LinguisticsContext -->

<owl:Class rdf:about="#LinguisticsContext">
    <rdfs:subClassOf rdf:resource="#DomainContext"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#MeasuresContext -->

<owl:Class rdf:about="#MeasuresContext">
    <rdfs:subClassOf rdf:resource="#DomainContext"/>
```

```
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#NonRepudiationContext -->

<owl:Class rdf:about="#NonRepudiationContext">
    <rdfs:subClassOf rdf:resource="#SecurityContext"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#OutputContext -->

<owl:Class rdf:about="#OutputContext">
    <rdfs:subClassOf rdf:resource="#SyntaxContext"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#PerformanceContext -->

<owl:Class rdf:about="#PerformanceContext">
    <rdfs:subClassOf rdf:resource="#RuntimeContext"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#PlatformContext -->

<owl:Class rdf:about="#PlatformContext">
    <rdfs:subClassOf rdf:resource="#Context"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#PostConditionContext -->

<owl:Class rdf:about="#PostConditionContext">
    <rdfs:subClassOf rdf:resource="#EffectContext"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#PreConditionContext -->

<owl:Class rdf:about="#PreConditionContext">
    <rdfs:subClassOf rdf:resource="#EffectContext"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#ProtocolContext -->

<owl:Class rdf:about="#ProtocolContext">
    <rdfs:subClassOf rdf:resource="#FunctionalContext"/>
</owl:Class>
```

```
<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#QoSContext -->

<owl:Class rdf:about="#QoSContext">
    <rdfs:subClassOf rdf:resource="#Context"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#RegulatoryContext -->

<owl:Class rdf:about="#RegulatoryContext">
    <rdfs:subClassOf rdf:resource="#BusinessContext"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#ReliabilityContext -->

<owl:Class rdf:about="#ReliabilityContext">
    <rdfs:subClassOf rdf:resource="#RuntimeContext"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#ReputationContext -->

<owl:Class rdf:about="#ReputationContext">
    <rdfs:subClassOf rdf:resource="#BusinessContext"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#RuntimeContext -->

<owl:Class rdf:about="#RuntimeContext">
    <rdfs:subClassOf rdf:resource="#QoSContext"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#SecurityContext -->

<owl:Class rdf:about="#SecurityContext">
    <rdfs:subClassOf rdf:resource="#QoSContext"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#SemanticContext -->

<owl:Class rdf:about="#SemanticContext">
    <rdfs:subClassOf rdf:resource="#DomainContext"/>
</owl:Class>
```

```
<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#Service -->

<owl:Class rdf:about="#Service"/>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#StandardsContext -->

<owl:Class rdf:about="#StandardsContext">
    <rdfs:subClassOf rdf:resource="#MeasuresContext"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#SyntaxContext -->

<owl:Class rdf:about="#SyntaxContext">
    <rdfs:subClassOf rdf:resource="#FunctionalContext"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#TrustContext -->

<owl:Class rdf:about="#TrustContext">
    <rdfs:subClassOf rdf:resource="#QoSContext"/>
</owl:Class>



<!-- http://www.w3.org/2002/07/owl#Thing -->

<owl:Class rdf:about="&owl;Thing"/>


<!--
/////////////////////////////////////////////////////////////////////////////
//
// Individuals
//
/////////////////////////////////////////////////////////////////////////////
 -->



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#0.1 -->

<owl:Thing rdf:about="#0.1">
    <rdf:type rdf:resource="#CostContext"/>
    <hasCurrencyContext rdf:resource="#GBP"/>
</owl:Thing>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#0.2 -->
```

```
<owl:Thing rdf:about="#0.2">
    <rdf:type rdf:resource="#CostContext"/>
    <hasCurrencyContext rdf:resource="#Euro"/>
</owl:Thing>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#0.3 -->

<CostContext rdf:about="#0.3">
    <rdf:type rdf:resource="&owl;Thing"/>
    <hasCurrencyContext rdf:resource="#USD"/>
</CostContext>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#1 -->

<owl:Thing rdf:about="#1">
    <rdf:type rdf:resource="#ReputationContext"/>
</owl:Thing>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#AuthLevel1 -->

<AuthenticationContext rdf:about="#AuthLevel1">
    <rdf:type rdf:resource="&owl;Thing"/>
</AuthenticationContext>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#AuthLevel2 -->

<AuthenticationContext rdf:about="#AuthLevel2">
    <rdf:type rdf:resource="&owl;Thing"/>
</AuthenticationContext>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#ConfLevel1 -->

<ConfidentialityContext rdf:about="#ConfLevel1">
    <rdf:type rdf:resource="&owl;Thing"/>
    <isPartOf rdf:resource="#ConfLevel1"/>
</ConfidentialityContext>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#ConfLevel2 -->

<ConfidentialityContext rdf:about="#ConfLevel2">
    <rdf:type rdf:resource="&owl;Thing"/>
    <isPartOf rdf:resource="#ConfLevel2"/>
</ConfidentialityContext>
```

```
<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#CurrencyTypeVerification -->

<owl:Thing rdf:about="#CurrencyTypeVerification">
    <rdf:type rdf:resource="#PreConditionContext"/>
</owl:Thing>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#E-ClientFinancialService -->

<owl:Thing rdf:about="#E-ClientFinancialService">
    <rdf:type rdf:resource="#Service"/>
    <hasCostContext rdf:resource="#0.2"/>
</owl:Thing>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#EncryptionVerification -->

<PreConditionContext rdf:about="#EncryptionVerification">
    <rdf:type rdf:resource="&owl;Thing"/>
</PreConditionContext>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#Euro -->

<owl:Thing rdf:about="#Euro">
    <rdf:type rdf:resource="#CurrencyContext"/>
</owl:Thing>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#False -->

<NonRepudiationContext rdf:about="#False">
    <rdf:type rdf:resource="#AuthenticationContext"/>
    <rdf:type rdf:resource="#ConfidentialityContext"/>
    <rdf:type rdf:resource="#IntegrityContext"/>
    <rdf:type rdf:resource="&owl;Thing"/>
</NonRepudiationContext>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#GBP -->

<owl:Thing rdf:about="#GBP">
    <rdf:type rdf:resource="#CurrencyContext"/>
</owl:Thing>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#GlobalBankingService -->

<owl:Thing rdf:about="#GlobalBankingService">
    <hasCostContext rdf:resource="#0.2"/>
</owl:Thing>
```

```
<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#HSBCBankingService -->

<Service rdf:about="#HSBCBankingService">
    <rdf:type rdf:resource="&owl;Thing"/>
    <hasCostContext rdf:resource="#0.1"/>
    <hasReputiationContext rdf:resource="#1"/>
    <hasPreConditionContext rdf:resource="#CurrencyTypeVerification"/>
    <hasPreConditionContext rdf:resource="#EncryptionVerification"/>
    <hasSecurity rdf:resource="#1111"/>
    <hasProtocolContext rdf:resource="#Pro1-HSBCBankingService"/>
    <hasTrustContext rdf:resource="#TCert1"/>
    <hasPreConditionContext rdf:resource="#UserAccountVerification"/>
    <hasInputContext rdf:resource="#accountNumberIns"/>
    <needConnectivityContext rdf:resource="#connectivity"/>
    <hasPerformanceContext rdf:resource="#perf-HSBCBankingService"/>
    <hasInputContext rdf:resource="#transactionAmountIns"/>
</Service>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#IntegriLevel1 -->

<IntegrityContext rdf:about="#IntegriLevel1">
    <rdf:type rdf:resource="&owl;Thing"/>
</IntegrityContext>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#IntegriLevel2 -->

<IntegrityContext rdf:about="#IntegriLevel2">
    <rdf:type rdf:resource="&owl;Thing"/>
</IntegrityContext>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#1111 -->

<SecurityContext rdf:about="#1111">
    <rdf:type rdf:resource="&owl;Thing"/>
    <hasNon-RepudiationContext rdf:resource="#True"/>
    <hasConfidentialityContext rdf:resource="#True"/>
    <hasAuthenticationContext rdf:resource="#True"/>
    <hasIntegrityContext rdf:resource="#True"/>
</SecurityContext>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#NonRepLevel1 -->

<owl:Thing rdf:about="#NonRepLevel1">
    <rdf:type rdf:resource="#NonRepudiationContext"/>
</owl:Thing>
```

```
<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#NonRepLevel2 -->

<owl:Thing rdf:about="#NonRepLevel2">
    <rdf:type rdf:resource="#NonRepudiationContext"/>
</owl:Thing>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#Pro1-HSBCBankingService -->

