

A POLICY BASED GOVERNANCE FRAMEWORK FOR CLOUD SERVICE PROCESS ARCHITECTURES

MingXue Wang

BSc Computing Science

MSc Software Engineering

A Dissertation submitted in fulfilment of the
requirements for the award of
Doctor of Philosophy (Ph.D.)

to the



Dublin City University

Faculty of Engineering and Computing

Supervisor: Dr. Claus Pahl

November, 2011

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 No.: 56210716

Date: 17 Nov 2011

Contents

Table of contents	ii
Abstract	viii
Acknowledgements	ix
List of Figures	x
Acronyms	xi
I Foundations	1
1 Introduction	2
1.1 Overview	2
1.2 Research issues and contributions	5
1.3 Thesis outline	7
2 Background and related work	10
2.1 Introduction	10
2.2 Service-Oriented Architecture	10
2.2.1 SOA elements	11
2.2.2 SOA style	11
2.2.3 SOA reference architecture	14
2.2.4 SOA specification and architectural framework	17
2.2.5 Other related work	18
2.2.6 Discussion	19
2.3 Service based business processes	20
2.3.1 Service composition	20
2.3.2 WS-BPEL	23
2.3.3 Process adaptation	26
2.3.4 Business processes in cloud computing	28
2.3.5 Discussion	31
2.4 Policy based service computing	32

2.4.1	Autonomic computing	32
2.4.2	SOA governance	33
2.4.3	Policy modelling and approaches	34
2.4.4	Other related work	41
2.4.5	SPL and Variability descriptor	42
2.4.6	Discussion	42
2.5	Transaction and coordination	45
2.5.1	Coordination	45
2.5.2	Business transaction	46
2.5.3	OASIS WS-TX specifications	47
2.5.4	Other related work	49
2.5.5	Discussion	50
2.6	AOP and service computing	51
2.6.1	Aspect-Oriented Programming	51
2.6.2	Related work	53
2.6.3	Discussion	55
2.7	Conclusion	55

II Designing an architectural style 56

3 Problem statement as an architecture problem 57

3.1	Introduction	57
3.2	A purchase order checkout business process scenario	58
3.3	Process as a service	61
3.3.1	The need from process consumers	61
3.3.2	The need from process providers	63
3.4	Business process governance	64
3.4.1	SOA governance for business process	64
3.4.2	Business process delivery in cloud	64
3.5	A new architectural style and framework	66
3.5.1	The need of a new architectural style	66
3.5.2	The need of new architecture framework	67
3.6	Conclusion	68

4 Service Process Architecture style 69

4.1	Introduction	69
4.2	SPA basic concepts and elements	70
4.3	SPA principle	72
4.3.1	Process governability	73
4.3.1.1	Governability explained	73
4.3.1.2	Profiling the principle	74
4.3.1.3	Measure of governability	74
4.3.1.4	Type of process governability	76

4.3.2	Governability and process design	77
4.3.2.1	Process design and development	77
4.3.2.2	Governability with impact on SOA principles	78
4.4	Roles and activities for business process automation	81
4.5	Case study	81
4.6	Discussion of related work	85
4.7	Conclusion	86

III Designing an architecture framework 89

5 Policy model 90

5.1	Introduction	90
5.2	The information model	92
5.3	The language model	93
5.3.1	Rule categorisation	94
5.3.2	Rule	97
5.3.2.1	Objects	98
5.3.2.2	ActivityStates	101
5.3.2.3	Conditions	102
5.3.2.4	Actions	103
5.3.2.5	FaultHandler	108
5.3.2.6	Obligations	109
5.3.3	Policy	109
5.3.4	PolicySet	110
5.4	Related algorithms	111
5.4.1	Semantic matching algorithm	111
5.4.2	Sequencing algorithm	111
5.4.3	Policy combining algorithms	112
5.4.3.1	Constraint combining algorithm	113
5.4.3.2	Remedy combining algorithm	117
5.5	Case study	121
5.5.1	Objective	121
5.5.2	Approach	121
5.5.2.1	Case 1: configuration on service references of activities	121
5.5.2.2	Case 2: configuration on flow logic and resource message	125
5.5.2.3	Case 3: protection aspect	128
5.5.2.4	Case 4: optimization and healing aspect	129
5.6	Discussion with related work	132
5.6.1	The primary requirements	133
5.6.2	The language model complexity	135
5.6.3	The fault handling ability	136
5.7	Conclusion	137

6	Coordination	139
6.1	Introduction	139
6.2	Coordination model	140
6.2.1	The model	140
6.2.2	Coordination context	142
6.2.2.1	CoordinationContext	143
6.2.2.2	Cache	144
6.2.2.3	Example	145
6.2.3	Coordination cache mechanism	145
6.3	Process activity protocol	149
6.3.1	Protocol message schema	150
6.3.2	FSM of protocol	151
6.3.2.1	FSM of <i>COOR^c</i> and <i>COOR^p</i>	152
6.3.2.2	FSM of protocol design for runtime governance	157
6.3.3	Cache of process activity protocol	168
6.4	Coordination implementation with BPEL	169
6.4.0.1	The wrapper service template	171
6.4.1	The process template	173
6.5	Case study	176
6.5.1	The effectiveness on the coordination framework	176
6.5.1.1	Objective	176
6.5.1.2	Approach	176
6.5.1.3	Result and discussion	180
6.5.2	The performance overhead on coordination framework	181
6.5.2.1	Objective	181
6.5.2.2	Approach	182
6.5.2.3	Result and discussion	187
6.6	Discussion with related work	188
6.7	Conclusion	191
7	AOP Enhanced policy framework	192
7.1	Introduction	192
7.2	Policy AOP motivation and concept	194
7.2.1	AOP motivation and capabilities	194
7.2.2	Policy aspect model	196
7.3	Policy aspect specification	198
7.3.1	Join point model	198
7.3.2	Pointcut language	199
7.3.3	Advice Specification	201
7.3.3.1	Advice type	201
7.3.3.2	Advice language	202
7.3.3.3	Advice template	203
7.3.3.4	Aspect and lifecycle	204
7.3.3.5	Fault handling	205

7.4	Aspect deployment and weaving	206
7.4.1	Aspect deployment	207
7.4.2	Aspect weaving	210
7.5	Case study	212
7.5.1	Objective	212
7.5.2	Approach	212
7.5.2.1	Case 1: extension with high level policy for time windows	212
7.5.2.2	Case 2: extend with Jess rule for rule reasoning	215
7.5.3	Result and Discussion	217
7.5.4	Comparison with related work	218
7.6	Conclusion	222
8	Conclusion	223
8.1	Work summary	224
8.2	Future work	225
	Bibliography	228
	Appendix	242

Abstract

In today's environment, the day to day business operations of organisations heavily rely on the automated business processes from enterprise IT infrastructures. The dynamic business environment and the problems with the long time implementation, high cost, etc. of process development and maintenance, are pushing organisations as process consumers to look for ready to use and shared business processes from IT providers for on demand requirements. This is manifest in the rising of Cloud Computing and Business Process Outsourcing with the development of the new concept of (business) Process as a Service.

Service-Oriented Architecture (SOA) is an architectural style commonly adopted for enterprise IT infrastructures and the implementation of service based business processes. However, the SOA style and current specifications do not intend for the case of business processes sharing with cross organisational consumers, since various requirements or the business policies of different organisations, are unmanageable to meet on a business process at the same time.

In this thesis, we present an architectural solution to address the above issues for the Process as a Service. It consists of a Service Process Architecture (SPA) architectural style designed to extend the SOA style, and a supported architecture framework designed for the specific style. The proposed SPA style has a defined principle for the goal of process customizability and adaptability on process design and development with providers. The supported architecture framework consists of three main parts: a policy specification entails expressing business policies or the requirements of consumers regarding business processes in the cloud; a coordination framework aim to enforce expressed policies on process executions with adaptive business processes for different consumers; finally, an AOP enhanced extension is responsible for the extensibility of the framework to satisfy consumers' possible additional requirements.

Our SPA style could extend the SOA style, and has an impact on the SOA principles. With the supported architecture framework, we provided a complete architectural solution for the Process as a Service.

Acknowledgements

Firstly, I would like to thank my supervisor, Dr. Claus Pahl, for the invaluable time time you have spared for me over the last few years. I have not only gained from your impeccable instruction and helps but, more than that, your attention to detail, professionalism, patience, even with my foolish arguments, and the many other abilities I have tried to learn from you will prove invaluable in my further work and life .

Also, I would like to extend a big thanks to all of my colleagues in DCU with whom I worked or had a good time. You are (in alphabetical order): Aakash, Darren, Declan, Edmond, Haiying, Jie, Kosala, Javed, Maurice, Mark, Michal, Murat, Oisin, Paul, Pooyan, Ronan, Veronica, Yalemisew. It was a great benefit for me to be able to learn from their multi-cultural experiences.

I would like to thank all people who gave comments to my work, including the anonymous paper reviewers, and my transfer report examiners (Dr. Markus Helfert and Dr. David Sinclair). Also, the authors of works I have read include that have not been referenced in the thesis, it would have been impossible for me to have completed my work without the comments and knowledge you have contributed to it.

Finally, I am forever indebted to my family members.

List of Figures

1.1	The high level of contribution	5
2.1	SOA operational model of SOA style	11
2.2	S3 or OpenGroup SOA-RA [1]	15
2.3	CCOA diagram [2]	19
2.4	A BPEL example	24
2.5	Business rule approach with BPEL development	38
2.6	A coordinator of WS-Coordination framework [3]	48
4.1	Process governability related to the SOA principles	79
4.2	Application architecture diagram	83
5.1	Information model and framework elements	92
5.2	Core components of the policy language model	94
5.3	Rule categorization related to process execution	95
6.1	The schematic coordination example	142
6.2	$COORD_{context}$: CoordinationContextType	143
6.3	Message flow diagram	152
6.4	Transition graph	155
6.5	Transition graph	156
6.6	Activity life cycle	158
6.7	Pattern graphs	159
6.8	Exception situation description	172
6.9	Activity scope BPEL template	174
6.10	Process scope BPEL template	175
6.11	Screen shot of coordination log	178
7.1	Policy based governance framework stack	195
7.2	Aspect deployment and weaving	207
7.3	Aspect : AspectType	208

Acronyms

ACID	Atomicity, Consistency, Isolation, Durability
AOP	Aspect-Oriented Programming
AOSD	Aspect-Oriented Software Development
BAM	Business Activity Monitoring
BPM	Business Process Management
BPO	Business Process Outsourcing
BPMN	Business Process Modelling Notation
BRMS	Business Rule Management System
BTP	Business Transaction Protocol
CEP	Complex Event Processing
EAI	Enterprise Application Integration
ECA	Event Condition Action
ESB	Enterprise Service Bus
KPI	Key Performance Indicator
LRT	Long Running Transaction
MTBF	Mean Time Between Failures
PCD	Pointcut Designator
QoS	Quality of Service
RA	Reference Architecture
RBAC	Role Based Access Control
REST	Representational State Transfer
SaaS	Software as a Service
SBVR	Semantics of Business Vocabulary and Business Rules
SLA	Service Level Agreement
SCA	Service Component Architecture
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SPL	Software Product Line
UDDI	Universal Description Discovery and Integration
WFM	Workflow Management
WS-AT	Web Services Atomic Transaction
WS-BA	Web Services Business Activity
WS-BPEL/BPEL	Web Services Business Process Execution Language
WS-TX	Web Services Transactions
WSDL	Web Services Description Language
XACML	eXtensible Access Control Markup Language
XPath	W3C XML Path language

Part I

Foundations

Chapter 1

Introduction

1.1 Overview

Business processes are considered as the centre of the business of organisations [4]. A *business process* is a collection of interrelated tasks or activities, which are designed to deliver a particular result or complete a business goal [5]. A business process could be broken down into several sub-processes mapping to activities of the overall process.

Today, business processes of organisations are generally automated or supported with advanced *workflow management* (WFM) or *business process management* (BPM) systems [6]. Business reactions to ever changing market conditions normally need changes to existing business processes, or the development of new processes for current systems. Consequently, this results in substantial IT projects on business process development, which lead to long implementation time, high costs, etc., often inhibiting rapid reactions in a highly dynamic business environment [7].

As a consequence, organisations, from process consumer perspective would benefit from ready to use business processes from IT providers for on demand requirements. On the other hand, from a provider perspective, organisations have developed business processes that can be shared with others to reduce the operational cost or gain profit. This exactly

falls in the concept domain *Software as a Service* under the scope of *Cloud Computing* [8] [9], which is about delivering IT services to clients over the Internet. More specifically, *Process as a Service* in this case. It is increasingly required by the Service outsourcing [10] or Business Process Outsourcing (BPO) business paradigms.

Service-Oriented Architecture (SOA) is a business-centric IT architectural style [11] [12], which aims to use services as basic building blocks to rapidly construct low-cost applications. It reuses developed services, which may come from different service providers when a new business process arises, also as a way of business collaboration between organisations.

Because of various requirements such as monitoring, security, etc, on a micro service level of business processes, or different business requirements or policies of organisations with regard to business processes, business processes generally are scoped and resided within one organisation [9] [13]. Service consumers themselves are concerned with developing and hosting business processes based on services from service providers [1] [13]. Business processes are not available for sharing between cross-organisational service consumers in SOA.

Business policies describe business requirements that are expressed in formal *policy* statements and are focused in the domain of SOA governance in SOA reference architectures [1] [14]. Since it is based on the SOA paradigm above, current work such as WS-Policy [15], in the SOA governance domain only addresses policies with regard to services or task services for cross-organisational consumers. SOA treats policies of business processes as an internal organisational problem. With related work such as [16] [17] [18], processes only comply with single party policies, and are not applicable for sharing processes hosted outside organisations with multi-tenancy capability for cloud computing. *Multi-tenancy* means different tenants or organisations could have isolated and customized behaviours on shared software resources [19] [20], or business processes in this case.

In our conceptualisation, we leverage SOA and with the cloud computing concept, lift

up software artefacts as the building blocks for cross-organisational collaboration in SOA from a micro service level to a process level. The business processes are expected to be available from process providers for sharing with consumers with on demand needs and for self-service. The business policies regarding business processes would be addressed by on-the-fly process customization through runtime governance.

The goal of this PhD work is to design an architectural style that promotes sharing of business processes for cross organisational consumers or tenants in process development, particularly on service processes in SOA styles with a supported architecture framework. The central hypothesis of this thesis is that business processes can be shared for on-demand requests from cross-organisational consumers for self-service. The main research questions derived from the central hypothesis are:

1. Can the SOA architectural style be adapted or extended to accommodate the needs of process sharing for multiple tenants?
2. Can a supporting architecture framework for the style be developed? And what components, protocols, etc., are needed?

This thesis introduces,

1. An architectural style with a defined principle - process governability, to extend the SOA style to guide software engineers in service process and infrastructure design to address the problem of sharing business processes.
2. And an architecture framework that will support engineers implementing applications in the style to demonstrate the feasibility of our concept.

The architecture framework consists of three main parts to address the requirements on business processes as a policy based process runtime governance problem.

1. First, we provide a policy model specification to allow process consumers to express their own business policies or requirements with regard to business processes.

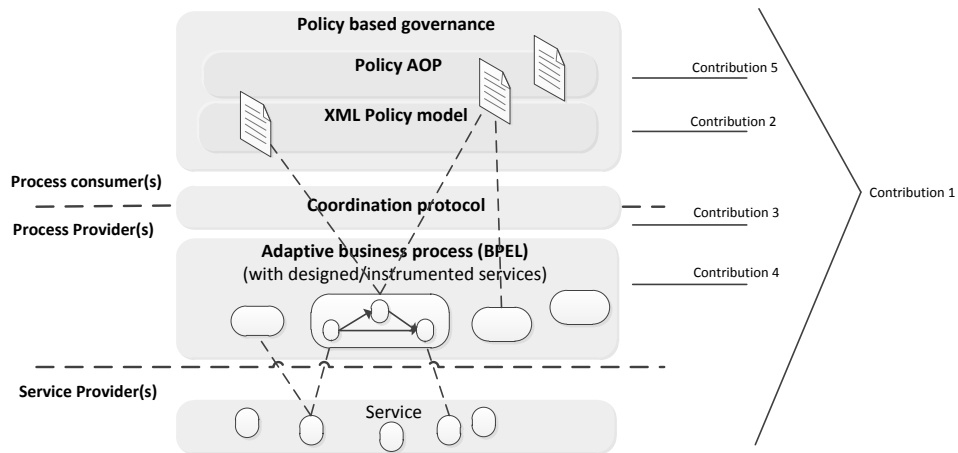


Figure 1.1: The high level of contribution

2. The policies will be enforced on process execution by process providers for consumers through a coordination protocol. The coordination framework implementation will form the second part.
3. The last part, an AOP (Aspect-Oriented Programming) specification is introduced as an extension for advanced requirements of process consumers with regard to process governance.

1.2 Research issues and contributions

In the following subsections, we briefly describe the key research issues and main contributions. Figure 1.1 summarises our contributions to give a larger picture.

- **An architectural style to extend the SOA style for Process as a Service**

The main problem of the SOA style is that the current principles are defined with little consideration for the service customization for multi-tenancy cloud applications. This makes orchestrated task services or business processes in SOA almost impossible to be

shared across organisations if different business policies apply to the processes. Hence, the development of business processes is more of a concern for service consumers than service providers. A solution at an architectural level is needed to address the problem of sharing business processes across organisations. The style would extend the SOA style to offer *(Business) Process as a Service*.

Contribution 1: The architectural style consists of a principle focusing on the adaptation and customization of business processes, and a study of the principle with regard to SOA principles for service design with business processes.

- **A policy model specification for business process governance as customization**

Business processes might be customized to address the business policies of consumers regarding the processes. Policies defined by consumers would act as customization metadata of business processes by means of policy based process governance. A policy specification for process governance is desired, as most current Web service policy specifications, such as [15] [21] [22], neglect the business processes. The issue is how a policy can be facilitated for process consumers on shared business processes.

Contribution 2: A policy specification is defined for formalizing different categories of business policies as customization metadata of business processes of providers.

- **Coordination model and protocol**

Defined policies of processes consumers must be enforced on process execution when consumers consume the processes. For reason such as policy centralization, privacy concerns of cloud consumers [23] [24], governance directly from process consumers needs a coordination protocol as a base for a service contract between providers and consumers to address the governance need.

Contribution 3: A coordination model and protocol serve as the base of the distributed and multi-tenant and process runtime governance.

- **Adaptive BPEL process development**

Since each process consumer might have its own policy requirements in a business process, consumers need to be served by the process at the same time but without interfering with each other. For sake of the process providers with a multi-tenancy requirement, the process should not require redevelopment and redeployment for enforcing policies for each consumer, since the process is shared by multiple consumers. The implementation of such a coordination framework is a challenge.

Contribution 4: A BPEL template is presented for governance enhanced BPEL process development implementing the coordination protocol to handle arbitrary policies of process consumers. The approach is platform independent.

- **An AOP specification for extendible policy model**

An extendible policy model will provide an advantage on extendibility and advanced requirements needed in process governance or customization. Other policy models and frameworks could be adopted and integrated with our framework. Process consumers would have more opportunities to utilize shared processes to fulfil their requirements.

Contribution 5: An AOP enhanced policy framework to address additional features that might be required for process runtime governance by consumers. A policy based AOP specification is defined on top of the policy model for extensibility.

1.3 Thesis outline

Our research falls in the category of design science [25], which addresses research through the building and evaluation of artefacts designed to solve the identified problems. We use one chapter (3) to state the main problem we identified with a scenario through analysis, which also is part of our design. The four following chapters (4, 5, 6, 7) show our design of solution components for different sub-problems of the main problem. An evaluation is

placed at the end of each of four chapters, using a case study based on the described scenario to demonstrate how the sub-problem of each chapter is addressed.

The remainder of the thesis consists of background and related work, two main parts, and a conclusion.

In Chapter 2, we present the necessary background knowledge and related work.

The first main part is designing the architecture style. We describe the design process of the architectural style.

1. In Chapter 3, we present the problem statement of the SOA architectural style. Through a basic scenario, we show that the business processes in current SOA cannot be shared or used by any consumers outside the organisations since a restriction regarding business policies. Current approaches do not fully solve the problem.
2. In Chapter 4, we present an architectural style as an abstract solution to address the problem. One principle - process governability is defined in the architectural style to extend the SOA style. We give a detailed discussion of the principle in particular and also in relation to SOA principles.

The second part is designing the architecture framework. We describe the concrete solution as a framework of the architectural style.

1. In Chapter 5, we present a policy model for formally expressing business policies regarding business processes. The language model and related algorithms on semantic matching, sequencing, and combining will be detailed. A case study will follow at the end of the chapter to demonstrate the policy model covering various business policies.
2. In Chapter 6, we first describe the coordination model and protocol (process activity protocol) in detail. The protocol design will be illustrated in detail. Also, the business

process development for a platform independent coordination framework implementation will be described. Finally, effectiveness and performance of implementation will be evaluated and discussed.

3. In Chapter 7, we firstly explain the motivation scenarios of the need of extensibility of our framework. Then we describe an AOP framework, which is located on top of our policy model. Process consumers can adopt other policy models or frameworks on top of our policy model, which will be illustrated with a case study to demonstrate the extensibility our framework.

The last chapter of the thesis (Chapter 8) contains conclusions and an outlook.

Chapter 2

Background and related work

2.1 Introduction

In this chapter, we give a discussion of related work with the necessary background.

The necessary background knowledge, such as SOA elements, styles, service composition, BPEL, cloud computing, etc., will be described.

We will describe in detail related research work, which has been recognised as standards and influenced our work, such as OpenGroup SOA-RA, XACML, WS-BA. Other related work also will be briefly described.

In following, we describe the background and related work in relation to different domains as sections (Section 2.2 Service-oriented architecture, Section 2.3 Service based business processes, Section 2.4 Policy based service computing, Section 2.5 Transaction and coordination , Section 2.6 AOP and service computing), and will give a short discussion with regard to our research at the end of each domain.

2.2 Service-Oriented Architecture

Software architecture is concerned with software systems development to assure the satisfaction of systems' requirements [26]. As noted by [27], software architecture could be

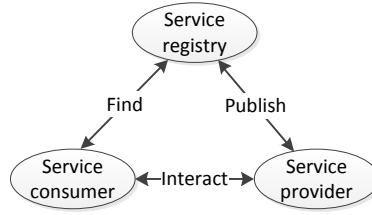


Figure 2.1: SOA operational model of SOA style

defined in different forms, such as elements, styles, etc. Consequently, we describe the SOA architecture with respect to different forms.

2.2.1 SOA elements

A software architecture can be defined by a configuration of *architectural elements* - components, connectors, and data constrained in their relationships in order to achieve a desired set of architectural properties [27].

The SOA triangular operational model (Figure 2.1) [13] or the conceptual model of the SOA architectural style [28] describes the basic SOA elements. The building blocks of SOA are business tasks, or services, which are self-contained, self-describing, and platform-independent computational components. These business services realized as Web services are described and published by service providers, can be discovered and invoked by service consumers through standard Web protocols.

2.2.2 SOA style

An *architectural style* is a coordinated set of design principles, and constraints that dictate how architectural elements can be composed, behave, and communicate [29] [27] [30]. Architectural styles are identified to be used for guiding the design of software systems [31]. The SOA style is defined as a set of flexible principles, which are basic generalizations that are accepted as true and that can be used as a basis for SOA system design. As a consequence, it allows to make a balance between different principles of service design, and

differs from many architectural styles, such as REST [27] which contains a set of constraints that must be satisfied.

SOA is defined as an architectural style, consisting of a set of design principles used for *service-oriented* development for SOA systems. Many SOA vendors have specified different principles, such as [32]. Here, we reference the commonly accepted and widely referenced eight principles from published research by Thomas Erl [33] [11].

1. *Standardized Service Contract* - Services in compliance with the same contract design standards within a service inventory. A service contract can consist of a group of service description documents, which includes technical documents (such as WSDL) and non-technical documents. The goal of a standardized service contract is enabling service interoperability within a service inventory and to increase service interpretability and predictability.
2. *Service Loose Coupling* - Dependencies between the surrounding environment of services themselves and their consumers are only limited to conformance to the service contract. Services are loosely coupled to programming languages, technology implementation, outside software components, etc. reflected on service contract independent from service implementations. The goal of the service loose coupling is enabling service and consumers to be adaptively evolved with minimal impact between each other.

As basic example, the service provider can move the service host machine from Windows to Unix for security reasons, i.e., adaptively evolve the service as long as the host machine is not defined in the contract.

3. *Service Abstraction* - Only essential information is published in service contracts, and is the only information visible to the outside world of service. More information published outside causes consumers-to-contract coupling to become deeper, and affords less space to evolve the service over time. The goal of service abstraction is preventing the publication of unnecessary service information and balancing with

other principles, such as service discovery which emphasizes publishing more service information.

The information can be any information regarding to services, such as which developer developed the service or where the service is hosted.

4. *Service Reusability* - Services are designed with agnostic logic ¹ as a potential reusable enterprise resource. Reusable services have capabilities useful for more than one purpose. It opens the door to increase the ROI. However, requirements of multiple purposes increase complexity, cost, etc. to deliver the service. The goal of service reusability is increasing ROI, rapid fulfilment of future requirements, and more agnostic services.

More than one purpose is necessary. For example an operation *getPerson()* could be used to get a person name, get the age, or get an address. Reusable could mean only to be used more frequently, but not necessary for multiple purposes.

5. *Service Autonomy* - Services have governance over their underlying runtime execution environment, and are not dependent on other services for it to execute its governance. The more independent a service is from unpredictable outside influences, the more reliable it will be. The goal of service autonomy is increasing runtime reliability, performance, predictability and increasing the amount of control over the runtime environment.
6. *Service Statelessness* - A service should be designed to a maximum of statelessness, deferring the management of state information if necessary. A service constantly consuming computation resources for processing and retaining state information could drain the system resources when numerous service instances exist concurrently. The goal of service statelessness is increasing service scalability and supporting agnostic service logic for improving service reuse.

7. *Service Discoverability* - Services are supplemented with description meta data allowing it to be effectively discovered and interpreted by machines or humans. Dis-

¹without need to know the service logic

covery is a process that searches and finds suitable services for given criteria. It is important for making decisions that if solutions exist or need to be developed. The goal of service discoverability is emphasizing and clearly expressing the service purpose and capability, and as a high discoverable enterprise resource.

8. *Service Composability* - Services are effective composition participants, allowing logic to be represented at different levels of granularity. Software composition enables decomposability solution logic to be recomposed into a new configuration for various problems. Service reuse is realized by service effectively and repeatedly composed by others. The goal of service composability is to increase service reusability and allow extensions for future business requirements.

2.2.3 SOA reference architecture

A *Reference Architecture* (RA) is developed to provide a conceptual framework for describing architectures and showing how components are related to each other [27].

SOA-RA extends the fundamental SOA principles, provides a worked design of an enterprise-wide SOA implementation, with detailed architecture diagrams, etc. There are a number of SOA-RAs that have been developed. Here, we describe two SOA-RAs from standardisation bodies (*OpenGroup* and *OASIS*).

OpenGroup SOA-RA

IBM research published a layered SOA-RA named S3 (Figure 2.2) [1], which was adopted as the OpenGroup draft standard for SOA-RA [34]. The S3-RA is a layered architecture from a consumer and provider perspective. There are five horizontal layers that are more functional in nature and relate to the functionality of the SOA solution.

The lower layers (services, service components, and operational systems) are provider concerns,

1. *Operational Systems Layer* - captures the new and existing organization infrastruc-

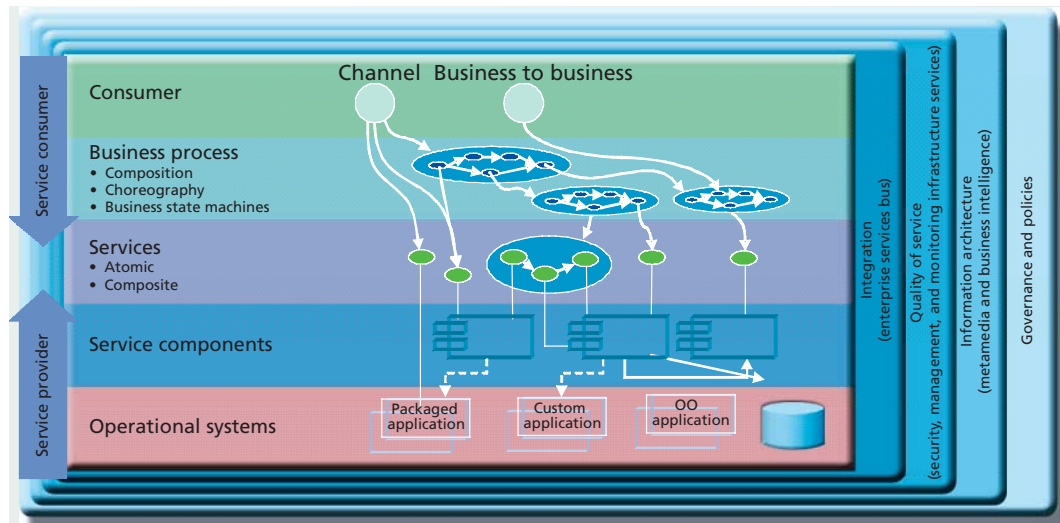


Figure 2.2: S3 or OpenGroup SOA-RA [1]

ture, including those involving actors, needed to support the SOA solution. A number of existing software systems are part of this layer.

2. *Service Components layer* - contains software components, each of which provides the implementation or operation of a service; hence the name Service Component.
3. *Services Layer* - consists of all the services defined within the SOA. The service layer contains the service descriptions (service contracts) and the container for implementing the services.

The upper layers (services, business process, and consumer) are consumer concerns.

4. *Business Process Layer* - In this layer, the organization composes the services exposed in the services layer into composite services as business processes, which provide significant business applicability.
5. *The Consumer Layer* - handles interaction with the user or with other programs in the SOA ecosystem. It provides the capability to quickly create the front end of the business processes and composite applications.

The four vertical layers are *non-functional requirements* (NFRs) in nature and support

various cross-cutting concerns of the architectural building blocks and principles that support the realizations of SOA.

6. *Integration Layer* - is a key enabler for a SOA as it provides the capability to mediate, transform, route and transport service requests from the service consumers to the correct service provider. It also provides the support of a common business rules capability.
7. *Quality of Service Layer* - provides the service SOA solution lifecycle processes with the capabilities required to ensure that the defined policies and NFRs (such as availability) are adhered to.
8. *Information Architecture Layer* - includes stored metadata content. It captures all the common cross industry and industry-specific data structures, and business protocols for exchanging business data, etc.
9. *Governance Layer* - ensures that the services and SOA solutions within an organization adhere to the defined policies, guidelines and standards that are defined as a function of the objectives, strategies and regulations applied in the organization.

OASIS SOA-RA

OASIS documents the SOA-RA [14] from a service ecosystem perspective rather as a complex system. A service ecosystem is a space which people, machines and services inhabit in order to further both their own objectives and the objectives of the larger community. It describes architecture in terms of models, views, and viewpoints.

A *View* is a representation of the whole system from the perspective of a related set of concerns. The reference architecture has three main views:

1. *Service Ecosystem view* which focuses on the way that participants are part of a SOA ecosystem;
2. *Realizing Services view* which addresses the requirements for constructing a Service Oriented Architecture;

3. *Owning SOA view* which focuses on the governance and management of SOA-based systems.

A *Viewpoint* is a specification of the conventions for constructing and using a view.

1. *The Service Ecosystem viewpoint* is intended to capture what using a SOA-based system means for people using it to conduct their business.
2. *The Realizing SOAs Viewpoint* focuses on the infrastructure elements that are needed to support the construction of SOA-based systems.
3. *The Owning SOAs Viewpoint* addresses the issues involved in owning a SOA as opposed to using one or building one.

A *Model* is an abstraction or representation of some aspect of a system. Each architectural model is developed using the methods established by its associated architectural viewpoint. UML class diagram is used to represent a visual model depiction in the document.

2.2.4 SOA specification and architectural framework

SOA is only an architectural style itself that does not specify or provide any methodology and framework to create services. An *Architecture Framework* is a software that helps application developers to correctly implement applications in a particular or a family of architectural styles [29].

Heterogeneous SOA frameworks developed or chosen from different vendors by organisations would cause interoperability problems between different organisations without common standards. Thus, there are a set of common Web service specifications (such as SOAP, WSDL.) that have been established by standardization bodies (such as W3C, OASIS.) to form open standards. Different frameworks (such as Apache Axis2², Apache CFX³.) are developed based on these standards by SOA vendors or communities.

²Apache Axis2, available at <http://axis.apache.org/axis2/java/core/>

³Apache CFX, available at <http://cxf.apache.org/>

The service is referred to as the W3C SOAP Web service [35] specification in general, also in our work. Other technical specifications on service developments for the SOA style, such as CORBA, will be not discussed in our work.

There is a large amount of Web service specifications which cover various features needed for enterprise SOA systems, such as security, reliable messaging, and transactions. Specifications may complement, overlap, and compete with each other. For example, WS-Security, WS-SecureConversation, XACML, are all in the security domain for Web services systems.

A framework might only focus or cover a certain number of specifications, but could be integrated as a module of large frameworks or SOA solutions. For example, the Apache Axis2 Web service engine (with most current version 1.54) implements such as SOAP, WSDL, WS-Security specifications, but not the WS-Coordination, WS-Discovery specifications. Our architectural framework is also only implementing the specifications or the problems we addressed in this thesis.

2.2.5 Other related work

The *Service Component Architecture* (SCA) [36] provides a set of specifications which describe a model for composing applications that follow SOA principles. It is developed by IBM, Oracle and others, and submitted to OASIS. Apache Tuscany⁴ is an example of a supporting architecture framework. Service components in business applications is the main concern of SCA, such as what components are; how they connected and communicated; and policies applied to each of the components. In contrast, BPEL is concerned with business logic and tasks of business processes. SCA could work together with BPEL as an extension of SOA [37], but SCA and SCA variants, such as [38] [39], do not address our problem.

IBM research proposed a Cloud Computing Open Architecture (CCOA) [2] by formally adopting the SOA and virtualization technologies. It consists of seven architectural princi-

⁴Apache Tuscany, available at: <http://tuscany.apache.org/>

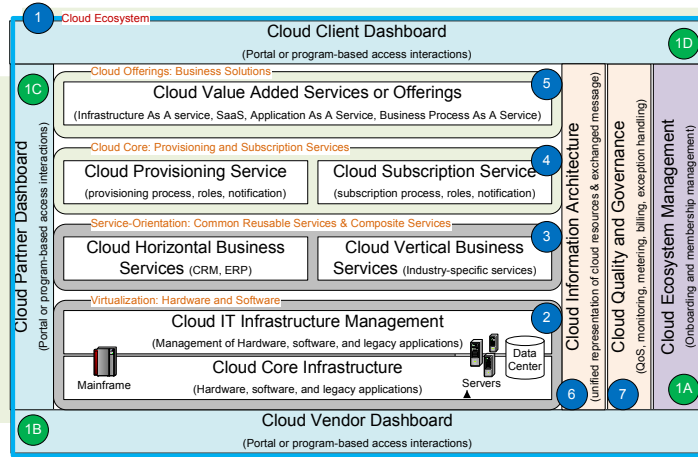


Figure 2.3: CCOA diagram [2]

ples and derives ten interconnected architectural modules (Figure 2.3), which are derived from the layered S3 or OpenGroup SOA-RA. The CCOA proposes an integrated collaboration framework for cloud vendors and consumers to work together based on seven principles.

[40] [41] specified an Enterprise Cloud Service Architectural style (ECSA) by merging the SOA style with the cloud computing concept. ECSA specifies the vocabulary of ECSA architectural elements and constraints of the elements and their relationships. The ECSA architectural elements are modelled by a 7-tuple: service, consumer, data element, infrastructure, management, process, and quality attributes. ECSA constraints focus on quality constraints such as performance, transaction, which the architecture must satisfy during the design.

2.2.6 Discussion

For consumers outside of organisations of service providers, there are different perceptions regarding services and business processes, as they sit in different layers defined in OpenGroup SOA-RA [1] [34], or have different viewpoints defined in the OASIS SOA-RA [14]. Service consumers share services, but not business processes in the SOA architecture.

Business processes are concerns of consumers themselves. How can business processes be shared for consumers outside of provider organisations? The SOA style, currently defined RAs which extend the principles of the SOA style, and SCA specifications leave this aspect and protocol details undefined, inhibiting the sharing of business processes. The cloud architectures, such as CCOA, give high level guidelines on a cloud infrastructure, but without detailing the problem of sharing business processes with SOA.

Software architectures use a number of commonly-recognized 'styles' to guide their design of system structures [26]. Each of these is appropriate for some classes of problems, but none is suitable for all problems. We address our problem as a need of an architectural style. The style defines an additional principle to extend the SOA style for service process design as a solution of the above problem. More discussion about the problem and style will be presented in a later chapter.

2.3 Service based business processes

Business processes are defined by service composition in SOA. In following, we describe the background of service composition and WS-BPEL, and also related work for adaptive processes and process delivery in the cloud.

2.3.1 Service composition

The service composition is a key concept in SOA. It realizes the business process by combining individual business services in composite services. It also realizes business collaboration through composite services from different business partners.

Service Orchestration and Choreography Composite Web services for creating business processes can be described in two perspectives [42] [43]:

1. *Orchestration* represents control from one party's perspective. The process may use

both internal and external Web services. The process is described in term of message exchange and execution order.

2. *Choreography* tracks the message sequences among multiple parties and sources rather than single party execution. The process is described as interaction between multiple parties involved in the process.

The primary difference between orchestration and choreography is execution and control within a single party and multiple parties, i.e., if the business process is described with a centralized execution and control. A choreography can be implemented as an orchestration for each party involved in it [42]. The BPEL is an orchestration language. Service composition for business processes in our work is addressed only from the orchestration perspective.

Composition implementation types There are three types of implementation approaches for service composition [44].

1. Programmatic implementation of composite service - It uses a general purpose programming language to composite services. WS-CAF [45] is a framework example for supporting such implementation. It is a straightforward approach, but suffers from many drawbacks. The hard-coded orchestration is very inflexible to maintain and change. Implementation could be complex for aspects such as service conversational requirements and supporting service context.
2. Service interaction through Publish/Subscribe - A Pub/Sub engine is an intermediary between service consumers and providers. An event sent by a service or service consumer will be delivered through the pub/sub engine to a set of services that have subscribed to this event. A service might also send an event for handling a received event. This sequence of events effectively creates a composite service. ESB (Enterprise Service Bus) products are examples.

3. Service composition through an orchestration engine - In this case, an orchestration language such as BPEL is used instead of a general programming language for service composition implementation. Visual designers are available for building composition logic with orchestration languages, and deployment in an orchestration engine.

The orchestration engines provide built-in capabilities for asynchronous invocations, compensation support, etc. for service process development. Our work is focused on this type of composition, i.e., BPEL service processes.

Composition method types Depending on human involvement in composition processes, there are three types of composition methods:

1. Manual composition - Users need to program the composition logic. Desired business processes or goals are translated to programs manually. Various tools or frameworks from SOA vendors could be used to support this manual implementation, WS-CAF and BPEL designer are such examples.
2. Automatic composition - In this case, processes are produced by the machines with AI technology. The composition process could be regarded as an AI planning problem, which is solved by situation calculus [46] or hierarchical task network [47] planners. The planner will find a plan as a process containing a set of actions that, when performed starting in the initial state, will terminate in a goal state.
3. Semi-automatic composition - Instead of giving a complete process by an automatic approach, semi-automatic composition requires or asks human users' decisions in the middle steps of compositions. For example [48], available service choices are automatically discovered by semantic annotated Web services and presented to the user at each step by the composer when a user creates a process.

Our work focuses on manual composition, the problem that might occur in automatic and semi-automatic composition will not be discussed.

Composition binding types There are two types of service processes that could result from composition processes.

1. *Abstract service process* - It contains the interaction protocol between services which are composed, but without covering the concrete service details which are needed for execution.
2. *Concrete service process* - In contrast to the abstract service process, it covers service details. A concrete service process can be directly executed.

A concrete service process can be viewed as instantiation of an abstract service process. It requires binding concrete services for every service in an abstract process. And it results two types of compositions depends on the binding approaches.

1. *Static binding composition* - The composition process is with concrete services to provide a concrete service process directly.
2. *Dynamic binding composition* - The composition process is with abstract services to have an abstract process first, and then bind concrete services based on additional requirements.

The dynamic service composition has the potential to realize flexible and adaptable applications by selecting the concrete services based on the user request, especially in the context of QoS requirements. [49] [50] [51] are works focussed on services selection with QoS constraints on business processes.

2.3.2 WS-BPEL

BPEL is an industry standard language for expressing business processes with Web services [52] [53]. It has rich and comprehensive semantics to address the complex requirements for service solutions. BPEL has strong roots in traditional workflow models [54], plus

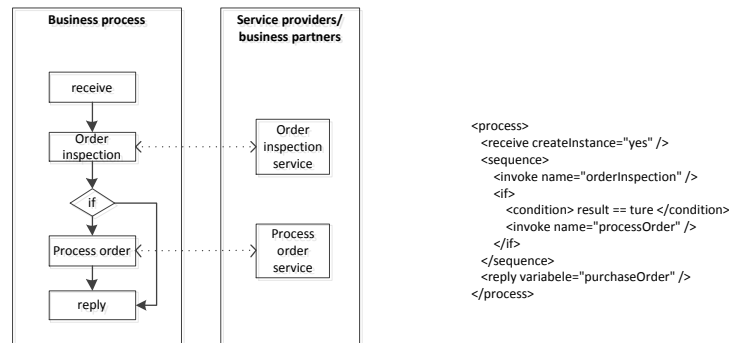


Figure 2.4: A BPEL example

many concepts from structured programming languages. It is developed based on several composition languages from different SOA vendors (IBM, etc.), and finally became an OASIS standard. The latest BPEL version 2.0 specification is published by OASIS in 2007 [52]. BPEL is a type of XML language. Figure 2.4 shows a simplified process diagram and code.