<ProtocolContext rdf:about="#Pro1-HSBCBankingService">
    <rdf:type rdf:resource="&owl;Thing"/>
    <hasDefinition rdf:datatype="&xsd;string"
        >Pre -
            EncryptionVerification,
            UserAccountVerification,
            CurrencyTypeVerification.
    </hasDefinition>
</ProtocolContext>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#TCert1 -->

<owl:Thing rdf:about="#TCert1">
    <rdf:type rdf:resource="#TrustContext"/>
</owl:Thing>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#True -->

<AuthenticationContext rdf:about="#True">
    <rdf:type rdf:resource="#ConfidentialityContext"/>
    <rdf:type rdf:resource="#IntegrityContext"/>
    <rdf:type rdf:resource="#NonRepudiationContext"/>
    <rdf:type rdf:resource="&owl;Thing"/>
</AuthenticationContext>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#USD -->

<CurrencyContext rdf:about="#USD">
    <rdf:type rdf:resource="&owl;Thing"/>
</CurrencyContext>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
 ServiceContextOntology.owl#UserAccountVerification -->

<PreConditionContext rdf:about="#UserAccountVerification">
    <rdf:type rdf:resource="&owl;Thing"/>
</PreConditionContext>



<!-- http://www.semanticweb.org/ontologies/2010/3/28/
```

```
     ServiceContextOntology.owl#accountNumberIns -->

    <InputContext rdf:about="#accountNumberIns">
        <rdf:type rdf:resource="&owl;Thing"/>
        <hasData rdf:datatype="&xsd;string">ISS1234</hasData>
    </InputContext>



    <!-- http://www.semanticweb.org/ontologies/2010/3/28/
     ServiceContextOntology.owl#connectivity -->

    <ConnectivityContext rdf:about="#connectivity">
        <rdf:type rdf:resource="&owl;Thing"/>
        <hasConnectivityStrength rdf:datatype="&xsd;string">&gt; 5 Mbps
         </hasConnectivityStrength>
    </ConnectivityContext>



    <!-- http://www.semanticweb.org/ontologies/2010/3/28/
     ServiceContextOntology.owl#perf-HSBCBankingService -->

    <owl:Thing rdf:about="#perf-HSBCBankingService">
        <rdf:type rdf:resource="#PerformanceContext"/>
        <hasResponseTime rdf:datatype="&xsd;string">&lt; 600 ms</hasResponseTime>
    </owl:Thing>



    <!-- http://www.semanticweb.org/ontologies/2010/3/28/
     ServiceContextOntology.owl#transactionAmountIns -->

    <owl:Thing rdf:about="#transactionAmountIns">
        <rdf:type rdf:resource="#InputContext"/>
        <hasData rdf:datatype="&xsd;string">3000</hasData>
        <hasContext rdf:resource="#USD"/>
    </owl:Thing>
</rdf:RDF>

<!-- Generated by the OWL API (version 2.2.1.1138) http://owlapi.sourceforge.net -->
```

A.7 Analysis of scenarios from classical business domain

We developed and analysed case-study scenarios from classical business domain, e-learning domain and convenience services domain with the help of domain experts in our research group, [Pahl 2008]. In here, we extracted some scenarios from the project delivarable D1 [Pahl 2008].

1). Scenario:

A user pays a utility bill through a service broker using his/her mobile device. The broker is responsible for composition and execution of required services while respecting dynamic requirements.

Assumptions:

The broker uses services provided by service providers and the broker is charged for usage of services. Dynamic requirements attached to SLAs are monitored at broker. The broker respects dynamic requirements so that it needs to set constraints for each process instance, such as that the cost of a process needs to be less than 0.5 USD, the security setting of a process instance needs to be "1111", the process execution time needs to be less than 400 ms, etc.

Dynamic requirements:

- Cost of a service
- Response time of a service
- Security setting of a service and process
- The user device needs to support a process output message type (e.g., process output is a MMS message)
- The user device connection needs to support a message type (e.g., connection need to support MMS messaging)
- The broker's middleware updates need to support services
- The currency type parameter of a banking service, and user information service should be similar (i.e., semantics of input and output parameters need to be similar).
- The user information service and banking service should use message encryption.

2). Scenario:

A client is booking a holiday in a remote country through a broker.

Assumptions:

A broker use services from service providers and the broker is charged for usage of services.

Dynamic requirements:

- The hotel information service should provide trusted information.
- The hotel information service can provide information based on user preferences (such as, parking facility, charges, shuttle services - the service uses its own semantic frameworks to provide requested information)
- A local banking service may be needed for the payment process and a currency conversion service need to be used as a pre-condition of the banking service.
- The currency type of input/output parameters of services needs to be matched.
- The local banking service and user information service need to use message encryption.
- The execution time of the local banking service should not exceed process execution-time constraint set by the broker.
- The linguistic information of services may need to be considered.

3). Scenario: E-learning courseware generator will generate a database courseware.

Assumptions: A courseware generator is the application system. Based on information and knowledge resources, course content is automatically generated. This case study reflects a knowledge- and information-intensive application.

Dynamic requirements:

- user device is a PDA
- a low speed GPRS connection
- trustworthiness of the content
- size of the database courseware suits for mobile device

4). Scenario: Multilingual convenience services - Technical support system (convenience service) for a software system (tool).

Assumptions: A sample technology-oriented service is considered in a multilingual environment. Convenience services are integral to recently emerging service and social networking platforms. These often span countries and languages. This domain reflects a service-based application in modern information, communications and social networking environments.

Dynamic requirements:

- language of requested information
- user device information
- trustworthiness of the content
- regulatory compliance of the content

Appendix B

Context constraints

B.1 Logical view of an ECVC

We developed explicit context validation constraints using the Message Automation Workbench. The logical view of a cost constraint can be viewed as in figure B.1,

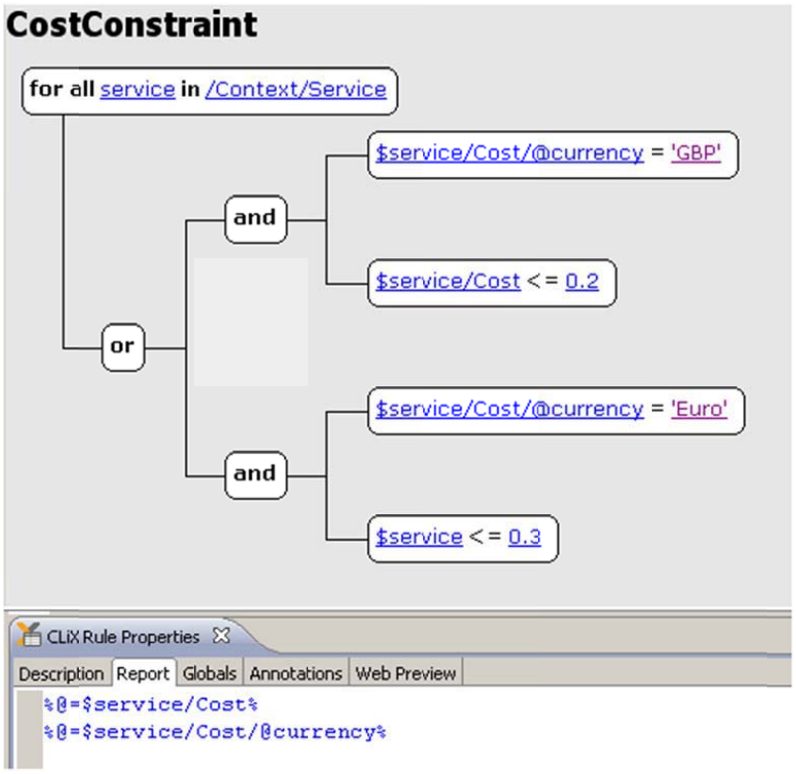
If a constraint fails at validation, a failure report is generated. The failure report can be defined as in figure B.1. These failure reports can be used for logging and monitoring processes introduced in the validation monitoring architecture in chapter 7.

B.2 Context reasoning service

ICVCs are implemented as reasoning services. A complete implementation of the reasoning service used in the case study in chapter 6 (Example 3), can be presented as follows,

```
package device;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import org.semanticweb.owlapi.apibinding.OWLManager;
import org.semanticweb.owlapi.model.*;
```



```
%@=$service/Cost%
%@=$service/Cost/@currency%
```

Figure B.1: A logical view of a cost constraint

```
import java.io.File;
import java.util.Iterator;
import java.util.Set;


@WebService()
public class Constraint_DeviceSupportMMS {
    //Web service operation