The BPEL specification includes the following key concepts:

Variables: BPEL variables are XML elements declared within processes that store messages and hold state information of BPEL business processes during runtime. The Name of a variable has to be unique in its own scope. Messages stored in variables could come from and are sent to business partners. For example, sending a purchase order information to a shipping company, then getting a message back with shipping free.

Activities: BPEL has two types of activities: basic and structured. Structured activities can contain other activities and define the business logic between them. In contrast, basic activities do not include other activities. The important basic activities are `<receive>`, `<invoke>`, `<reply>`. They are used for message exchange between business partners (Web services). Other basic activities include such as `<waiting>` (waiting an amount of time or deadline), `<empty>` (No-op instruction), `<throw>` (signalling faults). Structured activities are such as `<sequence>` (sequential execution), `<if>` and `<switch>` (specify conditional behaviours), `<scope>` (split the process up into several parts).

Message correlation: When a set of process instances of the same business process run concurrently, it's important to make sure that each process instance always exchanges messages with the right instance of a service. BPEL addresses this problem by making use of a message correlation mechanism. Key variables of message exchanged, such as purchaseOrderId, buyerId, between a business process and outside world, are marked as correlation variables in the service interface and the BPEL process to uniquely identify a process instance. When the process is invoked, these variables make sure messages are processed in the right process instance.

Fault handling: BPEL offers fault handlers <faultHandlers> that can be attached to a scope, define a set of fault handling activities. A <faultHandlers> can have a set of <catch> constructs to catch specific faults and maximally one <catchAll> construct to catch any unspecific fault. Any unhandled fault in a process scope will be thrown to the parent scope.

Compensation handling: There is no guarantee that every process instance can reach the end point to complete the business transactions. There is a need to roll back activities which were successfully completed at the point where the problem occurs. For example, a payment needs to be refunded to a buyer, if a seller cannot send a receipt. The BPEL compensation handler is able to define a set of activities that have to be executed to rollback a process scope. A <compensate> activity causes all immediately enclosed scopes to be compensated and <compensateScope> activities to compensate specific immediately enclosed scopes in a Default Compensation Order. The default order follows two rules: If a successful scope B has a control dependency on successful scope A, the compensation on scope A will start after the completion of the compensation of scope B; scopes A and B cannot have cycles in a peer-scope dependency relation.

Event handling: An event handler can specify what to do when certain events happen. There are two types of events: <onMessage> and <onAlarm> event. Message events point to Web service send and receive messages. The alarm event has a specified point in

time or a time interval.

2.3.3 Process adaptation

An adaptive system means the system can be changed for user requirements or fulfil the same requirement in changing environments [55]. Process adaptation is closely related to policy systems (which we will detail in the next subsection), as changing of processes is normally realized by enforcing policies. More specifically, business process adaptation is needed for the following reasons in [56] [16]:

1. Configuration - to add/remove/replace activities specific to business processes [57] [58].
2. Correction - to handle faults or exceptions occurred during the process execution [59] [17] [60].
3. Optimization - to improve extra-functional (usually performance) issues noticed during execution, and it might be addressed through the correction [61] [62] [60].
4. Prevention - to prevent future faults or extra-functional issues before they occur [62] [17].

[57] utilizes the event handling ability available in BPEL `<eventHandlers>` and provides adaptation by performing predefined actions if certain events occur. They introduce a new namespace-qualified element `<cc ns:alt activity>` to extend BPEL. The element allows specifying a choice of actions that could be performed on receiving an event. Similarly, in [63] a BPEL *onMessage*-clause is added into BPEL *eventHandlers* to catch exceptions, and generate process exception events.

[59] [64] proposes a Service Relevance and Replacement Framework (SRRF). SRRF modules include a SRRF pre-processor, which analyses its input BPEL scenario to identify invocations of web services and arranges for complementing invocations. A pre-processor

will create a SRRF-aware BPEL scenario. Firstly, the pre-processor adds the appropriate declaration of *partnerLinkType* in BPEL for the new Alternate WS Service Locator binding within the *partnerLinks* construct. Then, the pre-processor uses a scope construct to provide the appropriate fault handlers for each service invocation within the BPEL scenario. Scopes are employed to enable the definition of different fault handlers for different activities. When a system fault occurs, the handlers generated by the SRRF pre-processor invoke the alternate WS locator module to retrieve a list of services which can replace the failed one.

The Dynamo project [17] developed a supervision framework for the ActiveBPEL engine. The framework provides a *Callback(eventHandler, input)* operation that allows direct access to the internal state of the process. This action allows complex logic, embedded in the process by means of an event handler (<eventHandlers>). The event handler executes in an independent thread with respect to the main business process, which meanwhile continues to remain synchronously blocked by the supervision framework. Once the event handler thread completes, the supervision framework is warned to unblock the main business process.

[65] offers a TWSO (Transactional Web Service Orchestrations) framework that addresses process transactions. The framework provides TWSOL (Transactional Web Service Orchestration Language) as an extension for orchestration languages. It can be used to describe transaction logic in addition to BPEL processes. The defined TWSOL is bound to orchestration languages by utilizing a built-in extension mechanism. A transaction monitor is attached with the BPEL engine for monitoring the transaction states of process execution and handles the TWSOL language as well.

[60] [51] defines a flexible process as one that can change its behaviour dynamically according to variable execution contexts, such as QoS constraints. They offer a PAWS (Processes with Adaptive Web Services), a framework for flexible and adaptive execution of managed service-based processes. The ActiveBPEL engine is extended for managing

adaptive actions. There are also a number of similar frameworks such as [66] [67] [68] [56] that extend a BPEL engine for process adaptation.

2.3.4 Business processes in cloud computing

Cloud computing is a newly emerging trend in the IT industry about delivering hosted IT services (hardware and software resources) to clients over the Internet. One great advantage is that the clients can purchase IT services on demand in real time with a pay-per-use model, and without having to worry about hardware and software hosting. There are challenges and opportunities by combining the two computing paradigms: service-oriented computing and cloud computing [9].

Cloud service categories

The services provided by the cloud can be broadly divided into three categories [69]: *Infrastructure as a Service (IaaS)*, *Platform as a Service (PaaS)* and *Software as a Service (SaaS)*. With regard to BPEL processes:

1. IaaS offers basic infrastructure to clients, such as a hardware environment for a BPEL engine. Common cloud services, such Amazon EC2⁵, GoGrid⁶, etc., are in this category.
2. PaaS offers a deployment platform, such as a BPEL engine for business processes of clients. Sample work [70] introduces a BPEL engine compliance interface that allows enterprises to gather process evidence from a BPEL engine as well as enforcing rules on the process. The compliance interface includes signalling, runtime monitoring, enforcement, and assessment services. The Apache ODE BPEL engine is extended for the implementation.

⁵Amazon Elastic Compute Cloud (Amazon EC2), <http://aws.amazon.com/ec2/>

⁶GoGrid cloud infrastructure service, <http://www.gogrid.com/>

3. SaaS offers a software product to clients, such as a payment business process. It can be a front-end ready Web based application, or a Web service as a software component, etc. In this category, the Cafe project [71] [72] [73] [74] [7] is an example, which proposes a composite application based BPEL delivery, where the BPEL process is wrapped in an end user based application. QoS configuration in its case is made available by process variability descriptors, where application clients or process consumers can customize the application according to their needs.

Multi-tenancy capability

In the SaaS model, software applications might require some degree of isolation for different customers. This is discussed in terms of multi-tenancy [20] in software engineering.

The essence of multi-tenancy in a software system is about sharing and isolate resources between different tenants, or application users. Multi-tenancy applications could have different level of sharing. For example, a data architecture [75]: it could have shared schemas, separated schemas, or event separated databases for each tenant. [20] defines multi-tenancy applications in four maturity levels. Higher numbers indicate higher levels of resource sharing. Table 2.1 describes the SaaS maturity level with regard to business processes.

Multi-tenancy capability in SaaS architecture generally means the level 4 maturity. It gives great benefits to SaaS vendors [76] [20] [77]. For example, supporting more tenants on fewer hardware components; quicker and simpler on application updates, etc. These benefits will trickle down to the tenants, in the form of lower service fees, quicker access to new functionality, etc. Many SaaS vendors have pointed out that multi-tenancy is a requirement of any SaaS system [76].

Currently, the multi-tenancy capability of BPEL processes is achieved by the Web service components dynamically bound to the different process instances, resulting in different QoS behaviours for different tenants. [78] [77] present the WSO2 Business Process Server based on this idea. The extended Axis2 Web service engine will intercept and inject the message into the extended ODE BPEL engine runtime, which takes care of creating the

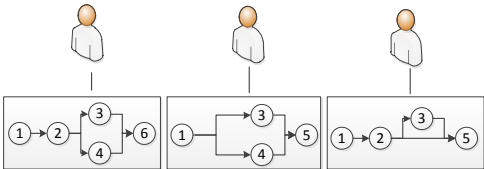
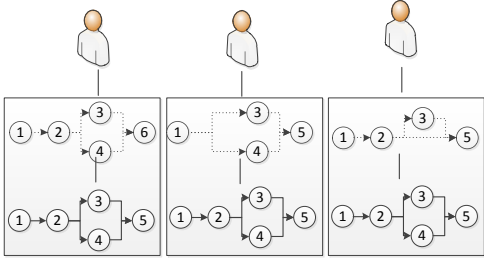
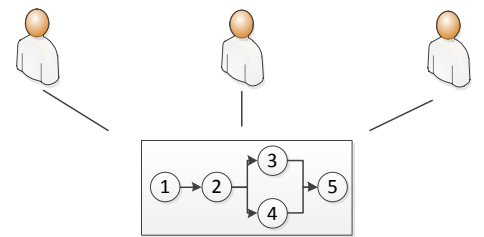
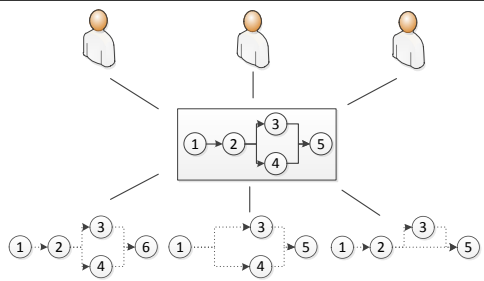
SaaS maturity level	Description
	<p><i>Level 1</i> provides a customized software instance per tenant. It is similar to the traditional <i>application service provider</i> (ASP). In this case, each process consumer has its own customized version of a BPEL process developed by the process provider.</p>
	<p><i>Level 2</i> provides a set configurable instance clone for tenants. A single version of BPEL process is hosted in multiple isolated instances for process consumers, i.e., the same BPEL process is deployed separately for each consumer with configuration options.</p>
	<p><i>Level 3</i> runs a single instance that serves all tenants. A single BPEL process is deployed to serve all process consumers. General Web services are in this case.</p>
	<p><i>Level 4</i> enables level 3 to scale up by running multiple instances for unique user experience by configurable meta-data, e.g., a single version BPEL process is offered, but process execution behaviours would be very different for each process consumers, such as QoS proprieties.</p>

Table 2.1: SaaS maturity level with regard to business processes

necessary process instances or routing the message to an already running instance.

2.3.5 Discussion

Business or BPEL processes could be offered as services for consumers in the SaaS model. Higher levels of resource sharing gives more economy of scale for SaaS providers, so consumers could have the best cost saving services. Our work tries to provide multi-tenancy capability business processes, falling in the category of level 4 maturity. A single version BPEL process is offered and deployed for all process consumers, but customization functionality is also offered at the same time to offer a unique experience for process consumers.

Related work such as [70] is trying to solve the problem in the PaaS layer, provides a BPEL engine rather than BPEL processes for consumers. process consumers still need to find or develop their own business processes in that case. In the SaaS layer, the Cafe project [71] [72] [73] [74] [7] offers the composited end user application for process consumers rather than software components. Business processes as software components should be free to be integrated with the end user application or as sub-processes by process consumers just as Web services.

Business policies might cover a wide range of requirements on business processes, rather than common QoS properties addressed in current work on multi-tenant business processes, which include the WSO2 and the Cafe project discussed. Adaptive processes are needed, so that process can be changed to meet the various change requirements of process consumers, i.e., various business policies of process consumers, as the base of customization. However, the multi-tenancy problem needs to be addressed, and it is not covered in the current adaptive process approaches, such as [17] [59], we discussed.

2.4 Policy based service computing

Policies play a key role in autonomic computing and SOA governance. We will first briefly describe the two domains (autonomic computing and SOA governance) to give the background context of our work within both domains, then consider related work. Software product line (SPL) also will be discussed in this section, as we look at variability descriptors of SPL as a kind of policy.

2.4.1 Autonomic computing

The goals of *autonomic computing* are to let systems manage themselves according to an administrator's goals, thus minimize human intervention in system administration [79] [80]. The fundamentals of autonomic computing revolve around self-governing or self-managing components. IBM frequently cites the following four aspects for self-managing components [80]. We consider these aspects as possible consumer requirements in our work.

1. *Self-configuration* - Components of the system automatically configure themselves according to high level policies. For example, Web services from trusted business partners are assigned for process execution.
2. *Self-healing* - The system automatically detects, diagnoses, and repairs fault occurrences. For instance, a non-response failure of Web service in a process is notified and remedied.
3. *Self-optimization* - Components automatically seek opportunities to improve system performance and efficiency. For example, skipping a redundant activity in a process.
4. *Self-protection* - A system automatically applies measures against malicious attacks or cascading failures. It refers to security aspects. For instance, cancel the business process if the buyer information is incomplete or unknown.

Policy based computing has been recognised as a core technology to achieve self-management for autonomic service computing [81]. These aspects are similar to common functional areas which have been discussed in policy based distributed system management [16] [82]: *Configuration management*, *Fault management*, *Performance management*, *Security management*. It also similar to the four reasons needed in adaptive business process development discussed: *Configuration*, *Correction*, *Optimization* and *Prevention*. So, the four autonomic aspects are a main concern with our policy modelling, since the same problems are defined in different related domains of software engineering.

2.4.2 SOA governance

Organizations need a consistent way to manage SOA to enable solutions to their business problems, to ensure it gives the results the enterprise envisions [83] [14]. The topic of SOA governance applies it, which might include decision rights, measurement, policy and control mechanisms, all placed around the services lifecycle [83].

SOA governance has been defined as a key part and given definitions in both OpenGroup and OASIS SOA-RAs, and policy is a key word in SOA governance. With OpenGroup, SOA governance is a vertical layer (*governance layer*) of the SOA-RA we described. SOA Governance ensures that the services and SOA solutions within an organization are adhering to the defined policies, guidelines and standards that are defined as a function of the objectives, strategies and regulations applied in the organization [84]. In addition, OpenGroup published a separated draft technical standard on SOA Governance Framework in 2009 [85]. The Framework covers: *The SOA Governance Reference Model (SGRM)* establishes a foundation of understanding, and is utilized to expedite the process of tailoring the SOA Governance Regimen for an organization; *The SOA Governance Vitality Method (SGVM)* is a process that starts with the SOA Governance Reference Model and then follows a number of phased activities to customize it for the organization's variants.

In the OASIS standard [14], the SOA governance model is described under *the owning*

SOA view. Governance is the prescribing of conditions and constraints consistent with satisfying common goals and the structures and processes needed to define and respond to actions taken towards realizing those goals. Governance expressed through policies, which are the formal characterizations of the conditions and constraints that governance deems as necessary to realize the goals. Goals are expressed by the participants within the organisations.

The description of SOA governance in these technical standards provides context and definitions to enable organizations to understand and deploy SOA governance. The governance activities and approaches described are at a high corporate level. The detailed approach at a software system level in our work is not targeted and covered in these specifications. And our view of governance is not restricted to 'within an organization' (with OpenGroup) or 'owning SOA' (with OASIS) as per these concepts or specifications.

SOA governance could involve a wide range of SOA activities in the lifecycle of SOA governance [86]. SOA Governance is viewed as the application of Corporate Governance, IT Governance and Enterprise Architecture Governance to SOA [85] [14]. In our work, we focus on the governance approach and technology on service processes or BPEL processes at software system level.

Policies in SOA governance technology can be separated into two types (design-time, runtime) [85]: Design-time policies ensure that the service registry/repository contains only approved, standards-compliant services. Runtime policies govern the service executions. Our work is restricted to runtime policy governance for on-the-fly customization by process consumers.

2.4.3 Policy modelling and approaches

Policy in general

[87] introduced a unified framework for defining autonomic computing policies that are based on the notions of states and actions, and is used in our policy modelling. In general,

a state represents a system component or characteristics at a given moment, and can be described as a vector of attributes. A policy will directly or indirectly cause an action to be taken, the result of which is that the system or component will make a deterministic or probabilistic transition to a new state. These types of policies can be modelled within the framework [87]:

1. *Action Policies* - describe actions that should be taken in a given state, such as, if condition, then action; ECA rules. Our policy model falls into this category.
2. *Goal Policies* - describe the desired state of a system. The system will decide to transit from a current state to a desired state.
3. *Utility Function Policies* - Objective function that expresses the value of each possible state.

Organisations generally have different administrative levels, such as managers, operators. Policies might be defined for different levels especially for security aspects. Many works on policies address this problem with a *Role-Based Access Control* (RBAC) model [88], which is also adopted for the BPEL process [89]. In RBAC, different access or execution rights are assigned to roles which represent the administrative positions in an organisation. Users are assigned to roles to have different access or action rights.

Various logic languages also have been developed or adopted for formalizing business policies, such as *business rules* [90] and ontological policies [81]. Business rules encode business policies with *If-Then* statements in pseudo natural language. Business developers can easily understand and edit the business rules. Ontology allows a formal representation of knowledge as a set of concepts, so they can be machine reasoned.

In many BPMs, policies include descriptions of the monitoring activities on processes, which are associated with the goal state of policies. Business process monitoring can be generally divided into Technical Monitoring and *Business Activity Monitoring* (BAM). Technical monitoring provides information about if technical requirements are met, such

as Web service *mean time between failures* (MTTF) report. BAM intends to provide a real-time summary of business activities to operations managers and upper management to help enterprises overcome *IT blindness* [91]. It used for tracking and to assure the progress of business processes with *key performance indicators* (KPIs) on dash boards, predicting violations of KPIs, etc. Technical monitoring could also be an underlying part or a support of BAM. In such cases, a high-level description of goals is associated with KPI values in policies. It makes it easier for non-technical people and business users to define technical monitoring requirements.

Business rules and approach

Business rule statements are commonly used for expressing business policies of business processes in SOA governance, incorporated into business processes to support *business process agility* [90] [92] [93]. According to the Business Rules Group [94], a rule statement must be either a term or fact (described below as a structural assertion), a constraint (described below as an action assertion), or a derivation. A large majority of business rules are expressed using the *If-Then* format [92] [90], for example, *if an order amount with a total over 200 euros, then we give 5% discount on the order*. In production systems, formal rule languages of business rules, such as Jess rules [95], are executed in rule engines included in the *Business Rule Management System* (BRMS) solutions. Some business rule formal languages have natural English language syntax, such as the OMG standard on *Semantics of Business Vocabulary and Business Rules* (SBVR) [96] and IBM ILOG JRules [97], which gives advantage to business analysers and policy developers without programming knowledge. Business rules can be classified in different types. Table 2.2 shows studies for business rule classification.

For a *business rules approach* for business processes, rules are in a form that is used by and does not have to be embedded in a business process. The business rules approach for BPEL process development is shown in Figure 2.5 [101]. The basic steps for both new and

Classification schemas	Re.
<i>Derivation rule</i> : a statement of knowledge that is derived from other knowledge in the business. <i>Structural assertion rule</i> : a defined concept or a statement of a fact that expresses some aspect of the structure of the enterprise. This encompasses both terms and the facts assembled from these terms. <i>Action assertion rule</i> : a statement of a constraint or condition that limits or controls the actions of the enterprise.	[94]
<i>Constraint rule</i> : a statement that expresses an unconditional circumstance that must be true or false. <i>Action enabler rule</i> : a statement that tests conditions and upon finding them true, initiates another business event, message, or other activity. <i>Computation rule</i> : a statement that provides an algorithm for arriving at the value of a term. <i>Inference rule</i> : is a complete statement that tests conditions and upon finding them true, establishes the truth of a new fact.	[90] [98]
<i>Derivation rule</i> : represents a statement of knowledge that is derived from other knowledge by an inference or a mathematical calculation. <i>Integrity rule</i> : represents an assertion that must be satisfied in all evolving states. <i>Reaction rule</i> : causes a constructive action when a certain event occurs and/or when a certain condition is met.	[99] [100]

Table 2.2: Classification of business rules

upgrade process development are: 1.Develop business rules. 2.Generate rule tasks services. 3.Develop or modify BPEL processes. Rules are wrapped in rule task Web services. These rule task services will be integrated in BPEL processes.

In this case, since the part of process logic defined in business rules is separated from BPEL processes, processes could be continually refined and updated by changing the rules, but without changing or redeploying the BPEL processes.

OASIS XACML

The security policy of a large enterprise has many elements and many points of enforcement. XACML stands for *eXtensible Access Control Markup Language* [21]. It is a XML based security policy language for access control. IBM published its initial research in 2000, proposed a security policy language based on XML language [102]. It became an OASIS standard in 2003 by continue development from such as IBM, Sun Microsystems, and the latest version XACML 3.0 is published by OASIS in late 2010 [21]. The basic concepts of the latest version of the XACML policy system architecture are:

- *Access Request* - An access request consists of attributes that describe an operation on

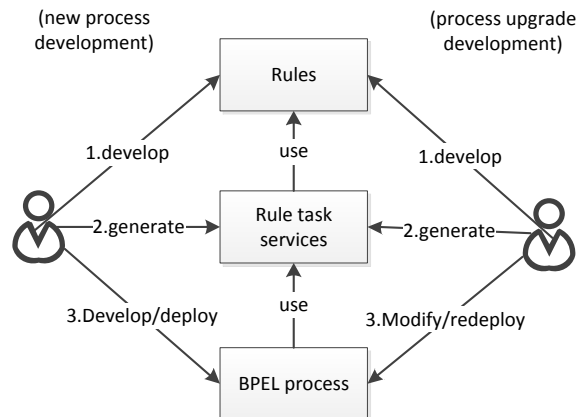


Figure 2.5: Business rule approach with BPEL development

a resource. These attributes provide information about the subjects (the information of a user who requests access), resources (e.g., a medical data record), and actions (type of access that is being requested e.g., read, write, delete).

- *Access response* - A decision action about the access request based on defined policies. A final decision (effect) is either 'Permit' or 'Deny'.
- *Policy Administration Point (PAP)* - The system entity that creates and manages policies, which are defined in XACML.
- *Policy enforcement point (PEP)* - The system entity that performs access control, by making decision requests and enforcing policy decisions. The access requests are generated by a PEP.
- *Policy decision point (PDP)* - The system entity that evaluates an applicable policy and renders an authorization decision. The PDP finds the applicable policy out of all of the policies created at PAP. The PDP then evaluates the access request against the policy, makes a decision, and informs the PEP.
- *Policy information point (PIP)* - The system entity that acts as a source of attribute value. The information needed to evaluate an access request at PDP, are as attribute queries sent to PIP. PIP responds to the attribute queries to provide the information for PDP.

The key concepts of latest version of XACML policy language model include:

PolicySet, Policy, and Rule: XACML is structured into three levels of policy elements. A PolicySet can contain a set of policies. Multiple rules can be associated to a policy. This nested policy structure allows more accurate policy definition with enterprise hierarchical administrative levels. For example, there could be organisation level policies, and department (HR, finance, etc.) level policies on a same resource. All three level elements also contain Target elements, which define the relative scope of policies. OASIS also defines an additional specification [103] with an XACML profile for the RBAC model.

Target, condition, effect: Each rule is composed of (a target, a condition, an effect). The target defines the set of requests to which the rule is intended to apply in the form of a logical expression on attributes in the request. Conditions are statements about attributes that upon evaluation access request return either True, False, or Indeterminate. Effect defines the consequence of the rule for access response. It can either be Permit or Deny.

Policy-combining algorithm: Since multiple rules and policies in nested policies may return different results when evaluated against the same request, there must a technique to solve the conflicts to determine a final authorization decision for an access response. XACML offers combining algorithms which are to be used for combining multiple decisions if that is the case. The XACML specification defines several standard rule combining algorithms, including 'deny-overrides' (return 'deny' if any decisions evaluate to 'deny') and 'permit-overrides' (return 'permit' if at least one decision evaluates to 'permit'). Other possible decisions that might result in middle of policy evaluation are also handled by combining algorithms, such as Indeterminate (an error occurred or some required value was missing, so a decision cannot be made) or Not Applicable (the access request can't be answered by this service).

W3C WS-Policy

The *Web Services Policy Framework* (WS-Policy) [15] is a W3C recommendation,

which provides a general purpose model and corresponding syntax to describe the policies of Web Services. WS-Policy defines a base set of constructs that can be used and extended by other Web services specifications to describe a broad range of Web service requirements and capabilities, for example, required security tokens, supported encryption algorithms, and privacy rules. The following shows a simple policy:

```
1 <wsp:Policy xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy"
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
2   <wsp:ExactlyOne>
3     <sp:WssUsernameToken10 />
4     <sp:WssUsernameToken11 />
5   </wsp:ExactlyOne>
6 </wsp:Policy>
```

The basic concept of WS-Policy includes:

Policy alternative: A policy is a collection of policy alternatives. A policy alternative is a logical construct for combining a collection of policy assertions. The policy alternatives are defined by policy operators: *<All>* defines a policy alternative, *<ExactlyOne>* defines a collection of policy alternatives. Policy operators may be recursively nested.

Policy assertion: A policy assertion represents a requirement (or capability) of a policy subject on Web services. A consumer could define service requirements for service providers. Service providers could publish service capabilities as service contracts. WS-Policy allows nested policies. Any policy assertion may contain a nested policy expression. For example, a QoS assertion of a policy defines a security requirement. Other WS-* specifications could be used as assertion languages for defining the policy assertion. For example, using WS-SecurityPolicy for security requirements, such as authentication tokens, encryption, a digital signature.

The key concept of WS-Policy is the policy alternative, so that it allows the negotiation between service providers and service consumers. For example, only one QoS assertion will be satisfied each time, either a high performance or a heavy encryption algorithm.

2.4.4 Other related work

The semantics of XACML is designed for general purposes rather than a particular application or environment. Still, it has been largely applied in Web service systems [104] [105]. Moreover, the *Web Services Policy Language* (WSPL) is introduced [22] [106] to cover various aspects and features that can be controlled or described for Web services. The syntax of WSPL is a strict subset of XACML. WSPL can specify a wide range of policies, including authorization, QoS, reliable messaging, privacy, and application-specific service options.

Many works such as [63] [107] [68] and the Oracle SOA fault management framework [108] only focus on policies with self-healing aspect of BPEL processes. The theory behind all these works in policy modelling is the *Event-Condition-Action* (ECA) paradigm. A fault as an event of the system will trigger a remedial action defined in the policy.

The Dynamo project [17] [109] [110] proposes the WSCoL assertion language and WSReL recovery language for BPEL processes. The WSCoL mixes typical propositional logic constructs with XML-based technology. The WSReL is designed by following the ECA paradigm. Both languages use a Java like programming language to enhance the language construction power.

[16] [111] introduce the WS-Policy4MASC language as an extension of WS-Policy, which developed for the specification of monitoring and adaptation policies for Web services and business processes. WS-Policy4MASC includes four types (Goal, Action, Utility, Meta influenced by [87]) of policy assertions for WS-Policy operators (e.g., *<ExactlyOne>*). The policy assertions may reference a *<When>* element to specify further conditions to be satisfied before a policy assertion should be processed. Policies can target a service operation (e.g. GetStockPrice operation) or an execution event (e.g., ProcessDeployed, Before-SendRequest, a set event generated by the MASC middleware).

2.4.5 SPL and Variability descriptor

Software Product Line (SPL) refers to software engineering approaches for developing a collection of similar software systems from a shared set of software assets, so that cost, time, etc., can be reduced on individual software production [112]. One core approach is providing customized software systems based on a *software platform* for different consumers [113] [114]. The customization is done by configuration of the *Variability Point* developed for the software platforms. The variability points of software platforms could be modelled as variability descriptors to allow to describe the need of customization [115].

We view variability descriptors as a kind of policy in the sense of configuration description of system or user requirements for the goal of software flexibility. Variability modelling generally defines the possible changes of software systems. In contrast, the policy modelling generally defines the needs of changes.

In the context of BPEL as the software platform of SPL, several variability descriptor frameworks and associated frameworks have been proposed. [116] [115] introduces the VxBPEL BPEL extension. It includes constructs, such as <VPChoice> and <BpelCode> that extend BPEL elements allow to redefine the fragment of an original BPEL process or codes.

[117] developed a *locator* and *alternative* variability descriptor constructor. The locator declares the variability point of BPEL processes. The alternative specifies one or more possible values for a variability point, such as a service reference of a BPEL invoke activity.

In SaaS approaches, such as the Cafe project [7] we mentioned before, the *SCA assembly model specification* [118] is directly adopted as a variability descriptor for customizing composite applications [72] [117].

2.4.6 Discussion

Policy based computing becomes one of our key research domains for two main reasons. Firstly, a policy language specification allows consumers to define their requirements in-

dependently from providers' implementations. Secondly, consumers can update policies at runtime without pre-notifying and effecting providers' implementations, as policies are interpreted by a policy engine at runtime. The reasons are fundamental for our work to achieve the multi-tenancy capability.

Policies play a key role in autonomic computing and SOA governance. The four autonomic aspects are a main concern with our policy modelling. Our work only focuses on runtime governance in the context of SOA governance.

Policies express requirements which includes monitoring and control. The business rules approach can expose those parts of the process, which contain decisions that change often in a rule language, to increase the capacity for change and the flexibility of the overall process. However, business rules and processes are tightly coupled in the approach. Business processes are the last step of the development life cycle with integrated rule task services. As a consequence, the general business rules approach is not applicable in our case, in which business processes are produced from providers at the start. Additionally, the business rule approach assumes an organisation owns the business process. It is not designed for multi-tenancy cloud environments; the business processes can not deal with different business rules from different tenants.

Business rule formal languages can be used to express business policies. However, these might be too general, and do not cover the necessary syntax for different aspects of business process governance, such as self-healing. Many policy specifications such as XACML do not cover completely the four automatic aspects of requirements. So we define our own policy language. Still, our policy model allows other rule languages on top of our policy language model. The details will be described in a later section.

The study of policy languages for Web services can be classified in two categories: The first category is self-understanding policies. The defined policies by a service consumer or provider are not necessary to be understood by each other, such as business rules and XACML. The second category is mutual-understanding policies. These policies focus on

defining the formal service contracts between service consumers and providers as *Service Level Agreement* (SLA). As in WS-policy, agreements could be established between a service provider and a consumer on mutually acceptable policy if all assertions match. Other work, such as WS-agreement [119] and [120] that focus on service agreements and contracts are also in the second category.

Our policy model falls into the first category, as process providers are not expected to see the concrete policies to support the process consumers' privacy and flexibility. The agreement negotiation mechanism centralized in WS-Policy is also not needed in our policy model.

Our policy model is influenced by XACML, as our policy needs cover self-protection or constraint aspects of business processes and also cover the requirements of policies that might be defined in different administrative levels of organisations. Still, firstly, XACML and also WS-Policy are not process centric policy languages. They only deal with independent Web services. Secondly, they do cover the complete four autonomic aspects we need. Thus we developed our own policy language for expressing business policies for process consumers.

Business rule classifications (described in Section 2.4) only show different types of formal expression of business rules. The purpose of these classifications is to help rule developers to discover, analyse, and design business rules [90], which are derived from business policies. The final goal is formulating these business policies in a formal rule language for rule engines. These classifications do not give any concrete meaning to business processes. To develop our own policy language, we need a classification to find a common connection between policies (rules) and processes that can be used for our policy and process development later on. After the rules are categorized at high level, our policy language could be designed for process consumers to define the concrete policies with different categories of rules.

2.5 Transaction and coordination

Business transactions for web services are specified with coordination specifications or service process specifications. In following, we describe the background of coordination and business transactions, and related work for web service coordination specifications.

2.5.1 Coordination

Coordination is the act of the coordinator disseminating information to a number of participants or system parts for a variety of reasons, e.g., to reach consensus on a decision, or guarantee that all participants obtain a specific message. It is a fundamental requirement in distributed systems that many applications use either explicitly or implicitly, e.g., workflow, transactions, caching, security, auctioning, and business-to-business activities [121].

A general service coordination framework comprise the following key players [121] [122] [3] [123] [124]:

1. *Participant*: The service operation or operations that are performed as part of a coordination conversation.
2. *Coordinator*: The coordinator is responsible for communicating the participants in a coordination conversation based on a coordination model.
3. *Coordination protocol*: Defines the behaviour of a coordination conversation for a specific coordination model.

A successful coordination is based on an agreement between participants or coordination protocols. Coordination protocols are usually realized as a state machine to define the system behaviour of coordination conversation [125] [126]. The implementation of coordination frameworks [121] [3] applies the *Context Object* pattern [127], propagates additional information with state as (coordination) *context* to the participants. In a coordination framework [3], the coordination context defines the message communication between coordinator and its participants. A coordination framework [3] might be decoupled from coordination

protocols [128] [126]. In this case, the coordination context is relaxed and can extend to support a number of coordination protocols.

2.5.2 Business transaction

A business transaction is a consistent change in the state of the business that is driven by a well-defined business function [129].

Two kinds of business transactions for business applications are proposed [123] [129]:

1. *Atomic transactions* - follow *ACID* (Atomicity, Consistency, Isolation, Durability) [130] semantics and therefore assume that resources are locked for the transaction's duration and guarantees that all participants will see the same outcome (atomic). In case of a success, all services make the results of their operation durable (commit). In case of a failure, all services undo (roll-back) operations that they invoked during the transaction.
2. *Long-running transactions (LRT)* - typically occur in business-to-business interactions, they do not necessarily have a common outcome to avoid locks on non-local resources. They are potentially aggregations of several atomic transactions and have the behaviour of *open nested transaction scopes* [131]. The compensation that restores the original state in LRT is business-specific in contrast to the roll-back of ACID transactions. For example, the compensation of a flight booking (cancel a flight booking) will only refund half of the original payment.

There are a number of published specifications for business transactions with Web services, such as WS-TX [132] or BTP (Business Transaction Protocol) [133] .

Business transactions always involve a recovery process when exceptions or faults occur, where the transactions cannot be completed as default or as expected. In general, potential failure sources of business processes comprise [134]:

1. *Process engine failure* - breakdown of process engine leads to an abnormal termination of business process execution.
2. *Activity failure* - comprises failures within an activity, such as invalid responses from a Web service for an activity execution.
3. *Communication failure* - frequent problems of network based distributed systems. A down or slow network causes unresponsiveness in the message exchange.

Two types of recovery models on business processes are introduced [129]:

1. *Backward recovery* - A business process will return to the consistent state that existed before the execution, or cancel the effect made by the process after execution, which includes the subprocesses of the process if any. The business processes require defining the compensation logic that will cancel the effects of the failed transaction.
2. *Forward recovery* - which comprises backward recovery and forward execution and is used in LRT only [134]. The consistent state is stored from a transaction state, and its execution can continue past the point of failure.

2.5.3 OASIS WS-TX specifications

OASIS Web Services Transactions (WS-TX) specifications are the outcome of R&D by IBM and others. The recent version 1.2 is completed and approved by the OASIS WS-TX Technical Committee in 2009 [132], defining three protocol specifications for coordinating the outcome of distributed application actions:

- WS-Coordination [3]
- WS-AtomicTransaction (WS-AT) [128]
- WS-BusinessActivity (WS-BA) [126]

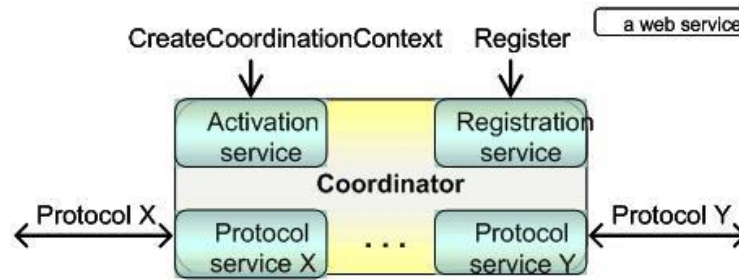


Figure 2.6: A coordinator of WS-Coordination framework [3]

The WS-Coordination specification defines an extensible framework for coordinating activities using a coordinator and set of coordination protocols. The coordination protocols are provided in additional specifications (WS-AT, WS-BA).

The WS-Coordination framework consists of these component services (Figure 2.6): an Activation service, a Registration service and protocol specific services.

These component services define three forms of interactions between a coordinator and its participants:

1. *Activation* - A participant requests a coordinator to create a coordination context. When a participant wants to initiate a coordination conversation, a new coordination context is created, for example, initiating an atomic transaction.
2. *Registration* - A participant registers with a coordination protocol in a coordination conversation. By registering, the participant will be notified for participation in corresponding steps in a coordination conversation as defined by the protocol. For example, a web service can be registered with an atomic transaction.
3. *Protocol specific interaction* - The coordinator and its participants exchange messages that are specific to a coordination protocol. For example, a commit message is sent by a coordinator to its participants in an atomic transaction.

The WS-AT specification defines a set of protocols for atomic transactions that follow ACID semantics for short duration transactions. These protocols in this coordination type, are executed in sequence or in alternative depending on what must be done during the

different phases of a distributed transaction [122]. These protocols include: *Completion*: The completion protocol initiates commit processing to complete a transaction. *Two-Phase Commit (2PC)*: to verify the outcome of the transaction and ask participants for a commit or abort decision, to reach an agreement on the outcome. Two variants of the 2PC protocol are *Volatile 2PC* and *Durable 2PC*.

The WS-BA specification defines a set of protocols for LRTs without having to lock resources. All participants inform the coordinator about the status of their execution (such as Exited, Faulted). The coordinator responds with a Close, Complete, etc., event to all the participants. Two protocols are defined in this coordination type based on the above business agreement. *BusinessAgreementWithParticipantCompletion*: A participant knows when it has completed all work for a business activity. *BusinessAgreementWithCoordinatorCompletion*: A participant relies on its coordinator to tell it when it has received all requests to perform work within the business activity.

2.5.4 Other related work

Since the WS-Coordination framework is extendible with other coordination protocols, an auction coordination protocol [125] for a coordinated distributed activity is introduced that fits with WS-Coordination. An auction represents a market institution, which is based on competition between its participants. The auction coordination protocol between the client and the coordinator is similar to the completion protocol of WS-BA. It gives the client the opportunity to start or terminate the auction and defines the messages returned to the client after the coordination, which provides the client with the outcome of the auction.

Coordination protocols, such as WS-BA, assume that a transaction has an initiator and that this initiator is also the one who is able to decide on the closure of a transaction. i.e., the participant initiating the process maintains a controlling position throughout the lifetime of the process. However, [124] argues that the initiator of a process is not always the one who is able to decide whether to commit or cancel a transaction in some scenarios. It extends the

WS-BA by enabling defining a set of rules to decide who and when decides the transaction process closure.

WS-BA is designed for Web service transactions, but no WS-BA based interaction between a process and contained services is assumed. It is not possible for a BPEL process to participate in a WS-BA coordination. [135] proposes WS-BA4BPEL which extends WS-BA to allow parts of a BPEL process to participate in a coordination. The modified BPEL engine supports the WS-BA4BPEL, which allows a BPEL scope registered as a participant and response for a coordination activity of BPEL sub-scopes.

2.5.5 Discussion

Since most business processes often involve long-running computations, loosely coupled systems, and components that do not share data, location, or administration [123] [122], and compensation mechanism of BPEL specification under the open nested transaction model supports LRT [131], our work on coordination and transaction focuses on LRTs. The data-centric ACID transactions are mainly used within task services, and are not considered in our work. Moreover, forward recovery will be addressed for transaction failures in our work, and it is critical to the fault policies of process consumers.

The WS-Coordination, WS-BA and extended works, such as [123] [124] [125], are about transactional activity control with distributed Web services. They are not designed for transactions of BPEL processes and contained services. However, without standard protocols, it is impossible to coordinate a transaction with various processes distributed in one or many different providers. Still, different aspects of policies as requirements needs a more comprehensive protocol rather than those that only deal with transaction management. Such work on coordination with policy enforcement for consumers and process providers is still lacking, but is needed for business process sharing in the cloud paradigm. Additionally, the multi-tenancy capability needs to be taken care of in coordination framework implementation. This problem needs to be addressed in our coordination implementation, but has not

been focused on in current work.

2.6 AOP and service computing

AOP has been widely applied in service computing to address the technical or crosscutting concerns in a flexible and modular way, including concerns or policies on business processes. In our case, we utilize AOP for the extensibility of our framework, allowing other policy models and frameworks to be adopted and integrated.

In the following, we give the background of AOP, and describe the related work with web service systems.

2.6.1 Aspect-Oriented Programming

Enterprise applications need to address many crosscutting functionalities: transaction management, security, SLA monitoring, error handling and so on. *Code tangling*⁷ and *Code scattering*⁸ are the problems with conventional implementations, such as OO or functional programming, with crosscutting concerns [136]. Core and crosscutting concerns are tangled in each module. Each crosscutting concern is scattered in many modules. AOP [136] is a programming paradigm that specifically targets the management of crosscutting concerns.

AOP encapsulates crosscutting concerns in a special type of class or module, called *Aspect*. The fundamental concepts of aspects are:

- *Join points* - There are a number of identifiable points during the execution of the system. These may include the execution of methods, creation of objects, or throwing of exceptions. Such identifiable points in the system are called join points.
- *Pointcut* - Implementing a crosscutting concern requires selecting a specific set of join points. For example, the fault handling aspect will be applied on a point of

⁷Code tangling is caused when a module is implemented to handle multiple concerns simultaneously.

⁸Code scattering is caused when a single functionality is implemented in multiple modules.

throwing of exceptions. The pointcut construct selects any join point that satisfies the criteria. A pointcut may use another pointcut to form a complex selection.