    @WebMethod(operationName = "OP_DeviceSupportMMS")
    public String OP_DeviceSupportMMS(@WebParam(name = "device")
    String device, @WebParam(name = "dFeature")
    String dFeature, @WebParam(name = "uriOntology")
    String uriOntology) {
        //TODO write your implementation code here:
        String ret="";
        OWLClass c = null;
        OWLNamedIndividual ins = null;
        OWLObjectProperty Oprop = null;

        try{
            // Create the manager that we will use to load ontologies.
            OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
            // Let's load an ontology
            //File file = new File("C:/NewProtoType/ServiceContext.owl");
            File file = new File(uriOntology);
            OWLOntology ontology = manager.loadOntologyFromOntologyDocument(file);

            Set classes = ontology.getClassesInSignature();
            for(Iterator itins = classes.iterator(); itins.hasNext();){
                  c = (OWLClass)itins.next();
                  if( c.getNNF().toString().contains("MobileDevice")){
                      //ret = c.getIndividuals(ontology).toString();
                      Set<OWLIndividual> individuals = c.getIndividuals(ontology);
                      //ret = individuals.toString();
                      //check for device instance
                      for(Iterator it2 = individuals.iterator(); it2.hasNext();){
                        ins = (OWLNamedIndividual)it2.next();
                        //ret = ins.toString();
                        if(ins.toString().contains(device)){
                           //ret = ins.toString();
                            Set<OWLObjectPropertyExpression> props =
                             ins.getObjectPropertyValues(ontology).keySet();
                            for(Iterator it3 = props.iterator(); it3.hasNext();){
                                Oprop = (OWLObjectProperty)it3.next();
                                if(Oprop.toString().contains("hasFacilities")){
                                    //ret = Oprop.toString();
                                   Set <OWLIndividual> ins2 =
                                    ins.getObjectPropertyValues(Oprop, ontology);
                                   for(Iterator it4= ins2.iterator(); it4.hasNext();){
                                       if(it4.next().toString().contains(dFeature)){
                                            ret = "true";
                                       }else{
                                             ret = "false";
                                       }
                                   }//for Feature

                                }//if object property

                            }//for object property

                        }//if individual


                      }//for select individual

                  }//if MobileDevice
            }//for select class


        }
        catch (Exception e) {
            ret = e.toString();
        }

        return ret;
    }//end of Web method

}//end of class
```

Appendix C

Prototype : Constraints instrumentation and validation monitoring

C.1 Overview

The overall architecture for dynamic instrumentation of context constraints and their validation monitoring is illustrated in figure 7.1. We implemented a prototype to illustrate dynamic instrumentation of context constraints and their validation monitoring at process run-time. The implementation aspects are illustrated in the following sections.

C.2 Instrumentation and validation monitoring

The implementation of IVS (figure 7.1) with its operations and a data collector are detailed. The IVS in chapter 7 is used to instrument and validate both ECVCs and ICVCs described in chapter 6.

C.2.1 Instrumentation and validation operation for ECVCs

This section describes the implementation of `OperationValidation`, which enables dynamic instrumentation and validation monitoring of ECVCs (chapter 6).

For example, the pre-conditions attached to `PaymentService` is illustrated.

The configuration file PreCon-PaymentService.mavconfig is hard coded, which can pass as an input parameter.

Input: DynamicContext.xml, PreCon-PaymentService.mavconfig

Output: True or False

```java
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import javax.xml.parsers.*;
import com.systemwire.xlinkit.api.*;
import org.w3c.dom.*;

@WebService()
public class WebService_Validation {

    @WebMethod(operationName = "OperationValidation")
    public String operation(@WebParam(name = "DynamicContext")
    String DynamicContext,@WebParam(name = "url_config") String url_config) {
     //the configuration file is hard coded for the illustration purpose
     String url_config = "PreCon-PaymentService.mavconfig";
     String output = "";
     Document doc = null;
     try{
         DocumentBuilderFactory fact = DocumentBuilderFactory.newInstance();
         DocumentBuilder builder = fact.newDocumentBuilder();
          // DynamicContext contains context instances
         doc = builder.parse(DynamicContext);
      }catch (Exception e){output = e.toString(); System.exit(1);}

      Validator validator = new Validator();
      try{
        validator.initialise(".../personalDomain/config",url_config);
      }catch(ValidatorException e){
        output = "Failed to initialize";
        System.exit(1);
      }

      // perform the validation
      try{
         ValidatorResult result = validator.check(doc, Validator.REPORT_HTML);
         switch (result.getResult()){
            case ValidatorResult.RESULT_PASS:
                 output= "Validation True";
                 break;
            case ValidatorResult.RESULT_RULE_FAILED:
                 output = "Validation False" ;
                 break;
           }//switch
      }catch(ValidatorException e){
```

```
                output = "Failed to validate";
                System.exit(1);
            }

        return output;
        }// end of validation method
}//end of class
```

The xlinkit.api is imported into the validation service. The `Validator` class is used to create a validator instance, and `initialise()` method is used to initialise the validator instance to a given configuration file. Once the validator instance has been initialised, then the `check()` method is used to validate constraints attached in the configuration file with dynamic context instances attached in the document object `doc`.

C.2.2 Instrumentation and validation operation for ICVCs

This example illustrates dynamic instrumentation of an ICVC (implemented as a reasoning service) to reason about a device property (or connection data). The ICVC profile attached to a *para_Name* is selected dynamically. The binding and SOAP message information of the ICVC is captured from the ICVC profile. The binding and SOAP message are generated, and the context reasoning service is invoked. The reasoning service, reasons a user device using the given ontology for MMS message facility. This reasoning service outputs True or False accordingly.

Input: MobileDevice, N95, MMS, url_ContextOntology

Output: True or False

```

// Web service operation
 @WebMethod(operationName = "operation-ReasoningInvoke")
     public String operation_decision(@WebParam(name = "para_Name")
     String para_Name, @WebParam(name = "para_Value")
     String para_Value, @WebParam(name = "para_Feature")
     String para_Feature, @WebParam(name = "url_paraOntology")
     String url_paraOntology) {

//selecting ICVC profile (ICVCprofile),
//this info can be stored in a database table
     if (para_Name.equalsIgnoreCase("device")){
            url_ICVCprofile_ConstraintService = "../constraints/ICVCprofile_DeviceSupportMMSConstraint.xml";
        }else if (para_Name.equalsIgnoreCase("connection")){
            url_ICVCprofile_ConstraintService = "../constraints/ICVCprofile_ConnectionDataConstraint.xml";
        }


        File file = new File(url_ICVCprofile_ConstraintService);
        if(file.exists()){
            try{
              //read ICVC profile and get binding and soap message infor
               DocumentBuilderFactory fact = DocumentBuilderFactory.newInstance();
               DocumentBuilder builder = fact.newDocumentBuilder();
               Document doc = builder.parse(file);

               endPointUrl = doc.getElementsByTagName("endPointUrl").item(0).getTextContent();
               serviceName = doc.getElementsByTagName("serviceName").item(0).getTextContent();
               portName = doc.getElementsByTagName("portName").item(0).getTextContent();
               operationName = doc.getElementsByTagName("operationName").item(0).getTextContent();
               messagePrefix = doc.getElementsByTagName("messagePrefix").item(0).getTextContent();
               messageNameSpace = doc.getElementsByTagName("messageNameSpace").item(0).getTextContent();
               paraName = doc.getElementsByTagName("paraName").item(0).getTextContent();
               paraValue = para_Value;//N95, N12, O2connection etc.
               paraNameFeature = doc.getElementsByTagName("paraNameFeature").item(0).getTextContent();
               paraValueFeature = para_Feature;//MMS
               paraNameOntoUrl = doc.getElementsByTagName("paraNameOntoUrl").item(0).getTextContent();
               paraValueOntoUrl = url_paraOntology;//ontology url of device, connection


              //invoking external service
               QName QserviceName = new QName(serviceName);
               QName QportName = new QName(portName);

                      //Create a service and add at least one port to it
               Service service = Service.create(QserviceName);
               service.addPort(QportName, SOAPBinding.SOAP11HTTP_BINDING, endPointUrl);
               Dispatch dispatch = service.createDispatch(QportName, SOAPMessage.class, Service.Mode.MESSAGE);

               MessageFactory messageFactory = MessageFactory.newInstance();
               SOAPMessage message = messageFactory.createMessage();

              //Create objects for the message parts
               SOAPPart soapPart = message.getSOAPPart();
               SOAPEnvelope envelope = soapPart.getEnvelope();
               SOAPBody body = envelope.getBody();