- *Advice* - After a pointcut selects join points, additional or alternative behaviour needs to be defined to address the crosscutting concern. This facility is provided by an advice construct in AOP.
- *Advice type* - Different types of behaviours can be added with regard to join points, such as adding behaviour before, after the selected join points. These types are advice type. Common advice types are: the *Before* advice executes before the join point, whereas the *After* advice executes after it; the *Around* advice surrounds the join point execution and may execute it zero or more times.

These implemented crosscutting concerns as aspects will be combined with core concerns in order to form the final system, and is called *Weaving* in AOP. Different weaving mechanisms are available depending on the AOP frameworks, such as compile time weaving [137] or runtime time weaving [138] [139].

To support *Aspect-oriented software development* (AOSD) [140] for different programming languages, systems or purposes, differed AOP frameworks, such as [136] [141] [142], are developed. These frameworks might have different AOP specifications under the fundamental concepts of AOP for their own purposes. Two core parts are needed in a full AOP specification.

1. *Aspect specification* - specify how to implement the individual concerns for the target system.
2. *Weaving specification* - specify how to combine the implemented concerns in order to form the final system.

2.6.2 Related work

[143] argues that non-functional features, such as security, routing, reliability, and transactions, which are implemented with a SOAP context handler approach, cannot be defined once for all when developing or deploying an application. This makes traditional middleware usually developed as monolithic and non-evolvable entities, resulting in a lack of flexibility and interoperability. The approach describes the non-functional requirements as policies, and implements aspects for the policies. The policy engine selects the appropriate aspects depending on the policies deployed.

[144] aims to minimize middleware participation in non-functional properties management. It describes how aspect-oriented techniques can be used in conjunction with WS-Policy to achieve the aim. The approach uses WS-Policy for the description of non-functional properties of Web services, and aspects for the implementation of the mentioned properties in WS-policy. The approach allows non-functional properties for Web services to be completely decoupled at description and implementation level. As a result, it provides a modularized, standardized and reusable way of describing and implementing the non-functional properties within a Web Service environment.

[145] discusses the significance of business rules segregation for responding to ever changing business requirements in shorter cycles. They propose segregation of business rules from other business aspects like business entities and business processes for dynamic process management. A practical Aspect-Oriented Framework is developed for rule-based business process management where business rules can be defined and managed dynamically.

Both [146] and [147] have discussed problems with the business rules approach. The problems can be traced down to the lack of modularity in the implementation of business rules with business process. [146] explains how to describe business rules in aspects. [147] demonstrates how to implement business rules in BPEL processes with their AO4BPEL framework as a separated module.

[148] [149] provides an AOP based Web Services Management Layer (WSML). This layer decouples Web Services from client applications and enables hot-swapping between semantically equivalent Web Services based on availability to address reliability and flexibility of service systems.

[150] uses AOP to deal with Service Domain adaptation based on context for a BPEL process. Three modules are contained in its framework: Context Manager Module (CMM), Service Orchestration Module (SOM) and finally an Aspect Activator Module (AAM). An aspect defines the adaptation behaviours of a BPEL process. CMM is used to catch context information changes. Aspects are activated by the AMM according to the context change.

Different from the above work which applies AOP for system implementations, AO4BPEL and A4B are two AOP frameworks specially designed for BPEL processes.

AO4BPEL [141] is an AOP framework which treats BPEL as the target programming language. Aspect and weaving specification is based on the specification of BPEL to obtain an aspect-oriented workflow language. BPEL elements, such as `<invoke>`, `<scope>`, `<reply>` activities are used to model the join point of AO4BPEL. BPEL itself acts as an advice language.

A4B [58] is an extension of the WS-Policy framework with respect to BPEL processes. The join point model is expressed in terms of events the BPEL engine needs to generate and notify, such as *ActivityReady* or *Link_Evaluated*. Advices are mapped to WS operations. The association of aspects to process models or instances is described in WS-Policy attachments. An AOP broker is added between BPEL engine and ESB. It uses WS-Notification for publishing events to the AOP broker, which is able to weave the aspect with business processes, i.e., invoking WS operations defined in an advice *before*, *after*, or *instead* of the activity in the business processes.

2.6.3 Discussion

AOP aims to increase modularity by allowing the separation of crosscutting concerns. Many works have applied AOP with policy systems on Web services, such as [142] [144] [143] and business processes, such as [147] [16] [145] to leverage the advantages of AOP.

Current research has shown that AOP can be applied for policy enforcement. Aspects implement policies, and are weaved into business processes to address the policy enforcement in a modular way. The approach also is suitable for our policies in the policy framework. However, further research still is needed for some questions to be answered: how process consumers are able to use other policy models or frameworks, such as business rules, with our policy model and framework together; how policies defined in various policy models can be integrated without conflicts. Simply translating policies to aspects does not answer the questions. Our work will address the problems as an extensibility problem of our policy framework into an AOP enhanced policy framework.

2.7 Conclusion

In this chapter, background and related work including existing standard specifications were investigated. Several domains of software engineering including such as software architecture, business process systems, policy based computing, with regard to service computing and related with our work have been taken into account. The related issues of current work concern to the goal of our research is discussed.

Part II

Designing an architectural style

Chapter 3

Problem statement as an architecture problem

3.1 Introduction

In this chapter, we analyse the problem we observed, and identify it as a software architecture problem in need of an architectural style and framework.

There are many business processes which could be used across many application domains and organisations. For example, a purchase order business process could be used in supply chain systems, online retail applications, etc. A recruitment business process could be used for many recruitment agents or HR departments of many organisations. As a consequence, many business processes have a high potential to be shared by many organisations.

In this chapter, we introduce a purchase order business process scenario. The scenario describes a service process offered from process providers for the checkout process. We assume that some organisations as process consumers are looking for the business process. However, different business policies of organisations make it difficult to share any business processes or large Web services for any process consumers outside of the organisation.

Based on the scenario storyline, we will study some examples. From that, we can

observe that our work, which offers Process as a Service in the cloud paradigm, is needed for an organisation as either a process provider or a process consumer. Process governance can act as a way to address various policies for business processes. However, through our analysis, we will show that many critical issues arise in current cloud solutions, such as policy centralization.

The essential problem we identified and abstracted is as an architecture problem of the SOA style. That problem is how business processes can be shared by many organisations or tenants with different requirements regarding processes. The solution is proposed as an architectural style for the problem, and an architecture framework for the style.

This chapter is organized as follows. In Section 3.2, we describe the purchase order checkout business process scenario. In Sections 3.3 and 3.4, we show the need for our work and the issues with current work. In Section 3.5, we explain the architectural style and framework as our solution. Finally, we present a conclusion (Section 3.6).

3.2 A purchase order checkout business process scenario

In this section, we describe a general purchase order business process scenario [151], which commonly appears in the e-commerce domain for our case study. This scenario will be used throughout the case study sections in the following chapters of the whole thesis for proving the concept. The description is in two parts, the first part is about processes from process providers, then about potential process consumers with business policies.

In the following, we briefly describe concrete BPEL processes using the BPMN notation (Table 3.1). In this case, orchestrated task services are BPEL composite services for BPMN sub-process activities, and task services are atomic services for the BPMN task activities. All these BPEL processes are general Web services available on the Internet, and belonging to different organisations as service/process providers. For example, the *purchase order checkout* process could be from *Salesforce*, the *shipping* process could be from *FedEx*.


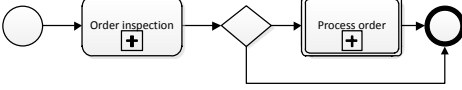

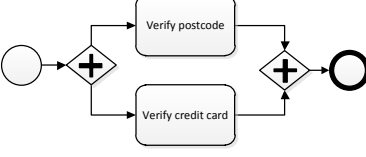




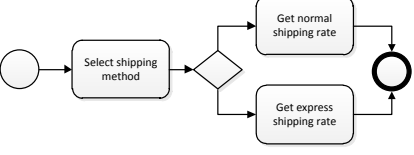

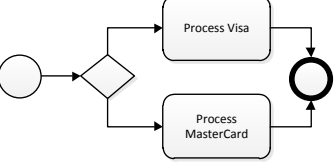
<p>Activity</p>  <p>Purchase order Checkout</p> <p>Buyer checks out an purchase order</p>	<p>Process 1</p>  <p>From: provider_1 <i>Order inspection</i> sub process refers to process 2. <i>process order</i> sub process refers to process 3</p>
<p>Activity</p>  <p>Order inspection</p> <p>Inspect buyer information. Cancel the checkout process if buyer information is not validated</p>	<p>Process 2</p>  <p>From provider_1</p>
<p>Activity</p>  <p>Process order</p> <p>Processing the order</p>	<p>Process 3</p>  <p>From provider_1 <i>Payment</i> sub process refers to process 6</p> <p>Process 4</p>  <p>From provider_2 <i>Shipping</i> sub process reference to process 5</p>
<p>Activity</p>  <p>Shipping</p> <p>Add shipping cost to the order</p>	<p>Process 5</p>  <p>From Provider_2</p>
<p>Activity</p>  <p>Payment</p> <p>Transfer money for buyer's credit card account to seller's merchant account.</p>	<p>Process 6</p>  <p>From provider_3</p>

Table 3.1: Processes of providers

Consumer_1	Business policies: <ul style="list-style-type: none"> • Receipt needs be issued to buyers for every checkout. • Shipping needs to be calculated for before payment. • Retry the Web service for card processing if it fails, but no more 5 times in the last minute, and no more 30 times in the last 5 minutes. • Free parcel shipping for orders with a total over 2000 euro. • Customer account information is not allowed to be passed to business partners that have low security (>3). • Credit card processing should be completed quickly (expected less than 700 ms) without fault). • Average time cost for purchase order inspection activity executed in the last hour for each order should less than 5 seconds. • Item partNumber'32541' is a hazard item. Item partNumber'1234' is a hazard item. Sellers with an Irish address are in a controlled area for selling. Buyers with a UK address are in a controlled area for buying. Any hazard item in any controlled area is a controlled transaction. The transaction will be approved if it is not a controlled transaction.
Consumer_2	Business policies: <ul style="list-style-type: none"> • All payment transactions should be processed by <i>Bank of Ireland</i>. • Cancel all transactions for orders outside the Republic of Ireland and N.Ireland. •

Table 3.2: Policies of consumer

In the following, we briefly describe some organisations as service/process consumers who require a purchase order business process for their systems. They expect that the business process can be used as software components in form of Web services. The process can either be a sub-process of parent processes or for their consumer applications. However, the process must meet the business policies defined in their organisations. The business policies regarding the process for the process Consumer_1 covers different aspects (autonomic aspects) described in Table 3.2. Other consumers could have different policies. Some examples are given in the example of Consumer_2.

3.3 Process as a service

As we discussed, the SOA style and RAs are not designed for the scenario of sharing business processes. Business processes are the concerns of consumers, who should develop their own processes by utilizing available services. In this section, we briefly describe the benefits of our work from a software process [152] aspect.

3.3.1 The need from process consumers

Even if the checkout process is available for process consumers, it might not meet the business policies of Consumer_1 in the example, with the exception of the first two business policies. Since the process consumer that cannot find a suitable checkout process as a Web service from processes offered by service providers, this requires these organisations who are consumers to develop their own BPEL processes. They might utilize some sub-processes from the providers. However, self-development is not always a good option for organisations.

Time constraint

A time constraint deals with the time necessary to complete a business process development. A time constraint is a common restriction for a software project. In many cases, there is an absolute deadline. Failing to deliver software on time is not only costly, it also could crash the whole software project or damage the organisation.

For various reasons, the time scheduled for business process development might be just inadequate. These reasons could result from business decisions, or from unforeseen events. For example, Consumer_1 needs to complete the business process before next week, so the business can start summer holiday sales. Such examples could result in self-development failing to meet the requirement of time-to-market. By using shared business processes available from providers for on-demand needs, we could expect the time required on business process development to be reduced.

Cost constraint

Cost represents the resources required for completing the business process development. It includes many elements, such as labour cost, software cost, etc. In many organisations, the cost is considered as the top priority factor.

The same as the time constraint, the cost of self-development might be too much for many organisations. For example, the old version of the BPEL developing tool is not supported, Consumer_1 must pay an expensive license for a new version to start the BPEL development. In this case, the organisations might not have enough budget for the cost of self-development. By sharing business processes available from providers with a pay-per-use-model, we could expect the cost to be reduced for individuals.

Investment containment

There are inevitable costs associated with business process development. It is important to protect the investment of organisations in business process development. As business and technology continues to change at a faster and faster rate, protecting investment for software or processes development is difficult.

Organisations must respond to the fast changing trend of business and technology to successfully keep profit and manage risks. This might require organisations dropping current business and starting business in a new area, or completely redeveloping the business process with new process technology. Accordingly, old business processes might be discarded. For example, Consumer_1 decides to drop the online selling process after few months, as the performance is not as expected. Investment in process development by the organisations suddenly vanishes in this example. With large investments in the complex process development, this will be a significant loss. By sharing business processes available from providers, the organisations could simply have pay-per-use-model business processes.

3.3.2 The need from process providers

Even if process providers have offered the checkout processes for consumers, because of different business policies with different organisations, it is of little use for process consumers. In fact, it is hard to find large services shared in the real world. Because of the large process logic inside the checkout process, it is difficult to satisfy another organisation. Such processes are generally only developed for self-use purpose within organisations' SOA infrastructure.

Increase ROI for self-use

Return on investment (ROI) of automated solutions is a critical factor in determining just how cost effective a given application or system actually is [11]. Organisations would benefit more with a greater return from the process development and maintenance. However, a self-use only process would only erode the budget and profit of the organisations. By having business processes fulfil the self-need and favour other organisations would increase financial returns.

As suppliers in a cloud supply chain

Cloud Supply Chain (CS-C) [153] or the value chain [2] concept has been advocated as a new business model in enterprise computing. Hardware, database storage, applications and other IT resources could be obtained from different suppliers for end-user systems in the cloud with short development circles. A standard such as ebXML [154] could be a basis for process development. An organisation or process provider might not need the business processes for themselves, but offering the processes to others as suppliers in the supply chain could be a great business opportunity.

3.4 Business process governance

In this section, we analyse current approaches in cloud computing for processes sharing. Related work has been described in the Chapter 2.

3.4.1 SOA governance for business process

Work regarding SOA governance ([155] [147], etc.) has addressed the problem of business policies and processes. However, SOA governance is only about governing business processes within organisations, i.e., an organisation is not only the process consumer, but also the provider of the process. It is also from the *owning SOA viewpoint* defined in the SOA-RA [34]. Business policies are tightly coupled with business process development and infrastructure within organisations, such as the business rule approach. We are not talking about reuse or governing pre-developed business processes from an inter-organisational process repository. In our case, process consumers belong to many different organisations. Processes are shared across enterprise boundaries in the cloud environment.

3.4.2 Business process delivery in cloud

Current approaches such as the Cafe project [156], and industry SaaS vendors such as *salesforce.com* for business process delivery in the cloud are built on top of provider side SOA governance approach. A configuration database is added on top of SOA governance for different tenants. Each registered tenant/process consumer could login to a user control panel to configure or define policies or descriptors. However, the problems are:

- A pre-registered account is required for all process consumers

To create separate configuration data for a process, each process consumer must be a registered user to keep a unique account ID in the configuration database. This registration procedure limits the open accessibility of business processes. It is impossible for many scenarios for service applications, such as dynamic process discovery and invocation [157].

- Policy enforcement completely relies on process providers

Registered tenants could configure the business process by setting configured data in the configuration database. After consumers set their policies in the configuration database, the policy enforcement completely relies on the process provider, as process consumers have no monitoring or control of processes any more. This means the process provider has to be fully trusted, without a satisfactory verification preferred by many businesses [158].

- Problems with policy centralization and reuse

Policies should be centralized in a SOA management system to avoid redundancy and inconsistency problems for policies with multiple services and processes [23]. However, saving policies at process providers reintroduces this problem. Especially, a company might have many different providers for different processes. In addition, each process provider may have a different policy or descriptor models for the configuration. This means that existing policies of process consumers might need to be re-formalized for each of the process providers, making it is very difficult to change process providers. This is also a vendor lock-in problem of cloud computing, which is feared by many organisations [24] [159].

- Concerns over privacy with business policies

Business policies contain confidential information, which might concern the competitive advantage of an organisation. Storing or exposing these policies to process providers, i.e., outside the organisation, raises privacy concerns [159]. Also, the business partner relationships might change in dynamic business situations. A process provider as a business partner may be trusted today, but might become a competitor tomorrow. Some business policies details may be forbidden to be exposed to process providers.

3.5 A new architectural style and framework

In the last section, we presented the problem which has driven our motivation. In this section, we abstract the problem from service based business processes to generic programs or applications in the cloud environment as a software architecture problem. We outline the requirements needed from an architecture level approach. We identify requirements as a need of a new architectural style and framework for a systematic solution. Additionally, we could extend our contribution, not only restricted to Web service systems, but also to other types of application development in the cloud trend.

3.5.1 The need of a new architectural style

Architectural styles (described in Section 2.2) can be organized by their key focus areas [160], [31], i.e., the contribution area of software engineering. For example, SOA and *Message Bus* focus on the communication of software. *Client/Server*, *3-tiers* and *N-tiers* focus on the deployment of software. *Component-Based* and *Object-Oriented* focus on the structure of software. A large software application development often adopts several architectural styles to meet all requirements.

One architectural style may be developed or derived from other architectural styles to address a new problem that emerges in the software engineering or business world. In some cases, two architectural styles may look similar, but should not be mixed as their focus or contribution areas are different. For example, the SOA style is derived from Component-Based and distributed computing [13]. The building blocks of SOA are not arbitrary components or distributed objects [13]. Instead, they are reusable contracted services accessible from the Internet or services registries. Services in SOA are focused on providing a schema and message-based interaction with an application through interfaces that are application scoped, and not component or object-based [160]. An SOA service should not be treated as a component-based service provider [160].

The SOA style can package business processes into composite services and expose them

to process consumers. However, how the business process can be an interoperable process for consumers with different policies is not addressed by SOA, or any other architectural style. The essential problem is that when a scoped program or application, such as a service process, contains large process logic, it becomes very difficult to meet various consumers' requirements at the same time. This is a distinctive software engineering problem related to the multi-tenant character of cloud applications.

The new architecture style needs focus on governing developed applications at runtime for application tenants' various requirements or policies, but also addresses the issues with current approaches we observed. More specific to business processes and SOA, the architecture style focuses on governance of the service process for process consumers' business policy requirements. Although our study and framework developed are made for service processes or BPEL, the programs should not be restricted to BPEL. The business policies are only our form of representing application tenant requirements. The architectural style could be applied to other types of software application design, for example Java programs in the cloud. The architectural style will be described in the next chapter.

3.5.2 The need of new architecture framework

The SOA style only contains a set of principles for service design, but not a framework for creating running services. The architecture framework that brings the SOA to reality is a set of open standards and software components, thus software developed by different languages and organisations can be integrated and collaboration between organisations can be built. Apache Axis [161] and JBossWS [162] are examples of software frameworks which implement the standards for service development within the SOA style. It enables interoperability and integration of complex software systems [13]. Without support from the SOA architecture frameworks, the SOA style applications cannot be implemented.

Similarly, we need a new architecture framework for software implementation for the new architectural style. The framework will include one or more proposed standards and

software components for the new style. In general, standards are not required for an architecture framework. Like the SOA style, our new architectural style emphasizes collaboration between multiple organisations. Some standards or protocols must be established between these organisations. Software components will implement the standard for our prototype development. The architecture framework will be described in later chapters.

3.6 Conclusion

In this chapter, we described the problem using a purchase order business process scenario, which will also be used as a case study scenario in the following chapters. We observed and identified the problem as a software architecture problem. The problem is a need for a new architectural style to guide the software design and the supporting architecture framework for software development.

Chapter 4

Service Process Architecture style

4.1 Introduction

This chapter discusses a Service Process Architecture (SPA) architectural style that aims at sharing business processes with multiple tenants who have various business policies regarding the processes.

How business processes can be delivered in the cloud environment is gaining attention in academia recently [69]. The SOA style or RAs are not designed for solving this problem. Some work related to critical concerns in cloud computing communities, has been discussed in the last chapter. The SPA is an architectural style defined for sharing large programs or processes for multiple tenants who might have various requirements regarding the programs, to offer a solution of process as a service for software design.

We introduce the SPA architectural style to extend SOA as a solution for the above problem. The basic concept will be introduced in this chapter. As an architectural style to guide software design for software engineers, we define a principle of SPA - *Process Governability*. It extends the SOA style to enable the process as a service in the cloud. The SPA principle itself and its connection to principles of the SOA style will be discussed in this chapter.

The overall organisation of this chapter is as follows. In Section 4.2, we introduce the basic concepts and elements of SPA. Then, we describe in detail the process governability principle in Section 4.3. In Section 4.4, we describe the roles and activities of process consumers in on-demand self-service business process automation. In Section 4.5, we give a case study in application architecture. In the remaining Sections (4.6 and 4.7), we compare with related work and give some conclusions.

4.2 SPA basic concepts and elements

SPA is a style for distributed computing that promotes sharing of large programs or service processes for different tenants. Service processes are the programs in our context. One objective of the SPA style is an attempt to provide a plug and play EAI solution for business processes for enterprise-wide collaboration. The design of SPA is derived from the notion of code mobility [163] and SOA principles.

The *code mobility* styles define a *general principle* that the code segment, the execution state, and the data space of computing units might be relocated to different computation environments [163]. It addresses a wider range of needs and requirements, such as service customization, dynamic extension of application functionality, fault tolerance, etc. A set of architectural styles fall under the concept, such as remove evaluation [163] [27], code on demand [163] [27]. SPA is under the scope of the code mobility concept, allowing relocation of the computing units which are available for process governance to the computation environments external to the process consumers' side. It defines a principle -*Process governability* to advocate this relocation for business processes of providers on system design. Moreover, it has its own characteristics which differ from other architectural styles [163] [27] under the code mobility concept and also the SOA style.

Components are the primary building blocks of architectures. SPA identifies two different basic computational components based on behaviours and responsibilities for programs

and tenants in different computation environments for governance, i.e., process provider and process consumers in the context of business processes. It is modelled as a tuple:

$$SPA = \langle \langle BP, PG \rangle, CP \rangle, \text{ where,}$$

- $bp \in BP$ is a SOA service process or subprocess component in the cloud

Process components on the process provider side offer business processes. They execute business activities within the process logic to serve a particular goal, and hold process runtime information resource need for policy evaluation or weaving.

- $PG = PG^e \cup PG^i$,

$pg^e \in PG^e$ is a process governance component external to the provider of a bp , or owned by external consumers

$pg^i \in PG^i$ is a process governance component internal to the provider of a bp

Governance components govern processes runtime for process consumers. They hold the policies of process consumers and evaluate or weave the policies.

- $cp \in CP$ is a coordination protocol

A *coordination protocol* within a service or process contract defines the connectors and behaviour between any process and governance components. The protocol will be described in a later chapter within the architecture framework design.

Process components send process runtime information resources to the governance components for a governance request as is defined by the contract. Governance components respond with guidance actions or decisions which are defined by the contract to govern process execution after policy weaving. The governance components could be viewed as autonomic managers which have the functions of *Sensors* and *Effectors* from the perspective of an autonomic computing architecture [53]. The process components are not tied to any governance components but comply with coordination protocols. One governance

component is responsible for one process consumer that has a separate set of policies. The connections from any governance component to any process component are dynamic on demand through coordination protocols to offer a *mess architectural topology* [26] between components. It supports three different implementation patterns for meeting various business scenario needs:

1. $pg \in PG^e$: *Consumer driven pattern* - The policies are implemented by external process consumers. Each consumer freely defines their own policies for the business process.
2. $pg \in PG^i$: *Provider driven pattern* - The policies are implemented by the process provider. The process provider defines policies for different process consumers. For example, in a scenario with internal consumers of a large organisation, several policy models for different regional branches have different policies.
3. $pg_1 \in PG^e \wedge pg_2 \in PG^i$: *Hybrid (Consumer & Provider) driven pattern* - The policies are implemented by both consumers and providers. For example, in addition to applying the process internally, the process also provides this service for external customers.

4.3 SPA principle

In SPA, the processes are the central focus. This is unlike architecture approaches such as the SOA style and the SCA framework, where the services as process components are the focus so that applications rely on available services to facilitate business processes [13]. In SPA, the process micro level is focused on rather than the service micro level. In other words, orchestrated task services or processes are our core concerns. It concentrates on a sequence of business activities within a defined process logic rather than a single activity.

SPA defines one principle - *Process governability*, which extends the SOA to enable Process as a Service. The whole architecture is formed by the fusion of SOA and SPA to

offer shared service processes. While the SPA principle serves its own goal, similar to the SOA style, the SPA contains a flexible principle rather than constraints defined in many architecture styles [27] [29]. It gives flexibility for software engineers incorporated in other SOA principles. They could balance between different principles with the final goal of the service or process design. In the following section, we study the SPA principle and also in connection with SOA principles defined by Thomas Erl [11].

4.3.1 Process governability

4.3.1.1 Governability explained

Governability in abstract

Governability represents the ability for *external* monitoring and control as governance of multiple consumers. The governance involvement has the control to make guidance, decisions on processes based on observations from external of the processes.

If a software program is in a governance runtime state, the consumer of the process is capable of observing and influencing the behaviour or affect what the process is carrying out. If monitoring and control is desired from external of the process to govern the process at runtime, the more monitoring and control of the process is offered to external, then the more governability the process can achieve. To achieve greater governability requires that the process implementation are more open and flexible in its internal states and components to increase the levels of governability. The result of achieving enhanced governability in software programs or processes is increased customizability and adaptability due to the increased external control available in which the programmes operate.

Origins of governability

Customizability is the ability for software to be changed by the user [164]. *Adaptability* is the ability for software changes to fit to the environment or requirements [55]. Governability is a combination of customizability and adaptability for the multi-tenancy capability.

It is the ability that software changes fit the changes required from outside of the software by the same or different aspects from multiple users, and also provides the observation of the software execution. The change represents the result of control of governability. The observation represents the result of monitoring of governability for needs of the control or general business monitoring.

The more governability of software is available, the more requirements on the software could be made from different tenants or groups of tenants for their own needs, the the more customizability that it will be for individual tenants. The more adaptability it has, the better software is able to change to fit to the changes needed by multiple tenants. Customizability and adaptability are two key factors to make governability a principle of the SPA architectural style.

4.3.1.2 Profiling the principle

For cloud services, more tenants with various requirements are desired to use or share the programs of providers. Customizability and adaptability need to be offered for consumers to tailor and adjust processes for their own needs. To provide this, processes must be governable for tenants. This requires processes to give a significant degree of monitoring and control to external of the processes, for tenants or their delegates. Table 4.1 describes the principle profile.

4.3.1.3 Measure of governability

The measure of governability as the result of an ability of software on software design, can help process developers to set their goals of process design in relation to customizability and adaptability, also can help process tenants to discover and select their processes. The measure of governability can from tenants' viewpoint, the level of governability as the degree of possible requirements of tenants achieved through program governance. The measure could be from different perspectives:

Short definition	Processes are governable
Long definition	A high level of monitoring and control over the underlying process and runtime environments is available for tenants or consumers externally.
Goals	<ul style="list-style-type: none"> • To attract the potential tenants with requirements. • To increase the process customizability and adaptability for individual tenants and multiple tenants.
Design Characteristics	<ul style="list-style-type: none"> • Process has a contract that expresses a well-defined process behaviour and governability for tenants. • Governability should be comprehensive, and governance from tenants could be applied on demand with self-service. • Process behaviour instances and governance actions are isolated for individual tenants.
Implementation Requirements	<ul style="list-style-type: none"> • A data model design for the process and process behaviour control depending on the need of governability. Extra process logics or data attributes might be developed for a process to support a great level of governability. • An infrastructure capable of supporting distributed governance environment for direct governance available for tenants. • A multi-tenant infrastructure capable of supporting process and governance behaviour for multiple tenants without interfering with each other.

Table 4.1: Principle Profile

From a technical software engineering perspective, requirement analysis [165] [166] can be applied to the measure. The governability could be classified into aspects such as functional, non-Functional requirements, and domain specific requirements.

From a business analysis perspective for business processes, business requirements or business policies could apply to the measure. With our rule of policy categorization (will be described later), the governability could be classified based on availability of enforcing, such as flexibility rules or constraint rules.

However, the comprehensive metrics and approaches for measuring process governability require more research, and we note it as our future work.

4.3.1.4 Type of process governability

In this section, we discuss two primary forms of governability: Process component and flow governability regarding service processes. They have different primary objectives, but share the same goal. The more governability is offered by processes, the more opportunities the processes can be governed to meet various or future requirements. This also means more monitoring and control needs to be supported by the process or infrastructure.

1. Process component governability

Component governability means that the monitoring and control is available for components of the process. In generally, components are service component for service processes, but could also be workflow language specified components, e.g., BPEL *scope* as BPEL component. The primary objective of offering component governability for process consumers is to

- Meet the non-functional requirements for processes or individual business activities, such as performance and security.

Control on components could be of different types: assigning parameters of components, such as specifying a performance requirement in a WS-Policy expression for a ser-

vice; allocating more CPU resources for a component; replacing a component, like replacing a service reference with a trusted service.

2. Process flow governability

Process flow governability refers to the monitoring and control available for the flow of the process, which includes data flow and control flow. The primary objective of offering flow governability for process consumers is to

- Meet the functional requirements for business process automation

Control on process flow also could occur in different types for a business process, such as allowing to decide on an execution path, skipping or adding an activity in the process flow.

Both components and flow governabilities have overlapping concerns in their objectives. For example, skipping an unnecessary activity in a process flow also could improve the performance and cost, which are non-functional requirements. Replacing a service reference with a different logic also changes the process flow of the overall process.

4.3.2 Governability and process design

Architectural styles are used to guide the software design. In this section, we discuss the governability principle in connection with process design and also the SOA style.

4.3.2.1 Process design and development

Governability is one principle that is applied to the analysis design of processes in addition to the physical development design. In this section, we discuss the relationship between governability and process design and development.

The governability of a process could be offered by the underlying infrastructure with frameworks. Hence, a certain level of governability is automatically added to processes by

frameworks. However, for some reasons such as platform-independence, a restriction of frameworks in many cases processes might need to be especially developed to offer a certain level of governability. For example, processes are specially developed with integrated control logic to enable alternative replacement [57] [117], proxy services are required for processes to enable validation logic [167]. In this case, the final level of governability is highly dependent on the process development.

As we discussed, more governability directly increases process capability for meeting various requirements of process consumers. The governability is also closely related to process design. For example, alternative process control flow is allowed to be decided by the governance of consumers and should be designed in the process logic. The process design is also related with the supported framework of governance. For example, the process design does not have credit card number validation logic before a payment activity, and only a skipping activity action is offered by the framework for process governance. This process would be less capable, as most process consumers in most cases would like to add the validation logic before the payment activity. In the same way, if only adding activity is allowed in process governance, then adding many unnecessary or uncommon activities could also decrease the capability. Hence, process governability highly affects the process design.

4.3.2.2 Governability with impact on SOA principles

Business processes are composite services in SOA, and the governability and SOA principles have different goals in service design. In the following, we discuss the connection with SOA principles (which are described in Section 2.2), see Figure 4.1.

Governability and standardized service contract

Governability could be viewed as added functionality to increase the capability of original services for process consumers. Therefore, a process contract should include the base

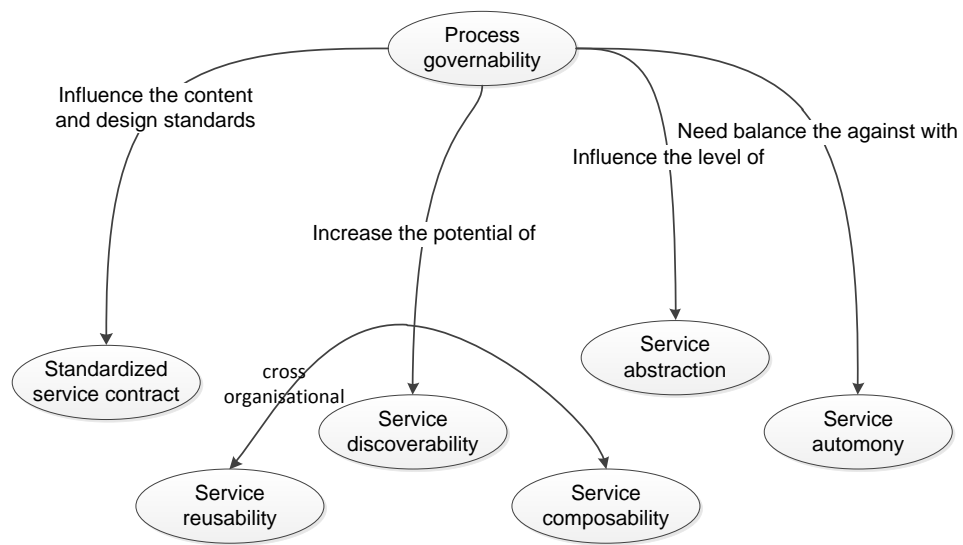


Figure 4.1: Process governability related to the SOA principles

information about governability that enables process consumers to govern the process. The original contract structure is impacted by technology, logic, etc., which is related to the governance approach offered. Both the content of the contract and the contract design will be directly influenced.

Governability and service abstraction

This principle gives emphasis to the need for exposing more process internal information to process consumers. This directly provides available and adequate information to enable process consumers to judge and govern the process behaviours. More information might need to be exposed to the outside for increasing the governability required. While service abstraction always looks for hiding information from others to minimize the contract coupling.

Governability and service autonomy

Service autonomy defines services exercising a high level of control over their underly-

ing runtime execution environment. Two primary benefits of raising the level of autonomy within a program are to increase its reliability, performance and behavioural predictability [11], as autonomy services are independent from external influences. A pure autonomy level service has an isolated and dedicated underlying logic and data resources from other parts of the enterprise [11]. The governability has an emphasis on transferring the control from self-governance by process itself to consumer-governance. As a consequence, process consumers could improve reliability, performance and predictability through governance. For example, a governance action that makes a service replacement within the process to an alternative service which is dedicated for the process only. However, it is difficult to achieve a high level autonomy without influences from other process consumers, since the primary character of cloud computing is sharing of resources to reduce the cost to individuals, underlying logic and data resources of processes are expected to be shared with other users with other processes, autonomy could be treated as second class in an SPA style architecture.

Governability and service discoverability, reusability, composability

The primary purpose of emphasizing process governability is also to support service reusability, but with across organisations. Therefore, when pursuing the application of this principle, we need to remain aware of the ultimate impact that effective process governability will have on realizing service process reuse in the cloud.

By increasing the governability, the demands of process consumers are easy to match in process discovery. Governability enlarges the coverage of the capability contract of the process, as additional goals can be achieved through governance. Increased discoverability can be realized on improved reusability and composability [11].

Role	Description
Business analyst	Determining the business policies or policies and processes needed for the business goals.
Developer	Implements the policies determined by the business analyst.

Table 4.2: User roles involved on process consumers

4.4 Roles and activities for business process automation

In the following section, we briefly describe roles and activities involved in the approach for business process automation under the SPA concept for process consumers in our vision for general cases. The governance in our case is policy based. The business policies are deployed in the governance components.

In general, there are two different types of roles that are involved at the consumer side (Table 4.2).

The basic activities of the roles involved are represented graphically in Table 4.3. The process could be customized by defined policies from process consumers. The policies also define the adaptations needed at process runtime. Other activities may also be included if needed. For example, a process verification activity [168] [169] is added before process integration. However, the process development, modification, re-/deployment activities will never be involved. It is clear that this approach is different from the business rules approach or policy first approaches with SOA governance (described in Figure 2.5).

4.5 Case study

In this section, we show a case study of the SOA application architecture applied to the SPA style.

The objective is to illustrate that the SPA style architecture offer process as a service in the cloud for multiple consumers with different requirements or policies. Also the issues of current approaches we observed (described in the problem statement chapter) are overcome.

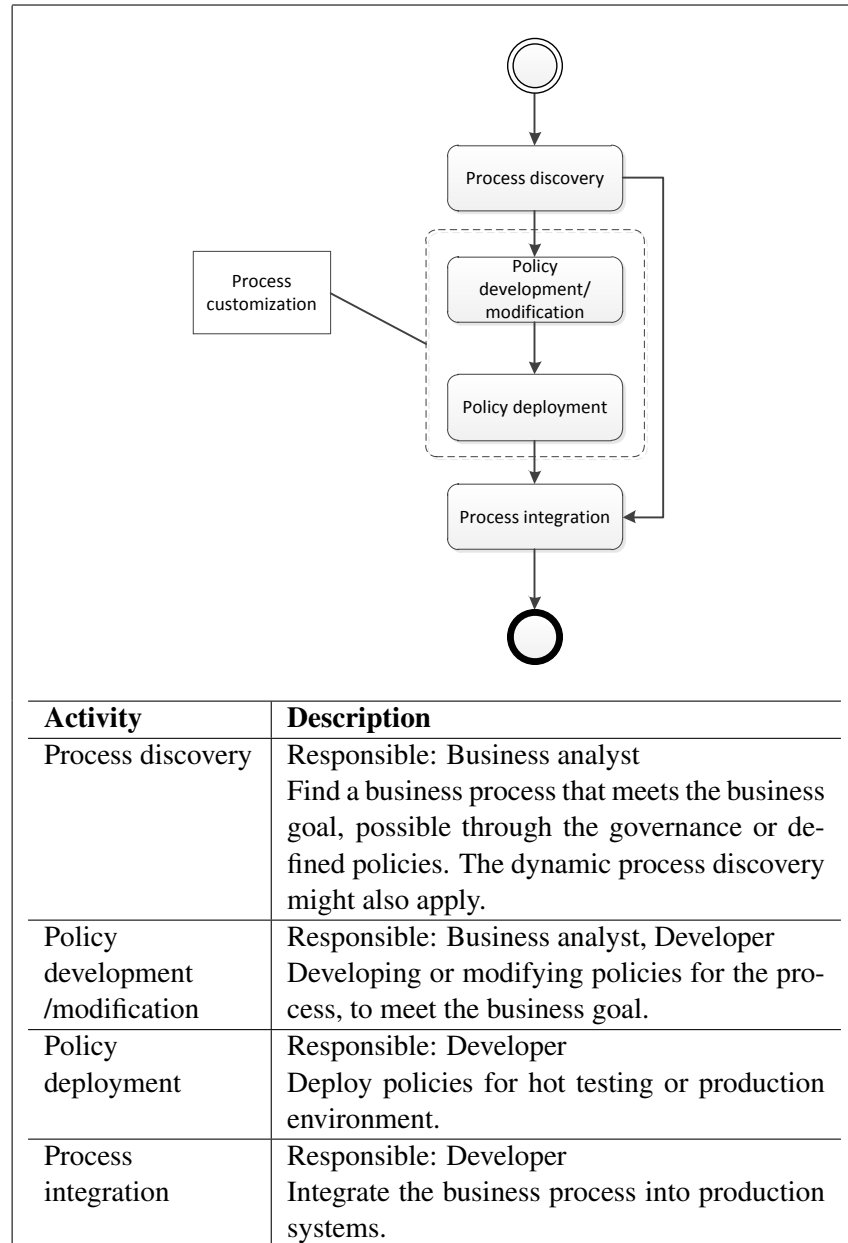


Table 4.3: Activities for automate a business process

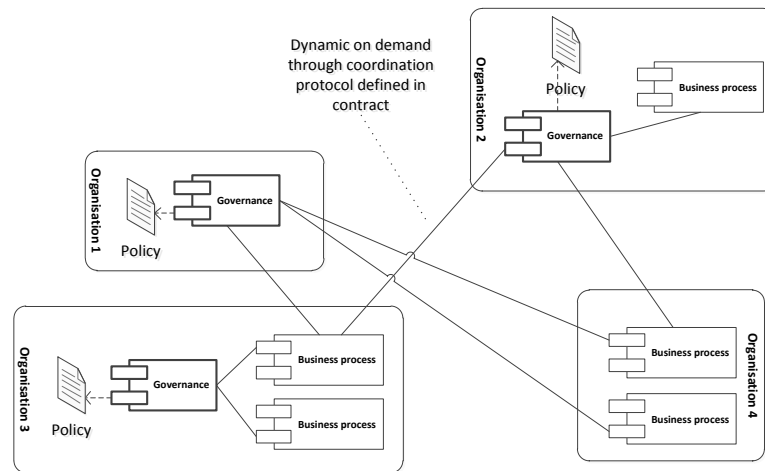


Figure 4.2: Application architecture diagram

We draw an SPA application architecture diagram of the case study, then we analyse the diagram and demonstrate how the objective can be achieved.

This is a generic scenario that involves different organisations which are process providers and/or consumers (Figure 4.2). The architecture elements are described in Section 4.2. We assume a standard coordination protocol is used for all organisations. All business processes are service processes with the process governability principle applied. SOA principles still apply, so the coordination protocol for governability for consumers is defined in the service contract.

From Figure 4.2, we can see these business processes are shared across organisations. In addition:

- The process governance will be associated with process requests from the process consumers dynamically on demand. The governance through protocols is defined in service contracts between providers and consumers. Process consumers could discover and request processes without pre-registration required to create a configuration database.

- Governability is described in the contract, but governance is the responsibility of process consumers. Process consumers govern the processes on their own. In our approach, policy compliance can be verified by consumers themselves.
- Process consumers freely define their own forms of policies on the consumer side. Defined policies of a consumer will be enforced in organisation wide processes, which include processes within organisations, and processes from external organisations. Policies are centralized and not process vendor specific.
- The policies stay inside of process consumers, hidden from process providers or any other parties. Only necessary controls or governance actions are sent to process providers through a protocol, but policies are not exposed to providers. For example, a provider is asked to cancel a process as a policy decision, but the provider does not know anything about the policies or the reason for cancelling. This maintains the privacy of policies.

From the above analysis, we can see the SPA style application architecture extending the SOA style offers process sharing in the cloud and overcomes the issues of current approaches we discussed (Chapter 3.4).

However, we assume the framework is built on standardized protocols to avoid vendor lock-in. As a consequence, the challenge might not only come from the technical side, but also from the business side in protocol standardization. In fact, an amount of draft specifications for cloud open standards are published in late 2010, for example specifications¹ from the OpenGroup. Moreover, there could be different degrees of governability offered by different process providers, so there are possible restrictions for process consumers switching processes from high governability processes to low governability processes.

¹details at <http://www.opengroup.org/cloudcomputing/>

4.6 Discussion of related work

In this section, we discuss work related to architectural styles. We also compare actual application architectures with policy based service systems, since our application architecture has similar characteristics to other policy approaches.

There are 21 network-based architectural styles in 5 categories which have been identified and studied by Roy Fielding [27]. To focus and narrow the scope, we only study styles under the code mobility concept, as they are network based architectural styles able to provide a degree of customizability for software architectures [163] [27]. These styles [163] include *remote evaluation*, *code on demand*, and *mobile agent*. The styles are distinguished in terms of interaction patterns that define the relocation and coordination among the components needed to perform a service, and give design paradigms for fundamental component interaction in distributed computing. In terms of component interactions from the view of distrusted computing, SPA is similar to remote evaluation, or might be viewed as policy based remote evaluation. However, all these styles just extract from the application scenarios, but do not define any principles of service and process design, like SOA or our work on SPA. The SPA style does not only add a remote governance component for a scenario of distributed computing. What is more important is that we define and profile the principle in connection with principles of the SOA style.

[2] [40] [41] as we discussed earlier do not detail the problem of process sharing with SOA. The FP7 NEXOF-RA ² specification development [170] adds *Service quality* in addition to the original eight principles as basic principles for enterprise SOA architecture design. Service quality defines a principle for service design with respect to quality characteristics in addition to functional requirements. *Governance* means services management which is mentioned in the RA as a factor that should be considered, but is not defined as a principle and is not analysed. The FP7 SOA4All project ³ [171] extends the eight SOA

²The NEXOF Reference Architecture <http://www.nexof-ra.eu/>

³Service-Oriented Architectures for All (SOA4All) <http://www.soa4all.eu/>

principles for the problem of services accessible for third-party usage in a global, dynamically changing environment. They define five additional principles (*distributed*, *openness*, *interoperability*, *user-centric* and *semantic* principles) to enhance the SOA style, which are different from the principle we defined.

Since our actual application architecture addresses the problem by means of policy based computing, we also compare it with policy enforcement application architectures in service systems. Table 4.4 shows the application architectures of XACML [21], business rules and other related work [172] [16] [173] [174] for policies of business processes, WS-Policy [15], and our policy approach.

We can see that with both XACML and the business rules approach, the policies are defined by process providers. The policies do not represent the requirements of external process consumers or multiple consumers. With WS-Policy approaches, only mutually accepted policies will be enforced on the provider side. Moreover, WS-Policy only focuses on policies with service components rather than business processes. In our approach, the external process consumers could define their own policies on business processes. The provider could also define policies for internal consumers. The details of a policy model in our approach will be described in the next chapter.

4.7 Conclusion

We presented the SPA architectural style with a principle - process governability aimed at enabling Process as a Service in the cloud. The principle is for the goal of attracting potential tenants with various requirements, and to increase the process customizability and adaptability for individual tenants and multiple tenants. Process consumers could use pre-defined processes, customize and adapt the processes according to the consumer needs through runtime governability, and remotely execute the processes in the cloud. SPA extends the SOA style on service process design. As a consequence, the process governability

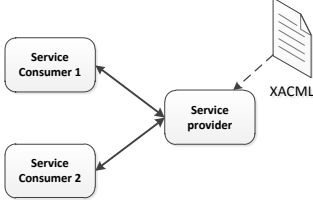
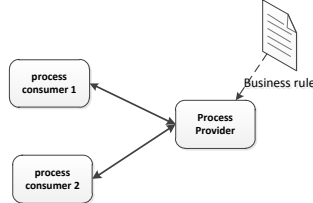
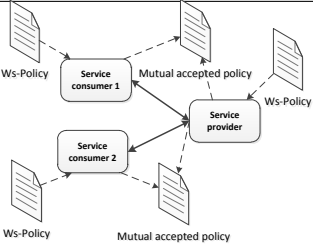
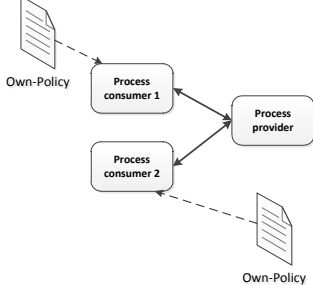
Policy approach	Application architecture	Policy focus	Policy by consumers	Policy by providers	Policy mutual understanding required
XACML		service	Not available	Available	Not required
Business rule and others		process	Not available	Available	Not required
WS-Policy		service	Available	Available	Required
Our SPA approach		process	Available	Available	Not required

Table 4.4: Comparing policy frameworks in Web service system

principle will affect orchestrated task services design with regard to SOA principles which have different goals.

Part III

Designing an architecture framework

Chapter 5

Policy model

5.1 Introduction

This chapter presents a policy model for process consumers to formalize business policies as a customization of business processes of process providers.

With the Process as a Service approach, the business policies are defined after business processes are ready for process consumers. This makes the policy-first process development approach for internal organisational processes, such as the conventional business rules approach, not applicable. We need a new policy model for consumers to formalize the business policies in pre-developed business processes. The new policy language could be viewed as a customization language of prepared business processes. The customization is achieved by means of runtime governance of business processes.

The defined policies are enforced in business processes of providers for the consumers. This is a superficial process level contract between process consumers and providers. The mechanism for process providers to carry out the superficial contract is a real contract defining the process governability. Hence the development of the policy model is based on a coordination protocol for runtime governance between process consumers and providers to achieve on-the-fly customization. The policy model is an approach for consumers in our

architectural framework. However, this is not a protocol which all consumers and providers must comply with.

The core of the policy model is providing a language model for process consumers to express business policies for existing business processes of providers as process customization metadata. Since XML is generally used in SOA to structure data [173], the language model is defined as a XML schema like other SOA specifications to enforce the syntax and format. It is used by business analysts and developers of process consumers (described in Chapter 4.4) to express different rule categories of policies in the XML language. In the first main part of this chapter, we are going to present the policy language model. This will introduce the rule categorization and the core components of the model, such as rule, policy, and the language syntax for each component of the model.

The policy language model provides features for policy developers for goals such as defining nested policies, defining policy sequences and resolve conflicts of multiple policies. This allows policy developers to express complex business policies, such as policy hierarchies. The implementations behind these features are a set of algorithms defined in the policy model. A section of this chapter will describe the related algorithms in the policy model.

This chapter also includes a case study section for the policy language model. In this section, we use the policy model to define the business policies of process Consumer_1 (as described in the problem statement chapter). The case study will demonstrate how various business policies are expressed in concrete policy languages.

This chapter is organized as follows. In Section 5.2, we introduce the basic information model and framework of the policy model. In Section 5.3 and 5.4, we describe the policy language model and related algorithms. Section 5.5 details the case study. In the remaining Sections (5.6, 5.7), we compare with related work and give conclusions.

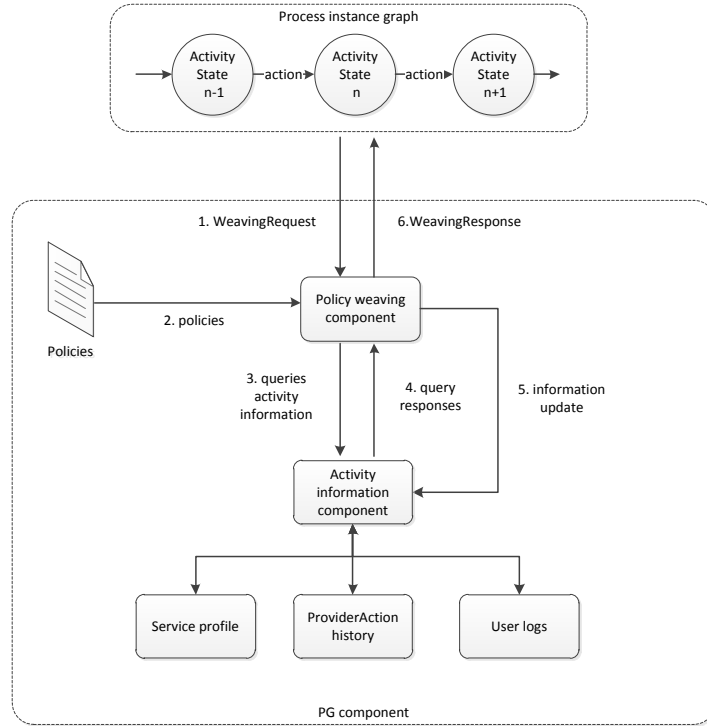


Figure 5.1: Information model and framework elements

5.2 The information model

Before we describe the language model of the policy, we first would like to describe the basic information model. Figure 5.1 shows the information model. This also describes the basic elements of the framework with a policy weaving of a process governance component *pg*.

- *Process instance graph* - It is a process runtime execution instance derived from an activity based process instance graph defined by the coordination protocol. The protocol will be detailed in a later coordination chapter. It sends a weaving request $weaving_{request} \in Weaving_{request}$ to the policy weaving component at governance states of activities.

- *Policies* - The requirements or customization of processes are described in policies, which will be carried out by both the process consumer and provider through a coordination framework.
- *Policy weaving component* - It weaves defined policies of the process consumer at process runtime in governance states of activities of the process. The weaving response $weaving_{response} \in Weaving_{response}$ as a part of a policy decision is sent back to the process instance as a part of a contract which needs to be carried out by the provider of the business process.
- *Activity information component* - It operates on information sources of activities for policy weaving.
- *Service profile SP* - It is an information source, providing service information of activities. It includes a service endpoint reference and service context information. The context information covers subcategories such as QoS or Platform.
- *Weaving history WH* - It is an information source, stores the $Weaving_{response}$ history of policy weaving.
- *User logs UL* - It is an information source, stores which relevant information created by user log actions.

5.3 The language model

Our policy model is influenced by the XACML specification, which also influenced many other proposed SOA policy models, such as [174] [173] [22]. The influence is especially with the three level structure (Rule, Policy, PolicySet) to support nested policies for different administrative levels, which would be a required feature of many organisations in policy development. Still, the XACML only focuses on the access control security aspect, and

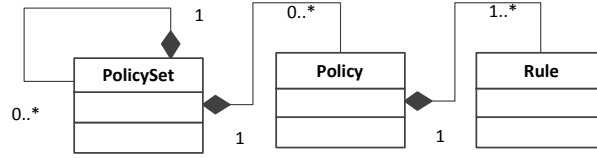


Figure 5.2: Core components of the policy language model

also extended work such as [22], are not process aware policy language as we discussed. The core components of our policy model are shown in Figure 5.2

These core components are described in the following subsections. The following two prefixes will be used in the policy language syntax description (more details in Appendix A).

spap is a prefix for policy schema namespace:

xmlns : spap = http : //www.computing.dcu.ie/mwang/spap

xsd is a prefix for W3C XML schema [175] namespace:

xmlns : xsd = http : //www.w3.org/2001/XMLSchema

5.3.1 Rule categorisation

Since the rules are used as the basic policy elements of the policy model, our policy modelling starts with different categories of rules needed for different aspects of business policies.

Business policies can be formalized as business rules for SOA governance. Business rule classifications (described in Chapter 2.4) only show different types of formal expressions of business rules. The purpose of these classifications is helping rule developers to discover, analyse, and design business rules [90], which are derived from business policies. The final goal is formulating the business policies in a formal rule language for a rule engine. These classifications do not give any concrete meaning to a business process. To develop our own policy language, we need a classification to find a common connection be-

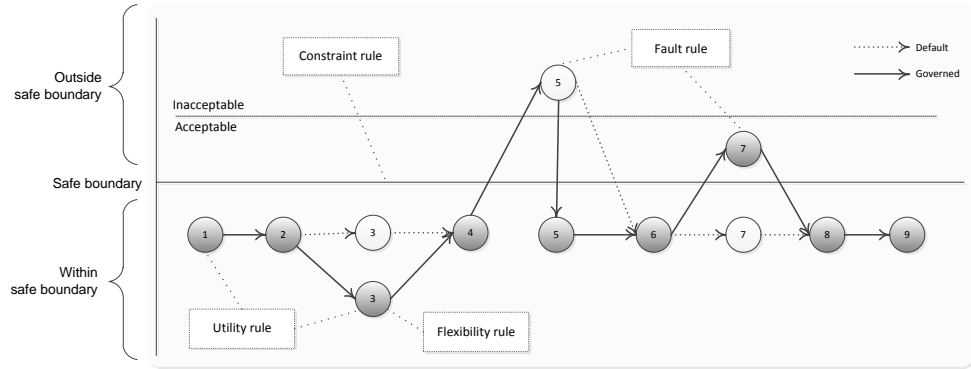


Figure 5.3: Rule categorization related to process execution

tween rules and processes that can be used for our policy model and coordination protocol development later on.

Based on aspects of autonomic computing [80] and state-action policy modelling [87], we have developed a categorization schema that allows us to categorise rules for processes into four different categories based on the safe boundary of a business process execution. The safe boundary is defined in terms of rules, derived from business regulations and requirements which the business must conform to. Figure 5.3 is used to explain the rule categories. It shows an execution example of a process which has nine execution steps. The circles represent the steps of process execution in different domains of the rule category. The numbers represent the sequenced numbers of steps.

We define $RU = RU^{flexibility} \cup RU^{constraint} \cup RU^{fault} \cup RU^{utility}$, where

1. $ru^{flexibility} \in RU^{flexibility}$ is a flexibility rule in the policy model

Flexibility rules are for business within the safe boundary - This rule category expresses the business decisions within the safe boundary of the execution. The execution steps continue forward after the decisions are made. It is used to specify variable business decision logic for various expected business scenarios such as different customer types, different types of post method use or frequently changing strategies (e.g., different discount rate over

times).

The business dynamics is the driving force. The purpose of this rule category is configuring business operations for business versatility and different business conditions.

2. $ru^{constraint} \in RU^{constraint}$ is a constraint rule in the policy model

Constraint rules are for the business *safe boundary* - this rule category defines the safe boundary of the process execution to restrict business behaviours. Constraint rules that specify assertions that must be satisfied in all steps of the process execution, e.g., the availability of the payment service must be above 99%.

The purpose of this rule category is to make sure that the business complies with relevant laws, regulations, and agreements, etc.

3. $ru^{fault} \in RU^{fault}$ is a fault rule in the policy model

Fault rules are for business *outside the safe boundary* - this type of rule defines the system responses when the process crosses the safe boundary, i.e., the constraints are violated. The business needs to decide what remedial strategy is required to avoid potential subsequent failure of the business goal. Since the constraint violations are viewed as 'faults' of process executions, this rule category is also known as the fault rule. The fault rule can be further divided for acceptable and unacceptable business cases outside the safe boundary.

The purpose of a fault rule is handling the violations of business regulation compliance that may have occurred.

4. $ru^{utility} \in RU^{utility}$ is a utility rule in the policy model

The *Utility rule* is the last category of rule that does not control or affect the process execution. It defines the additional or utility actions that might need to be associated with process execution.

The purpose of utility rules are, such as in the case of data collection for BAM, event notification.

After rules are categorized at a high level, our policy model can be modelled based on different categories of rules.

5.3.2 Rule

A *Rule* element $ru \in RU$ specifies the actual conditions under which defined governance actions are allow to be performed. It follow the ECA paradigm like other policy models [63] [107] [68] [111] [172]. Each rule contains *applicability predicates* and/or *condition predicates* as conditions to determine whether governance actions defined in the rule will be performed for a *weaving_{request}*.

Rules are building blocks of a policy. They must be encapsulated in a policy. A rule is made up of the tuple $\langle \underline{os}, \underline{ss}, \underline{cs}, \underline{acs}, \underline{fh}, \underline{rui}, \underline{pr}, \underline{de} \rangle$.

The main elements of a rule are:

- An *Objects* element $os \in OS$ and an *ActivityStates* element $ss \in SS$ define the *applicability predicates* of the rule, i.e., the E part of the ECA.
- A *Conditions* element $cs \in CS$ defines the *condition predicates* of the rule, i.e., the C part of the ECA.
- If either the *applicability predicates* or the *condition predicates* evaluate to *false* or *fault*, the governance actions contained in the *Actions* element $acs \in ACS$ of the ru will not be performed, i.e., the A part of the ECA.
- A fault handler element $fh \in FH$ which contains actions when faults occur during rule weaving, will be described later.

In addition to the main elements, a ru is defined and has the following attributes and elements.

- RuleId $ruid$ - a string identifies this ru .
- Priority pr - a positive integer denotes the priority weight of the ru . Default and minimal value is 0.

- Description $de \in DE$ - a description of this ru from policy developers.

5.3.2.1 Objects

An *Objects* element $os \in OS$ defines the governance targets of the business process. It specifies what the rule applies to. $os = \{(os'_k, sma) | k = 1, \dots, n; os'_k \in OS'; sma \in SMA\}$, where,

$sma \in SMA$ is a *SemanticMatchingAlgorithm* element, which will be described later.

$os'_k \in OS'$ is a disjunctive sequence element *ObjectsAnyOf*. $os'_k = \{os''_k, sma | k = 1, \dots, n; os''_k \in OS''; sma \in SMA\}$, where,

$os''_k \in OS''$ is a conjunctive sequence element *ObjectsAllOf*. $os''_k = \{o_n, sma | k = 1, \dots, n; o_n \in O, sma \in SMA\}$, where,

An *Object* element $o_n \in O$ represents a fundamental process element as a single governance target. $O = A \cup P \cup R \cup V$, where,

- An *Activity* $a \in A$ is an implementation of a business task through a Web service and defined by a tuple $\langle na, sma \rangle$. An a is identified by its name na . With $sma \in SMA$ as above.

- A *Process* $p \in P$ contains a set of activities executed in a specific sequence and defined by a tuple $\langle wso, wsa, sma \rangle$. A p is implemented by a composite service, which is identified by the *WSOperation* wso and/or the *WSAddress* wsa of the service reference of the process. $sma \in SMA$ as above.

A process itself could be an activity, not different from other activities. But in our policy language modelling, it is used to specify the policy scope. A p specific policy will only apply to the process p itself, but not to subprocesses of p .

- A *Resource* $r \in R$ is a business object in a process for transferring data between business partners or activities, and defined by a tuple $\langle na, sma \rangle$. A resource is identified by the *Name* na of the business object. $sma \in SMA$.

- A *Violation* $v \in V$ is an occurrence of violating constraints, and defined by a tuple $\langle tp, sma \rangle$. A violation is identified by the constrained aspect of a business process, i.e., $tp \in TP$. $sma \in SMA$.

$TP = VT \cup EVT$ is the *Type of violation*, and is defined as an extensible enumeration list:

- The *violation* VT is defined as a set of predefined violation types in a policy model for general business domains. These predefined violation types cover Functional, Quality of Service, Domain, and Platform context violation.

- * **Functional:** describes the violations of operational features of Web services. It is grouped into Syntax, Effect and Protocol violation.

1. Syntax violation: includes violation of input/output parameters that define the operations' messages and the data types for the parameters for invoking the service.
2. Effect violation: includes faults in terms of pre-conditions and post-conditions on service semantics, i.e. functional failure during an operation execution.
3. Protocol violation: refers to faults related to the consistent exchange of messages between services involved in a service composition to achieve their goals.

- * **QoS:** violation of end-to-end quality in service compositions including local services and global processes. It is grouped into QoS runtime, Financial/Business, Security, and Trust violation.

1. QoS runtime violation: violation of properties related to the execution of a service. This includes Performance, Reliability and Availability violations.
2. Financial/business violation: violation relates to the financial context

- which allows the assessment of a service from a financial or business perspective. This includes Cost, Reputation and Regulatory violations.
3. Security violation: violation of security requirements. This includes Integrity, Authentication, Nonrepudiation and Confidentiality violations.
 4. Trust violation: violation refers to failed establishment of trust relationships between a client and provider.
- * **Domain:** refers to application domains that need specific requirements to be met for services. It is grouped into semantic, linguistic, measures and standard violation.
1. Semantic violation: violations related to the semantic framework (i.e. concepts and their properties) in terms of vocabularies, taxonomies or ontologies.
 2. Linguistic violation: violation related to the language used to express queries, functionality and responses.
 3. Measures and standards violation: violation relates to locally used standards for measurements, currencies, etc.
- * **Platform:** violation related to the technical environment a service is executed in (includes classical technical platform faults). It is grouped into Device and Connectivity violation.
1. Device violation: refers to violations regarding the computer/hardware platform on which the service is provided.
 2. Connectivity violation: refers to violation regarding the network infrastructure used by the service to communicate.
- * **Unknown:** Violation is not determined. This could be defined by policy developers or results from constraint combining algorithms at runtime. The combining algorithms are described in a later section.

- The *ExtendViolationTypeStringPattern EVT* is defined as a free extendible violation type for policy developers. Extended types allows for late binding of new violation types, or further specifies the above predefined violation types from policy developers for special application requirements.

The Abstract syntax is shown in the following *string pattern* [176]: *pattern* =
Extend:\S.*

5.3.2.2 ActivityStates

An *ActivityStates* element $ss \in SS$ defines the governance states of activities of the business process. $ss = s_1 \vee \dots \vee s_n$ for $s_k \in S_g$ and $k = 1, \dots, n$ is a disjunctive set of *ActivityState* element. $S_g = S_g^{flexibility} \cup S_g^{constraint} \cup S_g^{fault}$, where,

- $S_g^{flexibility} = \{s_{man_{pre}val_{pre}}, s_{man_{pre}val_{post}}, s_{man_{post}val_{pre}}, s_{man_{post}val_{post}}\}$, a set of governance states for the $RU^{flexibility} \cup RU^{utility}$.
- $S_g^{constraint} = \{s_{validating_{pre}}, s_{validating_{post}}\}$, a set of governance states for the $RU^{constraint} \cup RU^{utility}$.
- $S_g^{fault} = \{s_{handling_{pre}}, s_{handling_{post}}, s_{cancelling}\}$, a set of governance states for the $RU^{fault} \cup RU^{utility}$.

Where,

1. *Validating-Pre/Post* $s_{validating_{pre}} / s_{validating_{post}}$ is a state of an activity execution for a *pg* component which enforces constraint rules defined for the activity. A Pre and Post denote the kind of validation that happens before and after the activity execution.
2. *Manipulating-Pre/Post-Validating-Pre/Post* $s_{man_{pre}val_{pre}} / s_{man_{pre}val_{post}} / s_{man_{post}val_{pre}} / s_{man_{post}val_{post}}$ is a state of an activity execution for a *pg* component which enforces

the flexibility rules defined for the activity through message manipulation. It contains a condition, Pre and Post denote that the manipulation happens before and after a validating pre/post state.

3. *Handling-Pre/Post* $s_{handling_{pre}} / s_{handling_{post}}$ is a state of an activity execution for a *pg* component enforcing the fault rules defined for the activity when violations occur. Pre and Post denote handling violations occurring at the $s_{validating_{pre}} / s_{validating_{post}}$ states.
4. *Cancelling* $s_{cancelling}$ is a state of an activity execution for a *pg* component enforcing the fault rules defined for the process if cancels its previous execution effect.

There are a number of states defined by the coordination protocol for an activity in business processes. S_g comprises the nine governance states involved with *PG* components. It is a core concept of our policy modelling. The remaining activity states of the protocol are also involved with policies, but processes do not interact with the *PG* components in the remaining states. More details of the states will be described in a later coordination chapter on the coordination protocol.

5.3.2.3 Conditions

A *Conditions* element $cs \in CS$ defines additional conditions for triggering actions on business processes. $cs = ce_1 \wedge \dots \wedge ce_n$ for $ce_k \in CE$ and $k = 1, \dots, n$ is a conjunctive sequence element.

A *ConditionExpression* element $ce \in CE$ is an XPath expression specifying a condition requirement on a data source $ds \in DS$. It returns a Boolean value on its evaluation. $ds = \langle weaving_{request}, SP, UL, WH \rangle$.

The XPath Expression complies with XPath 2.0 query syntax and should return a Boolean value. Boolean values result from utilizing XPath implicit conversion and specific Boolean expressions.

5.3.2.4 Actions

An *Actions* element $acs \in ACS$ defines a sequence of final actions on business processes for governance. $acs = \{ac_k | k = 1, \dots, n; ac_k \in \{CA \cup PA\}; \#\{ac_k | ac_k \in PA\} \leq 1\}$.

An *Action* element $ac \in AC$ defines a type of governance action. An ac can be either a consumer action CA or provider action PA , but at most one provider action for an acs .

Consumer action

A *ConsumerAction* element $ca \in CA$ is defined as an action performed within PG components or available on the consumer side for governance without directly controlling process executions. They are needed for $RU_{utility}$ of policies. For example, it is used to collect data required for subsequence control or monitoring. All consumer action elements as direct children of the acs will be weaved and executed immediately within a pg component when the rule is weaved. $CA = CA_{log} \cup CA_{suspend} \cup CA_{alert}$ is a set consumer actions supported by the framework and included in the policy language model.

- A log action $ca_{log} \in CA_{log}$ is to store information from a weaving request to the user log UL . The log level is an attribute to specify how much information needs to be stored.
- A suspend action $ca_{suspend} \in CA_{suspend}$ is to suspend the current service for the process consumer through updating the *ActiveTime* of the service of the service profile SP .

The *Time* attribute $t \in ca_{suspend}$ specifies a suspending time from the current time for the service. If $currentTime + t \leq ActiveTime$, then $ca_{suspend}$ will be ignored.

- An alert action $ca_{alert} \in CA_{alert}$ is to notify a relevant stakeholder of the processes about the current situation. It has a *MailTo* attribute specifying an email address of the stakeholder.

Provider Action

A *ProviderAction* element $pa \in PA$ is defined as an action in the policy model to directly control process executions on the provider side for governance requirements. They are needed for $RU_{flexibility}$, $RU_{constraint}$, and RU_{fault} of policies. For the policy framework, it also includes provider action types PA' resulting from policy combination or weaving, but they are not available in the policy language model for policy developers.

Both PA and PA' are defined based on and comply with the coordination protocol. The following gives the details of PA and PA' respectively:

PA contains a set of different provider action types in the policy model that are designed for different rule categories, thus are expected for different activity states. The activity states with rule categories are described above. The following table defines expected provider actions for PA from policy developers and their weaving in relation to activity states.

S_g	expected PA
$S_g^{flexibility}$	$PA_{manipulate}$
$S_g^{constraint}$	$PA_{validate} \cup PA_{violate}$
$s_{handling_{pre}} \in S_g^{fault}$	$PA_{ignore} \cup PA_{replace} \cup PA_{cancel} \cup PA_{skip}$
$s_{handling_{post}} \in S_g^{fault}$	$PA_{ignore} \cup PA_{replace} \cup PA_{cancel} \cup PA_{retry} \cup PA_{compensate}$
$s_{cancelling} \in S_g^{fault}$	$PA_{compensate}$

These provider action elements are described as follows:

- $pa_{manipulate} \in PA_{manipulate}$ - is a manipulate action to manipulate the *Resource* data $r \in weaving_{request}$ for the message adaptation requirement of the flexibility rules. The $pa_{manipulate}$ will be executed immediately during the rule weaving with the consumers, but the manipulated resource will be sent back to the process providers.

It contains a set of *Copy* operations that will be executed in an *all or none* manner. Exceptions caused by $pa_{manipulate}$ will trigger the fault handler, which will be described later.

Each copy operation can modify a single node element of resource data as a destination, which is specified by a *query* attribute of a *To* element of a *Copy* operation. A *query* attribute is an XPath 2.0 expression with a *Resource* as an input data source. The *From* element of a *Copy* can be either a *Literal* or an *XsltTrans* element. The *Literal* element allows giving a literal value to replace the destination node. The *XsltTrans* allows doing more complex data transformations or manipulations by utilizing the XSLT language [177].

- $pa_{validate} \in PA_{validate}$ - is a validate action defined for constraint rules to allow process execution steps to continue forward, if the current process instance is within the business safe boundary.
- $pa_{violate} \in PA_{violate}$ - is a violate action defined for constraint rules to guide the process execution into a violated state, if current process instance is outside the business safe boundary.

A $pa_{violate}$ contains a set of child elements, which denote a set violation types *TY* of the current process being violated. This has been described previously.

- $pa_{ingore} \in PA_{ingore}$ - is a remedial action defined for fault rules to guide the current process instance back to the business safe boundary without additional recovery. It ignores specified faults which do not affect the overall business goal.
- $pa_{replace} \in PA_{replace}$ - is a remedial action defined for fault rules to guide the process instance to replace the service reference of the current activity by an alternative. $pa_{replace} = \langle io, scs \rangle$, where the *InstanceOnly* *io* attribute is a Boolean value denoting two types of replace action, which are the Temporarily and Permanently replacement.

- Temporarily replace (*InstanceOnly=true*) is a process instance adaptation action for the activity instance of the current process instance, and is the default.

The service replacement is only applied for current activity in the current execution instance.

- Permanently replace (*InstanceOnly=false*) is a process adaptation action for continuous process improvement. The activity replacement is applied for the current instance and the following request instances in the current process.

The *ServiceConditions* *scs* element is used to specify a service reference for the activity implementation. $scs \in SCS$ will be described later.

- $pa_{compensate} \in PA_{compensate}$ - is a remedial action defined for fault rules to guide the current process instance to take a compensation action for the current activity by executing a compensate activity. A $scs \in pa_{compensate}$ is used to specify the service reference for the compensate activity.

Services are implementations of activities that must be assigned for every activity. A service should be specified with $PA_{replace}$ or $PA_{compensate}$ action type which contains service references of activities. A service selection mechanism in the language model is used to specify a service through the *ServiceConditions* *SCS* element.

A $scs \in SCS$, $scs = \{sce_k | k = 1, \dots, n; sce \in SCE\}$ is defined as a conjunctive sequence of conditions, which a service needs to satisfy for the activity to be executed. A $sce \in SCE$ is a *ServiceConditionExpression*, $sce = \langle ex, foc \rangle$ is defined as a single condition on a service. The *expression* attribute *ex* is an XPath 2.0 expression with the service profile *SP* as the data source. The *Force* attribute *foc* indicates if this condition is mandatory for a service selection. The default value is true.

- $pa_{cancel} \in PA_{cancel}$ - is a remedial action defined for fault rules to govern a current process instance to cancel the process execution.
- $pa_{retry} \in PA_{retry}$ - is a remedial action defined for fault rules to wait an amount of time before retrying the current fault causing activity. It has a *waitFor* attribute

complying with BPEL time expressions, which denotes the amount of time to wait before continuing the current activity execution. Immediate retry without waiting can be achieved by setting zero as the waiting time of a pa_{retry} .

The following shows that provider action types of PA' are not defined in the policy language model, but can be result from policy/rule combination and weaving. The reason is the coordination protocol taking composite provider actions and additional provider actions for a cache mechanism and the fault handling. More detail will be provided in the combining algorithms and coordination descriptions.

- $pa_{com+ign} \in PA_{com+ign}$ is a composite provider action which is composed of a $pa_{compensate}$ and a $pa_{ignnonre}$ action in a sequence.
- $pa_{com+rep} \in PA_{com+rep}$ is a composite provider action which is composed of a $PA_{compensate}$ and a $pa_{replace}$ action in a sequence.
- $pa_{undefined} \in PA_{undefined}$ indicates no Policy/Rule defined for the related activity state of an activity on the policy weaving, i.e. all policies or rules fail on an activity state evaluation for a $weaving_{request}$
- $pa_{unexpected} \in PA_{unexpected}$ indicates defined policies or rules which do not have any expected provider action in policy weaving. Thus, all specified provider actions in the defined rules or policies are not expected for a $weaving_{request}$. A $pa_{unexpected}$ becomes the result provider action for a $weaving_{request}$ on policy weaving in this case.
- $pa_{undetermined} \in PA_{undetermined}$ indicates a situation which cannot determine between $pa_{undefined}$, $pa_{unexpected}$ and expected provider actions for a $weaving_{request}$.

5.3.2.5 FaultHandler

A *FaultHandler* element $fh \in FH$ specifies what should be done if exceptions occur when evaluating a *Conditions* element $cs \in CS$, or executing a $pa_{manipulate} \in PA_{manipulate}$ action of a rule. Since these elements involve XPath and XSLT expressions defined by policy developers, exceptions may occur during rule weaving when that are mistakes in these expressions. If exceptions occur, the fault handler will be called and involves the current rule weaving.

A fault handler contains the *Actions* element $acs \in ACS$ which specifies a set of actions for fault handling on policy weaving. $acs = \{ac_k | k = 1, \dots, n; ac_k \in \{CA \cup PA^{fh}\}, \#\{ac_k | ac_k \in PA^{fh}\} \leq 1\}$, where $PA^{fh} \subset PA$. The following table defines the expected provider actions for fault handling with regard to different activity states of a weaving request.

S_g	expected PA^{fh}
$S_g^{flexibility} \cup S_g^{constraint}$	$PA_{validate} \cup PA_{violate}$
$s_{handling_{pre}} \in S_g^{fault}$	$PA_{ignore} \cup PA_{replace} \cup PA_{cancel} \cup PA_{skip}$
$s_{handling_{post}} \in S_g^{fault}$	$PA_{ignore} \cup PA_{replace} \cup PA_{cancel} \cup PA_{retry} \cup PA_{compensate}$
$s_{cancelling} \in S_g^{fault}$	$PA_{compensate}$

A provider action of a fault handler of a rule is expected to be in the same rule category as the provider action of the rule, except $RU^{flexibility}$. $PA_{manipulate}$ can not be defined in a fault handler. For handling exceptions for $PA_{manipulate}$ of a rule, $PA_{validate}$ or $PA_{violate}$ is expected in a fault handler.

If the fault handler is *Absent*, or an *Expected* provider action pa^{fh} is not included in the defined fault handler of a rule, a $pa_{undetermined}$ will be the provider action of the rule when the exceptions occur, i.e., a $pa_{undetermined}$ is the default expected provider action of a fault handler.

5.3.2.6 Obligations

An *Obligations* element $obs \in OBS$ contains a set of obligations. $obs = \{ob_k | k = 1, \dots, n; ob_k \in OB\}$.

An *Obligation* element $ob \in OB$, $ob = \{(ca_k, pa_t) | k = 1, \dots, n; ca_k \in CA; pa_t \in \{typeOf(PA \setminus PA_{manipulate})\}\}$ is specified as a set of consumer actions, which will only be executed on the consumer side when a type of provider action will be executed on the provider side for a weaving request on policy weaving. The type attribute pa_t specifies a type of provider action which the obligation is associated with. The provider actions are described in a previous section.

For a rule component, it should only have at most one obligation, as a rule can have only at most one provider action. It can not be associated with $PA_{manipulate}$ provider actions as they are executed on the consumer side on policy weaving.

The obligation elements might be merged when a Rule/Policy/PolicySet is weaved for a weaving request. When a type of provider action is finally decided for a weaving request for the provider, all obligation elements associated with the provider action will be selected and merged, then executed. For example, if two obligations defined for logging details when a process instance is cancelled, the details will only be logged once when the process is cancelled.

5.3.3 Policy

A *Policy* element $po \in PO$ is made up of the tuple $\langle os, ss, RU, obs, sa, cca, rca, poi, pr, de \rangle$. $os \in OS$ is an *Objects* element, $ss \in SS$ is an *ActivityStates* element, RU is a set of *Rule* element, $obs \in OBS$ is an *Obligations* element, $sa \in SA$ is a *SequencingAlgorithm* element, $cca \in CCA$ is a *ConstraintCombiningAlgorithm* element, $rca \in RCA$ is a *RemedyCombiningAlgorithm* element, poi is a string identifying the policy, pr is a positive integer denoting the priority weight of the policy, with a default and minimal value of 0. $de \in DE$ is a description element.

The *RU* and the other elements have been described before. We briefly describe the three algorithm elements in the following. More detail will be provided in the later algorithms section (Section 5.4).

- A $sa \in SA$ specifies the weaving sequence of Rule/Policy/PolicySet components within a Policy/PolicySet.
- A number of policies or rules contain constraint rules may applicable for a single weaving request. A $cca \in CCA$ specifies a procedure for combining possible multiple provider actions into a single provider action for the process provider in a weaving response.
- The $rca \in RCA$ specifies the combining algorithm for combining multiple provider actions defined for fault rules resulting from multiple Rule/Policy components to a single provider action on a weaving request.

5.3.4 PolicySet

A *PolicySet* element $pos \in POS$ is used to combine separate policies into a single combined policy. It allows policy developers to have nested policies. A policy set $pos \in POS$ is made up of the tuple $\langle os, ss, PS, POS, obs, sa, cca, rca, posi, pr, de \rangle$, $os \in OS$ is an *Objects* element, $ss \in SS$ is an *ActivityStates* element, PS is a set of *Policy* elements, POS is a set of *PolicySet* elements, $obs \in OBS$ is an *Obligations* element, $sa \in SA$ is a *SequencingAlgorithm* element, $cca \in CCA$ is a *ConstraintCombiningAlgorithm* element, $rca \in RCA$ is a *RemedyCombiningAlgorithm* element, poi is a string identifying the policy set, pr is a positive integer denoting the priority weight of the policy set, with a default and minimal value of 0. $de \in DE$ is a description element. All elements have been described before.

5.4 Related algorithms

There are a set of algorithm elements defined in the policy model. They allow policy developers to specify or configure the weaving behaviours. These algorithms are described in the following.

5.4.1 Semantic matching algorithm

The *SemanticMatchingAlgorithm* element $sma \in SMA$ specifies the algorithms used for semantic similarity measurements between policy objects and attributes of a weaving request. A policy can target a wide range of objects without matching the exact identification.

The reason behind this is that, some policy objects, such as activities, could have similar or same semantics, but not the same identification with different process providers. Through a semantic matching configuration, the policy can easily apply to interesting objects of all processes from different providers. For example, a new policy would be applied on ‘payment’ related activities. A semantic matching configuration could easily apply the policy on ‘process payment’, ‘pay’, or ‘repayment’ named activities.

A $sma = \langle ty, de \rangle$, where ty denotes the type of built-in algorithms, and de denotes the matching degree. The simple *Levenshtein distance* [178] is a built-in algorithm in our policy framework. Other algorithms which have such as a better accuracy on semantic similarity, performance, also could be used within the framework. However, defining and developing such algorithms is not in the scope of our work.

5.4.2 Sequencing algorithm

The *SequencingAlgorithm* element $sa \in SA$ specifies the weaving sequence on a collection of *Rule/Policy/PolicySet* components within a *Policy/PolicySet* component. Thus, this allows policy developers to specify an action execution sequence in a governance state of policy weaving.

A $sa \in SA$ has a type attribute indicating the type of build-in algorithms for sequencing. *Ordered* and *PriorityBased-QuickSort* are two basic build-in algorithms in our policy framework.

- The *Ordered* specifies a collection of *Rule/Policy/PolicySet* components within a *Policy/PolicySet* that are weaved in the order they are listed in a component.
- The *PriorityBased-QuickSort* specifies a collection of *Rule/Policy/PolicySet* components within a parent component that are weaved from a priority order with high priority coming first. The priority order is determined by the quicksort algorithm based on the *priority* attribute of each component. Components with same priority are weaved in the order they are listed by the quicksort algorithm.

5.4.3 Policy combining algorithms

In the case of multiple policies developed in different time periods, or developed by different policy developers, and nested policies, a potential problem is conflicting provider action types from multiple rules on a weaving request for process providers. As a consequence, we need policy combining algorithms to combine multiple provider actions into a single provider action for process providers on a *weaving_request* as a final decision.

The following defines the combining algorithms with regard to different activity states for actions .

S_g	Defined combining algorithm
$S_g^{flexibility}$	n/a
$S_g^{constraint}$	<i>CCA</i>
$S_g^{fault}/s_{cancelling}$	<i>RCA</i>
$s_{cancelling} \in S_g^{fault}$	default

The combining algorithm defined in a *PolicySet/Policy* will be simply ignored for an unrelated activity state of a weaving request.

Two policy combining algorithms are defined in the policy language model for constraint and fault rules respectively. These combining algorithms require policy developers involved to specify a type of combining algorithm. One way to solving policy conflicts is to assign explicit priority values to policies to define a precedence ordering [179]. This is done by a set of designed combining algorithms in XACML. The approach is also used in our policy model. The policy combining algorithms allow policy developers to give priority to different types of provider actions regarding constraint and fault rules.

The combining algorithm is not required for $S_g^{flexibility}$, as $PA_{manipulate}$ actions will be executed immediately when weaving the rules.

For the $s_{cancelling}$ state, since there is only one type of a provider action ($PA_{compensate}$) that is expected, a simple default combining algorithm is assigned without policy developer involvement. It does not combine different types of actions, but merges the same type of actions. In this case, the combination is based on the union of the child elements of $PA_{compensate}$ actions. Similarly, consumer actions of obligations are also merged and executed when a provider action is decided for a $weaving_{response}$.

In the following subsections, we describe the CCA and RCA .

5.4.3.1 Constraint combining algorithm

The *ConstraintCombiningAlgorithm* $cca \in CCA$ element is defined for combining provider actions with constraint rules. A cca has a type attribute that denotes the type of built-in algorithms which have different behaviours resulting in different combining conclusion. A cca is defined to be one of following types in our framework:

1. Pa-Violate-Override-Through-All
2. Pa-Validate-Override-Through-All
3. Pa-Violate-Unless-Pa-Validate-Through-All
4. Pa-Validate-Unless-Pa-Violate-Through-All

These types are described in the following subsections.

Pa-Violate-Overrides-Through-All

The *Pa-Violate-Override-Through-All* gives priority to $PA_{violate}$ actions over $PA_{validate}$ actions. The *Through-All* means that all of the rules or policies are weaved, even when the type of provider action has been decided. The purpose is

1. Gathering complete violation information which is needed for violation handling.
2. Making sure all necessary consumer actions are weaved.

The formal algorithm behaviour is defined in Algorithm 1.