              // Construct the message payload.
               SOAPElement operation = body.addChildElement(operationName, messagePrefix,messageNameSpace);
               SOAPElement device = operation.addChildElement(paraName);
               device.addTextNode(paraValue);
               SOAPElement feature = operation.addChildElement(paraNameFeature);
               feature.addTextNode(paraValueFeature);
               SOAPElement uriOntology = operation.addChildElement(paraNameOntoUrl);
               uriOntology.addTextNode(paraValueOntoUrl);
               message.saveChanges();

              //invoke the reasoning service
               SOAPMessage response = (SOAPMessage)dispatch.invoke(message);
               ret = response.getSOAPBody().getFirstChild().getTextContent();

            }
            catch(Exception e){ret=e.toString()+" : while invoking";}

         }else{ret="File not found"};

      return ret;

    }//end of web method
```

C.2.3 ICVC Profile

ICVC profile used in the above ICVC is,

```

<Service id = "ICVCprofile_DeviceSupportMMSConstraint">
    <Binding>
        <endPointUrl>http://localhost:8080/ProTypeV1_WebApp/Constraint_DeviceSupportMMSService</endPointUrl>
        <serviceName>Constraint_DeviceSupportMMSService</serviceName>
        <portName>Constraint_DeviceSupportMMSPort</portName>
    </Binding>
    <SOAP>
        <operationName>OP_DeviceSupportMMS</operationName>
        <messagePrefix>ns2</messagePrefix>
        <messageNameSpace>http://device/</messageNameSpace>
        <paraName>device</paraName>
        <paraValue>temp</paraValue>
        <paraNameFeature>dFeature</paraNameFeature>
        <paraValueFeature>temp</paraValueFeature>
        <paraNameOntoUrl>uriOntology</paraNameOntoUrl>
        <paraValueOntoUrl>temp</paraValueOntoUrl>
    </SOAP>
</Service>
```

The default value of each field is temp.

C.2.4 Data collector

We used monitoring directives and data collectors to monitor, collect and facilitate transitory context to the context space of the service process execution. The response time data collector described in the case study in chapter 7 was implemented as,

Input: InTime, OutTime

Output: ResponseTime context

```

@WebService()
public class WebService_ResponseTime_DataCollector {

 //Web service operation

    @WebMethod(operationName = "operation_ResponseTime")
    public String operation_ResponseTime(@WebParam(name = "InTime")
    String InTime, @WebParam(name = "OutTime")
    String OutTime) {

        String t1MS = InTime.substring(9,11);
        int t1MSint = Integer.parseInt(t1MS);

        String t2MS = OutTime.substring(9,11);
        int t2MSint = Integer.parseInt(t2MS);

        String t1S = InTime.substring(6,8);
        int t1Sint = Integer.parseInt(t1S);

        String t2S = OutTime.substring(6,8);
        int t2Sint = Integer.parseInt(t2S);

        int out=0;
        if(t2MSint≥t1MSint){
            out = ((t2MSint − t1MSint)+(t2Sint − t1Sint)∗1000);
        }else{
            out = ((t2MSint + (t2Sint−t1Sint)∗1000) − t1MSint);
        }

        //write data to an XML file
        try{
            OutputStream fout = new FileOutputStream(".../personalDomain/config/RuntimeContext.xml");
            OutputStream bout = new BufferedOutputStream(fout);
            OutputStreamWriter outW = new OutputStreamWriter(bout);

            outW.write("<?xml version=\"1.0\" ");
            outW.write("standalone=\"no\"?>\r\n");

```

```
                outW.write("<Context>\r\n");
                    outW.write("<RuntimeContext>\r\n");
                        outW.write("<ResponseTime>");
                            outW.write(Integer.toString(out));
                        outW.write("</ResponseTime>\r\n");
                    outW.write("</RuntimeContext>\r\n");
                outW.write("</Context>\r\n");

                outW.flush();
                outW.close();

            }catch(UnsupportedEncodingException e){}
             catch(IOException e){}

            return ("RuntimeContext.xml");
        }

    }
```

C.3 Performance evaluation

C.3.1 Design view of the instrumented process

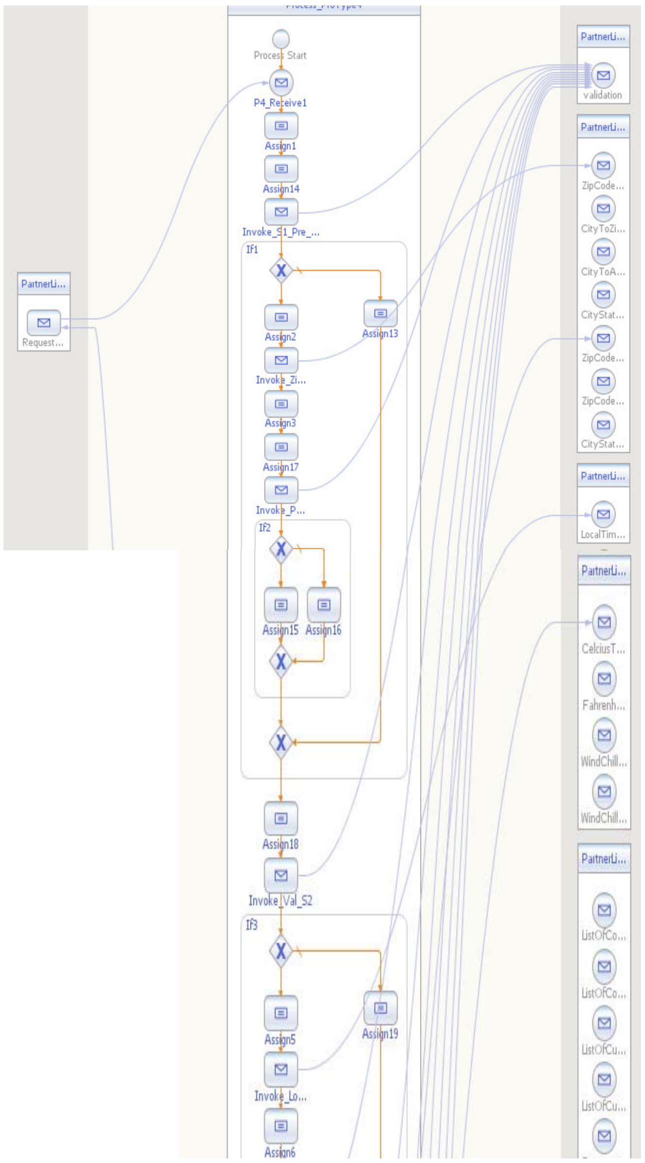


Figure C.1: A part of instrumented process

This is a design view of an instrumented process, which we developed using NeBeans IDE and its integrated BPEL designer.

C.3.2 Process coding