Algorithm 1: Pa-Violate-Overrides-Through-All

input : a list of provider actions PA
output: a provider action pa

```

1  $V \leftarrow \emptyset$ ;
2  $One_{validate}, One_{undetermined}, One_{unexpected} \leftarrow false$ ;
3 foreach  $pa \in PA$  do
4   if  $pa \in PA_{violate}$  then  $V \leftarrow V \cup getViolations(pa)$ ;
5   if  $pa \in PA_{validate}$  then  $One_{validate} \leftarrow true$ ;
6   if  $pa \in PA_{undetermined}$  then  $One_{undetermined} \leftarrow true$ ;
7   if  $pa \in PA_{unexpected}$  then  $One_{unexpected} \leftarrow true$ ;
8 if  $V \neq \emptyset$  then
9   new  $pa_{violate}$ ;
10   $setViolations(pa_{violate}, V)$ ;
11  return  $pa_{violate}$ ;
12 if  $One_{validate} = true$  then return  $new\ pa_{validate}$ ;
13 if  $One_{undetermined} = true$  then return  $new\ pa_{undetermined}$ ;
14 if  $One_{unexpected} = true$  then return  $new\ pa_{unexpected}$ ;
15 return  $new\ pa_{undefined}$ ;
```

Informative description of the algorithm behaviour:

1. If any provider action is of $PA_{violate}$ type, then the result is a $pa_{violate}$ with merged violation elements of all $PA_{violate}$ type actions.

2. Otherwise, if any provider action is of $PA_{validate}$ type, then the result is a $pa_{validate}$.
3. Otherwise, if any provider action is of $PA_{undetermined}$ type, then the result is a $pa_{undetermined}$.
4. Otherwise, if any provider action is of $PA_{unexpected}$ type, then the result is a $pa_{unexpected}$.
5. Otherwise, the result is a $pa_{undefined}$.

Pa-Validate-Overrides-Through-All

The *Pa-Validate-Overrides-Through-All* is intended for those cases where a $PA_{validate}$ action should have priority over a $PA_{violate}$ action. It is similar to *Pa-Violate-Overrides-Through-All*, but gives priority to a $PA_{validate}$ action.

The algorithm behaviour is similar to *Pa-Violate-Overrides-Through-All* and is defined in Algorithm 2.

Algorithm 2: Pa-Validate-Overrides-Through-All

input : a list of provider actions PA
output: a provider action pa

```

1  $V \leftarrow \emptyset$ ;
2  $One_{undetermined}, One_{unexpected} \leftarrow false$ ;
3 foreach  $pa \in PA$  do
4   if  $pa \in PA_{validate}$  then return  $pa$ ;
5   if  $pa \in PA_{violate}$  then  $V \leftarrow V \cup getViolations(pa)$ ;
6   if  $pa \in PA_{undetermined}$  then  $One_{undetermined} \leftarrow true$ ;
7   if  $pa \in PA_{unexpected}$  then  $One_{unexpected} \leftarrow true$ ;
8 if  $V \neq \emptyset$  then
9    $new\ pa_{violate}$ ;
10   $setViolations(pa_{violate}, V)$ ;
11  return  $pa_{violate}$ ;
12 if  $One_{undetermined} = true$  then return  $new\ pa_{undetermined}$ ;
13 if  $One_{unexpected} = true$  then return  $new\ pa_{unexpected}$ ;
14 return  $new\ pa_{undefined}$ ;
```

Pa-Violate-Unless-Pa-Validate-Through-All

The *Pa-Violate-Unless-Pa-Validate-Through-All* is intended to give a strict final decision with $PA_{violate}$ as the default. The *Through-All* makes sure that all rules or policies are

weaved, even if the type of provider action has been decided.

The formal algorithm behaviour is defined in Algorithm 3

Algorithm 3: Pa-Violate-Unless-Pa-Validate-Through-All

input : a list of provider actions PA

output: a provider action pa

```

1  $V \leftarrow \emptyset$ ;
2 foreach  $pa \in PA$  do
3   if  $pa \in PA_{validate}$  then return  $pa$ ;
4   if  $pa \in PA_{violate}$  then  $V \leftarrow V \cup getViolations(pa)$  ;
5 new  $pa_{violate}$ ;
6 if  $V = \emptyset$  then  $V \leftarrow v_{unknow} \in VT$ ;
7  $setViolations(pa_{violate}, V)$ ;
8 return  $pa_{violate}$ ;
```

Informative description of the algorithm behaviour:

1. If any provider action is of $PA_{validate}$ type, then the result is a $pa_{validate}$.
2. Otherwise, if any provider action is of $PA_{violate}$ type, then the result is $pa_{violate}$ with merged violation elements of all $PA_{violate}$ actions.
3. Otherwise, the result is a $pa_{violate}$, with an *Unkown* type violation.

Pa-Validate-Unless-Pa-Violate-Through-All

The *Pa-Violate-Unless-Pa-Validate-Through-All* is intended to give a strict final decision with $PA_{validate}$ as default.

The formal algorithm behaviour is similar to the *Pa-Violate-Unless-Pa-Validate-Through-All*. It is defined in Algorithm 4.

Algorithm 4: Pa-Validate-Unless-Pa-Violate-Through-All

input : a list of provider actions PA

output: a provider action pa

```
1  $V \leftarrow \emptyset$ ;  
2 foreach  $pa \in PA$  do  
3   if  $pa \in PA_{violate}$  then  $V \leftarrow V \cup getViolations(pa)$  ;  
4 if  $V \neq \emptyset$  then  
5   new  $pa_{violate}$ ;  
6    $setViolations(pa_{violate}, V)$ ;  
7   return  $pa_{violate}$ ;  
8 return new  $pa_{validate}$ ;
```

5.4.3.2 Remedy combining algorithm

The *RemedyCombiningAlgorithm* $rca \in RCA$ element is defined for combining provider actions resulting from fault rules. A $rca = \langle ty, DS \rangle$, where ty is an attribute denoting the type of built-in algorithms which have different behaviours resulting from different combining conclusions. DS specifies a defined sequence of provider actions as an input parameter of one type of algorithm. A rca is defined to be one of following types in our framework:

1. Defined-Sequence-Overrides-Through-All
2. Pa-Ignore-Unless-Defined-Sequence-Through-All
3. Pa-Cancel-Unless- Defined-Sequence-Through-All

These are described in the following.

Defined-Sequence-Overrides-Through-All

The *Defined-Sequence-Overrides-Through-All* gives a priority ranking according to the sequence of the defined provider action types for fault rules. The first action type in the sequence has the highest priority. Hence, when a list of remedies is available from defined related policies, the one with the highest priority will be chosen finally. The *Through-All* denotes that all rules or policies are weaved, even if the type of provider action has been decided.

The formal algorithm behaviour is defined in Algorithm 5.

Algorithm 5: Defined-Sequence-Overrides-Through-All

input : a list of provider actions PA , a sequence of provider actions DS

output: a provider action pa

```

1  $One_{ignore}, One_{retry}, One_{skip}, One_{cancel}, One_{unexpected}, One_{undetermined} \leftarrow$ 
    $false$ ;
2  $CE_{replace}^{instance}, CE_{replace}, CE_{comp} \leftarrow \emptyset$ ; /* CE denotes ConditionExpressions */
3  $Time_{waitFor} \leftarrow 0$ ;
4 foreach  $pa \in PA$  do
5   if  $pa \in PA_{ignore} \vee pa \in PA_{com+ign}$  then
6      $One_{ignore} \leftarrow true$ ;  $CE_{comp} \leftarrow CE_{comp} \cup getConExpsForCom(pa)$ ;
7   if  $pa \in PA_{retry}$  then
8      $One_{retry} \leftarrow true$ ;
9     if  $getTime(pa) > Time_{waitFor}$  then  $Time_{waitFor} \leftarrow getTime(pa)$ ;
10  if  $pa \in PA_{comp}$  then  $CE_{comp} \leftarrow CE_{comp} \cup getConExpsForCom(pa)$ ;
11  if  $pa \in PA_{replace} \vee pa \in PA_{com+rep}$  then
12     $CE_{comp} \leftarrow CE_{comp} \cup getConExpsForCom(pa)$ ;
13    if  $isInstanceOnly(pa)$  then
14       $CE_{replace}^{instance} \leftarrow CE_{replace}^{instance} \cup getConExpsForRep(pa)$ ;
15    else  $CE_{replace} \leftarrow CE_{replace} \cup getConExpsForRep(pa)$ ;
16  if  $pa \in PA_{skip}$  then  $One_{skip} \leftarrow true$ ;
17  if  $pa \in PA_{cancel}$  then  $One_{cancel} \leftarrow true$ ;
18  if  $pa \in PA_{undetermined}$  then  $One_{undetermined} \leftarrow true$ ;
19  if  $pa \in PA_{unexpected}$  then  $One_{unexpected} \leftarrow true$ ;
/* continued with next part on the next page */ ;

```

Informative description of the algorithm behaviour:

1. If any provider action is the first action type defined in the sequence, the result is an instance of the type action with merged child elements.
2. Otherwise, checking for the second action type defined in the sequence. It iterates until the last type of action defined in the sequence.
3. Otherwise, if any provider action is of the type of $PA_{compensate}$, the result is a $pa_{compensate}$ with merged children elements.
4. Otherwise, if any provider action is of $PA_{undetermined}$ type, then the result is a

```

/* continue from the previous page                                     */
20 foreach  $pa \in DS$  do
21   if  $pa \in PA_{ignore} \wedge One_{ignore}$  then
22     if  $CE_{comp} \neq \emptyset$  then
23        $new\ pa_{com+ign}; setConExps(pa_{com+ign}, CE_{comp});$ 
24       return  $pa_{com+ign};$ 
25     return  $new\ pa_{ignore};$ 
26   if  $pa \in PA_{retry} \wedge One_{retry}$  then
27      $new\ pa_{retry}; setTime(pa_{retry}, Time_{waitFor});$ 
28     return  $pa_{retry};$ 
29   if  $CE_{replace} \neq \emptyset$  then
30     if  $CE_{comp} \neq \emptyset$  then
31        $new\ pa_{com+rep}; setConExps(pa_{com+rep}, CE_{comp}, CE_{replace});$ 
32        $setInstanceOnly(pa_{com+rep}, false);$ 
33       return  $pa_{com+rep};$ 
34      $new\ pa_{replace}; setConExps(pa_{replace}, CE_{replace});$ 
35      $setInstanceOnly(pa_{replace}, false);$ 
36     return  $pa_{replace};$ 
37   if  $CE_{replace}^{instance} \neq \emptyset$  then
38     if  $CE_{comp} \neq \emptyset$  then
39        $new\ pa_{com+rep}; setConExps(pa_{com+rep}, CE_{comp}, CE_{replace});$ 
40       return  $pa_{com+rep};$ 
41      $new\ pa_{replace}; setConExps(pa_{replace}, CE_{replace});$ 
42     return  $pa_{replace};$ 
43   if  $pa \in PA_{skip} \wedge One_{skip}$  then return  $new\ pa_{skip};$ 
44   if  $pa \in PA_{cancel} \wedge One_{cancel}$  then return  $new\ pa_{cancel};$ 
45   if  $CE_{comp} \neq \emptyset$  then
46      $new\ pa_{compensate}; setConExps(pa_{compensate}, CE_{comp});$ 
47     return  $pa_{compensate};$ 
48   if  $One_{undetermined} = true$  then return  $new\ pa_{undetermined};$ 
49   if  $One_{unexpected} = true$  then return  $new\ pa_{unexpected};$ 
50   return  $new\ pa_{undefined};$ 

```

$pa_{undetermined}$.

5. Otherwise, if any provider action is of $PA_{unexpected}$ type, then the result is a $pa_{unexpected}$.
6. Otherwise, the result is a $pa_{undefined}$.

Pa-Ignore-Unless-Defined-Sequence-Through-All

The *Pa-Ignore-Unless-Defined-Sequence-Overrides-Through-All* is intended to give a strict final decision with PA_{ignore} as default. The algorithm is similar to the *Defined-Sequence-Overrides-Through-All*, which gives a priority ranking according to the sequence of defined actions. The algorithm assigns a pa_{ignore} as a default provider action. In this case, if no remedy is found from defined related policies or rules, the violations will be ignored.

The algorithm behaviour is similar to Algorithm 5. The formal algorithm behaviour is defined in Algorithm 6.

Algorithm 6: Pa-Ignore-Unless-Defined-Sequence-Through-All

input : a list of provider actions PA , a sequence of provider actions DS
output: a provider action pa

```

42 ... /* same as Algorithm 5 until line 42                                */
43 if  $CE_{comp} \neq \emptyset$  then
44     new  $pa_{com+ign}$ ;
45     setConExps( $pa_{compensate}$ ,  $CE_{com+ign}$ );
46     return  $pa_{com+ign}$ ;
47 return new  $pa_{ignore}$ ;

```

Pa-Cancel-Unless-Defined-Sequence-Through-All

The *Pa-Cancel-Unless-Defined-Sequence-Through-All* is intended to give a strict final decision with PA_{cancel} as default. The algorithm is similar to the *Pa-Ignore-Unless-Defined-Sequence-Overrides-Through-All*. The algorithm assigns a pa_{cancel} as the default provider action. Hence, it will cancel the process instance which has violations not covered by any fault policies or rules.

The algorithm behaviour is similar to Algorithm 5. The formal algorithm behaviour is defined in Algorithm 7.

Algorithm 7: Pa-Cancel-Unless-Defined-Sequence-Through-All

input : a list of provider actions PA , a sequence of provider actions DS

output: a provider action pa

42 ... /* same as Algorithm 5 until line 42

*/

43 **return** new pa_{cancel} ;

5.5 Case study

In this section, we show some case studies on expressing concrete business policies with our policy language.

5.5.1 Objective

The objective is to further explain the policy model with examples to demonstrate how business policies are expressed in our policy language, and evaluating if various aspects of business policies can be covered.

We use the process Consumer_1 as the case study setup (described in Chapter 3). We use four different cases to cover configuration, protection, optimization, and healing all four autonomic aspects requirements that arise from business policies. Using these cases, we can demonstrate that the policy model can express various aspects of business policies.

The following are policy examples for Consumer_1.

5.5.2 Approach

5.5.2.1 Case 1: configuration on service references of activities

Business policy:

A receipt needs be issued to buyers for every checkout.

The discovered BPEL processes logic prepared by process providers has met the requirements of the above business policies as we assumed. Still, we could configure or lock default settings on Web service endpoint references assigned with activities of the process. For this case, we express the business policy in a policy set containing two policies. The first policy contains constraint rules for the correct references of activities. The second policy contains fault rules for handling the violations of first constraint policy.

Listing 5.1 shows a defined PolicySet *orderInspectionLockingPolicySet3* that contains two policies for this case study. The policy set targets the *purchase order inspection* activity with any activity states. We only demonstrate the policies for one activity in this case. Policies can be defined for other activities in a similar way.

Listing 5.1: orderInspectionLockingPolicySet3

```

1
2 <p1:PolicySet policySetId="orderInspectionLockingPolicySet3" priority="0">
3
4   <p1:Description>locking provider default setting on a activity</
5     p1:Description>
6   <p1:Objects>
7     <p1:ObjectsAnyOf>
8       <p1:ObjectsAllOf>
9         <p1:Activity>
10          <Name>purchase order inspection</Name>
11        </p1:Activity>
12      </p1:ObjectsAllOf>
13    </p1:ObjectsAnyOf>
14  </p1:Objects>
15  <p1:ActivityStates/>
16
17  <p1:Policy policyId="constraintOrderInspectionPolicy3" priority="0">...</
18    p1:Policy>
19  <p1:Policy policyId="orderInspectionFaultPolicy3" priority="0">...</
20    p1:Policy>
21  <p1:ConstraintCombiningAlgorithm type="Pa-Violate-Override-Through-All"></
22    p1:ConstraintCombiningAlgorithm>
23  <p1:RemedyCombiningAlgorithm type="Pa-Cancel-Unless-Defined-Sequence-Through
24    -All">
25    <DefinedSequenceElement>Pa-Replace</DefinedSequenceElement>
26    <DefinedSequenceElement>Pa-Ignore</DefinedSequenceElement>
27    <!-- ... more DefinedSequenceElement ... -->
28  </p1:RemedyCombiningAlgorithm>
29  <p1:SequencingAlgorithm type="Ordered"></p1:SequencingAlgorithm>
30 </p1:PolicySet>

```

Listing 5.2 shows the first policy - *constraintOrderInspectionPolicy3*. The policy is

restricted to the $s_{validating_{pre}}$ state, defines the constraint validation before the activity execution. Two constraint rules are defined in the policy. There are two *ConditionExpression* elements defined as two conditions on the first rule *constraintValidateRule3* (line 8). The first condition checks if the operation name of the service reference is correct (line 10). The second condition checks if the service address of the reference is correct (line 11). If both conditions are true, a $pa_{validate}$ action is expected in this case. The second rule - *constraintViolateRule3* (line 18) defines a *Functional:Protocol* violation with a $pa_{violate}$ action as an expected action for incorrect service reference of the activity. Since the policy is specified with the *Pa-Violated-Unless-Pa-Validate-Through-All* constraints combining algorithm (line 28), a $pa_{violate}$ will be the final provider action for this policy if the reference cannot be validated.

Listing 5.2: constraintOrderInspectionPolicy3

```

1  <p1:Policy policyId="constraintOrderInspectionPolicy3" priority="0">
2    <p1:Objects/>
3    <p1:ActivityStates>
4      <p1:ActivityState>Validating-Pre</p1:ActivityState>
5    </p1:ActivityStates>
6
7    <p1:Rule priority="0" ruleId="constraintValidateRule3">
8      <p1:Conditions>
9        <p1:ConditionExpression>/WeavingRequest/Activity/serviceReference/
10         Operation='orderInspection'</p1:ConditionExpression>
11        <p1:ConditionExpression>/WeavingRequest/Activity/serviceReference/Ws
12         -address='http://localhost:8080/ws/OrderInspectionService'</
13         p1:ConditionExpression>
14      </p1:Conditions>
15      <p1:Actions>
16        <p1:Pa-Validate/>
17      </p1:Actions>
18    </p1:Rule>
19
20    <p1:Rule priority="0" ruleId="constraintViolateRule3">
21      <p1:Conditions/>
22      <p1:Actions>
23        <p1:Pa-Violate>
24          <p1:Violation>
25            <Type>Functional:Protocol</Type>
26          </p1:Violation>
27        </p1:Pa-Violate>
28      </p1:Actions>
29    </p1:Rule>
30
31    <p1:ConstraintCombiningAlgorithm type="Pa-Violate-Unless-Pa-Validate-
32      Through-All"></p1:ConstraintCombiningAlgorithm>
33    <p1:RemedyCombiningAlgorithm type="Pa-Cancel-Unless-Defined-Sequence-
34      Through-All">

```

```

30     <DefinedSequenceElement>Pa-Cancel</DefinedSequenceElement>
31     <DefinedSequenceElement>Pa-Replace</DefinedSequenceElement>
32     <!-- ... more DefinedSequenceElement ... -->
33 </p1:RemedyCombiningAlgorithm>
34 <p1:SequencingAlgorithm type="Ordered"></p1:SequencingAlgorithm>
35 </p1:Policy>

```

Listing 5.3 shows the second policy - *orderInspectionFaultPolicy3*. It contains a fault rule to reconfigure the expected service reference of the activity by handling the violation we defined above. More specifically, handling violations are caused by incorrect service references before the *purchase order inspection* activity execution. The fault rule *remedyRule3* (line 6) permanently (*instanceOnly = false*) assigns a correct endpoint reference to the activity by a *pa_replace* action if the *Functional:Protocol* violation occurring is caused by the above constraint policy. The *ServiceConditions* of the *pa_replace* specifies the correct endpoint reference.

Listing 5.3: orderInspectionFaultPolicy3

```

1  <p1:Policy policyId="orderInspectionFaultPolicy3" priority="0">
2    <p1:Objects></p1:Objects>
3    <p1:ActivityStates></p1:ActivityStates>
4
5    <p1:Rule priority="0" ruleId="remedyRule3">
6      <p1:Objects>
7        <p1:ObjectsAnyOf>
8          <p1:ObjectsAllOf>
9            <p1:Violation>
10              <Type>Functional:Protocol</Type>
11            </p1:Violation>
12          </p1:ObjectsAllOf>
13        </p1:ObjectsAnyOf>
14      </p1:Objects>
15      <p1:ActivityStates>
16        <p1:ActivityState>Handling-Pre</p1:ActivityState>
17      </p1:ActivityStates>
18      <p1:Conditions/>
19      <p1:Actions>
20        <p1:Pa-Replace InstanceOnly="false">
21          <p1:ServiceConditions>
22            <p1:ServiceConditionExpression force="true" expression="/
23              serviceReference/Operation='orderInspection'"/>
24            <p1:ServiceConditionExpression force="true" expression="/Ws-
25              address='http://localhost:8080/ws/OrderInspectionService'"/>
26          </p1:ServiceConditions>
27        </p1:Pa-Replace>
28      </p1:Actions>
29    </p1:Rule>
30
31    <p1:ConstraintCombiningAlgorithm type="Pa-Violate-Override-Through-All">
32      </p1:ConstraintCombiningAlgorithm>
33  </p1:Policy>

```

```

30     <pl:RemedyCombiningAlgorithm type="Pa-Cancel-Unless-Defined-Sequence-
      Through-All">
31         <DefinedSequenceElement>Pa-Replace</DefinedSequenceElement>
32         <DefinedSequenceElement>Pa-Ignore</DefinedSequenceElement>
33         <!-- ... more DefinedSequenceElement ... -->
34     </pl:RemedyCombiningAlgorithm>
35     <pl:SequencingAlgorithm type="Ordered"/>
36 </pl:Policy>

```

5.5.2.2 Case 2: configuration on flow logic and resource message

Business Policy:

Free parcel shipping for orders with a total over 2000 euro

This business policy could be implemented with the activity instance and message adaptation on the *Assign shipping method* activity. We defined 4 rules in a policy for this objective. Listing 5.4 shows the policy - *freeShippingPolicy5*. The policy has both an *Assign shipping method* activity and an *Order* resource as the policy objective (lines 2-13).

Listing 5.4: freeShippingPolicy5

```

1 <pl:Policy policyId="freeShippingPolicy5" priority="0">
2   <pl:Objects>
3     <pl:ObjectsAnyOf>
4       <pl:ObjectsAllOf>
5         <pl:Activity>
6           <Name>Assign shipping method</Name>
7         </pl:Activity>
8         <pl:Resource>
9           <Name>Order</Name>
10        </pl:Resource>
11      </pl:ObjectsAllOf>
12    </pl:ObjectsAnyOf>
13  </pl:Objects>
14  <pl:ActivityStates></pl:ActivityStates>
15
16  <pl:Rule priority="0" ruleId="constraintFreeShippingRule5">...</pl:Rule>
17
18  <pl:Rule priority="0" ruleId="skipFreeShippingRule5">...</pl:Rule>
19
20  <pl:Rule priority="0" ruleId="manipulateFreeShippingRule5">...</pl:Rule>
21
22  <pl:Rule priority="0" ruleId="remedyFreeShippingRule5">...</pl:Rule>
23
24  <pl:ConstraintCombiningAlgorithm type="Pa-Violate-Override-Through-All"></
    pl:ConstraintCombiningAlgorithm>
25  <pl:RemedyCombiningAlgorithm type="Pa-Cancel-Unless-Defined-Sequence-
    Through-All">
26    <DefinedSequenceElement>Pa-Cancel</DefinedSequenceElement>
27    <DefinedSequenceElement>Pa-Skip</DefinedSequenceElement>
28    <!-- ... more DefinedSequenceElement ... -->

```



```

29     </p1:RemedyCombiningAlgorithm>
30     <p1:SequencingAlgorithm type="Ordered"></p1:SequencingAlgorithm>
31 </p1:Policy>

```

The first and second rules are shown in Listing 5.5. They are used for activity adaptation - skipping an unwanted activity for some process instances. The *constraintFreeShippingRule5* (line 1) specifies a condition on the total amount of the order before the object activity. If order totals > 2000, then a *pa_violate* action would be executed at the process provider. In this case, an extend violation type *Extend:FreeShipping:Skip* is specified for the *pa_violate* action. The second rule - *skipFreeShippingRule5* (line 17) is a fault rule that is defined for handling the extended violation type with a *pa_skip* action. The process instance is asked to skip the current activity execution.

Listing 5.5: *constraintFreeShippingRule5* and *skipFreeShippingRule5*

```

1  <p1:Rule priority="0" ruleId="constraintFreeShippingRule5">
2    <p1:ActivityStates>
3      <p1:ActivityState>Validating-Pre</p1:ActivityState>
4    </p1:ActivityStates>
5    <p1:Conditions>
6      <p1:ConditionExpression>//Order/Total>2000</p1:ConditionExpression>
7    </p1:Conditions>
8    <p1:Actions>
9      <p1:Pa-Violate>
10        <p1:Violation>
11          <Type>Extend:FreeShipping:Skip</Type>
12        </p1:Violation>
13      </p1:Pa-Violate>
14    </p1:Actions>
15  </p1:Rule>
16
17  <p1:Rule priority="0" ruleId="skipFreeShippingRule5">
18    <p1:Objects>
19      <p1:ObjectsAnyOf>
20        <p1:ObjectsAllOf>
21          <p1:Violation>
22            <Type>Extend:FreeShipping:Skip</Type>
23          </p1:Violation>
24        </p1:ObjectsAllOf>
25      </p1:ObjectsAnyOf>
26    </p1:Objects>
27    <p1:ActivityStates>
28      <p1:ActivityState>Handling-Pre</p1:ActivityState>
29    </p1:ActivityStates>
30    <p1:Conditions></p1:Conditions>
31    <p1:Actions>
32      <p1:Pa-Skip></p1:Pa-Skip>
33    </p1:Actions>
34  </p1:Rule>

```

The third and fourth rules are designed for the message adaptation (Listing 5.6). They assign free shipping on the Order resource or business object. The *manipulateFreeShippingRule5* is a flexibility rule to manipulate the *Order* resource at the *s_{man}postval_{pre}* state. It uses an *XsltTrans* operation to assign the *Parcel* value to the shipping method, and the *0* value to the shipping fee of the Order resource (lines 8-17). The rule also includes a fault handler to specify the actions on the fault situation with the rule weaving (lines 18-25). It will first log the fault detail immediately, and then guide the process instance to a violation state with a defined extended violation type. The last rule - *remedyFreeShippingRule5* (line 28) defines a provider action to handle the extended violation type by cancelling the process instance.

Listing 5.6: manipulateFreeShippingRule5 and skipFreeShippingRule5

```

1  <p1:Rule priority="0" ruleId="manipulateFreeShippingRule5">
2    <p1:ActivityStates>
3      <p1:ActivityState>Manipulating-Post-Validating-Pre</p1:ActivityState>
4    </p1:ActivityStates>
5    <p1:Conditions>
6      <p1:ConditionExpression>//Order/Total>=2000</p1:ConditionExpression>
7    </p1:Conditions>
8    <p1:Actions>
9      <p1:Pa-Manipulate>
10        <p1:Copy>
11          <p1:From>
12            <p1:XsltTrans source="//Order" xslt="assignFreeShipping.xsl"></p1:XsltTrans>
13          </p1:From>
14          <p1:To query="//Order"></p1:To>
15        </p1:Copy>
16      </p1:Pa-Manipulate>
17    </p1:Actions>
18    <p1:FaultHandler>
19      <p1:Ca-Log level="5"></p1:Ca-Log>
20      <p1:Pa-Violate>
21        <p1:Violation>
22          <Type>Extend:FreeShipping:MessageManipulating</Type>
23        </p1:Violation>
24      </p1:Pa-Violate>
25    </p1:FaultHandler>
26  </p1:Rule>
27
28  <p1:Rule priority="0" ruleId="remedyFreeShippingRule5">
29    <p1:Objects>
30      <p1:ObjectsAnyOf>
31        <p1:ObjectsAllOf>
32          <p1:Violation>
33            <Type>Extend:FreeShipping:MessageManipulating</Type>
34          </p1:Violation>

```

```

35         </p1:ObjectsAllOf>
36     </p1:ObjectsAnyOf>
37 </p1:Objects>
38 <p1:ActivityStates>
39     <p1:ActivityState>Handling-Pre</p1:ActivityState>
40 </p1:ActivityStates>
41 <p1:Conditions/>
42 <p1:Actions>
43     <p1:Pa-Cancel/>
44 </p1:Actions>
45 </p1:Rule>

```

5.5.2.3 Case 3: protection aspect

Business policy:

Customer account information is not allowed to be passed to business partners that have low security (>3)

For the above business policy, we defined a security policy - *SecurityAccountPolicy7* (Listing 5.7). The policy is triggered for any activity involved with the *CustomerAccount* resource (lines 3-11) before the activity execution (lines 12-14). A constraint rule is defined in the policy and has a condition to check the security level of the service or process provider before proceeding with the activity (lines 17-19). A fault handler is defined to violate the process instance, in the case that security condition checking cannot be executed or exceptions are caused (lines 23-29).

Listing 5.7: SecurityAccountPolicy7

```

1 <p1:Policy policyId="SecurityAccountPolicy7" priority="0">
2   <p1:Objects>
3     <p1:ObjectsAnyOf>
4       <p1:ObjectsAllOf>
5         <p1:Resource>
6           <Name>CustomerAccount</Name>
7         </p1:Resource>
8       </p1:ObjectsAllOf>
9     </p1:ObjectsAnyOf>
10  </p1:Objects>
11  <p1:ActivityStates>
12    <p1:ActivityState>Validating-Pre</p1:ActivityState>
13  </p1:ActivityStates>
14
15  <p1:Rule priority="0" ruleId="constraintRule7">
16    <p1:Conditions>
17      <p1:ConditionExpression>exists(//ServiceProfile//ServiceReference[ (
        child::Ws-address =//WeavingRequest/Activity//Ws-address and

```

```

18         child::Operation =//WeavingRequest/Activity//Operation) and
19         descendant::Security>=3])</p1:ConditionExpression>
20     </p1:Conditions>
21     <p1:Actions>
22         <p1:Pa-Validate/>
23     </p1:Actions>
24     <p1:FaultHandler>
25         <p1:Pa-Violate>
26             <p1:Violation>
27                 <Type>QoS:Security</Type>
28             </p1:Violation>
29         </p1:Pa-Violate>
30     </p1:FaultHandler>
31 </p1:Rule>
32 <p1:ConstraintCombiningAlgorithm type="Pa-Violate-Unless-Pa-Validate-
    Through-All"/>
33 <p1:RemedyCombiningAlgorithm type="Pa-Cancel-Unless-Defined-Sequence-
    Through-All">
34     <DefinedSequenceElement>Pa-Cancel</DefinedSequenceElement>
35     <DefinedSequenceElement>Pa-Skip</DefinedSequenceElement>
36     <!-- ... more DefinedSequenceElement ... -->
37 </p1:RemedyCombiningAlgorithm>
38 <p1:SequencingAlgorithm type="Ordered"/>
</p1:Policy>

```

5.5.2.4 Case 4: optimization and healing aspect

Business policy:

Credit card processing should be completed quickly (expected less than 700 ms) without fault.

In this case, we have three rules in a policy - *cardProcessingPolicy10* (Listing 5.8) for the business policy. The policy targets both *Visa* and *MasterCard Card Processing* activities.

Listing 5.8: cardProcessingPolicy10

```

1 <p1:Policy policyId="cardProcessingPolicy10" priority="0">
2     <p1:Objects>
3         <p1:ObjectsAnyOf>
4             <p1:ObjectsAllOf>
5                 <p1:Activity>
6                     <Name>Visa Card Processing</Name>
7                 </p1:Activity>
8             </p1:ObjectsAllOf>
9             <p1:ObjectsAllOf>
10                 <p1:Activity>
11                     <Name>MasterCard Card Processing</Name>
12                 </p1:Activity>
13             </p1:ObjectsAllOf>

```

```

14     </p1:ObjectsAnyOf>
15 </p1:Objects>
16 <p1:ActivityStates/>
17
18
19 <p1:Rule priority="0" ruleId="constraintRule10">...</p1:Rule>
20
21 <p1:Rule priority="0" ruleId="retryRemedyRule10">...</p1:Rule>
22
23 <p1:Rule priority="0" ruleId="replaceRemedyRule10">...</p1:Rule>
24
25
26 <p1:ConstraintCombiningAlgorithm type="Pa-Violate-Unless-Pa-Validate-Through
27   -All"/>
28 <p1:RemedyCombiningAlgorithm type="Pa-Cancel-Unless-Defined-Sequence-Through
29   -All">
30   <DefinedSequenceElement>Pa-Retry</DefinedSequenceElement>
31   <DefinedSequenceElement>Pa-Replace</DefinedSequenceElement>
32   <!-- ... more DefinedSequenceElement ... -->
33 </p1:RemedyCombiningAlgorithm>
34 <p1:SequencingAlgorithm type="Ordered"/>
35 </p1:PolicySet>

```

The first rule is a constraint rule - *constraintRule10* (Listing 5.9) to check the service performance of the two activities. The rule specifies a condition to check if the service performance for the current activity is less than 700 ms (lines 5-7). If the performance is slower than expected, a *pa_violate* action is expected to be executed to guide the process instance to a performance violation state. A *ca_suspend* consumer action is also defined. It will suspend the service to avoid the service to be selected for any activity or process for the next 5 hours. The rule has a fault handler (lines 16-22) specifying the performance violation which is expected if condition checking of the rule is faulty, but the *ca_suspend* actions will not be performed, as this is not defined in the fault handler.

Listing 5.9: constraintRule10

```

1 <p1:Rule priority="0" ruleId="constraintRule10">
2   <p1:ActivityStates>
3     <p1:ActivityState>Validating-Pre</p1:ActivityState>
4   </p1:ActivityStates>
5   <p1:Conditions>
6     <p1:ConditionExpression>exists(//ServiceProfile/ServiceReference[(
7       child::Ws-address =//WeavingRequest/Activity//Ws-address and
8       child::Operation =//WeavingRequest/Activity//Operation) and
9       descendant::Performance>700])</p1:ConditionExpression>
10   </p1:Conditions>
11   <p1:Actions>
12     <p1:Ca-Suspend Time="P0Y0M0DT5H"/>
13     <p1:Pa-Violate>
14       <p1:Violation>

```

```

12         <Type>QoS:Performance</Type>
13     </pl:Violation>
14 </pl:Pa-Violate>
15 </p1:Actions>
16 <p1:FaultHandler>
17     <pl:Pa-Violate>
18         <pl:Violation>
19             <Type>QoS:Performance</Type>
20         </pl:Violation>
21     </pl:Pa-Violate>
22 </p1:FaultHandler>
23 </p1:Rule>

```

The second and third rules are fault rules (Listing 5.10), define the remedy actions for both effect violation and performance violation. The second rule - *retryRemedyRule10* targets the *Functional:Effect* violation (lines 1-18). It specifies a *pa_retry* remedial action for the violation with two conditions. The first condition (line 12) specifies for a current service that a *pa_retry* action is executed less than 5 times on the provider side within the last minute. The second condition (line 13) specifies for current service that a *pa_retry* action is executed less than 30 times on the provider side within the last 5 minutes. Otherwise, *pa_retry* is not expected. The third rule - *replaceRemedyRule10* defines a *pa_replace* for both *Functional:Effect* and *QoS:Performance* violations. The replacement service is defined with a mandatory condition on the trust context, a weak condition on preferred service performance for the activity. Hence, a fast performance service is selected, if it is available as an optimization. The approach for optimization is similar to the protection aspect of the business policy, but with different remedial actions. An obligation is also defined for the rule (lines 37-42). It will log the replacement event, if the *pa_replace* will be executed on the provider side. The remedy combining algorithm of the policy (Listing 5.8 line 27) specifies that *pa_replace* is the preferred remedy over the *pa_retry*, if both remedies are applicable.

Listing 5.10: retryRemedyRule10 and replaceRemedyRule10

```

1 <p1:Rule priority="0" ruleId="retryRemedyRule10">
2   <p1:Objects>
3     <pl:ObjectsAnyOf>
4       <pl:ObjectsAllOf>
5         <pl:Violation>
6           <Type>Functional:Effect</Type>
7         </pl:Violation>
8       </pl:ObjectsAllOf>

```

```

9      </p1:ObjectsAnyOf>
10    </p1:Objects>
11    <p1:Conditions>
12      <p1:ConditionExpression>count (//Pa-ActionLog/Pa-Action[@type="Pa-Retry
        " and @time > (current-dateTime()- xdt:dayTimeDuration('PT1M'))
        and descendant::ServiceReference])&lt;=5</p1:ConditionExpression>
13      <p1:ConditionExpression>count (//Pa-ActionLog/Pa-Action[@type="Pa-Retry
        " and @time > (current-dateTime()- xdt:dayTimeDuration('PT20M'))
        and descendant::ServiceReference])&lt;=30</p1:ConditionExpression>
14    </p1:Conditions>
15    <p1:Actions>
16      <p1:Pa-Retry WaitFor="PT0M"/>
17    </p1:Actions>
18  </p1:Rule>
19
20  <p1:Rule priority="0" ruleId="replaceRemedyRule10">
21    <p1:Objects>
22      <p1:ObjectsAnyOf>
23        <p1:ObjectsAllOf>
24          <p1:Violation>
25            <Type>QoS:Performance</Type>
26          </p1:Violation>
27        </p1:ObjectsAllOf>
28        <p1:ObjectsAllOf>
29          <p1:Violation>
30            <Type>Functional:Effect</Type>
31          </p1:Violation>
32        </p1:ObjectsAllOf>
33      </p1:ObjectsAnyOf>
34    </p1:Objects>
35    <p1:Conditions/>
36    <p1:Actions>
37      <p1:Pa-Replace InstanceOnly="false">
38        <p1:ServiceConditions>
39          <p1:ServiceConditionExpression force="false" expression="/Context
        //Performance&lt;700"/>
40          <p1:ServiceConditionExpression force="true" expression="/Context//
        Trust&gt;5"/>
41        </p1:ServiceConditions>
42      </p1:Pa-Replace>
43    </p1:Actions>
44    <p1:Obligations>
45      <p1:Obligation Type="Pa-Replace">
46        <p1:Ca-Log level="4"/>
47      </p1:Obligation>
48    </p1:Obligations>
49  </p1:Rule>

```

5.6 Discussion with related work

In this section we discuss related work on policy models in service computing in comparison to our policy language model.

5.6.1 The primary requirements

Many policy models have been developed. We state the primary requirements of a policy model needed in our case first. Then, we discuss related work with regard to these requirements, also to strengthen the need for our work. Our policy language model is designed to satisfy two primary requirements for the architecture framework.

- A policy model covers all rule categories.

The policies represent the business policies or requirements of process consumers on business processes. The primary goal of the policy model is to allow process consumers to specify comprehensive requirements on business processes, i.e. the policy model should cover all the types of requirements, identified as four types of rule categories. Additionally, the policy model should be refined to reduce redundancy which is caused by covering multiple categories.

- A policy model targets on business processes.

A business process is not an autonomic Web service or business activity. A list of services or activities are connected with the control flow structures in a business process. Actions or decisions made on a single Web service might result in changing or additional actions on other Web services. The policy model needs to target business processes, and consider control flow graphs of business processes rather than only on the Web services. For example, cancelling an activity and cancelling a process is different.

In the following we present some work on policy modelling in service computing, and compare it with ours.

WS-Policy[15] and XACML[21] are two major policy models that have matured from research and have become standard specifications. These policy specifications can be viewed as in the same family that only focuses on the constraint aspect of Web services. They do not satisfy both primary requirements we described above. Extended work on XACML for BPEL processes, such as [89], still only covers the constraint aspect.

[63] [107] [68] are different projects that offer fault rules and policies for BPEL processes. The theory behind the above work in policy modelling is the ECA paradigm. XML is used in all policy languages. These only focus on the fault aspect of BPEL processes.

Dynamo and the MASC framework are two works offering the most comprehensive policy model within our context. The Dynamo project [17] [109] [110] proposes a WSCoL assertions language and WSReL recovery language for BPEL processes. The WSCoL mixes typical propositional logic constructs with XML-based technology. The WSReL is designed by following the ECA paradigm. Both languages use a Java like programming language syntax. The MASC framework [111] [172] [16] proposes a WS-Policy4MASC to extend the WS-Policy, and follows the ECA paradigm. The policy model covers all rule categories we defined and has XML syntax. Both these target services or BPEL processes. However, we still have a very different policy model. Comparing them with our work, we have following advantages:

Their work intends to ask policy developers to define policies for every single concrete BPEL process. A policy target or object is a concrete service reference. In our policy model, the objects of policies are abstract such as business processes and activities. This makes more sense for policy developers to formalize business policies in a policy language from a business perspective. Once policies are defined with our model, these policies are still applicable regardless of service references when the process consumer switches processes or service providers. This is important for supporting process consumers to discover and switch process providers at runtime. Also in our policy model, policy developers could refine the policy objects on any concrete service by utilizing the conditions element of the policy model.

Their policy models are highly dependent on their policy frameworks which are integrated with the BPEL execution platform, for example, QoS degradation events, configuration actions on the platform, a callback event. Having such policy frameworks tightly coupled with the execution platform is very difficult for processes with multi-tenancy capa-

bility, as the platform is shared with multiple consumers. Our policy model clearly defines actions available for process control in a process instance graph layer. The policy enforcement operates on the process instance layer rather than any other layers, such as a process itself or execution environment platform, etc. Our policy model is designed to naturally support the multi-tenancy environment, which they did not consider.

In addition, our policy model allows to define nested policies, and has the fault handling capability. These are not available in the Dynamo and MASC policy model.

5.6.2 The language model complexity

A policy language is a formal language for expressing business policies which are in natural languages, and it acts as customization or configuration metadata of business processes. It should be a simple high level language for the goal of customization or configuration of software, rather than a complex low level programming language for the goal of the software development.