```
<?xml version="1.0" encoding="UTF-8"?>
<process
    name="Process_ProType4"
    targetNamespace="http://enterprise.netbeans.org/bpel/BpelModule_ProType4/Process_ProType4"
    xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:sxt="http://www.sun.com/wsbpel/2.0/process/executable/SUNExtension/Trace"
    xmlns:sxed="http://www.sun.com/wsbpel/2.0/process/executable/SUNExtension/Editor"
    xmlns:tns="http://enterprise.netbeans.org/bpel/BpelModule_ProType4/Process_ProType4"
    xmlns:ns0="http://www.ripedev.com/"
    xmlns:sxxf="http://www.sun.com/wsbpel/2.0/process/executable/SUNExtension/XPathFunctions"
    xmlns:ns1="http://microsoft.com/webservices/"
    xmlns:ns2="http://webservices.daehosting.com/temperature"
    xmlns:ns3="http://www.oorsprong.org/websamples.countryinfo">
    <import namespace="http://j2ee.netbeans.org/wsdl/RequestReply"
    location="Partners/RequestReply/RequestReply.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://enterprise.netbeans.org/bpel/ZipCode.asmxWrapper"
    location="Partners/ZipCode.asmx/ZipCode.asmxWrapper.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://www.ripedev.com/"
    location="Partners/ZipCode.asmx/ZipCode.asmx.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://enterprise.netbeans.org/bpel/LocalTime.asmxWrapper"
    location="Partners/LocalTime.asmx/LocalTime.asmxWrapper.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://www.ripedev.com/"
    location="Partners/LocalTime.asmx/LocalTime.asmx.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://enterprise.netbeans.org/bpel/TemperatureConversions.wsoWrapper"
    location="Partners/TemperatureConversions.wso/TemperatureConversions.wsoWrapper.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://webservices.daehosting.com/temperature"
    location="Partners/TemperatureConversions.wso/TemperatureConversions.wso.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://enterprise.netbeans.org/bpel/CountryInfoService.wsoWrapper"
    location="Partners/CountryInfoService.wso/CountryInfoService.wsoWrapper.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://www.oorsprong.org/websamples.countryinfo"
    location="Partners/CountryInfoService.wso/CountryInfoService.wso.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://enterprise.netbeans.org/bpel/WebService_ValidationServiceWrapper"
    location="Partners/WebService_Validation/WebService_ValidationServiceWrapper.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://ProType1/"
    location="Partners/WebService_Validation/WebService_ValidationService.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://enterprise.netbeans.org/bpel/
    WebService_ProcessExecutionTimeInMillisecondsServiceWrapper"
    location="Partners/WebService_ProcessExecutionTimeInMilliseconds/
    WebService_ProcessExecutionTimeInMillisecondsServiceWrapper.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://ProType1/"
    location="Partners/WebService_ProcessExecutionTimeInMilliseconds/
    WebService_ProcessExecutionTimeInMillisecondsService.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/"/>
    <partnerLinks>
        <partnerLink name="PartnerLink_Validation"
        xmlns:tns="http://enterprise.netbeans.org/bpel/WebService_ValidationServiceWrapper"
        partnerLinkType="tns:WebService_ValidationLinkTypeP4"
        partnerRole="WebService_ValidationRole"/>
        <partnerLink name="PartnerLink_Zip"
        xmlns:tns="http://enterprise.netbeans.org/bpel/ZipCode.asmxWrapper"
        partnerLinkType="tns:ZipCodeSoapLinkType" partnerRole="ZipCodeSoapRole"/>
        <partnerLink name="PartnerLink_LocalTime"
        xmlns:tns="http://enterprise.netbeans.org/bpel/LocalTime.asmxWrapper"
        partnerLinkType="tns:LocalTimeSoapLinkType" partnerRole="LocalTimeSoapRole"/>
        <partnerLink name="PartnerLink_TempConversion"
        xmlns:tns="http://enterprise.netbeans.org/bpel/TemperatureConversions.wsoWrapper"
        partnerLinkType="tns:TemperatureConversionsSoapTypeLinkType"
        partnerRole="TemperatureConversionsSoapTypeRole"/>
        <partnerLink name="PartnerLink_CountryInfo"
        xmlns:tns="http://enterprise.netbeans.org/bpel/CountryInfoService.wsoWrapper"
        partnerLinkType="tns:CountryInfoServiceSoapTypeLinkType"
        partnerRole="CountryInfoServiceSoapTypeRole"/>
        <partnerLink name="PartnerLink_results"
        xmlns:tns="http://enterprise.netbeans.org/bpel/
        WebService_ProcessExecutionTimeInMillisecondsServiceWrapper"
        partnerLinkType="tns:WebService_ProcessExecutionTimeInMillisecondsLinkTypeR"
        partnerRole="WebService_ProcessExecutionTimeInMillisecondsRole"/>
        <partnerLink name="PartnerLink1" xmlns:tns="http://j2ee.netbeans.org/wsdl/RequestReply"
        partnerLinkType="tns:RequestReply" myRole="RequestReplyPortTypeRole"/>
    </partnerLinks>
    <variables>
        <variable name="ZipCodeToAreaCode_Out" messageType="ns0:ZipCodeToAreaCodeSoapOut"/>
        <variable name="ZipCodeToAreaCode_In" messageType="ns0:ZipCodeToAreaCodeSoapIn"/>
        <variable name="out_Time" type="xsd:string"/>
        <variable name="Operation_ProcessExecutionTime_Out" xmlns:tns="http://ProType1/"
        messageType="tns:operation_ProcessExecutionTimeResponse"/>
        <variable name="Operation_ProcessExecutionTime_In" xmlns:tns="http://ProType1/"
        messageType="tns:operation_ProcessExecutionTime"/>
        <variable name="S5_PostValidation_Out" xmlns:tns="http://ProType1/"
        messageType="tns:validationResponse"/>
        <variable name="S5_PostValidation_In" xmlns:tns="http://ProType1/"
        messageType="tns:validation"/>
        <variable name="S5_PreValidation_Out" xmlns:tns="http://ProType1/"
        messageType="tns:validationResponse"/>
        <variable name="S5_PreValidation_In" xmlns:tns="http://ProType1/"
        messageType="tns:validation"/>
        <variable name="S4_PostValidation_Out" xmlns:tns="http://ProType1/"
        messageType="tns:validationResponse"/>
        <variable name="S4_PostValidation_In" xmlns:tns="http://ProType1/"
        messageType="tns:validation"/>
        <variable name="S4_PreValidation_Out" xmlns:tns="http://ProType1/"
        messageType="tns:validationResponse"/>
        <variable name="S4_PreValidation_In" xmlns:tns="http://ProType1/"
        messageType="tns:validation"/>
        <variable name="S3_PostValidation_Out" xmlns:tns="http://ProType1/"
        messageType="tns:validationResponse"/>
        <variable name="S3_PostValidation_In" xmlns:tns="http://ProType1/"
        messageType="tns:validation"/>
        <variable name="S3_Pre_Validation_Out" xmlns:tns="http://ProType1/"
        messageType="tns:validationResponse"/>
        <variable name="S3_Pre_Validation_In" xmlns:tns="http://ProType1/"
        messageType="tns:validation"/>
        <variable name="Post_Validation_S2_Out" xmlns:tns="http://ProType1/"
        messageType="tns:validationResponse"/>
        <variable name="Post_Validation_S2_In" xmlns:tns="http://ProType1/"
        messageType="tns:validation"/>
        <variable name="Pre_Validation_S2_Out" xmlns:tns="http://ProType1/"
        messageType="tns:validationResponse"/>
        <variable name="Pre_Validation_S2_In" xmlns:tns="http://ProType1/"
        messageType="tns:validation"/>
        <variable name="S1_Post_Validation_Out" xmlns:tns="http://ProType1/"
        messageType="tns:validationResponse"/>
        <variable name="S1_Post_Validation_In" xmlns:tns="http://ProType1/"
        messageType="tns:validation"/>
        <variable name="PrePostTrue" type="xsd:string"/>
        <variable name="PostFail" type="xsd:string"/>
        <variable name="PreFail" type="xsd:string"/>
        <variable name="S1_Pre_Validation_Out"
        xmlns:tns="http://ProType1/" messageType="tns:validationResponse"/>
        <variable name="S1_Pre_Validation_In"
        xmlns:tns="http://ProType1/" messageType="tns:validation"/>
        <variable name="CountryCurrency_Out"
        xmlns:tns="http://www.oorsprong.org/websamples.countryinfo"
        messageType="tns:CountryCurrencySoapResponse"/>
        <variable name="CountryCurrency_In"
        xmlns:tns="http://www.oorsprong.org/websamples.countryinfo"
        messageType="tns:CountryCurrencySoapRequest"/>
        <variable name="CelciusToFahrenheit_Out"
        xmlns:tns="http://webservices.daehosting.com/temperature"
        messageType="tns:CelciusToFahrenheitSoapResponse"/>
        <variable name="CelciusToFahrenheit_In"
        xmlns:tns="http://webservices.daehosting.com/temperature"
        messageType="tns:CelciusToFahrenheitSoapRequest"/>
        <variable name="LocalTimeByZipCode_Out" messageType="ns0:LocalTimeByZipCodeSoapOut"/>
        <variable name="LocalTimeByZipCode_In" messageType="ns0:LocalTimeByZipCodeSoapIn"/>
        <variable name="in_Time" type="xsd:string"/>
        <variable name="ZipCodeToCityState_Out" xmlns:s0="http://www.ripedev.com/"
        messageType="s0:ZipCodeToCityStateSoapOut"/>
        <variable name="ZipCodeToCityState_In" xmlns:s0="http://www.ripedev.com/"
        messageType="s0:ZipCodeToCityStateSoapIn"/>
        <variable name="P4_RequestReplyOperation_Out"
        xmlns:tns="http://j2ee.netbeans.org/wsdl/RequestReply"
        messageType="tns:RequestReplyOperationResponse"/>
        <variable name="P4_RequestReplyOperation_In"
        xmlns:tns="http://j2ee.netbeans.org/wsdl/RequestReply"
        messageType="tns:RequestReplyOperationRequest"/>
    </variables>
    <sequence>
        <receive name="P4_Receive1" createInstance="yes" partnerLink="PartnerLink1"
        operation="RequestReplyOperation"
        xmlns:tns="http://j2ee.netbeans.org/wsdl/RequestReply"
        portType="tns:RequestReplyPortType" variable="P4_RequestReplyOperation_In"/>
        <assign name="Assign1">
            <copy>
                <from variable="P4_RequestReplyOperation_In" part="part1"></from>
                <to variable="P4_RequestReplyOperation_Out" part="part1"/>
            </copy>
            <copy>
                <from>sxxf:current-time()</from>
                <to variable="in_Time"/>
            </copy>
            <copy>
                <from>'Pre Conditions Fail : '</from>
                <to variable="PreFail"/>
            </copy>
            <copy>
                <from>'PrePostTrue : '</from>
                <to variable="PrePostTrue"/>
            </copy>
            <copy>
                <from>'PostconditionFail: '</from>
                <to variable="PostFail"/>
            </copy>
        </assign>
        <assign name="Assign14">
            <copy>
                <from>'Context_BillService.xml'</from>
                <to>$S1_Pre_Validation_In.parameters/datafile</to>
            </copy>
            <copy>
                <from>’config_BillService.xml’</from>
                <to>$S1_Pre_Validation_In.parameters/conf</to>
            </copy>
        </assign>
        <invoke name="Invoke_S1_Pre_Validation" partnerLink="PartnerLink_Validation"
        operation="validation" xmlns:tns="http://ProType1/"
        portType="tns:WebService_Validation"
        inputVariable="S1_Pre_Validation_In" outputVariable="S1_Pre_Validation_Out"/>
        <if name="If1">
            <condition>$S1_Pre_Validation_Out.parameters/return =
            ’Pre condition: rules are valid’</condition>
            <sequence name="Sequence1">
                <assign name="Assign2">
                    <copy>
                            <from>’12288’</from>
                                <to>$ZipCodeToCityState_In.parameters/ns0:ZipCode</to>
                        </copy>
                </assign>
                <invoke name="Invoke_ZipCode" partnerLink="PartnerLink_Zip"
                operation="ZipCodeToCityState" portType="ns0:ZipCodeSoap"
                inputVariable="ZipCodeToCityState_In" outputVariable="ZipCodeToCityState_Out"/>
                <assign name="Assign3">
                    <copy>
                            <from>concat($P4_RequestReplyOperation_Out.part1,
                             $ZipCodeToCityState_Out.parameters/ns0:ZipCodeToCityStateResult/
                             ns0:anyType)</from>
                                <to variable="P4_RequestReplyOperation_Out" part="part1"/>
                        </copy>
                </assign>
                <assign name="Assign17">
                    <copy>
                        <from>’Context_BillService.xml’</from>
                        <to>$S1_Post_Validation_In.parameters/datafile</to>
                    </copy>
                    <copy>
                        <from>’config_BillService.xml’</from>
                        <to>$S1_Post_Validation_In.parameters/conf</to>
                    </copy>
                </assign>
                <invoke name="Invoke_PostVal_S1" partnerLink="PartnerLink_Validation"
                operation="validation" xmlns:tns="http://ProType1/"
                portType="tns:WebService_Validation" inputVariable="S1_Post_Validation_In"
                outputVariable="S1_Post_Validation_Out"/>
                <if name="If2">
                    <condition>$S1_Post_Validation_Out.parameters/return =
                    ’Pre condition: rules are valid’</condition>
                    <assign name="Assign15">
                        <copy>
                            <from>concat($PrePostTrue, ’S1:’)</from>
                            <to variable="PrePostTrue"/>
                        </copy>
                    </assign>
                    <else>
                        <assign name="Assign16">
                            <copy>
                                <from>concat($PostFail, ’S1 : ’)</from>
                                <to variable="PostFail"/>
                            </copy>
                        </assign>
                    </else>
                </if>
            </sequence>
            <else>
                <assign name="Assign13">
                    <copy>
                        <from>concat($PreFail, ’:S1:’)</from>
                        <to variable="PreFail"/>
                    </copy>
                </assign>
            </else>
        </if>
        <assign name="Assign18">
            <copy>
                <from>’Context_BillService.xml’</from>
                <to>$Pre_Validation_S2_In.parameters/datafile</to>
            </copy>
            <copy>
                <from>’config_BillService.xml’</from>
                <to>$Pre_Validation_S2_In.parameters/conf</to>
            </copy>
        </assign>
        <invoke name="Invoke_Val_S2" partnerLink="PartnerLink_Validation"
        operation="validation" xmlns:tns="http://ProType1/"
        portType="tns:WebService_Validation"
        inputVariable="Pre_Validation_S2_In" outputVariable="Pre_Validation_S2_Out"/>
        <if name="If3">
            <condition>$Pre_Validation_S2_Out.parameters/return =
            ’Pre condition: rules are valid’</condition>
            <sequence name="Sequence2">
                <assign name="Assign5">
                    <copy>
                        <from>’12288’</from>
                            <to>$LocalTimeByZipCode_In.parameters/ns0:ZipCode</to>
                    </copy>
                </assign>
                <invoke name="Invoke_LocalTime" partnerLink="PartnerLink_LocalTime"
                operation="LocalTimeByZipCode" portType="ns0:LocalTimeSoap"
                inputVariable="LocalTimeByZipCode_In" outputVariable="LocalTimeByZipCode_Out"/>
                <assign name="Assign6">
                    <copy>
                            <from>concat($P4_RequestReplyOperation_Out.part1,
                            $LocalTimeByZipCode_Out.parameters/ns0:LocalTimeByZipCodeResult)
                            </from>
                                <to variable="P4_RequestReplyOperation_Out" part="part1"/>
                        </copy>
                </assign>
                <assign name="Assign20">
                    <copy>
                        <from>’Context_BillService.xml’</from>
                        <to>$Post_Validation_S2_In.parameters/datafile</to>
                    </copy>
                    <copy>
                        <from>’config_BillService.xml’</from>
                        <to>$Post_Validation_S2_In.parameters/conf</to>
                    </copy>
                </assign>
                <invoke name="Invoke_PostVal_S2" partnerLink="PartnerLink_Validation"
                operation="validation" xmlns:tns="http://ProType1/"
                portType="tns:WebService_Validation" inputVariable="Post_Validation_S2_In"
                outputVariable="Post_Validation_S2_Out"/>
                <if name="If4">
                    <condition>$Post_Validation_S2_Out.parameters/return =
                    ’Pre condition: rules are valid’</condition>
                    <assign name="Assign21">
                        <copy>
```

```
                        <from>concat($PrePostTrue, 'S2 : ')</from>
                        <to variable="PrePostTrue"/>
                    </copy>
                </assign>
                <else>
                    <assign name="Assign22">
                        <copy>
                            <from>concat($PostFail, 'S2 : ')</from>
                            <to variable="PostFail"/>
                        </copy>
                    </assign>
                </else>
            </if>
        </sequence>
        <else>
            <assign name="Assign19">
                <copy>
                    <from>concat($PreFail, 'S2 : ')</from>
                    <to variable="PreFail"/>
                </copy>
            </assign>
        </else>
    </if>
    <assign name="Assign23">
        <copy>
            <from>'Context_BillService.xml'</from>
            <to>$S3_Pre_Validation_In.parameters/datafile</to>
        </copy>
        <copy>
            <from>'config_BillService.xml'</from>
            <to>$S3_Pre_Validation_In.parameters/conf</to>
        </copy>
    </assign>
    <invoke name="Invoke_Val_S3" partnerLink="PartnerLink_Validation"
    operation="validation" xmlns:tns="http://ProType1/"
    portType="tns:WebService_Validation"
    inputVariable="S3_Pre_Validation_In" outputVariable="S3_Pre_Validation_Out"/>
    <if name="If5">
        <condition>$S3_Pre_Validation_Out.parameters/return =
        'Pre condition: rules are valid'</condition>
        <sequence name="Sequence3">
            <assign name="Assign7">
                <copy>
                    <from>'12288'</from>
                    <to>$ZipCodeToAreaCode_In.parameters/ns0:ZipCode</to>
                </copy>
            </assign>
            <invoke name="Invoke_pn" partnerLink="PartnerLink_Zip"
            operation="ZipCodeToAreaCode"
            portType="ns0:ZipCodeSoap" inputVariable="ZipCodeToAreaCode_In"
            outputVariable="ZipCodeToAreaCode_Out"/>
            <assign name="Assign8">
                <copy>
                        <from>concat($P4_RequestReplyOperation_Out.part1,
                         $ZipCodeToAreaCode_Out.parameters/ns0:ZipCodeToAreaCodeResult/
                         ns0:anyType)</from>
                            <to variable="P4_RequestReplyOperation_Out" part="part1"/>
                    </copy>
            </assign>
            <assign name="Assign25">
                <copy>
                    <from>'Context_BillService.xml'</from>
                    <to>$S3_PostValidation_In.parameters/datafile</to>
                </copy>
                <copy>
                    <from>’config_BillService.xml’</from>
                    <to>$S3_PostValidation_In.parameters/conf</to>
                </copy>
            </assign>
            <invoke name="Invoke_PostVal_S3" partnerLink="PartnerLink_Validation"
            operation="validation" xmlns:tns="http://ProType1/"
            portType="tns:WebService_Validation" inputVariable="S3_PostValidation_In"
            outputVariable="S3_PostValidation_Out"/>
            <if name="If6">
                <condition>$S3_PostValidation_Out.parameters/return =
                ’Pre condition: rules are valid’</condition>
                <assign name="Assign26">
                    <copy>
                        <from>concat($PrePostTrue, ’S3 : ’)</from>
                        <to variable="PrePostTrue"/>
                    </copy>
                </assign>
                <else>
                    <assign name="Assign27">
                        <copy>
                            <from>concat($PostFail, ’S3 : ’)</from>
                            <to variable="PostFail"/>
                        </copy>
                    </assign>
                </else>
            </if>
        </sequence>
        <else>
            <assign name="Assign24">
                <copy>
                    <from>concat($PreFail, ’S3 : ’)</from>
                    <to variable="PreFail"/>
                </copy>
            </assign>
        </else>
    </if>
    <assign name="Assign28">
        <copy>
            <from>’Context_BillService.xml’</from>
            <to>$S4_PreValidation_In.parameters/datafile</to>
        </copy>
        <copy>
            <from>’config_BillService.xml’</from>
            <to>$S4_PreValidation_In.parameters/conf</to>
        </copy>
    </assign>
    <invoke name="Invoke_PreVal_S4" partnerLink="PartnerLink_Validation"
    operation="validation"
     xmlns:tns="http://ProType1/" portType="tns:WebService_Validation"
     inputVariable="S4_PreValidation_In" outputVariable="S4_PreValidation_Out"/>
    <if name="If7">
        <condition>$S4_PreValidation_Out.parameters/return =
        ’Pre condition: rules are valid’</condition>
        <sequence name="Sequence4">
            <assign name="Assign9">
                <copy>
                    <from>30</from>
                        <to>$CelciusToFahrenheit_In.parameters/ns2:nCelcius</to>
                </copy>
            </assign>
            <invoke name="Invoke_TemperatureConvert"
            partnerLink="PartnerLink_TempConversion"
            operation="CelciusToFahrenheit" portType="ns2:TemperatureConversionsSoapType"
            inputVariable="CelciusToFahrenheit_In"
            outputVariable="CelciusToFahrenheit_Out"/>
            <assign name="Assign10">
                <copy>
                        <from>concat($P4_RequestReplyOperation_Out.part1,
                         $CelciusToFahrenheit_Out.parameters/ns2:CelciusToFahrenheitResult)
                         </from>
                            <to variable="P4_RequestReplyOperation_Out" part="part1"/>
                    </copy>
            </assign>
            <assign name="Assign30">
                <copy>
                    <from>'Context_BillService.xml'</from>
                    <to>$S4_PostValidation_In.parameters/datafile</to>
                </copy>
                <copy>
                    <from>'config_BillService.xml'</from>
                    <to>$S4_PostValidation_In.parameters/conf</to>
                </copy>
            </assign>
            <invoke name="Invoke_S4_PostVal" partnerLink="PartnerLink_Validation"
            operation="validation" xmlns:tns="http://ProType1/"
            portType="tns:WebService_Validation" inputVariable="S4_PostValidation_In"
            outputVariable="S4_PostValidation_Out"/>
            <if name="If8">
                <condition>$S4_PostValidation_Out.parameters/return =
                'Pre condition: rules are valid'</condition>
                <assign name="Assign31">
                    <copy>
                        <from>concat($PrePostTrue, 'S4 : ')</from>
                        <to variable="PrePostTrue"/>
                    </copy>
                </assign>
                <else>
                    <assign name="Assign32">
                        <copy>
                            <from>concat($PostFail, 'S4 : ')</from>
                            <to variable="PostFail"/>
                        </copy>
                    </assign>
                </else>
            </if>
        </sequence>
        <else>
            <assign name="Assign29">
                <copy>
                    <from>concat($PreFail, 'S4 : ')</from>
                    <to variable="PreFail"/>
                </copy>
            </assign>
        </else>
    </if>
    <assign name="Assign33">
        <copy>
            <from>'Context_BillService.xml'</from>
            <to>$S5_PreValidation_In.parameters/datafile</to>
        </copy>
        <copy>
            <from>'config_BillService.xml'</from>
            <to>$S5_PreValidation_In.parameters/conf</to>
        </copy>
    </assign>
    <invoke name="Invoke_S5_PreVal" partnerLink="PartnerLink_Validation"
    operation="validation" xmlns:tns="http://ProType1/"
```