With our policy language model, some policy elements are based on the XPath and Xslt specifications, such as the *Conditions* element, which requires XPath expressions. From the case studies, we can see XPath expressions are involved in many common cases in the policy development with our policy model. Thus, the XPath programming ability is required for policy developers. Since the XPath specification itself has a complex syntax, it adds significant additional complexity to our policy language. Compared to some other high level policy models, it might be more complex and not straightforward for policy developers initially, but the advantage of our policy language is that it is more powerful by utilizing the XPath expressions. It offers more flexibility for policy developers to define various policies which are not available in other current policy models. For example, the wait and retry remedy (PA_{retry}) for a type of fault or violation is a remedy available in most fault policy or rule languages for Web services. [67] only allows policy developers to define a retry remedy for a fault instance. [180] [17] [68] [63] allow to define a maximum

retry times. Policy modelling in [16] [107] has more parameters for the retry remedy. They allow specifying maximum retry time and waiting time for each retry. In the following, we show some examples which are only available in our policy language with retry remedies with XPath conditions.

1. Maximum retries 10 times for this service regardless of the business process in the last 3 minutes.
2. Maximum retries 10 times for this service for a business process in the last 3 minutes.
3. Maximum retries 10 times for this service for a business process instance in the last 3 minutes.
4. Waiting 2 seconds on the first and second retry, waiting 5 seconds on the remaining retry.

More examples could be listed. Policy developers could utilize the conditions, as above, for the PA_{retry} action to develop complex remedial strategies for complex business policies. For example, a smaller number of retries is expected for process instances with large orders before an instance adaptation to an expensive service.

5.6.3 The fault handling ability

When the policies are enforced at process runtime, the exceptions of policy weaving should be handled just as exceptions of the process execution. Within the related work we have studied, the XACML is the only policy language that has the fault handling ability for runtime exceptions of policies. It can handle exceptions occurring through combining algorithms when a policy is evaluated. A fault will result in an *Indeterminate* decision between *Permit* and *Deny* decisions. The combining algorithms defined in a policy would give a Permit or Deny decision for indeterminate decisions.

Our policy weaving does not intend to give one of two final decisions Permit or Deny as the XACML policy. It contains various consumer and provider actions covering different rule categories for governance requirements. We take the concept of the combining algorithm from the XACML policy to take care of nested policies and the policy confliction problem, but our combining algorithms are also differ substantially.

The combining algorithms could be used to determine the type of provider actions in exception situations. Still, there are some limitations. Firstly, the fault handling is not available at rule level. If a policy developer would define or update a fault handling strategy for a single rule in a policy which has multiple rules, he must encapsulate the rule in a new policy regardless of whether it is already in a policy, or analyse the whole policy and then decide if he needs to update the combining algorithm of the policy. Secondly, only using combining algorithms, it is difficult to specify a precise fault handling strategy in undetermined situations. For example, different violation types might need to be specified for exception situations of policy weaving, even the $PA_{violation}$ will still be decided by the combining algorithm. Different consumer or provider actions would need to be performed in exception situations of policy weaving. Thirdly, the combining algorithms are only suitable for provider actions in the same rule categories. In case of exceptions during $PA_{manipulate}$ action execution, it is difficult to apply the combining algorithm concept for the fault handling.

In our policy modelling, we introduced a *FaultHandler* for the rule component to handle expected exceptions on *Conditions* and $PA_{manipulate}$ elements which are largely based on the XPath or XSLT expressions. Expected consumer or provider actions for exception situations can be defined in the fault handler for a rule. However, with our current design, the fault handler is not intended for exceptions caused by other than *Conditions* and $PA_{manipulate}$ elements in policy weaving.

5.7 Conclusion

Policies are superficial process level contracts correlated with a real contract of process governability between the process consumers and providers. Our policy modelling is concerned with many issues which affect policy modelling, such as different aspects of business policies or requirements, distributed to consumers with multi-tenancy applications. We showed

our policy modelling approach and policy language model. A set of related algorithms with the policy model have also defined and explained. We used case studies to demonstrate various aspects business policies that are expressible in our policy language, also discussed related work.

Chapter 6

Coordination

6.1 Introduction

This chapter presents a coordination framework with protocols as real contracts to make process consumers and providers work together for governance to ensure that defined policies are enforced.

For a business transaction requested by a process consumer, there are a number of activities including those from subprocesses within a process that will participate in the transaction. The WS-Coordination specifications such as [126] [128] [3], are designed for transactions of distributed Web services rather than transactions of business processes. However, the adaptive processes, such as [59] [64] [17], for handling processes transactions lack a coordination mechanism for our case to guarantee all participants working together in a unified manner. The coordination framework we designed is a direct response to the above problems. It includes defined protocols as contracts for all participants for any business transactions of business processes.

We first develop the coordination model which focuses on message exchange or coordination contexts between participants and coordinators, and also a cache mechanism to reduce the overhead of coordination conversations caused by message exchange. A coordi-

nation protocol for addressing the policy enforcement in business transactions is defined in the second phase. Then, we design an approach which offers BPEL templates to implement the protocols with BPEL processes for providers, but also with the multi-tenancy capability.

This chapter also includes a case study section for the coordination framework. In this section, we continuously use the policies defined in the last chapter to evaluate the effectiveness and the performance overhead on our coordination framework.

This chapter is organized as follows: In Section 6.2, we show the coordinator model, and then detail the coordination protocol in Section 6.3. The implementation approach is described in Section 6.4. A case study is provided in Section 6.5 to evaluate the coordination framework. In the remaining sections, we compare with related work and give conclusions.

6.2 Coordination model

6.2.1 The model

The coordination model is inspired by the WS-Coordination and XACML policy framework, and is redefined for the specific need of our coordination protocol and mechanism for policy enforcement. We will detail this later. The coordination model defines two types of subcoordinators for process consumers and providers (Figure 6.1). Thus, each participant only interacts with its own type of coordinator. The coordination model is defined as $\langle COOR, COOR_{context} \rangle$, where

- $COOR = COOR^c \cup COOR^p$,
 $coor^c \in COOR^c$ is a coordinator associated with the consumers or PG
 $coor^p \in COOR^p$ is a coordinator associate with the providers or BP.
- $coor_{context} \in COOR_{context}$ is coordinaton context information.

Please note, a $coor^p \in COOR^p$ is required for all process providers, including the sub-process providers.

A $coor^c \in COOR^c$ consists of two component services:

1. An Activation service

It has an operation that enables an application to create a coordination instance or initial context. The $coor^c$ may support this activation service.

2. A Protocol service

It is for a specific coordination type which is defined in a separate coordination protocol. The protocol service allows protocol specific interactions.

A $coor^p \in COOR^p$ consists of two component services:

1. A Protocol service

It is the same as the one in the $coor^c$.

2. A Cache service

It has operations for accessing and updating the coordination cache. The $coor^p$ may support the cache service.

Figure 6.1 illustrates how a $coor^c$ and $coor^p$ interact in a coordination conversion. The protocol X and services X_c and X_p are specific to a coordination protocol. The following describes the coordination algorithm of Figure 6.1.

1. The process consumer sends a create coordination context request to the activation service of $coor^c$, getting back an initialized $coor_{context}$ (Cc) that contains the identification, a service reference of the $coor^c$'s protocol service and other information needed for starting a coordination conversation.
2. The process consumer then sends a process request to the provider or business process containing the $coor_{context}$.
3. The $coor_{context}$ is extracted from the SOAP message and passed to the protocol service X_p at $coor^p$. At this point, the protocol service X_c service reference is known by the protocol service X_p , and the communication between the protocol services can be established. In addition, X_p can determine if the coordination cache is enabled for the coordination conversation.

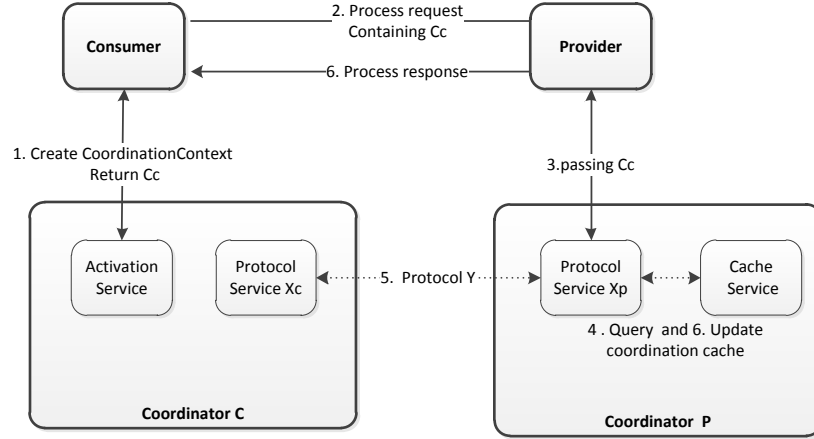


Figure 6.1: The schematic coordination example

4. In case the coordination cache is enabled, the protocol service X_b will send a request to the cache service of $coor^p$ at a point of process execution, getting back a cached coordination data result for the point of process execution.
5. Depending on the result of cached coordination data, the communication between the protocol service X_c and X_p may occur at the point of process execution.
6. The protocol cache is updated through the cache service if it is required.
7. The coordination conversation ends with the completion of the process execution or the business transaction.

It should be noted that several actions (4, 5, and 6) with the above description might be repeated in a coordination conversation. The operation interface of the activation and cache service are not defined as a protocol in the coordination model.

6.2.2 Coordination context

The $COOR_{context}$ defines the data structure of the message exchange in the context of coordination. All process consumers and providers must understand this information to enable coordination conversions. A schema defined for the coordination context is a protocol between process consumers and providers.

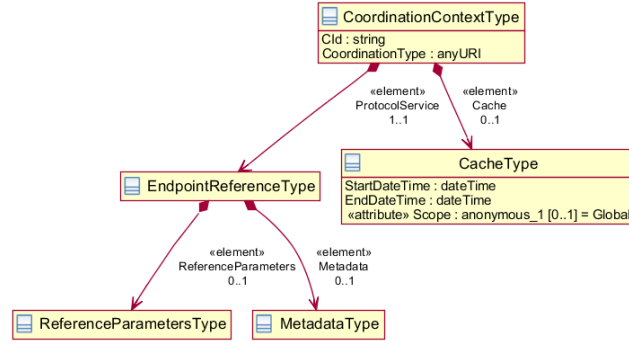


Figure 6.2: $COOR_{context}$: CoordinationContextType

$spac$ is a prefix for the schema namespace (more details are available in the appendix B):

$xmlns : spac = http : //www.computing.dcu.ie/mwang/spac$

6.2.2.1 CoordinationContext

A $coor_{context} \in COOR_{context}$ is defined as a tuple $\langle cid, pi, ps, cac, \alpha \rangle$ (Figure 6.2), where

- cid - specifies a unique identification of a coordination conversion for a process request or business transaction instance. It is generated by the Activation service.
- ct - denotes a coordination type specifying a coordination protocol of a coordination conversation. A coordination protocol is a separate protocol which will be discussed in the process activity protocol section (Section 6.3).
- $ps \in PS$ - denotes a protocol service specifying the service reference of a $coor^p$. It enables protocol service communication for the specified coordination type. It is of type EndpointReferenceType from WS-Addressing [181]. The address element of the endpoint reference is viewed as an identity of $coor^c$ for $COOR^p$.
- $cac \in CAC$ - specifies a cache configuration for the coordination type. It will be discussed in the next subsection.
- $x \in ALP$ - an extension needed for additional context information for the coordina-

tion, such as protocol specific messages and necessary message correlations.

The context contains a data field (cid, ct, ps, cac) initialized by the consumer side at the start of a coordination conversation with a process request. It can only be assigned by a $coor^c$. For activities as subprocesses in a business process, the above data of $coor_{context}$ will be propagated to participants, i.e., the subprocesses, regardless whether they belong to the same process provider or different providers. The process providers do not initialize a new context for subprocesses. This is important as the original source of $coor_{context}$ symbolizes the source of the business policies, i.e., all processes include subprocesses that are governed by the policies defined by the original process consumers, not by the policies from process providers. A $coor_{context}$ can be initialized by a process provider for subprocesses in a business transaction, the subprocesses would then be enforced with policies defined by the process provider. In such a case, the activities within the subprocesses of the overall process are Web services, i.e., atomic activities for process consumers. This is different compared with the distributed coordination of WS-Coordination, which is achieved by a chaining coordination [3]: A subcoordinator B of a coordinator A acts as a proxy responsible for passing all messages from coordinator C to A .

6.2.2.2 Cache

A cache $cac \in CAC$ specifies the configuration of the cache function on coordination conversations. The cache function will be described in the next section. If the cache element is absent in a $coor_{context}$, the coordination cache is disabled for the coordination conversation.

The *Scope* attribute specifies the cache scope, which is either Global or Process.

1. *Global scope* is a default scope, specifies that the coordination cache is $coor^c$ aware.

Caches are valid for a process request, only if the caches are made from the same $coor^c$.

2. *Process scope* specifies that the coordination cache is both $coor^c$ and process aware.

Caches are valid for a process request, only if the caches are made for the same $coor^c$

and process.

StartTime is an element used to specify a coordination cache operation start time.

EndTime is used to specify a coordination cache operation end time.

The defined coordination cache time (*CacheStart/EndTime*) will be exactly applied without awareness of the execution time of process instances, i.e., the cache could be enabled after the start time of a process instance, or could be disabled before the end of a process instance.

6.2.2.3 Example

The activation service provides the operation interface *CreateCoordinationContext* that is used to initialize a *coor_{context}* for a process request. Listing 6.1 shows an example of a *coor_{context}* instance.

Line 3 shows the identification (<http://www.computing.dcu.ie/mwang/spaa/sm/cache>) of a coordination protocol, which defines the coordination type of the coordination conversation. This will be described in the next section. Lines 4 to 15 show the protocol service endpoint reference. In this case, more detail of the endpoint reference is described as a *ServiceReference*, defined in the coordination protocol. Lines 16 - 19 shows the cache is enabled for one month time.

Context propagation is accomplished using an application-defined mechanism. The context message would be contained in a SOAP message together with an application message data sent to participants.

6.2.3 Coordination cache mechanism

The coordination cache mechanism is designed to try to improve coordination efficiency of the coordination model by trying to reduce protocol message communication between two types of subcoordinators. It caches message responses of a coordination protocol of coordination conversations when the coordination cache is enabled. For a *coor^P*, it remem-

Listing 6.1: An initialized coordination context example

```

1 <p1:CoordinationContext xmlns:p1="http://www.computing.dcu.ie/mwang/spac"
  xmlns:wsa="http://www.w3.org/2005/08/addressing" xmlns:xsi="http://www.w3.
  org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.computing.dcu.
  ie/mwang/spac_coordination.xsd">
2 <Cid>2134</Cid>
3 <CoordinationType>http://www.computing.dcu.ie/mwang/spaa/sm/cache</
  CoordinationType>
4 <p1:ProtocolService>
5 <wsa:Address>http://localhost:8080/ProcessRequestor1/
  RequestorCoordinatorService</wsa:Address>
6 <wsa:Metadata>
7 <p2:ServiceReference xmlns:p2="http://www.computing.dcu.ie/mwang/spaa"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
8 <p2:Address>http://localhost:8080/ProcessRequestor1/
  RequestorCoordinatorService</p2:Address>
9 <p2:Operation>Weaving</p2:Operation>
10 <p2:ServiceName xmlns:sere="http://coordinator/">
  sere:RequestorCoordinatorService</p2:ServiceName>
11 <p2:PortName xmlns:sere="http://coordinator/">
  sere:RequestorCoordinatorPort</p2:PortName>
12 <p2:SOAPAction>http://coordinator/RequestorCoordinator/WeavingRequest<
  /p2:SOAPAction>
13 </p2:ServiceReference>
14 </wsa:Metadata>
15 </p1:ProtocolService>
16 <p1:Cache Scope="Global">
17 <StartDateTime>2010-10-10T12:00:00-05:00</StartDateTime>
18 <EndDateTime>2010-11-10T12:00:00-05:00</EndDateTime>
19 </p1:Cache>
20 </p1:CoordinationContext>

```

bers policy information defined or a final provider action at a particular point of a business process by consumers, to decide what interaction pattern is needed between a $coor^c$ and $coor^c$ at the point. It aims at reducing the number of communications between protocol services of $COOR^c$ and $COOR^p$ in coordination conversations. As a consequence, the coordination overhead caused by communications would be reduced.

There are three types of interaction patterns defined as the foundation of the cache mechanism. It results in three default extra transition actions for all coordination protocols: $TA_c = \{ta_{undefined}, ta_{unexpected}, ta_{undetermined}\}$. They are mapped to $PA_{undefined}$, $PA_{unexpected}$, $PA_{undetermined}$ in the policy model which are discussed in Section 5.3 of the policy chapter. Other transition actions resulting from the specific coordination protocols are also associated with the interaction patterns. The three extra transition actions will

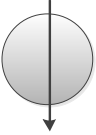
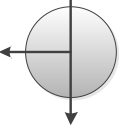
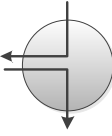
Transaction action	Interaction pattern	Description
$ta_{undefined}$		The protocol service of $coor^p$ will not try to communicate with the $coor^c$ on a governance state. A mapped transition action will be applied.
$ta_{unexpected}$		The protocol service of $coor^p$ will send a message or $weaving_{request}$ to $coor^c$ using a one-way interaction mode on a governance state. This makes sure the consumer actions defined in the policies are executed. A mapped transition action will be applied.
$ta_{undetermined}$ and others from specific protocol TA_g		The protocol service of $coor^p$ will communicate with the $coor^c$ and wait for a provider action from the consumer. The coordination cache is also updated with the provider action from the consumer. A mapped transition action will be applied if the returned transition action is one of $ta_{undefined}$, $ta_{unexpected}$, $ta_{undetermined}$.

Table 6.1: Transition actions and interaction patterns

actually be mapped to transaction actions specified in coordination protocol on governance, which will be detailed in a later subsection. The interaction patterns and association with transition actions are defined in Table 6.1 for the coordination model.

From the table, we can see the $COOR^p$ does not interact with the $COOR^c$ in case of $ta_{undefined}$. The $ta_{unexpected}$ uses a one-way notification interaction mode, process execution will not be blocked to wait for the consumer to complete the policy weaving. Hence, we could expect the performance overhead to be reduced in the first two cases. The detail of governance states, transition actions and provider actions will be described in the process activity protocol section.

The Cache service provides an operation interface for accessing and updating cached data (*getCacheResult*, *updateCacheResult*). A cached data $cad \in CAD$ is described as a tuple $\langle dt, psa, pn, p\alpha, r \rangle$, where

- dt - is the time when the cached result is created or updated.
- psa - denotes a protocol service URL, i.e., is an identification of the $coord^c$. It represents a unique policy source.
- pn - denotes the name of the process.
- $\alpha \in ALP$ - is an extended element which specifies additional conditions on using cached results. This is protocol specific based on the message schema of the coordination protocol. This is described in the next section.
- r - denotes the result or transition action cached.

The cache function is required to be implemented with the protocol service implementation to support the coordination cache mechanism. The following Algorithm 8 shows the cache function algorithm for the protocol service of a $coord^p$.

Algorithm 8: Algorithm for Cache function

input : $coord_{context} \in COOR_{context}$
output: $ta \in TA = \{TA_c \cup TA_g\}$

```
1 Initial  $ta \in TA$  ;
2 if  $isCacheEnabledOnCurrentTime(coord_{context})$  then
3   Initial  $cad \in CAD$  ;
4   if  $getCacheScope(coord_{context}) = global$  then
5      $cad \leftarrow getCacheResult(psa \in cad, \alpha \in cad, true)$  ;
6   else  $cad \leftarrow getCacheResult(psa \in cad, \alpha \in cad, false)$ ;
7   if  $dt \in cad \geq startDateTime \in coord_{context}$  then
8     if  $r \in cachedData = ta_{undefined}$  then  $ta \leftarrow ta_{undefined}$  ;
9     else if  $r \in cachedData = ta_{unexpected}$  then
10       $protocolServiceInvokeOneway(coord_{context})$  ;
11       $ta \leftarrow ta_{unexpected}$  ;
12    else  $ta \leftarrow protocolServiceInvoke(coord_{context})$  ;
13  else
14     $ta \leftarrow protocolServiceInvoke(coord_{context})$  ;
15     $updateCacheResult(psa, \alpha, ta)$  ;
16 else  $ta \leftarrow protocolServiceInvoke(coord_{context})$ ;
17 return  $ta$  ;
```

Depending on the coordination protocol defined for the coordination, a cache mechanism may be implemented in a $coord^c$. In this case, it will not reduce the communication overhead between the $coord^c$ and $COOR^p$, but the overhead of policy weaving. Since the policy weaving is not in the scope of coordination protocols, the cache mechanism in the $COOR^c$ is not defined in the coordination model.

6.3 Process activity protocol

The process activity protocol defines a coordination type for coordination conversations. It relies on the coordination model we described. A coordination conversation of a business process is established upon coordination of all activities which are within the overall process and subprocesses for the consumer. The conceptual modelling of the coordination protocol is activity centric, so it can be applied to any processes regardless of various flow logic, but

without losing the aspects related to business processes. This coordination protocol applies to all activities of business processes to be governed during execution.

A coordination protocol comprises three definitions in its identification ($ct \in \text{coor}_{context}$).

$$\frac{\text{http} : // \text{www.computing.dcu.ie/mwang/spaa}}{1} \frac{\text{/sm}}{2} \frac{\text{/cache}}{3}$$

1. a protocol message schema
2. a Finite State Machine (FSM) of $COOR^c$ and $COOR^p$
3. a cache function specification of $COOR^p$

These are described in the following subsections.

6.3.1 Protocol message schema

The protocol message schema defines the message data structure needed for protocol services communication between $COOR^c$ and $COOR^p$ for the extension element of the $COOR_{context}$.

spaa is a prefix for the schema namespace (more details are available in appendix C):

$\text{xmlns} : \text{spap} = \text{http} : // \text{www.computing.dcu.ie/mwang/spaa}$

Two main elements defined in the protocol message schema are $PAP_{request}$ and $PAP_{response}$.

They are the request and the response of the protocol services of coordinators.

A $pap_{request} \in PAP_{request}$ is defined as a tuple $\langle p, a, r, v', s \rangle$, which extends the $COOR_{context}$ to form the $Weaving_{request}$, where

- a process $p \in P$ contains the process name information, and a service reference information of the process.
- an activity $a \in A$ contains the activity name information, and a service reference information of the business service which implements the activity.
- a resource $r \in R$ constraints the business object involved in the activity for the current activity state. It is defined as a free extendible element (xsd:any) for any type of business objects.

- a set of violations $v' \subseteq V$ contains available violation information of the activity.
- a state $s \in S_g$ is the current governance state of the activity. The S_g is defined in the FSM part of the protocol.

A $pap_{response} \in PAP_{response}$ extends the $COOR_{context}$ to form the $Weaving_{response}$.

It is defined as $\langle ta \rangle$, where

- a transaction action $ta \in TA$ is an abstract type of a set of concrete transition actions mapped to provider actions, which are described in the policy model, and will be detailed in the next subsection as well.

The $Weaving_{request}$ is defined for request messages of protocol services of $COOR^p$, the $Weaving_{response}$ is defined for response messages of protocol services of $COOR^c$ (Figure 6.3). All messages are wrapped as coordination context information. We have seen the $Weaving_{request}$ and $Weaving_{response}$ used for the policy weaver in the policy chapter. In fact, the coordination protocol message schema definition is derived from our study of related work in the policy based computing. The policy weaver could be viewed as an implementation of a protocol service of a $coor^c$ depending on the coordination protocol, because of the difference between the policy model and the coordination message definition. The PA needs to be transformed to TA as the $Weaving_{response}$ before sending out from the policy weaver. Still, in our case, the policy weaver does not directly communicate with a $coor^p$, because the FSM of the coordination protocol is divided into two parts. A proxy service sends a $weaving_{request}$ to a policy weaver or a FSM of $COOR^c$ depending on the activity state. We will describe this in the next subsection.

6.3.2 FSM of protocol

Because of the FSM of the process activity protocol intends to cover the intricate runtime governance requirement, we present a complicated FSM, which might not be easy to follow. This makes it difficult to implement the protocol with BPEL processes, and complicates for future protocol customisation or improvement. We first present the FSM of protocol, then

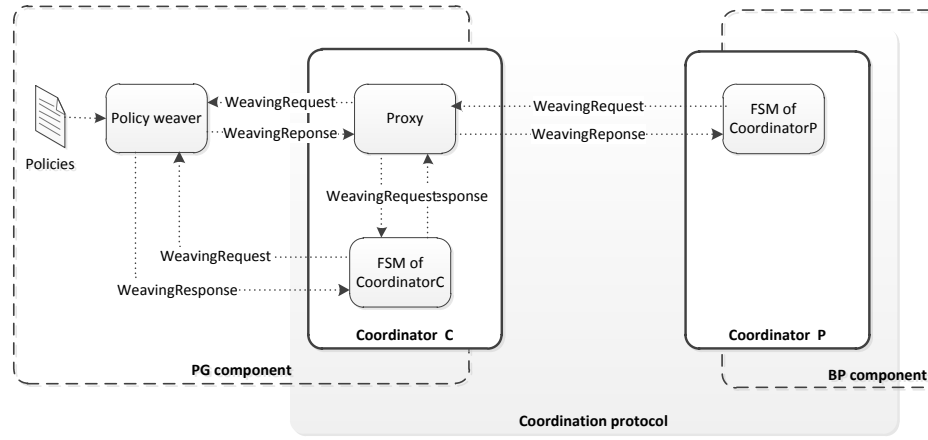


Figure 6.3: Message flow diagram

we further explain it in the later design subsection.

6.3.2.1 FSM of $COOR^c$ and $COOR^p$

The process activity protocol defines a certain level of runtime governability available from business processes and the responsibilities of process providers and consumers as a contract. The governability should satisfy the requirement of all categories of rules in the policy model. This is formalized as an FSM definition of the coordination protocol. It defines a completed FSM for every activity in the business processes, and describes the system behaviours of $COOR^c$ and $COOR^p$ on coordination conversations. The essential of the FSM design is to instrument the governance states into the process flow as these governance states are core to offer governability of business processes.

A complete FSM is divided into two parts for a protocol, which are responsible for $COOR^c$ and $COOR^p$ respectively. The FSM of $COOR^c$ is a submachine state of FSM of $COOR^p$. The process providers only follow the part of the protocol which is defined for $COOR^p$. The consumers follow the FSM of $COOR^c$. Since the implementation of the FSM will be executed at the consumer and provider separately, the $COOR^c$ must have

sufficient information about the process execution for its part of the state machine execution, as the process executes on the provider side. In our design of the entire FSM, the FSM of $COOR^c$ defined for the submachine state in FSM of $COOR^p$ is isolated from the business process. As a result, the protocol message schema only covers the complete information about the activity rather than the process state information. The execution of the FSM of $COOR^c$ does not require information other than the $Weaving_{request}$, which is defined in the protocol message schema. The execution of the FSM of $COOR^p$ does not require information other than $Weaving_{response}$. The reason behind this design is that, firstly, the same protocol message schema can be used for different coordination protocols. A process consumer can customize the FSM of $COOR^c$ for itself without affecting the FSM of $COOR^p$ and other process consumers. Secondly, it avoids possible complexity in state machine implementation for both sides. One side does not need to know the implementation details of other side for its own implementation.

The purpose of the two parts design is that it could reduce the number of governance states in the FSM of $COOR^p$, hence reduce the protocol message exchange times required between $COOR^c$ and $COOR^p$ on coordination conversations. The advantage is that it could reduce the performance overhead caused by communication between the protocol services. Depending on the network situations between a process consumer and providers, the message exchange between them could be expensive in some cases. Reducing required message exchange times could improve the overall coordination efficiency. The disadvantage is that it increases the complexity on the consumer side, because of the FSM of $COOR^c$ should be implemented by consumers. However, a different protocol can be defined with $COOR^p$ that has a complete FSM.

In the following, we give the FSM of $COOR^p$ and $COOR^c$ respectively.

FSM of $COOR^p$

The FSM of $COOR^p$ specifies the protocol which is responsible for $COOR^p$. It is defined as a 5-tuple $(S, s_{start}, F, TA, \delta)$, where

- $S = S_g \cup S_{\neg g}$ is a set of states. S_g is a set of governance states $\{s_{man_val_pre}, s_{man_val_post}, s_{handling_pre}, s_{handling_pre}, s_{cancelling}\}$, which are directly involved with process consumers or policies. The $S_{\neg g}$ is a set of non-governance states $\{s_{start}, s_{violated_pre}, s_{executing}, s_{replacing}, s_{waiting}, s_{skipping}, s_{violated_post}, s_{compensating}, s_{com+rep}, s_{com+ign}, s_{completed}, s_{end}\}$, which are not directly involved with the process consumers.
- $s_{start} \in S_{\neg g}$ is an initial state. The activity coordination can only be started by the process provider, and is not directly involved with consumers.
- $F \subseteq S_{\neg g}$ is a set of final states $\{s_{end}\}$.
- $TA = TA_g \cup TA_{\neg g}$ is a set of input symbols of transaction actions. TA_g is a set of transaction actions $\{ta_{violate}, ta_{validated}, ta_{ignore}, ta_{replace}, ta_{skip}, ta_{cancel}, ta_{compensate}, ta_{retry}, ta_{com+ign}, ta_{com+rep}\}$, and are expected from process consumers. $TA_{\neg g}$ is a set of transaction actions which are not expected from the process consumers $\{0, 1\}$. The input stream of the FSM regarding $TA_{\neg g}$ is decided by the process providers based on the process state information which is not covered by the FSM, as the FSM is only activity scoped. More details will follow in a later section 6.3.2.2.
- δ is a transition system $\delta : S \times TA \rightarrow S$, represented as a transition graph (Figure 6.4).

FSM of $COOR^c$

The FSM of $COOR^p$ introduces two submachine states: $s_{man_val_pre}$ before the $s_{executing}$, and $s_{man_val_post}$ after the $s_{executing}$ state. They contain the FSM which is responsible for $COOR^c$. It enables message adaptations and constraint validations before and after $s_{executing}$.

The FSM for the $s_{man_val_pre}$ submachine state specifies the protocol which is responsible for $COOR^c$. It is defined as a 5-tuple $(S, s_{start}, F, TA, \delta)$, where

- $S = S_g \cup S_{\neg g}$ is a set of states. S_g is a set of governance states, $\{s_{man_pre_val_pre}$

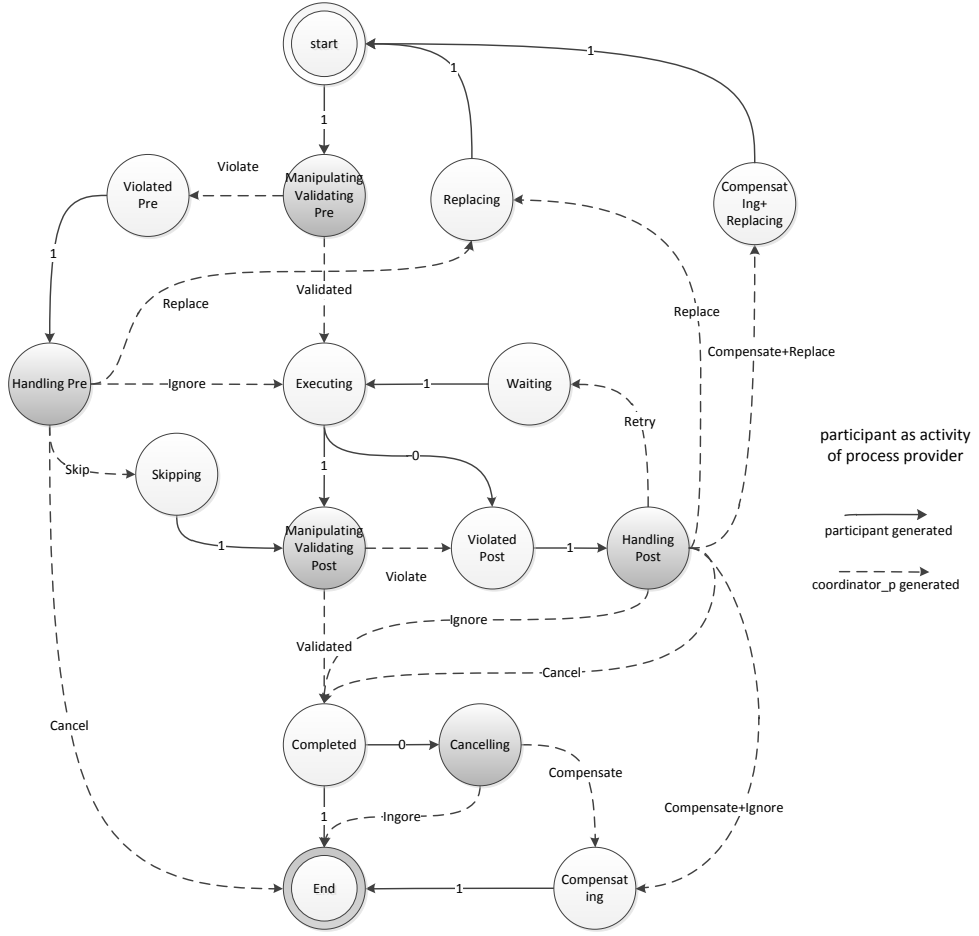


Figure 6.4: Transition graph

$s_{man_{post}val_{pre}}, s_{validating_{pre}}$. The S_{-g} is a set of non-governance states $\{s_{start}, s_{replacing}, s_{violated_{pre}}, s_{executing}\}$ from the parent FSM.

- $s_{start} \in S_{-g}$ is an initial state from the parent FSM.
- $F \subseteq S_{-g}$ is a set of final states $\{s_{violated_{pre}}, s_{executing}\}$ from the parent FSM.
- $TA = TA_g \cup TA_{-g}$ is a set of input symbols of transaction actions. TA_g is a set of transaction actions $\{ta_{validated}, ta_{violate}\}$, are expected from process consumers or policies. TA_{-g} is a set of transaction actions which are not expected from process consumers $TA_{-g} = \{1\}$.

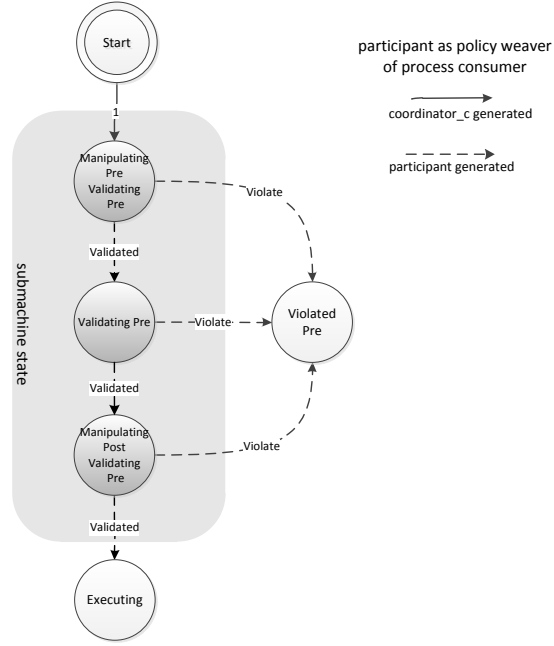


Figure 6.5: Transition graph

- δ is a transition system $\delta : S \times TA \rightarrow S$, represented as a transition graph (Figure 6.5).

The submachine state consists of three governance states allowing constraint rules validation before $s_{executing}$, and message adaptation before and after $s_{validating_{pre}}$. When a $coor_c$ receives a $weaving_{request}$, indicating an activity of a process is in the $s_{man_val_{pre}}$ state, the proxy service of $coor_c$ will enter the submachine of the $coor_c$ implementation defined for the $s_{man_val_{pre}}$ state. The FSM of $coor_c$ will send its $weaving_{request}$ to the policy weaver. The s of $weaving_{request}$ received by the policy weaver will be a governance state defined in the FSM of $COOR_c$. After completing the FSM of $COOR_c$, the final $weaving_{response}$ will be sent to the $COOR_p$.

The FSM of $COOR_c$ for $s_{man_val_{post}}$ submachine state is identical to $s_{man_val_{pre}}$, except $S_g = \{s_{man_{pre}val_{post}}, s_{man_{post}val_{post}}, s_{validating_{post}}\}$.

FSM Correctness

For the current and future updated versions of protocol correctness, the protocol should be validated to avoid problems during execution, such as deadlock. In our case, there must be a valid sequence of transitions leading to a desired state from a marking state as governance desires. We say the protocol is validated if it satisfies the reachability and liveness properties. Our protocol is validated as it satisfies the following properties.

Reachability: The protocol $(S, s_{start}, F, TA, \delta)$ for any type coordinator with a marking beginning state s_0 , a marking state s_n is reachable from s_0 if there exists a sequence of transitions $\omega = t_0, t_1, \dots$ that transits s_0 to s_n by $s_0 \times t_0 \rightarrow s_1 \times t_1 \rightarrow \dots \rightarrow s_n$.

Liveness: The protocol $(S, s_{start}, F, TA, \delta)$ for any type of coordinator with a desired state s_n , which is eventually reached from a marking state s_0 by inputting transactions belonging to the protocol.

6.3.2.2 FSM of protocol design for runtime governance

In the following, we give a further description of the FSM of the protocol to prove our design. We break the FSM into small FSM patterns for small problems we need to address. We introduce a special state called the place holder state s_* for FSM patterns. It is an abstraction of one or multiple concrete states in the FSM. Any concrete state(s) could be placed in a s_* state to overwrite it for protocol design.

For each pattern, we only focus on designing the related states for the problem addressed by the pattern, without considering the concrete states in the s_* . The details to specific implementations could be left to the final protocol design. The pattern description structure:

1. A name
2. A description of the problem context
3. A state machine (SM) pattern graph
4. The description of the SM template solution

For three categories of rules :

Pattern - Runtime governance



Figure 6.6: Activity life cycle

Problem: An activity's life cycle can be described by the activity states and the transitions among them. The life cycle of an activity in a business process could simply be described in three states without runtime governance executed on the provider side (Figure 6.6).

1. a start state s_{start} : An initial state of every activity in the business process. When an activity is reached in the process execution of the process flow, the activity is at the s_{start} .
2. a running state $s_{running}$: After the activity has started, the activity is executed to complete a task in the business process. The activity is in the $s_{running}$ for executing.
3. an end state s_{end} : After the activity is executed successfully, the activity is in the s_{end} . The life cycle of the activity is completed in the business process. With the process execution, the next activity placed in the process flow will be reached.

The first problem is that the three category rules which have provider actions. These rules are not involved.

The rule categories of policies only represent high level requirements. The actual requirements as policy enforcement are defined as a set of concrete actions that need to be performed with process execution as runtime governance. These actions include consumer and provider actions as described in the policy model. Before we address these actions, a primary problem is that these policies are defined at the consumer side and the process execution at the provider side are not aware of these actions or defined policies. This is the second problem.

Pattern: Figure 6.7

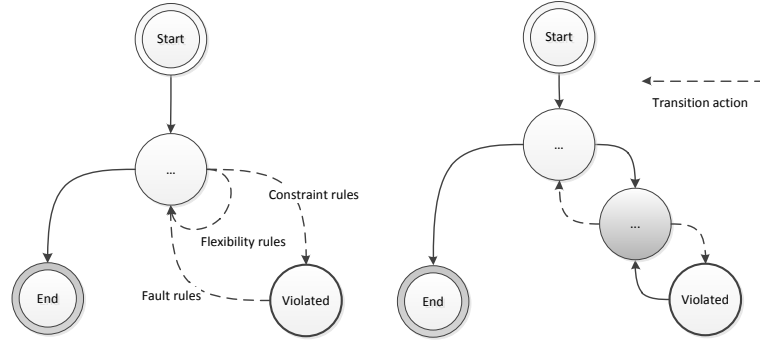


Figure 6.7: Pattern graphs

To enforce policies, we need to govern the activity between the s_{start} and the s_{end} . We expect more states in the activity life cycle to enforce the policies before the end of the activity life cycle. Through the proposed rule categorization, we can have a simple template for three rule categories in the first step (Figure 6.7 left). We add a violation state $s_{violated}$.

The runtime governance for three rule categories can be described as state transitions between the activity s_* and $s_{violated}$ state.

1. The flexibility rules define the transitions within the s_* states.
2. The constraint rules define the transitions from the s_* to the $s_{violated}$ state.
3. The fault rules define the transitions from the $s_{violated}$ to the s_* state.

Then, we introduce a new type of state called a governance state in the second step for the second problem.

- a governance state $s_g \in S_g$: is an identifier for starting available governance. The activity navigates to the s_g . The process consumer is notified that the expected consumer and provider actions of policies can be weaved. The expected provider action maps to a $ta \in TA_g$ transitions from the current state of the activity to a governed state. The consumer actions that have no effect on the business process will not be considered in the FSM.

In following, we are going to detail templates with the S_g and TA_g associated with the

three rule categories.

For flexibility rules :

Pattern - Manipulating

Problem: The governance of the data flow and control flow of a process by flexibility rules are achieved by message manipulation. Message adaptation needs message data manipulation. Through manipulating process data during process execution, the process execution path also can be adapted or altered. The problem is the need for message manipulation.

Pattern: Table 6.2

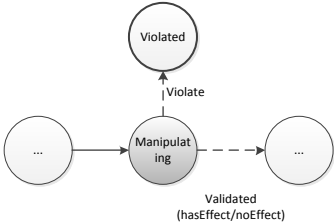
FSM pattern	transition action
	$ta_{violate}$ $ta_{validated}$

Table 6.2: Pattern graph

The process consumers should allow manipulating the resources or business objects processed by the activities of the process. We define a governance state as follows:

- a manipulating state $s_{manipulating} \in S_g$: indicates that the manipulate action could be performed. In case message manipulation needs to be applied by consumers, the activity is navigated to a $s_{manipulating}$.