```
portType="tns:WebService_Validation"
inputVariable="S5_PreValidation_In" outputVariable="S5_PreValidation_Out"/>
<if name="If9">
    <condition>$S5_PreValidation_Out.parameters/return =
    'Pre condition: rules are valid'</condition>
    <sequence name="Sequence5">
        <assign name="Assign11">
            <copy>
                <from>'LK'</from>
                    <to>$CountryCurrency_In.parameters/ns3:sCountryISOCode</to>
            </copy>
        </assign>
        <invoke name="Invoke_CountryInfo" partnerLink="PartnerLink_CountryInfo"
        operation="CountryCurrency" portType="ns3:CountryInfoServiceSoapType"
        inputVariable="CountryCurrency_In" outputVariable="CountryCurrency_Out"/>
        <assign name="Assign12">
            <copy>
                    <from>concat($P4_RequestReplyOperation_Out.part1,
                     $CountryCurrency_Out.parameters/ns3:CountryCurrencyResult)</from>
                        <to variable="P4_RequestReplyOperation_Out" part="part1"/>
                </copy>
        </assign>
        <assign name="Assign35">
            <copy>
                <from>'Context_BillService.xml'</from>
                <to>$S5_PostValidation_In.parameters/datafile</to>
            </copy>
            <copy>
                <from>'config_BillService.xml'</from>
                <to>$S5_PostValidation_In.parameters/conf</to>
            </copy>
        </assign>
        <invoke name="Invoke_PostVal_S5" partnerLink="PartnerLink_Validation"
        operation="validation" xmlns:tns="http://ProType1/" portType=
        "tns:WebService_Validation" inputVariable="S5_PostValidation_In"
        outputVariable="S5_PostValidation_Out"/>
        <if name="If10">
            <condition>$S5_PostValidation_Out.parameters/return =
            'Pre condition: rules are valid'</condition>
            <assign name="Assign36">
                <copy>
                    <from>concat($PrePostTrue, 'S5 : ')</from>
                    <to variable="PrePostTrue"/>
                </copy>
            </assign>
            <else>
                <assign name="Assign37">
                    <copy>
                        <from>concat($PostFail, 'S5 : ')</from>
                        <to variable="PostFail"/>
                    </copy>
                </assign>
            </else>
        </if>
    </sequence>
    <else>
        <assign name="Assign34">
            <copy>
                <from>concat($PreFail, 'S5 : ')</from>
                <to variable="PreFail"/>
            </copy>
        </assign>
    </else>
</if>
```