We define the following transition actions for the $s_{manipulating}$:

1. a validated action $ta_{validated} \in TA_g$: indicates message manipulation is completed (mapped to $PA_{validated}$). It transfers the $s_{manipulating}$ to the next expected state after all message manipulate actions ($PA_{manipulate}$) of the process consumer are executed, and possible effects are taken by the $PA_{validated}$.
2. a violate action $ta_{violate} \in TA_g$: leads the $s_{manipulating}$ to the $s_{violated}$ (mapped

to $PA_{violate}$) in the case of faults on message manipulating actions. In the policy model, $PA_{violate}$ is defined as an expected provider action in a fault handler for the $PA_{manipulate}$.

For Constraint rules :

Pattern - Validating

Problem: Constraint rules define the safe boundary of business processes. These constraints need to be validated through process execution for every business activities. As we described in the policy model, these constraints with security aspects are significant for the business.

Pattern: Table 6.3

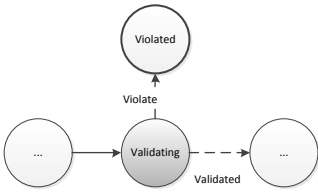
FSM pattern	Transition action
	$ta_{validated}$ $ta_{violated}$

Table 6.3: Pattern graph

All types of constraints could be viewed as *Assertions* in the business activity lifecycle that return true or false on validation. The process consumers should be allowed to validate their assertions before entering the next expected state, more specifically, before and after the $s_{executing}$. We define a governance state as follows:

- a validating state $s_{validating} \in S_g$: indicates the constraint validation actions could be performed. The activity is navigated to a $s_{validating}$ for constraint validation.

We define the following two transition actions for the $s_{validating}$. They are represented by two possible decisions made from constraint validation.

- a validated action $ta_{validated} \in TA_g$: indicates a decision on the constraint validation is validated (mapped to $PA_{validate}$). It transitions the $s_{validating}$ to the next expected state.

- a violate action $ta_{violate} \in TA_g$: indicates a decision on the constraint validation which is a violation (mapped to $PA_{violate}$). It will transfer the $s_{validating}$ to the $s_{violated}$ state.

For Fault rules :

Pattern - Handling-Pre

Problem: Fault rules define remedial actions for the $s_{violated}$ state. These remedial actions need to be applied on process executions, such as for service adaptation. The remedial actions are different and perform differently depending on whether violations occur before or after the $s_{executing}$ in the activity life cycle. The problem needs to be addressed separately. We first address the problem for violations occurring before the $s_{executing}$ state ($s_{violated_{pre}}$).

Pattern: Table 6.4

FSM pattern	Transition action
	ta_{skip}
	ta_{ignore}
	$ta_{replace}$
	ta_{cancel}

Table 6.4: Pattern graphs

Process consumers should allow to apply remedial actions when violations occur. We define a governance state as an after state of the $s_{violated_{pre}}$ state.

- a handling-pre state $s_{handling_{pre}} \in S_g$: indicates a remedial action could be performed for a violation occurring before the $s_{executing}$. The activity is navigated to the $s_{handling_{pre}}$ from the $s_{violated_{pre}}$ state for remedies.

We could define the following transition actions for available remedial actions

- a skip action $ta_{skip} \in TA_g$: will skip the $s_{executing}$ of the activity (mapped to PA_{skip}). The activity is transferred from $s_{handling_{pre}}$ to an s_* after the $s_{executing}$ state. With the process, the input message of the service is expected to be copied to the output message, or a new message is initialized for the output message if it is required for the transition actions.
- an ignore action $ta_{ignore} \in TA_g$: will ignore the violations of the activity (mapped to PA_{ignore}). The activity is transferred from $s_{handling_{pre}}$ to $s_{executing}$ state. Violations are considered acceptable, or do not affect the overall business goal achievement in this case.
- a replace action $ta_{replace} \in TA_g$: will replace the service reference of the activity as service adaptation (mapped to $PA_{replace}$). The activity is transferred from $s_{handling_{pre}}$ to an s_* before the $s_{executing}$ state. With the process, the alternative service reference supplied with ($PA_{replace}$) will be used as the replacement service reference for the current activity performing instance. If the attribute of the provider action indicates permanent service adaptation, the replacement service reference would be assigned for the activity for the $coor_c$ or consumer's service reference setting in the process provider.
- a cancel action $ta_{cancel} \in TA_g$: will cancel the activity and process. This happens when the cancellation of the process execution is required (PA_{cancel}). The current activity is transferred from $s_{handling_{pre}}$ to s_{end} , ending the activity life cycle.

Pattern - Handling-post

Problem: This is a continuing problem with fault rules. The problem is that violations could occur after the service $s_{executing}$ state ($s_{violated_{post}}$). The consumers need to be able to apply remedial actions for this type of violation.

Pattern: Table 6.5

FSM Pattern	Transition action
	ta_{ignore}
	$ta_{com+ign}$
	ta_{retry}
	$ta_{replace}$
	$ta_{com+rep}$
	ta_{cancel}

Table 6.5: Pattern graph

We define a governance state as an after state of $s_{violated_{post}}$.

- a handling-post state $s_{handling_{post}} \in S_g$: indicates that a remedy could be performed for violations occurring after the activity $s_{executing}$. The activity is navigated to $s_{handling_{post}}$ from $s_{violated_{post}}$ for remedying.

We define the following transition actions for available remedial actions.

- $ta_{ignore} \in TA_g$: will ignore violations of the activity. The activity is transferred from the $s_{handling_{post}}$ to the $s_{completed}$ state.
- a compensate+ignore action $ta_{com+ign} \in TA_g$: will compensate the activity before ignoring violations of the activity. The activity is transferred to the $s_{compensating}$ state, then ending the activity life cycle. In the $s_{compensating}$ state, the compensation service of the activity which is supplied by the $PA_{com+ign}$ will be executed.
- a retry action $ta_{retry} \in TA_g$: will wait an amount of time and then retry the activity (mapped to PA_{retry}). The activity is transferred to the $s_{waiting}$ state, then an s_* before the $s_{executing}$ state. It is mapped to PA_{retry} which defines the waiting time.
- $ta_{replace} \in TA_g$: will replace the service reference of the activity. The activity is transferred from $s_{handling_{post}}$ to $s_{replacing}$ first, then to the s_* . It is same as the $s_{handling_{pre}}$ state.
- $ta_{com+rep} \in TA_g$: is an extension of $ta_{replace}$ in this case. It will compensate the activity, and then replace the service reference of the activity (mapped to $PA_{com+rep}$). It transfers the activity from $s_{handling_{post}}$ to a $s_{com+rep}$ state, which has the role of both $s_{compensating}$ and $s_{replacing}$ states.
- $ta_{cancel} \in TA_g$: will cancel the current activity and process. The activity is transferred from $s_{handling_{post}}$ to the $s_{completed}$ state. All activities with service reference that have successfully executed, will be transferred from the $s_{completed}$ to a Cancelling State (this will be discussed in the following).

Pattern - Cancelling

Problem: Process consumers might want to cancel the current business transactions

for some cases with business processes. This is what the ta_{cancel} action is designed for. Cancelling the process does not only cancel the current activity which is executing. All activities performed in the current process execution (in $s_{completed}$ state) might need to be cancelled to compensate for the effects of the activities.

Pattern: Table 6.6

FSM pattern	Transition action
<pre> graph LR Start((...)) --> Completed((Completed)) Completed --> Cancelling((Cancelling)) Cancelling -- "Compensate" --> Compensating((Compensating)) Compensating -- "Ignore" --> End(((End))) Cancelling -- "Ignore" --> End </pre>	$ta_{compensate}$ ta_{ignore}

Table 6.6: Pattern graph

We define a governance state called $s_{cancelling}$ state to support process cancellation.

- $s_{cancelling} \in S_g$: indicates an activity compensation action that can be performed for compensating the effect. The activity is navigated to $s_{cancelling}$ from $s_{compensating}$ for compensation. The activity might transfer from $s_{completed}$ to s_{end} if no effect was created, for example, the $s_{executing}$ state was skipped.

We define the following transition actions for available remedies.

- a compensate action $ta_{compensate} \in TA_g$: will compensate the activity (mapped to $PA_{compensate}$). It transits from $s_{cancelling}$ to the $s_{compensating}$ state first before the end of the activity life cycle. In the $s_{compensating}$, the compensation service of the activity which is supplied by the $PA_{compensate}$, will be executed.
- $ta_{ignore} \in TA_g$: will ignore activity compensation. The effect is not cared about, or will be dealt with by the process consumer separately from the current process logic execution by the provider. The activity will end its life cycle.

6.3.3 Cache of process activity protocol

The cache of the process activity protocol defines the cache function specification on $COOR^p$. It includes a protocol specific data set of cached data ($PS \subset CAD$) and an action mapping table ($TA_c \rightarrow TA_g$).

As described, PS specifies additional conditions for using cached results. These additional conditions are relevant to the elements of a request of a protocol message of $COOR^p$ ($Weaving_{request}$). In this protocol, a $pa \in PS$ is defined as $\langle s \rangle$ and $s \in S_g$. In other words, a final provider action resulting from policy weaving is expected to be of the same interaction types (described in the coordination cache function) for any $weaving_{request}$ with the same s regardless of other elements when the cache is enabled. More conditions can be added for different types of coordination protocols, such as the activity name or a service reference for a activity. More conditions could further reduce protocol service interaction times on governance states, hence might reduce the performance overhead caused by blocking of governance states waiting for responses from policy weavers. However, more conditions would generate more cached data that needs to be handled by the cache service, and require a more complex algorithm for the policy weaver. This might increase the performance overhead on protocol service interaction each time. A process consumer could select a protocol with cache specification which suits its needs, depending on policy model, policy weaving algorithm, or network situation.

In the FSM definition of the protocol, transition actions are explicitly defined for transitions from a governance state to a governed state. However, process consumers could have three types of extra transition actions TA_c (defined in coordination cache function) for the cache function. They are not included in the TA_g defined in the process activity protocol. Hence, the protocol also defines the mapping (\mapsto) of transition actions from TA_c to TA_g .

In this protocol, the mapping for $COOR^p$ is defined as follows:

1. $\forall s \in \{s_{man_val_{pre}}, s_{man_val_{post}}\}, ta \in TA_c. \exists ta_{validated} \in TA_g. (ta \mapsto ta_{validated})$
2. $\forall s \in \{s_{handling_{pre}}, s_{handling_{post}}, s_{cancelling}\}, ta \in TA_c. \exists ta_{ignore} \in TA_g. (ta \mapsto$

$ta_{ignore})$

The mapping for $COOR^c$ is defined as follows:

1. $\forall s \in \{s_{man_{pre}val_{pre}}, s_{man_{post}val_{pre}}, s_{man_{pre}val_{post}}, s_{man_{post}val_{post}}\}, ta \in TA_c. \exists ta_{validated} \in TA_g. (ta \mapsto ta_{validated})$
2. $\forall s \in \{s_{validating_{pre}}, s_{validating_{post}}\}, ta \in TA_c. \exists ta_{validated} \in TA_g. (ta \mapsto ta_{validated})$

Even the mapping for $COOR^c$ is defined, but it is not restricted in the protocol for a consumer, if the $coor^c$ is on the consumer side. The process consumer is free to change the mapping of the $coor^c$ for its own implementations at any time, as the process providers does not need aware of the changes and it will not affect other process consumers either.

6.4 Coordination implementation with BPEL

The defined coordination protocol needs to be implemented to enable coordination. The difficulty is on the provider side implementation, since all activities within a business process need to comply with the protocol during the process or BPEL execution. From our study with related work on coordinated BPEL services and policy frameworks with adaptive BPEL, the implementations could be classified into two categories.

1. The implementation is separate from BPEL processes.

This approach is commonly used for protocols or policies defined for Web services only. Such as WS-Policy, WS-Coordination and extended protocols [124]. The advantage is that it maintains the simplicity of the BPEL process. The BPEL designers do not need to be aware of the protocol for Web services. The protocol is implemented as a wrapper or middleware for Web services. However, for protocols or policies designed for business processes, the implementation requires integration with the BPEL engine, such as [68]. The disadvantage of this approach is platform dependence and that a special BPEL engine is required.

2. The implementation is realized in BPEL processes.

In many policy frameworks and adaptive processes implementations, a set of templates or patterns for BPEL processes is designed, such as [17], [59]. The original BPEL processes need to be instrumented or developed according to these defined templates. The advantage of this approach is platform independence. No additional add-on or modification is required for the BPEL execution environment. However, the disadvantage of this approach is that it makes BPEL processes large and intricate.

In our approach, we designed a set of templates for BPEL development to avoid platform dependency. In this case, the protocol would be implemented with a BPEL process as a *coord^p* for activities. The BPEL contains the flow logic to be executed and could be driven by protocol messages. A process instance, not the BPEL process, is associated with a coordination conversation belonging to a consumer to provide the multi-tenancy capability.

We divide the FSM of *COOR_p* of the protocol in two parts. The first part of the FSM is process independent, i.e., does not require awareness of the states of the business process. The implementation of this part is wrapped up in a wrap service in the main BPEL process. The second part continues the FSM to the end state of activities of the main business process. The first part could be implemented in BPEL processes but separated from the main process. Through this hybrid design, we offer a platform independent approach, and keep the main BPEL relatively simple as well. The disadvantage is that BPEL processes are protocol specific.

The BPEL transaction scope concept [182] is applied for implementing the protocol with BPEL for supporting LRTs. LRTs in BPEL are centred on scopes and scopes can be nested. Nested scopes could be standalone BPEL subprocesses which are business activities of the parent process. When a fault occurs in a BPEL process, all previous committed activities either can be compensated within the fault process, or compensated as an activity in its parent process. This is defined in the BPEL process from process providers and exposed to process consumers.

We designed two templates for BPEL process development to minimise the need for development effort on protocol implementation. A template defines the program skeleton of an algorithm from the template method pattern [183]. One or more of the algorithm steps can be overridden by subclasses to allow differing behaviours while ensuring that the overarching algorithm or the protocol is still followed.

We extract the first part of FSM as the non-transactional requirement FSM for business activities of a process. The second part is an extension for business activities to support process transaction requirement. The FSM is separated in two implementation parts with two templates: the wrapper service template and the main process template. They are described in the following subsections.

6.4.0.1 The wrapper service template

The wrapper service is an implementation of the first part of the FSM of $COOR_p$ that contains activity states from the s_{start} to the $s_{completed}$ or s_{end} state. A while loop block is used in state machine implementation. The wrapper service will not be exited unless the activity which is in the $s_{completed}$ or s_{end} state, which indicates the activity is able to enter the second part of the FSM implementation.

According to the protocol, the process or coordinator needs to determine the transition with $s_{executing} \times 0 \rightarrow s_{violated_{post}}$ and $s_{executing} \times 1 \rightarrow s_{man_val_{post}}$. The activity should be navigated to the $s_{violated}$ from the $s_{executing}$ state when a runtime fault occurs during service execution for the activity. This is achieved by a ‘Catch’ block for the $s_{executing}$ state in the implementation (Figure 6.8). The exception message (e.g. ‘HTTP status code 404’) will initialize an extended violation type, which could be handled by fault policies. Hence, the protocol is designed to allow process consumers to define fault policies for both runtime and business faults.

After exiting the first part of the FSM, the final output message of the service execution would be returned to the main BPEL process. Still, the necessary context information for

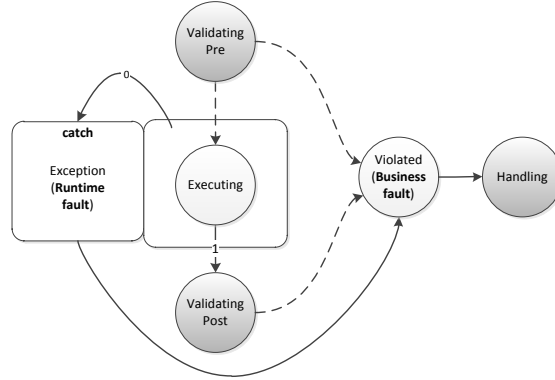


Figure 6.8: Exception situation description

the second part of the FSM execution also needs to be passed to the main BPEL. The output of the main BPEL process is $\langle \langle sere, resp \rangle, \langle canc, comp \rangle \rangle$, where

- *sere* - the service endpoint reference which is assigned for the activity execution. It is assigned at the s_{start} and $s_{executing}$ state.
- *resp* - the service response message after the $s_{executing}$.

The $\langle sere, resp \rangle$ provides a snapshot of the activity for activity compensation if needed.

- *canc* - a Boolean variable that indicates if the current process is in cancelling status
- *comp* - a Boolean variable that indicates if the activity needs to be navigated to $s_{cancelling}$ if the current process is in cancellation status.

The $\langle canc, comp \rangle$ provides the process status information, allows the processes to determine the transitions with $s_{completed} \times 0 \rightarrow s_{cancelling}$ and $s_{completed} \times 1 \rightarrow s_{end}$. The following defines the tuple value assignment in state transitions in the wrapper service template. (\diamond denotes to keep the previous value)

1. $\{0, 1\} \leftarrow s_{start} \times 1$
2. $\{\diamond, 1\} \leftarrow s_{handling_{pre}} \times ta_{skip}$
3. $\{1, 1\} \leftarrow s_{handling_{pre}} \times ta_{cancel}$
4. $\{\diamond, 0\} \leftarrow s_{executing} \times 0$

5. $\{\diamond, 0\} \leftarrow s_{handling_post} \times ta_{com+ign}$
6. $\{\diamond, 1\} \leftarrow s_{handling_post} \times ta_{retry}$
7. $\{1, \diamond\} \leftarrow s_{handling_post} \times ta_{cancel}$
8. $\{\diamond, \diamond\} \leftarrow other\ transitions$

6.4.1 The process template

The process template is an implementation of the second part of the FSM containing activity states from $s_{completed}$ to the s_{end} state. When the business process is in cancelling status, previous successfully executed activities should be compensated if necessary. The template is designed with an activity scope and a process scope, respectively.

Figure 6.9 shows the BPEL template for activity scope associated with activity states. The BPEL template for each activity is an isolated scope. There are two services inside the template indicated by grey boxes. The first service is the wrapper service for the first part of the FSM implementation. The necessary variables are passed into the BPEL process by a BPEL `<assign>` activity. With the following BPEL `<if>` control structure, a `<throw>` activity throws a defined fault if the `comp` variable is set to false. An attached BPEL `<catchAll>` handler catches the fault and does nothing, but to mark this scope as a faulty scope. The BPEL `<compensationHandler>` attachment would only be triggered by a successful scope if the process in cancelling status. In that case, such as the $s_{executing}$ is skipped in the first part of FSM, the compensation handler attached to the activity scope will not be triggered as the scope is marked as faulty. The last `<if>` control structure will mark the process in cancelling status, it throws a defined fault and will be caught in a `<catchAll>` handler defined in the process scope template. Hence, the `<compensationHandler>` handler at activity scope would be triggered. The activities of the process would be navigated from the $s_{completed}$ to the $s_{cancelling}$ state if required. A utility service inside the `<compensationHandler>` is responsible for transition from the $s_{cancelling}$ to the s_{end} state of the activity.

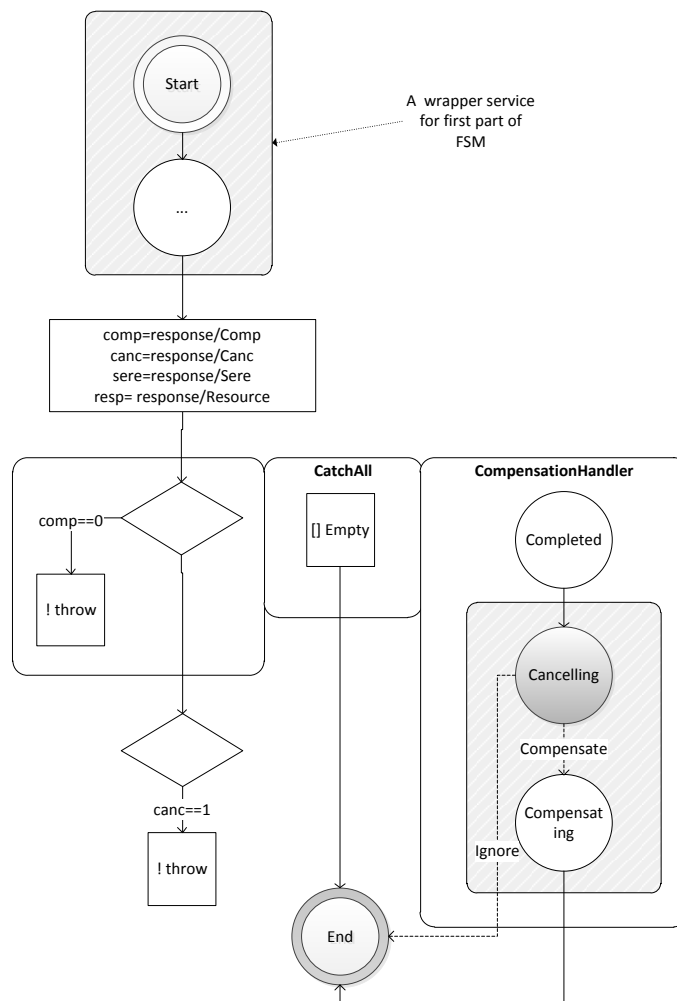


Figure 6.9: Activity scope BPEL template

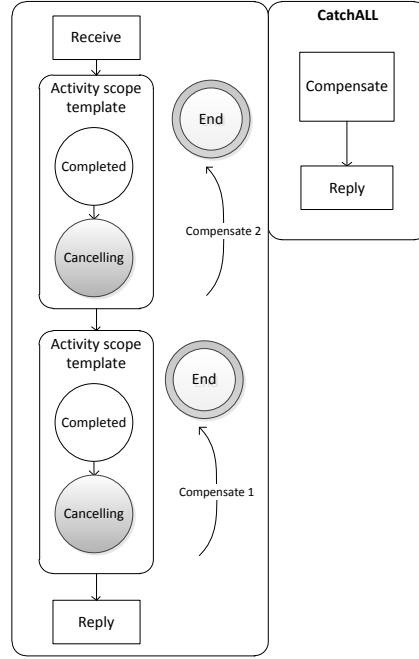


Figure 6.10: Process scope BPEL template

Figure 6.10 shows the BPEL template for the process scope. All activities of the process are inside a process scope, which is attached with a `<catchAll>` handler. If a defined fault for the process cancelling is caught by the handler with the process scope, all `<compensationHandler>` handlers of activity templates of fault-free activity scopes are executed in a backward order, which is specified in the process design. Activities in $s_{completed}$ will transition to the $s_{cancelling}$ state. If this process is a subprocess, after this subprocess cancelling is completed, the activity that represents this subprocess would transition to $s_{violated_{post}}$ in its parent process depending on constraint policies of the activity. The consequent violation handling would depend on the fault policy defined for the activity in the parent process.

6.5 Case study

In this section we are going to discuss a case study in which our coordination framework prototype was employed. The case study focuses on evaluating the following two aspects:

1. The effectiveness of the coordination framework
2. The performance overhead in the coordination framework

These are described in the following subsections titled objective, approach, and result.

6.5.1 The effectiveness on the coordination framework

6.5.1.1 Objective

The business process described in the previous section is developed with BPEL templates for use in this case study. Any limitations of the approach through the case study will be discussed.

As a primary research objective, we must provide an effective prototype in order to prove the concept. The effectiveness means that the business process can be governed in a distributed and multi-tenant environment using our approach, i.e., policies are enforced in business process executions for multiple consumers at the same time for their business goals to be achieved. We use a test case approach in this study. We design a number of test cases to determine whether the system is effective. We expect all cases to be successful to demonstrate the effectiveness of our approach.

Our study is divided in two stages: firstly, we verify its effectiveness for a single process consumer; in the second stage, we verify its effectiveness for multiple process consumers, examining its multi-tenancy capability.

6.5.1.2 Approach

The purchase order checkout BPEL process is developed for the experimental setup. A small set of alternative services are also developed for test cases related to $PA_{replace}$ reme-

dies. All of the service context information required for constraint validation and service selection are manually and randomly assigned.

A test case comprises of five parts of information,

1. Process: a target process of this test case. Some test cases are targeted on a sub process level.
2. Defined input: a section of SOAP message of the business process input that contains the business object information.
3. Defined policies: policies defined for the business process.
4. Expected process activity log: refers to expected activities and states information log in a process instance.
5. Expected output: a SOAP message referring to the expected output from the process instance.

For a target process, each part of the information used in the test case was also defined for the different process consumers in the second stage of the experiment. The following table displays a part of an example of the test case script.

When the real result matches with the expected process activity log and expected output of a test case, we state that the test case is successful. For comparing with the expected process activity logs of test cases, the real process execution instance is traced by implementing our own logging code, and the BPEL engine execution log as taken from the BPEL engine admin console. Our own logging code records every step of coordination conversations and policy weaving. The screen shot (Figure 6.11) shows a very small part of logging results.

In the first stage, we developed a total of 21 test cases for Consumer_1 only. These cases were designed to cover four categories of rules with different scenarios: for example, a test case with three constraints for validating the security context of activities. Afterwards, we compared the real process execution and coordination log, following the process execution, with the expected process activity log to verify whether the validations have occurred. In

```

output
Java DB Database Process * x GlassFish Server 3 * x
INFO: combining : 1
INFO: Combining algorithm: : defaultFlexibilityPolicyCombining
INFO: orderInspectionFaultPolicy3 policy_result : ie.dcu.computing.mwang.spap.PaUndefinedType@1b241
INFO: combining : 2
INFO: Combining algorithm: : defaultFlexibilityPolicyCombining
INFO: orderInspectionLockingPolicySet3 policySet_result : ie.dcu.computing.mwang.spap.PaUndefinedTy
INFO: Weaving Result : class ie.dcu.computing.mwang.spap.PaUndefinedType
INFO: resource updated : false
INFO:
Order Inspection : state >>> VALIDATING_PRE
INFO: wrPolicy : WrPolicy(process=ie.dcu.computing.mwang.spap.ProcessType@12a07d3activity=ie.dcu.com
INFO: Weaving PolicySet : orderInspectionLockingPolicySet3
INFO: Weaving ActivityStates : true
INFO: Weaving Policy : constraintOrderInspectionPolicy3
INFO: Weaving ActivityStates : true
INFO: Weaving Rule : constraintValidateRule3
INFO: Weaving ActivityStates : true
INFO: ConditionExpression : /WeavingRequest/Activity/ServiceReference/Operation='orderInspection'
INFO: ConditionExpression : /WeavingRequest/Activity/ServiceReference/Address='http://localhost:8080
INFO: Weaving Conditions : true
INFO: rule_result : ie.dcu.computing.mwang.spap.PaValidateType@ed9b24
INFO: Weaving Rule : constraintViolateRule4
INFO: Weaving ActivityStates : true
INFO: Weaving Conditions : true
INFO: rule_result : ie.dcu.computing.mwang.spap.PaViolateType@1648c40
INFO: combining : 2
INFO: Combining algorithm : : Pa-Violate-Unless-Pa-Validate
INFO: constraintOrderInspectionPolicy3 policy_result : ie.dcu.computing.mwang.spap.PaValidateType@e
INFO: Weaving Policy : orderInspectionFaultPolicy3
INFO: Weaving ActivityStates : true
INFO: Weaving Rule : remedyRule3
INFO: Weaving ActivityStates : false
INFO: combining : 1
INFO: Combining algorithm : : Pa-Violate-Overrides-through-All
INFO: orderInspectionFaultPolicy3 policy_result : ie.dcu.computing.mwang.spap.PaUndefinedType@33d40
INFO: combining : 2
INFO: Combining algorithm : : Pa-Violate-Overrides-through-All
INFO: orderInspectionLockingPolicySet3 policySet_result : ie.dcu.computing.mwang.spap.PaValidateTyp
INFO: Weaving Result : class ie.dcu.computing.mwang.spap.PaValidateType
INFO:
Order Inspection : state >>> MANIPULATING_POST_VALIDATING_PRE
INFO: wrPolicy : WrPolicy(process=ie.dcu.computing.mwang.spap.ProcessType@labbfid2activity=ie.dcu.com
INFO: Weaving PolicySet : orderInspectionLockingPolicySet3
INFO: Weaving ActivityStates : true
INFO: Weaving Policy : constraintOrderInspectionPolicy3

```

Figure 6.11: Screen shot of coordination log

Process	Purchase order checkout
Defined input	For Consumer1: <pre> <PurchaseOrder> <id>1234</id><buyer>...</buyer><seller>...</seller> <buyerAccount>...</buyerAccount><sellerAccount>...</ sellerAccount> <item>...</item>... </PurchaseOrder> </pre>
Defined policies	For Consumer1: <pre> <p1:PolicySet policySetId=" PurchaseOrderCheckoutPolicySet2010" priority="0"> <p1:Objects>...</p1:Objects>...<p1:Policy policyId="... " priority="0">...</p1:Policy>... </p1:PolicySet> </pre>
Expected process activity log	<pre> ProcessConsumer1: Order inspection : Manipulating- Validating-Pre ProcessConsumer1: Order inspection : Manipulating-Pre- Validating-Pre ... ProcessConsumer1: Order inspection : Validating-Pre ... </pre>
Expected output	For Consumer1: <pre> <PurchaseOrder> <id>1234</id> ... </PurchaseOrder> </pre>

Table 6.7: A test case script example

another similar test case, we manually changed the security context information under the requirement of a constraint rule. We traced process execution to verify whether the defined fault rules are weaved, and the final remedy is applied in the process execution. We also forced the undeployment of service applications, and made them return invalid responses to generate exceptions and violations.

In the second stage, we developed 10 test cases that involved two consumers (Consumer_1 and Consumer_2). Both consumers had different defined inputs, policies, expected process activity logs and expected outputs. For these test cases, we made two consumers continually and simultaneously send a number of process requests to verify if the policies of each consumer were enforced and whether there was interference between each other. We also forced the slowdown of the policy weaver on one of the consumers, and on one process request instance of one consumer, to ensure messages received by BPEL processes

did not follow a particular sequence.

6.5.1.3 Result and discussion

Our test cases cover all four types of rules and can be applied in various situations. With the successful test cases, we can demonstrate that our approach provides an effective coordination solution for governance in a distributed and multi-tenant environment. The activity centric process coordination protocol design can be applied to any business process. The process runtime governance is both process instance and consumer based. In addition, there was no interference between different consumers sharing a single BPEL process at the same time, so this highlights its multi-tenancy capability. This provides a great advantage for process providers offering business processes to multiple consumers, just like Web services.

Our approach assumes that coordinators and BPEL engines never fail. Failures of coordinators could result in un-handleable exceptions in process execution. With the failures of BPEL engines, BPEL engine solutions might support the restoration of persistent BPEL instances after the failure. However, this persistence could add additional overheads to the coordination and raise security concerns regarding storing information on the provider's side.

There needs to be some effort made to implement coordination frameworks on both the process provider and consumer side in our approach. However, once developed, the policy weavers and $COOR_c$ can be used for any business process. The only question that would be raised regarding the development is the difficulty of BPEL development with $COOR_p$. As we described, the process activity protocol is implemented with BPEL processes following the templates. That means additional efforts are required in BPEL development compared to conventional BPEL development. However, from our own experience with development for this case study, the effort required is small. The wrapper service development only requires a few lines of code for a business activity, once the first template is developed. As well as that, the main BPEL template is relatively simple.

Moreover, there is a restriction on BPEL development with our approach to protocol implementation. A business activity is in an isolated scope in BPEL processes with the template we designed. All BPEL activities for calling a service for the business activity must be placed in the scope, meaning that BPEL `<invoke>` and `<receive>` must be grouped together for asynchronous service calls. Hence, the business activity must be placed in parallel with other activities in the BPEL design to avoid blocking if it is needed.

6.5.2 The performance overhead on coordination framework

6.5.2.1 Objective

The execution aspect of our approach is inherently time consuming, but a performance overhead is also expected on coordination conversations. Once the activities of a process instance are in a governance state, the process would be blocked and wait for a provider action or policy decision from the process consumer. In this evaluation, we would like study the exact impact on process performance with coordination.

Our study commences with a single activity rather than a complete BPEL process, yet, the result still can be aggregated for the BPEL process. The time cost of an activity equals the duration of that activity's life cycle, and can be viewed as service execution time plus coordination time. The service execution time is the time spent in the *s_{executing}*. Both rest states and transition time are counted as the coordination time required governance. However, the time cost of during and after the *s_{completed}* is not counted as an overhead, since the process will not block the after activity in the same sequence. This activity would start without waiting for the previous activity to end. For each process instance, the same activity can travel through different sequences of states depending on the policies defined or customization made by the consumer. For example, *s_{violated}* only exists when constraint rules are defined. Hence, our study is divided into two different situations: violation free and violation occurring situations.

6.5.2.2 Approach

For both of these situations, two governance states ($s_{man_val_{pre/post}}$) must be passed for all activities in the FSM of $COOR_p$ to reach the $s_{completed}$ state. These two governance states are considered to be the coordination overhead in violation free situations.

We used a local machine for an in-lab experiment. The setup used 3.0 GHz single core process with 1 GB ram Windows XP VMware virtual machine. We used the *purchase order inspection* activity as a concrete activity for the case study. Policies involved with this activity are described in the policy case study section. The figure (Table 6.8 left) shows the result of 1000 test cases of the activity execution with a time cost of two governance states when the coordination cache was disabled. (Please note, the periodic peak time that occurred in test cases is not expected within our framework. It is related to the VMware VM operations (e.g. garbage collection) in our experiment setup.)

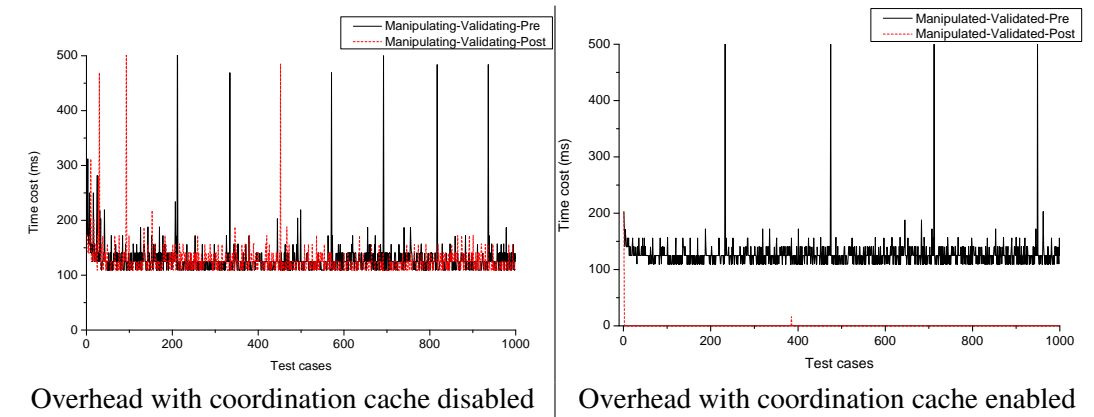


Table 6.8

The statistical test results are summarised in the following table,

State	Total test cases	Mean	Standard deviation	Minimum	Median	Maximum
$s_{man_val_{pre}}$	1000	129.46	35.59	109	125	610
$s_{man_val_{post}}$	1000	125.50	26.74	109	125	532

From the figure and table in above, we can see that the time costs for both governance states are similar, with less than 130 ms for the mean value. The time cost of the $s_{man_val_post}$ state is slightly lower in mean value, as the no policy is defined ($PA_{undefined}$) for the state of the activity. However, the difference is minimal, less than 4 ms in this case. Hence, for an activity, the overhead of governance states with no policy involved should not be expected to be much less compared to governance states which policies are involved.

The figure (Table 6.8 right) shows the result of the same 1000 test cases when the coordination cache was enabled. The cache is validated for all test cases, as we did not change any policies during this evaluation.

The statistical test results are summarised in the following table,

State	Total test cases	Mean	Standard deviation	Minimum	Median	Maximum
$s_{man_val_pre}$	1000	126.86	28.38	109	125	531
$s_{man_val_post}$	1000	0.217	6.38	0	0	201

From the figure and table in above, it is apparent that the time cost for the two governance states are significantly different. We used in-memory cache design as implemented by the singleton pattern. The time cost mean value of $s_{man_val_pre}$ state is similar to when the cache was disabled, but $s_{man_val_pre}$ has less than 1 ms overhead in this case. Hence, the coordination cache function could significantly reduce the performance overhead of governance states with no policy involved ($PA_{undefined}$).

To study time cost on different interaction patterns when the cache was enabled, we temporarily added a policy with a ca_{log} action to the activity on $s_{man_val_pre}$ state. Hence, the $coor_c$ needs to be notified at this state. The figure (Table 6.9 left) shows the result of the same 1000 test cases.

The statistical test results are summarised in the following table,

State	Total test cases	Mean	Standard deviation	Minimum	Median	Maximum
$s_{man_val_pre}$	1000	22.18	13.32	0	16	125

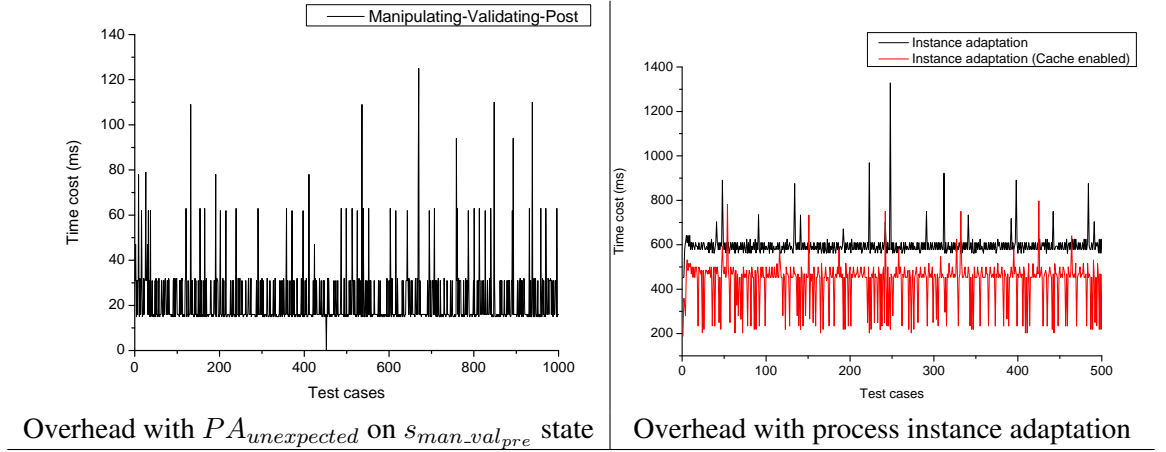


Table 6.9

From the figure and table above, we can see the overhead (22.18 ms) is smaller compared to when the cache was disabled (125.50 ms), even after we added a policy. When only consumer actions are defined for the governance state of the activity ($PA_{unexpected}$), a one way interaction without blocking the process could reduce the performance overhead on the governance state. In this case, the overhead is reduced 82.33% compared with when cache was disabled.

In the above study, there is only one policy *constraintOrderInspectionPolicy3* involved with the $s_{man_val_pre}$ state. Large amounts of policies are expected to be defined by the process consumers for each activity. We copied multiple *constraintOrderInspectionPolicy3* into the *orderInspectionLockingPolicySet3* PolicySet in the policy file in order to study the performance impact related to the number of policies on a governance state. The figure (Table 6.10 left) shows the results of time cost mean values (1000 test cases) with different number of copies of the policies.

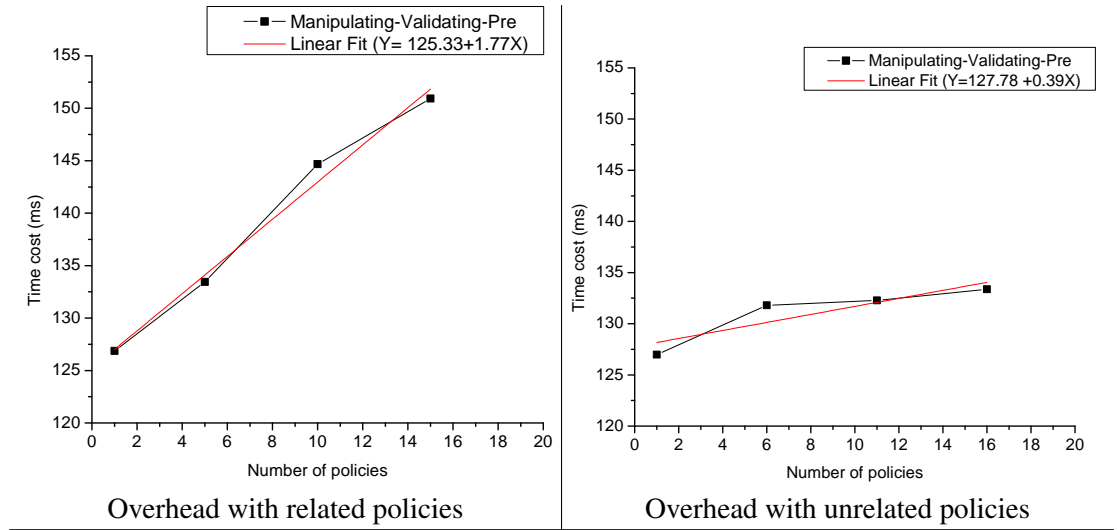


Table 6.10

From the figure, it was evident that the time cost increases linearly as the number of policies increase for an involved governance state. However, the increase is relatively small. The overhead increases by less than 1.77 ms for every new related policy in this case. Yet, the result may be different in different cases: for example, when a large number of *rules* are defined in a single policy.

We also copied multiple *orderInspectionFaultPolicy3* into the *PolicySet* in the policy file to study the performance impact for the unrelated governance state. The figure (Table 6.10 right) displays the result of time cost mean value (1000 test cases) with a different numbers of copies of the policies.

The *orderInspectionFaultPolicy3* is a fault policy that is not related to the $s_{man_val_pre}$ state, but that also increases the performance overhead. For each fault policy in this case, 0.39 ms overhead was added. This was done because fault rules of the policy will be weaved and will return $pa_{undefined}$ actions in this case. These actions still need to progress through the policy combining algorithm of the policies and policy sets defined for the $s_{manipulating}$ and $s_{validating}$ states, which causes an additional overhead. Hence, the addition of policies could also increase the performance overhead of unrelated governance states.