```
        <assign name="Assign40">
            <copy>
                <from>sxxf:current-time()</from>
                <to variable="out_Time"/>
            </copy>
        </assign>
        <assign name="Assign4">
            <copy>
                <from>concat($PrePostTrue, $PostFail, $PreFail,
                $P4_RequestReplyOperation_Out.part1,':In Time=', $in_Time,
                ':Out Time=', $out_Time)</from>
                <to variable="P4_RequestReplyOperation_Out" part="part1"/>
            </copy>
        </assign>
        <assign name="Assign39">
            <copy>
                <from variable="out_Time"/>
                <to>$Operation_ProcessExecutionTime_In.parameters/out_Time</to>
            </copy>
            <copy>
                <from variable="in_Time"/>
                <to>$Operation_ProcessExecutionTime_In.parameters/in_Time</to>
            </copy>
            <copy>
                <from variable="PrePostTrue"/>
                <to>$Operation_ProcessExecutionTime_In.parameters/results</to>
            </copy>
        </assign>
        <invoke name="Invoke_ProcessExecutionTime" partnerLink="PartnerLink_results"
        operation="operation_ProcessExecutionTime" xmlns:tns="http://ProType1/"
        portType="tns:WebService_ProcessExecutionTimeInMilliseconds"
        inputVariable="Operation_ProcessExecutionTime_In"
        outputVariable="Operation_ProcessExecutionTime_Out"/>
        <assign name="Assign38">
            <copy>
                <from>concat($P4_RequestReplyOperation_Out.part1, ' :
                Process Execution Time in MS = ',
                $Operation_ProcessExecutionTime_Out.parameters/return)</from>
                <to variable="P4_RequestReplyOperation_Out" part="part1"/>
            </copy>
        </assign>
        <reply name="P4_Reply1" partnerLink="PartnerLink1"
        operation="RequestReplyOperation"
        xmlns:tns="http://j2ee.netbeans.org/wsdl/RequestReply"
        portType="tns:RequestReplyPortType"
        variable="P4_RequestReplyOperation_Out"/>
    </sequence>
</process>
```

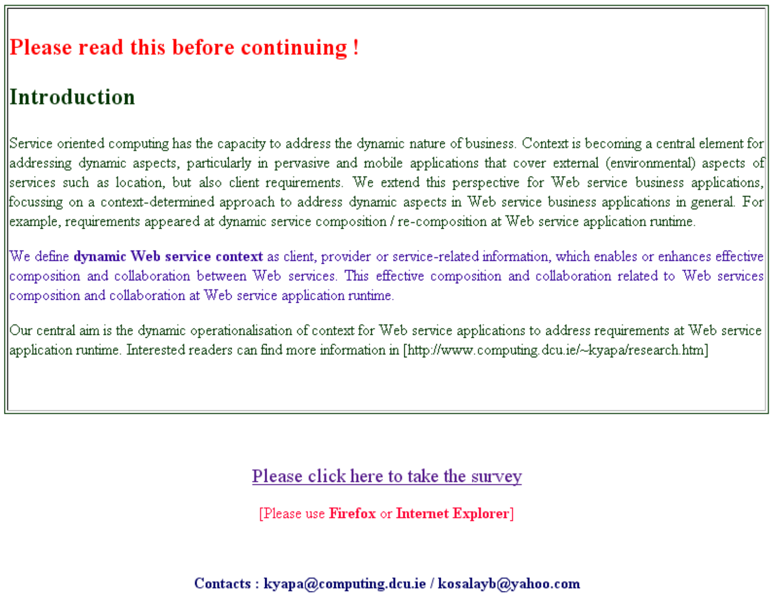
Appendix D

Questionnaire

We made the questionnaire available online, which can be found at

http : / / www.computing.dcu.ie / kyapa / MySurvey.htm.

The first page provides an introduction about our research so that participants can get an high level idea about the survey questions. We believe that will help to get a useful feedback from them. A part of the survey Web interface can be viewed as in figure D.1.



Please read this before continuing !

Introduction

Service oriented computing has the capacity to address the dynamic nature of business. Context is becoming a central element for addressing dynamic aspects, particularly in pervasive and mobile applications that cover external (environmental) aspects of services such as location, but also client requirements. We extend this perspective for Web service business applications, focussing on a context-determined approach to address dynamic aspects in Web service business applications in general. For example, requirements appeared at dynamic service composition / re-composition at Web service application runtime.

We define **dynamic Web service context** as client, provider or service-related information, which enables or enhances effective composition and collaboration between Web services. This effective composition and collaboration related to Web services composition and collaboration at Web service application runtime.

Our central aim is the dynamic operationalisation of context for Web service applications to address requirements at Web service application runtime. Interested readers can find more information in [http://www.computing.dcu.ie/~kyapa/research.htm]

Please click here to take the survey

[Please use **Firefox** or **Internet Explorer**]

Contacts : kyapa@computing.dcu.ie / kosalayb@yahoo.com

Figure D.1: Survey - Web interface

Questionnaire

This questionnaire is designed to determine the opinions of experts about requirements at Web service application runtime. The experts are selected from various organisations, in the service computing domain, around the world - Ireland, UK, USA, Sri Lanka, Iran, India etc. The feedback is used to evaluate the context model, which is one of the contributions in my PhD thesis. As a quid pro quo, I am happy to acknowledge you and your organisation in my thesis.

Name (Optional): Organisation (Optional): Current job role (Optional):
Qualification [BSc. MSc. PhD (Major)] (Optional): Are you happy to be acknowledged? [Yes/No]

Expertise:

[ ] Service based applications
[ ] Distributed systems applications
[ ] Cloud computing applications
[ ] Content-oriented service based application
[ ] Social network applications
[ ] Mobile/Telecommunication applications
[ ] Ubiquitous systems/applications
[ ] Context aware applications
[ ] Semantic Web applications
[ ] Other

Q1.

Services have quality of service properties.

Q1(a). Do you think the following properties of a service can be requirements in a service process at process runtime?

	Strongly Agree	Agree	No Opinion	Disagree	Strongly Disagree
[Performance – the measurement of the time behaviour of services in terms of response time]					
[Reliability – the ability of a service to be executed within the maximum expected time frame]					
[Availability – the probability of a service to be accessible]					
[Cost - the price of a service in a currency type]					
[Reputation - a measure of trustworthiness of a service]					
[Regulatory compliance - a measure of how well a service is aligned with government or organisational regulations]					
[Security - the integrity, authentication, non-repudiation and confidentiality aspects of a service]					
[Trust - refers to the trust relationships between client (user /broker) and provider]					

Q1(b). Can you think of any other quality of service properties, which can be requirements in a service process at process runtime?

Q2.

The ISO/IEC 9126 tries to develop a common understanding of objectives and goals of a project. The quality model of ISO/IEC 9126 classifies software quality in a structured set of characteristics and sub-characteristics. However, our concern is dynamic Web service context in Web service applications, in which some aspects which are not covered in ISO/IEC 9126 deserved to be addressed. We reused some aspects in ISO/IEC 9126, which we found necessary.

We define dynamic Web service context as client, provider or service-related information, which enables or enhances effective composition and collaboration between Web services. This effective composition and collaboration related to Web services composition and collaboration at Web service application runtime.

Q2(a). Do you think the properties defined in the above question (Performance, Reliability, Availability, etc.), properly cover the scope of dynamic Web service context attached to Quality of Service properties (in ISO/IEC 9126)?

Strongly Agree	Agree	No Opinion	Disagree	Strongly Disagree

Q2(b). Do you have any comment?

Q3.

Each application domain may need/have its own requirements for interacting with Web services or service processes.

Q3(a). Do you think the following aspects of an application domain can create requirements in a service process at process runtime?

	Strongly Agree	Agree	No Opinion	Disagree	Strongly Disagree
[Semantics - refer to semantic frameworks in terms of vocabularies, taxonomies and Ontologies]					
[Linguistics - refer to the languages used to express queries, functionalities and responses]					
[Measures and Standards - refer to locally used standards for measurements, currencies etc.]					

Q3(b). Can you think of any other application domain dependent properties, which can create requirements in a service process at process runtime?

Q4.

Platform refers to the technical environment in which a service or process is executed.
Q4(a). Do you think the followings can create requirements in a service process at process runtime?

	Strongly Agree	Agree	No Opinion	Disagree	Strongly Disagree
[Device - refers to the computer/hardware platform in which a service or process is executed]					
[Operating system - refers to the software platform of a device/hardware in which a service or process is executed]					
[Connectivity - refers to the network infrastructure used by the service to communicate]					

Q4(b). Can you find any other platform dependent properties, which can create requirements in a service process at process runtime?

Q5.

Functional properties describe operational features of services.
Q5(a). Do you think the following aspects attached to a service can create requirements in a service process at process runtime?

	Strongly Agree	Agree	No Opinion	Disagree	Strongly Disagree
[Semantics relevant to Input / Output parameters of a service]					
[Pre-conditions/Post-conditions of a service]					
[Protocol information - refers to invoking pre-conditions/Post-conditions in a particular order relevant to a Web service]					

Q5(b). Can you find any other functional properties, which can create requirements in a service process at process runtime?

Q6.

Context notion is widely used in pervasive computing and mobile computing applications to define dynamic aspects, such as temporal and locative aspects. We extended this concept and defined dynamic service context as client, provider or service-related information, which enables or enhances effective composition and collaboration between them. This effective composition and collaboration attached to service composition and collaboration at process runtime. The dynamic service context was modelled in a context model ontology, which facilitates shared conceptualization and reasoning of dynamic aspects attached to service processes. One concern is defining requirements, which emerge at process runtime as context constraints, focusing requirements validation at process runtime.

Q6(a). Do you think the definition of dynamic service context from the perspective of dynamic aspects of service processes is informative enough?

Strongly Agree	Agree	No Opinion	Disagree	Strongly Disagree

Q6(b). Do you have any comment?