In the following, we studied performance impact within a violation situation. In such cases, the $s_{handling_{pre}}/s_{handling_{post}}$ governance states need to interact with the $coor_c$ with the policy weaver, and subsequent states might also block process execution depending on the provider actions or remedies.

We used the same experiment setup with the same business activity. In this case, we changed the service reference of the activity on provider's side to simulate a violation situation. Since *constraintOrderInspectionPolicy3* includes the constraint policy on serviceReference, a violation situation was expected. The *orderInspectionFaultPolicy3* would assign a correct serviceReference through $pa_{replace}$ action for instance adaptation. The state travel before $s_{completed}$ in $coor_p$ would be $s_{start} > s_{man_val_{pre}} > s_{violated_{pre}} > s_{handling_{pre}} > s_{replacing} > s_{executing} > s_{man_val_{post}}$ state. We do not consider it to be a special case (i.e. exception or violation) at the moment. The performance overhead is considered to be an aggregation of the above states with the exception of the $s_{executing}$. The difference in performance between the replacement serviceReference and the original serviceReference is not counted in the coordination overhead. The figure (Table 6.9 right) shows overall time cost with 500 test cases.

The statistical test result is summarised in the following table,

State	Total test cases	Mean	Standard deviation	Minimum	Median	Maximum
$s_{man_val_{pre}}$	500	597.57	54.80	453	579	1328
$s_{man_val_{post}}$	500	431.44	101.95	188	454	797

From the above figure and table, it is clear that the mean time cost is a little smaller when the cache was enable, as the $s_{man_val_{post}}$ state can use cached values in this case. When the cache was disabled, instance adaptation takes an average of 597.57 ms. In the worst case scenario, it could take more than one second. Other remedies, such as PA_{ignore} , PA_{skip} , and PA_{cancel} are expected to take less or a similar time as this case. However, when PA_{retry} or $PA_{com+rep}$ are involved, the time cost could be much higher. This would

strongly depend on the wait time and the time for the execution of compensating activity in each case.

6.5.2.3 Result and discussion

When we added together the times cost of the two governance states, we can see the coordination overhead of the activity was around 245 ms with the cache disabled and around 127 ms with the cache enabled in a violation free situation. The overhead was significantly reduced in this case with coordination cache enabled. The actual overhead also depends on the number of policies defined by consumers, as more policies result in a greater overhead. It is less than 2 ms for a new related policy in this case. The overall overhead can increase when we apply it in real world networks with consideration of the network latency. However, we still consider the performance overhead is quite small, as long running business activities take a few hours or even a few days for execution in a process with LRT. In some cases with utility services (e.g. email notification), the business activity is expected with an instant activities response. For example, the average execution time of real world email notification service only takes 854 ms [184]. In this case, our coordination overhead would be greater than 29.7% with the cache disabled and 14.9% with the cache enabled. Since business processes usually are mixed with long running activities for LRTs, the performance overhead for the overall process again is very small and acceptable.

In a violation situation, the coordination overhead mean value for adaptation is around 598ms with the cache disabled. If we deduct the time cost in a violation free situation, the overhead on service adaptation itself would be $598-245=254$ ms for each process instance. Yet, it is possible to set a permanent adaptation to avoid remedy overhead on each process instance. However, the instance adaptation would avoid to store activity service reference information on the provider side anyway. This might be required for security reasons in some business activities by the consumer. Still, we consider the overhead to be acceptable compared with inherent time delays of long running activities. With some remedies, the

time cost could be much longer, such as when a compensation service needs to be executed. However, this is not expected to happen on a regular basis and the overhead is then normally considered to be the necessary price to pay to fix the problem in such cases [17].

6.6 Discussion with related work

In this section, we discuss the related work in service computing that relates to our coordination approach.

Our coordination approach satisfies the three following primary requirements for the architecture framework.

1. Transaction management for business processes

Business processes generally require a transaction feature, i.e., the all or nothing attribute. Business processes are parents of business activities. The changing status of business processes affects the states of business activities. Subprocesses are activities of its parent process. The changing state of activities also affects the status of business processes. The transaction management for a business process needs to consider both business activity and processes.

2. Process adaptation for flexible business processes

As a requirement of process consumers, business processes might need to be customized to satisfy consumers' needs, such as deleting an activity in a process, which is beyond the transaction or fault management. This entails that the coordination supports two types of operations on business processes. The first type of operation allows consumers to adapt processes, realized as flexible rules. The second type of operation enables consumers to determine the need for and to verify the adaptation of processes, realized as constraint rules enforcement.

3. Supporting multi-tenancy requirement

For cloud applications, a single version BPEL process is expected to be shared by mul-

multiple consumers with the multi-tenancy capability. Thus, coordination conversations with process execution instances should be isolated between each of the process consumers. The process provider offers a unified process interface and description for all consumers, but consumers' policies will be addressed and will not interfere with each other.

Now, we compare related work in service computing (detailed in the related work chapter) with our work regarding the above aspects. Firstly, we discuss the WS-TX and extension work [125][124], as our approach addresses protocols that are similar to WS-Coordination related specifications, which are designed for Web service transaction management. After that, we discuss related work in a larger scope, which involves process adaptation and policy enforcement frameworks.

In comparison with the WS-Coordination [3] and two additional protocols (WS-AT [128], WS-BA [126]) that extend the framework and our approach, the differences are:

Firstly, since WS-Coordination only deals with transactions and fault management, the design of the protocols is separated from concerns of policy enforcement. Regarding policies, they suggest using a separate policy framework (WS-Policy) for related problems [128, 126]. However, WS-Policy requires that all participants must mutually agree on a set of policy standards, and policies are completely exposed in plain text to all service providers. With our approach, policy enforcement is considered in the coordination protocol design. The related information for policy enforcement at each state is given to process consumers. Subsequently, the process consumers are free to define their own or customize their own policy specifications, and policies are completely hidden from other participants.

Secondly, the WS-Coordination and extended protocols focus on distributed service transactions rather than transactions of business process. For example, when a coordinator is notified that two services or activities are ready to commit, then a commit action notification could be given to both participants or services. By contrast, our work focuses on process level transactions. The participants of a coordination conversation are a policy weaver and a set of activities connected to a process and subprocesses. Even the proto-

col is designed to be activity centric. Activities are not addressed separately outside the process scope. Cancelling a process will cancel all executed activities rather than only a single activity. Since our approach does not solely on transaction management, the protocol design covers the entire states that are required for the different categories of rule enforcement, such as the manipulating state. to support process adaptation for flexible business processes. Additionally, our approach defines the cache function as being able to reduce the coordination overhead and provide BPEL templates for process development, which are not considered and available within the WS-Coordination framework.

Now, we discuss related work in the field of policy enforcement and adaptive BPEL process. Regardless different approaches, these works are not aware of the multi-tenancy problem. These approaches can be classified into two categories.

The first category is located at the BPEL layer, in which our approach falls. BPEL processes are specially designed or generated to serve the purpose and to provide a platform independent approach. [57] and [63] have a similar approach, where the BPEL specification itself is extended with a fault policy specification. Exception handling policies are bound into process schemas as a BPEL extension. The SRRF framework [59][64] generates SRRF-aware BPEL processes according to the defined policies. However, with these approaches, binding policies into business processes or static policies are certainly not an option for our objective, as it impossible to support multi-tenancy capability.

The second category is located at the BPEL engine layer, so the BPEL process is maintained to be simplified, but is platform dependent. The disadvantage of the Dynamo project [17] is that BPEL event handlers must be statically embedded into the process prior to deployment, meaning that the recovery logic is defined once and for all, and that it can only be personalized through the parametrization of the event handler itself [17]. This approach does not support dynamic policies and certainly, it does not support a multi-tenancy environment. The TWSO framework [65] addresses process transactions. The PAWS framework [60] extends the ActiveBPEL engine to provide a flexible process that can change its

behaviour dynamically, according to variable execution contexts. Similar frameworks [66] [67] [68] [56] also extend the BPEL engine for process adaptation, but without an awareness of the multi-tenancy requirement.

6.7 Conclusion

This chapter presented a coordinator framework with protocols that ensure defined policies are enforced during business transactions with business processes for consumers and providers. We defined the coordination model and protocol for the policy based governance of business processes on business transactions. The BPEL templates are offered for implementation with business processes. We used case studies to evaluate the effectiveness and performance overhead of the coordination framework. Our overall approach supports transaction management, adaptation for flexible processes, and multi-tenancy capability. Still, there are limitations identified with our approach. BPEL process implementations are protocol specific. The BPEL activities of a business activity must be placed in a BPEL scope.

Chapter 7

AOP Enhanced policy framework

7.1 Introduction

This chapter defines an AOP enhanced policy framework for the extensibility of our policy framework, to address additional requirements that might be needed for the process runtime governance by consumers.

We have provided an XML policy model and a coordination framework based on protocols for a policy based governance framework. We cannot expect our policy framework to offer process consumers all of the features expected in process governance. In other words, business policies or consumers' requirements might not easily be expressed by our policy model in some cases: for example, predicting supported policy decisions and the complex sliding time window conditions. This requires different policy modelling approaches for the policy language model and additional related algorithms for policy weaving using policy weavers. These features consist of many research domains that are not evident in our policy model and framework. It would be impossible to cover them all with a single policy framework. More details regarding the motivation behind our research will be discussed in this chapter.

The AOP framework is an extension of our policy framework. It is designed to offer

policy engineers a programming approach for policy development, so that it is possible to adopt other policy models or frameworks in addition to our model. Moreover, the advantages of AOP, such as modularity, and reuse, are kept in the overall governance system. The enhanced policy framework does not depend on AOP, meaning policy developers do not need to use AOP if they don't want to or it is not necessary. The AOP enhancement complements our schema-based XML policy development by providing an alternative and powerful policy development solution for policy developers.

The contribution comes from two perspectives:

1. Introducing an enhanced policy model with policy aspect extension

The existing policy model will be extended using the policy aspect model. This enables the adoption of other policy/rule models or frameworks, such as Jess rules, on top of our existing policy model using the AOP paradigm. This seamless approach allows policy developers to extend the policy framework for additional features without compromising on our XML policies, while also providing a master policy model for the overall governance framework.

2. Introducing a distributed and multi-tenant AOP framework

The aspect model has been developed based on the policy model, which sits on top of the coordination protocol. An upgraded policy weaver needs to cover the functions of aspect weaving. The AOP framework supports a distributed and multi-tenancy environment, which is not discussed and addressed in any other AOP framework to the best of our knowledge.

Our work is derived from a comparison between the policy based system and the AOP paradigm. The contribution of this chapter lies in the coherent mapping of the aspect concept onto the policy model: the business process is as the target program of aspects, which are realized as policies. Consequently, effects of aspects weaved in the process logic are policy decisions. This mapping is described in the first section of this chapter outlining the concept design of the policy AOP. The overall AOP framework consists of:

- Conceptual policy AOP modelling

- A detailed policy AOP specification

This chapter is organized as follows: in Section 7.2, we explain both our motivation and the concepts; in Sections 7.3 and 7.4, we describe our policy AOP specification, and aspect deployment and weaving; Section 7.5 details the case study; while in the remaining sections, we compare our work with related work and draw our conclusions.

7.2 Policy AOP motivation and concept

7.2.1 AOP motivation and capabilities

We provided an XML based policy model based on the coordination protocol. There are still some issues we can address:

- First, our current policy model and framework has limitations.

Our policy model may be unable to effectively or easily express some business policies. For example, it struggles to express conditions with time window based queries, results from logic reasoning of a set of rules. Although, our policy model and framework are unable to satisfy the requirements that arose in every aspect, we argue that this limitation is also apparent in any other single policy model or framework.

- Second, an integrated approach for multiple policy models and frameworks is absent.

There are others rule/policy systems, such as Jess rules, that might be required by process consumers to cover the limitation of our policy model. Yet, each policy approach has its own advantages and limitations, so it is difficult to find an all-in-one solution. Process consumers might directly practice on the coordination protocol to adopt multiple policy frameworks. However, simply adding multiple policy models on top of the coordination protocol can cause conflicts in any overlapping aspects. Policies in different subsystems can result in different decisions or provider actions. It can not be solved by the coordination protocol, and this causes process consumers to choose one policy approach and drop others to satisfy one aspect.

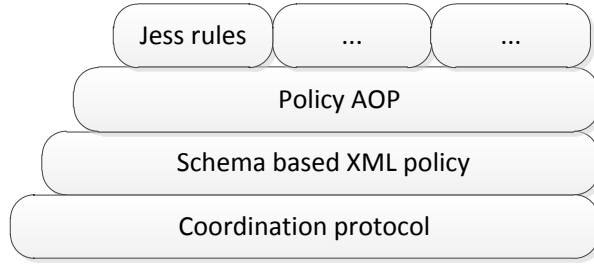


Figure 7.1: Policy based governance framework stack

Different from other joint work of policy and AOP, such as [143] [144], an AOP approach represents an implementation approach for policy enforcement to address modularity and implementation separation from the target system. The goal of our policy AOP is to provide an *aspect-oriented* programming approach on top of our XML policy (Figure 7.1). A more powerful programming language acts as a policy language syntax in the policy AOP to extend the XML policy model. Firstly, this allows defining more complex policies or interaction with additional systems by utilizing the power of the programming language. Secondly, the policy AOP allows for other policy models to work on top of our policy model, giving users the flexibility to adopt other forms of formal policies. Our policy model is underlying, and is extensible with other policy approaches, and functions as a master policy in the overall policy framework. It differs from the widely used policy handler chain pipeline approach [56] with multiple policy models on Web services. Each policy model only addresses one aspect, e.g., security, as a handler in the chain, and as a result, they acquire policy conflicts that never occur between policies defined in different models. Using our approach, policy developers could define policies for any aspects within the adopted policy models. All possible conflicts could be addressed by referring to our master policy model. For example, both the Jess rule and our XML policy could be used to define security aspect policies at the same time.

In the following, we describe two additional policy frameworks that might be required additionally for a governance system in a real world environment. Moreover, our policy

AOP is capable of adding these other frameworks into our governance framework for extensions. These two systems of integration will be examined in our case study in order to prove the concept.

- High level policy for process monitoring

For both technical and BAM monitoring (described in Chapter 2), the monitoring tools are similar or the same. The common technology behind the tools is Complex Event Processing (CEP) with event processing query languages for process events. However the *logging* approach is difficult to handle and provide real-time, or near real-time, conclusions on large amounts of real-time data [185]. In such cases, the CEP might be added to a policy framework for high level policies: for example, where there are constraints with time windows.

- Complex policy with rule reasoning

Although our XML policy model has provided a formal policy model, different approaches can offer unique advantages, which our policy model does not provide. For example, business rules allow reasoning about decisions from a set of asserted facts. Process consumers might need to adopt logic rule languages to address their problems relating to complex reasoning. In such a case, other formal languages, such as Jess rules [95] for business rules, might be required to express business policies.

7.2.2 Policy aspect model

Designs of many dynamic AOP frameworks are influenced by the ECA rule [186] [187]. Dynamic AOP can be achieved regarding a running target program as a series of events that signal the occurrence of join points. Many works use AOP as an implementation approach for policy enforcement [143][144], where aspects are integrated into a target program as a non-intrusive approach for policy injection. All of these works have shown that policies and aspects have similar concepts in many aspects, especially with the ECA rule based policies, and this provided the inspiration for our work. In a policy aspect model, an XML policy

and an aspect are mapped and provide a unified and consistent policy model. In fact, the mapping was considered within XML policy model design, thus resulting in a seamless mapping result.

In the policy aspect model, the requirements of all categories of rules are considered as crosscutting concerns. An aspect equals a policy as a module of concern. To realize a aspect as a policy, we defined a mapping pattern between *aspect* and *policy* elements in the policy aspect model (Table 7.1). Firstly, this provides an overall view of the policy model comprises of the XML based policy and the policy AOP extension. Secondly, it provides an aspect model for defining AOP specification, as will be described in the next section.

Aspect model	XML Policy model
Aspect	Policy
Join point	Object
Advice type	Activity state
Advice1 (Void return)	Consumer action(s)
Advice2 (Non-Void return)	Provider action with/without Consumer action(s)

Table 7.1: Policy Aspect model

1. Join point maps to Object. Interesting points in the target program refer to policy objects involved in the business processes.
2. Advice is mapped to Action(s) of rules. There are two categories of advices: the first category has a void return, meaning the rule only contains consumer actions. It does not expect any effect or provider action on the business processes (i.e., $PA_{unexpected}$); the second category has a return value, meaning that one provider action is defined in the rule.
3. Advice type maps to Activity state. In this case, available advice types depend on the join point, as not all join points are *activity* objects. Mapping join point only to

activity could fix this issue. However, the policy aspect model will have an inconsistent view with the XML policy model.

4. Aspect maps to Policy, which packages the rules. Since in a multi tenancy environment, similar to our XML policy, the ownership of aspects restricts its valid range. For each process consumer, the valid ranges of aspects are only the process instances that are created by its own requests.

7.3 Policy aspect specification

There are various AOP frameworks that are developed following the AOP concept. They generally use standard AOP terminology, which includes pointcut, advice, etc (described in Chapter 2). The differences between the frameworks are specifications about the terminology. The specifications describe the frameworks that are designed for different target environments or that have different capabilities. In this section, we describe the policy aspect specification. Also, the join point, pointcut, etc., in policy aspect specification are described in the following subsections.

7.3.1 Join point model

A critical part in the design of any aspect-oriented language is the join point model [188]. The join point model provides the common frame of reference that makes it possible to define the dynamic structure of cross cutting concerns. Our AOP framework supports the join point model that is identical to the *Object* element of the XML policy model. As a consequence, the different kinds of join points are *business process execution*, *business activity execution*, *resource/BO request/response*, and *violation occurrence*. They are basic situations or elements capturing business information in process executions, and have been described in the *Object* element of the XML policy section (Section 5.3).

In comparison with other AOP frameworks such as AspectJ and AO4BPEL, that tar-

get programming languages, our join point model functions at a high level and does not cover any program specific join points derived from the program's code structure. Examples are *constructor call* or *field get* join points in AspectJ or SpringAOP for Java [189] [190]; *invoke activity* or *sequence activity* join points in AOP4BPEL for BPEL [141] Our join point model remains at a general business process logic level rather than caring about any workflow or programming language implemented the process logic. Firstly, for a result of consistency with the XML policies, cross cutting concerns are implemented with aspects targeting business centric problems resulting from business policies, which are more about business level information. Secondly, our AOP framework is built on top of the coordination protocol, which does not specify any context information for any programming language for business process implementation. Since we are not restricted to any programming languages, the framework is not limited to BPEL processes. The process provider could use other workflow languages other than BPEL. However, our work will not discuss problems and solutions on the providers' side with regard to other workflow languages.

7.3.2 Pointcut language

In many AOP frameworks, such as AspectJ, pointcuts are designed by built-in Pointcut Designators (PCDs), which are predicates on join points. A PCD denotes a kind of join point, such as *call(method)* and *get(field)*. These fixed, built-in sets of PCDs have some disadvantages. They are not extensible and fail to provide operations to manipulate or reason about pointcuts beyond weaving [191]. For this reason, some AOP frameworks [191] [18] propose a functional query based pointcut language. In this case, the program source code structure is represented as a data model, for example in an XML data structure. A query language such as XQuery [192] or XPath [193] is then used as the pointcut language specification.

We provide an interconnecting approach in our pointcut specification. We offer a set of fixed PCDs and also support the use of query pointcut. In our approach, the AOP framework

Pointcut	<i>process</i> (name signature, sma(algorithm signature, parameter))
Description	Select join points whenever the specified business process is requested.
Pointcut	<i>activity</i> (name signature, sma(algorithm signature, parameter))
Description	Select join points whenever the specified business activity is requested.
Pointcut	<i>resource</i> (name signature, sma(algorithm signature, parameter))
Description	Select join points whenever the specified business object is requested.
Pointcut	<i>violation</i> (type signature, sma(algorithm signature, parameter))
Description	Select join points whenever the specified constraint violation has occurred.
Pointcut	<i>query</i> (XPath expression)
Description	Select join points whenever the query has returned true.

Table 7.2: PCDs definition

supports the semantic pointcut with extensible self-defined semantic matching algorithms with fixed PCDs for join points, and also the ability to utilize the power of standard functional query language for pointcut expressions. We define five PCDs in our approach (Table 7.2).

These PCDs include *process*, *activity*, *resource*, *violation*. These kinded PCDs matching are based on the kind of a join point. They support the semantic pointcut by assigning a semantic matching algorithm *sma* (optional) in pointcut expressions. The semantic matching algorithms would be the same as what we defined for the XML policy model. The policy developers can define a pointcut without a query language. In the following example, the pointcut expression refers to all of the process with name ‘order inspection’. Wildcards ‘*’ can also be used to match all signatures in pointcut designators.

process('order inspection')

Additionally, we add a *query()* designator to offer a query based pointcut language. A pointcut expression uses XPath2.0 to query the join point information. The data model of the query source is the same as for the *WeavingRequest*. However, the context information is restricted in *name* and *type* elements, while the *ServiceReferecne* is not available in the query data source. Firstly, as policies, our aspect objects are high level business elements rather than implementation details. Secondly, this applies to the coordination cache specification for aspect weaving. Otherwise, we do not know if the result of aspect behaviour is *PA_{undefined}* or something else when the pointcut does not match. The following is a query based example: the pointcut expression refers to all of the activity where the name contains either ‘pay’ or ‘account’.

$$\begin{aligned} & query('contains(//WeavingRequest/Activity/Name, 'pay') or \\ & contains(//WeavingRequest/Activity/Name, 'account'))' \end{aligned}$$

We also include the operators *and* (&&) and *or* (||) in the framework for the logical connective of PCDs. Policy developers can connect query and non-query based on pointcut expressions in a single pointcut expression. In the following example, a pointcut expression is a combination of two types of pointcut expressions.

$$process(' ') \&\& query(' ')$$

7.3.3 Advice Specification

The advice defines the crosscutting relationships within the aspect behaviour and locates the place in which to inject this behaviour (join point).

7.3.3.1 Advice type

Advice types are in exactly the same *activity state* of the XML policy model, i.e. the *S_g* defined in the coordination protocol, e.g., *s_{validating_{pre}}*, etc. This differs from the advice types in common AOP frameworks, which include *before*, *after*, and *around* advice types.

First, it is consistent with the XML policy model. Second, the traditional advice types do not satisfy the distributed AOP environment, as the location difference for the execution of aspects and the target program are not considered. For example, in order to replace a BPEL invoke activity join point with AO4BPEL, it must use the around advice. Instead of proceeding with the original invoke, a new invoke activity with the replaced service reference could be defined in the around advice. With this approach, all replaced business activities must be executed in a process at the consumer side, as all aspects are deployed on the consumer side with the XML policies. In our case, the business activities that are replaced would be executed at the process provider side. Still, our AOP framework does support replaced activities that are executed on the consumer side (PA_{skip} and then executing the replacement with $s_{man_{pre}val_{post}}$ advice).

7.3.3.2 Advice language

The behaviour of an advice needs to be described using an advice language. As a programming language is targeted by other AOP frameworks, such as AspectJ, the target programming language naturally becomes the advice language. For example, the advice language of AO4BPEL is BPEL and the advice language of AspectJ is Java.

Since our distributed AOP approach does not target any specific programming language and the execution of aspect behaviour execution is separated from the target program execution environment, the supported advice language depends on the AOP framework implemented by individual process consumers. In other words, the process consumers decide their own programming language for implementing aspect behaviours for business processes. The advice language could be Java, C#, etc. Both process consumers and providers are not restricted to any particular programming language, as long as they comply with the coordination protocol and the policy model. The AOP framework we developed for our case study currently uses the Java language.

7.3.3.3 Advice template

Just having a programming language is not enough for advice development as there are still some problems that need be solved in order to build a policy AOP framework.

1. First, developing an advice for aspect behaviours often requires context information regarding the current join point of the target program [141]. For example, log all customers who submitted orders in excess of 500 euros to the payment process activity. The pointcut defined for this advice is the payment process activity. However, without knowing the customer information that is processed by the activity, the advice cannot be completed.
2. Second, since the join point model is designed for high level business elements only, the pointcut may not always be able to clearly specify the required place for injecting the aspect behaviours. For example, an aspect relies on a service reference of a business activity; an aspect exists only between peak hours (e.g. 8am-5pm). This requires further filtering of the current join point. We need the same constructor similar to the *Conditions* element of the XML policy model.
3. Third, with the traditional AOP approach, such as AspectJ and AO4BPEL, the return of an advice is either Void, or a Resource/BO, which is the result of the *around* type of advices. Since aspects are treated as policies in our AOP concept, the behaviours of advices are expected to have an identical consequence to the XML policies. An advice would give a policy decision or a provider action after completion, so that aspect specification will also comply with the coordination protocol for the multi-tenancy requirement.

To address the above problems, we define the following advice template for our AOP framework (Listing 7.1). An advice contains the pointcut and the advice type metadata regarding advice behaviours, which is defined as a method. *Weaving_{request}* contains context information of the current join point and is defined as the input variable of the advice method. The advice could contain a conditions method with an if-else control structure to

further filter the current joint point. The conditions method is separated from the advice method, and can thus be reused for other advice methods. The advice behaviour is located after the conditions method and the advice method could return a *Void* or an *Object* as its behaviour result. The *Void* return means that it had no effect on the target program, i.e. only consumer actions are defined in a rule ($PA_{unexpected}$). There are three cases with the return *Object* o , and this will be checked and handled in aspect weaving.

$$o \left\{ \begin{array}{l} = \{\} \\ \in PA \setminus \{PA_{manuplate}\} \\ \in R \end{array} \right.$$

Case 1, the object has a null value, which is the same as a *void* return. Case 2, the object represents a provider action, meaning a decision is given from a policy to the target program. Case 3, the object is a *Resource*, meaning the aspect effects on the target program and needs to be taken by process providers. All un-handled exceptions will be thrown to the parent process of aspect weaving, and will be described in the subsection outlining the fault handling.

7.3.3.4 Aspect and lifecycle

In our framework, each aspect is either a Java class or bean. The advices of an aspect as Java methods are defined in the aspect Java class (Listing 7.2). The pointcut and advice type metadata of an advice are defined as Java annotation and retrieved by Java reflection. An aspect instance is unique to each advice method call in aspect weaving, i.e., a new aspect instance is created for each advice call. A singleton class can be defined outside the aspects when it is required and can be called by the aspects, thus providing a singleton instance that is shared across all advice calls.

Listing 7.1: Advice template

```

1  void adviceMethod1(WeavingRequest context) throws Exception {
2      if (conditionsMethod(context)) // optional
3      {
4          // behaviours
5      }
6  }
7
8  Object adviceMethod2(WeavingRequest context) throws Exception {
9      if (conditionsMethod(context)) // optional
10     {
11         // behaviours
12         // return behaviours result/null
13     }
14     else {
15         return new PaUndeterminedType();
16     }
17 }
18
19
20 Boolean conditionsMethod(WeavingRequest context) throws Exception {
21 }

```

7.3.3.5 Fault handling

Contrary to the XML policy model, the *FaultHandler* element is designed in the policy model for fault handling in policy weaving. The fault/exception of aspect weaving may also occur, but faults that occur in aspects are expected to be handled by the policy developers using the advice programs. The policy developers can use fault handling constructors that are offered by the advice programming language, such as the *try{} catch{} block* of Java. For example, a return statement with a *pa_{cancel}* could be found in the catch block for any exceptions that occur in the advice execution for a fault rule. In such a case, the fault rule would cancel the process execution in situations of exception during aspect weaving.

Since both aspects and the target program are executed separately on different participants in our case, the AOP framework considers two types of exceptions that are un-handled or re-thrown from advice and aspect weaving (Listing 7.3).

1. One type of exception that appears on the target program (*ViolationException*), or the violation of business processes. The exception means that the business process execution has moved outside the safe boundary defined by the constraint rules. The

Listing 7.2: Policy aspect

```
// policy1
@Aspect
public class MyPolicyAspect {

    // rule1
    @Pointcut(...)
    @AdviceType(...)
    void adviceMethodOfRule1() {
        // ...
    }

    // rule2
    //...

    // rule3
    //...
}
```

exception object from the aspect will be converted into a Violation element as a result of the aspect.

2. The others are the second type of exceptions (*Exception*). These exceptions do not directly appear in the target program. Exceptions are indicators of bugs in the aspect program development from policy developers. They are instances of the *Exception* class or subclasses of *Exception* in the programming language (Java in our case), for example, *NullPointerException*, etc. They result in a *pa_{undetermined}* provider action, which is expected to be handled in the policy combining algorithms.

Please note, *ViolationException* is itself a subclass of *Exception*. It extends *Exception* on implementation. *ViolationException* associates with the *ViolationTypeType* element that is defined in the XML policy mode.

7.4 Aspect deployment and weaving

In our overall framework, AOP is an extension of our policy model rather than an entirely separate module and the XML policy still plays a role for aspects deployment (Figure 7.2). Aspects are treated as policy elements in a set of policies that are defined in a *PolicySet*

Listing 7.3: Fault handling for policy aspects

```
try {
    // handling an aspect
    //

} catch (ViolationException ve) {
    PaViolateType pa = new PaViolateType();
    pa.getViolation().add(ViolationTypeType.fromValue(ve.value));
    return pa;

} catch (Exception e) {
    return new PaUndeterminedType();
}
```

element. Deployed aspects will be weaved through the upgraded policy weaver. In the following subsection, we detail aspect deployment and weaving.

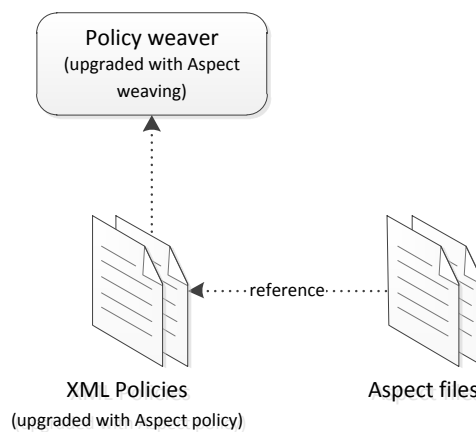


Figure 7.2: Aspect deployment and weaving

7.4.1 Aspect deployment

Deploying an aspect is similar to adding a new XML policy element. The *Aspect* element is added to the children elements of the *PolicySet* in addition to the XML *Policy* element to provide a choice between XML policy and Aspect in the upgraded policy schema. Figure 7.3 shows the upgraded *PolicySet* element in the policy schema with *Aspect* elements.

Please note, unrelated elements and attributes of *PolicySet* for this section are hidden in the Figure 7.3 and for full details of the *PolicySet* refer to the policy model chapter.

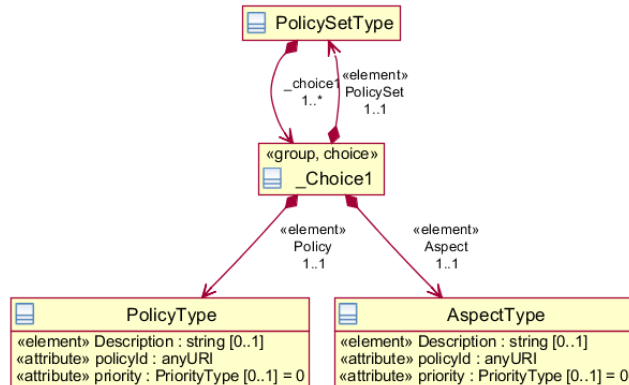


Figure 7.3: Aspect : AspectType

Aspect refers to the real aspect file and it has the following attributes and elements.

1. Description - a description of this aspect from the policy developer.
2. policyId - a unique identification of the Policy or Aspect. It refers to the real aspect class file developed by the policy developers. In our case, Java is used as the advice language. A completed aspect file results in a Java class file.
3. priority - a positive integer that denotes the priority weight of this aspect in the policy set. Default and minimal value is 0.

It is evident that aspect deployment relies on XML policies. From the XML policy point of view, an aspect represents a single policy description which assumes a different form in the policy set. From a pure AOP development perspective, the XML policy signifies both the aspects deployment and the management configuration file. The Combining and Sequencing algorithms defined in the PolicySet will still be applied to any deployed policy aspects. The XML schema based policy approach and the AOP approach are integrated seamlessly into the policy framework, and it is free for policy developers' choices. It provides a flexible way for policy developers to choose an appropriate approach. The following example (Listing 7.4) shows a deployed Aspect, which is integrated with XML

policies.

Listing 7.4: Fault handling for policy aspects

```
<p1:PolicySet ...>
  <p1:Objects/>
  <p1:ActivityStates/>

  <p1:Policy policyId="..." priority="0">...</p1:Policy>

  <p1:Aspect policyId="requestor1.policy.aspect.OrderInspectionPolicy4"
    priority="0">
    <p1:Description>performance constraint policy</p1:Description>
  </p1:Aspect>

  <p1:Policy policyId="..." priority="0">...</p1:Policy>

  ...
</p1:PolicySet>
```

The XML policy as an underlying master policy manages deployed aspects in the following facets.

- Advice precedence

Multiple advices can be defined on the same join point. *Advice precedence* determines the aspect weaving sequence [194]. Since different weaving orders can result in programs that behave in various manners, the weaver must determine the exact weaving order and the dependencies among the aspects. The XML policy declares the order in which aspects and XML policy elements are woven by policy *sequence algorithms*, e.g., *Ordered* or *PriorityBased-QuickSort*. (Please refer to the algorithms of the policy model chapter). Within an aspect, advices are woven in the textual order that they appear in the aspect class file for the same advice type. Advices with different advice types are referred to the coordination protocol.

- Advice combining

Advices can result in multiple returns when several advices are weaved at the same join points. In current AOP approaches, this potential problem is countered by using an atomic group - in ‘all or nothing’ manner [58], or with additional specifications, such as

constraints [195]. Since aspects are policies, multiple returns can result in varying policy decisions. The XML policy asserts that the combination of policy decision with aspects and XML policy elements are woven together by the policy combining algorithms, e.g., *Pa-Violate-Override-Through-All*, etc. (Please refer to the algorithms of the policy model chapter, Chapter 5). The combining process depends on the advice types with different policy combining algorithms.

7.4.2 Aspect weaving

Aspects are required to be integrated into the business process in order to address the separate concerns implemented by aspects for the business process, i.e. the aspect *weaving* mechanism in the AOP concept.

Aspect weaving can be classified into static weaving and dynamic weaving. The static weaving is done before target program deployment and the aspect code is compiled or built into the target program [137]. It is similar to binding policies with a BPEL file where the BPEL process is specialized for policies from a single consumer. This BPEL and policy coupled approach does not meet our multi-tenancy requirement. The dynamic weaving occurs at runtime [138] [139] and the change and deployed aspects do not affect the deployed target program. This is an especially important factor in the multi-tenancy environment where modifying or redeploying the business process is not allowed, as it could affect other current process consumers. Additionally, it is infeasible to stop an ongoing LRT process instance for editing aspects, as all previous completed tasks would require compensation after stopping the process instance [141] [138].

The essential of dynamic weaving is a program interceptor. It is able to find the points of program execution where an aspect is involved. It is typical for approaches to work on program execution platforms or engines; for example, an aspect-aware workflow engine for AO4BPEL [141], a JVM (Java VM) plug-in for a Java AOP [139]. Despite the fact that these approaches are platform-dependent, there are critical limitations that make it impossible to

adapt them for our approach. First, aspects need be delivered to the process providers who host the execution platforms. Hence, policies are exposed to the process providers which consumers might object to for security reasons. Second, the platforms are not aware of the multi-tenancy requirement. They fail to distinguish between the process instances and the aspects owned by different process consumers. In our case, the aspects of a process consumer should be exclusively weaved for the consumer only, but should not be involved with any other process consumers.

Our approach adopts the dynamic weaving for aspect weaving. We use a similar approach as with our XML policy weaving to address the limitations of the current weaving approaches we have discussed. Aspect weaving relies on the coordination protocol, as our AOP model is likewise designed to comply with. The aspect weaving feature is upgraded into the policy weaver component, and the weaver still remains in process governance components due to the multi-tenancy requirement.

The upgraded weaver component takes care of both the XML policies and aspects. During policy weaving with a policy set, if a policy element is an XML policy element, the weaving is as we described in the policy chapter. Moreover, a provider action and obligations are expected. If it is an aspect element, aspect information (such as pointcut) is retrieved from the aspect class file, and one or more advice methods might be executed. Once more, a provider action is expected after the aspect weaving. The returned provider actions from XML policies and aspects are not different and will be combined in the policy set. After all the policies and aspects are weaved, a final provider action will be returned to the process providers in the same manner as the XML policies we demonstrated in the earlier chapter.

7.5 Case study

In this section, we outline some case studies on concrete policy aspects as defined with an AOP extension for the business policies of Consumer_1 (Section 3.2). It provides evidence of the extensibility of our enhanced framework. Moreover, it allows for the adoption of other policy languages and frameworks to counter the limitations of our policy framework, and also the adoption is managed by our XML policy as the master policy.

7.5.1 Objective

The objective is to demonstrate how the AOP enhanced policy framework offers a great extensibility on the XML based policy model and show that additional policy models could be seamlessly added on top of our predefined XML policies. Also, adopted policy models or frameworks can be used to express the business policies which our XML policy model has difficulty handling.

The approach includes two case studies with two business policy examples as discussed in Section 7.2 to meet this objective. In the first case, we have a simple high level policy for slide time windows by utilizing event processing technology. In the second case, the Jess rule [95] is adopted to express and reason about business policies with complex logic. We do not argue that the adopted frameworks are the best options as they are only used to prove our concept in this case study. Through both case studies, we display evidence of the extensibility of our AOP enhanced policy framework.

7.5.2 Approach

7.5.2.1 Case 1: extension with high level policy for time windows

Business policy:

Average time cost for purchase order inspection activity executed in the last hour for each order should be less than 5 seconds

For the above business policy, we need a constraint rule on activity performance. As a pre-requirement, we need real-time monitoring of the activity execution events. As we discussed, it is difficult to express the time windows in our policy model while also handling large amounts of real-time information with our current framework. In this case, we develop a parametrized high level policy for performance constraints on the activity by utilizing event queries. It utilizes an event stream engine called Esper [196] for a policy framework, which is extended for our policy framework. From this case study, we can demonstrate that our policy framework is extended with the CEP engine with a simple high level policy for process monitoring.

Listing 7.5 shows the defined policy aspect. The code for support functions and classes are not shown here.

Listing 7.5: OrderInspectionPolicy4

```

1 package requestor1.policy.aspect;
2
3 import aspect.AdviceType;
4 ...
5
6 // >> Policy
7 @Aspect
8 public class OrderInspectionPolicy4 {
9
10     EPRuntime runtime = Esper.getProvider().getEPRuntime();
11
12     // >> rule 1
13     @Pointcut("activity('purchase_order_inspection')")
14     @AdviceType(ActivityStateType.MANIPULATING_PRE_VALIDATING_PRE)
15     private void monitoringRule1(WeavingRequestType context) throws Exception
16     {
17         ActivityEvent ae = new ActivityEvent();
18         ae.setTimeMs(System.currentTimeMillis());
19         ae.setActivity(context.getActivity());
20         ae.setActivityState(context.getActivityState());
21
22         runtime.sendEvent(ae);
23     }
24
25     // >> rule 2
26     @Pointcut("activity('purchase_order_inspection')")
27     @AdviceType(ActivityStateType.MANIPULATING_POST_VALIDATING_POST)
28     private void monitoringRule2(WeavingRequestType context) throws Exception
29     {
30         ActivityEvent ae = new ActivityEvent();
31         ae.setTimeMs(System.currentTimeMillis());
32         ae.setActivity(context.getActivity());

```



```

33         ae.setActivityState(context.getActivityState());
34
35         runtime.sendEvent(ae);
36     }
37
38     // >> rule 3
39     @Pointcut("activity('purchase_order_inspection')")
40     @AdviceType(ActivityStateType.MANIPULATING_PRE_VALIDATING_PRE)
41     private Object performanceConstraintRule(WeavingRequestType context)
42         throws Exception {
43         if (ActivityStatus.getInstance().getQoSPerformanceStatus().get("
44             purchase_order_inspection")) {
45             throw new ViolationException(ViolationTypeType.QO_S_PERFORMANCE);
46         }
47         return new PaValidateType();
48     }

```

The *OrderInspectionPolicy4* has three advices and all of them will be triggered by the purchase order inspection activity as defined in the pointcut. The first two advices signify utility rules, which collect the system time for performance calculation. The *monitoringRule1* (lines 12-23) sends an activity start event at $s_{man_{pre}val_{pre}}$ state, while the *monitoringRule2* (line 15-36) sends an activity end event at $s_{man_{post}val_{post}}$ state. Afterwards, the sent events will be correlated for each purchase order and the activity in order to create a new activity performance result event by a defined Esper EPL (Event Processing Language) [197] query. Another EPL defined in an Esper *UpdateListener* will query the average performance of the activity in the last 1 hour time window and update the activity performance violation status periodically. The sizes of the time window and performance requirement parameter are adjustable in a separated file as a simple high level policy, which defines the input parameters of predefined EPL query. The third *validating-pre* Advice - *performanceConstraintRule* (lines 38-48), will check the performance violation status of the activity. If the violation status is true, the *ViolationException* will be thrown. A $pa_{violate}$ with QoS performance violation will be returned as the policy decision or provider action. The provider action will then be combined in the *PolicySet* where the aspect is deployed.

7.5.2.2 Case 2: extend with Jess rule for rule reasoning

Business policy:

Item partNumber'32541' is a hazard item. Item partNumber'1234' is a hazard item. Sellers with an Irish address are in a controlled area for selling. Buyers with a UK address are in a controlled area for buying. Any hazard item in any controlled area is a controlled transaction. The transaction will be approved if it is not a controlled transaction.

The above business policy defines controlled transactions. This could be achieved by a policy with a constraint rule before the order inspection activity. The business policy contains a set of rules for controlled transactions, and one rule also depends on other rules. Thus, it is complex and difficult to formalise them using our XML policy model. We expect that the business policy can be expressed in a rule language with reasoning capability and this also makes it easier to maintain and update the policy for the policy developer, for example, adding a new controlled area, etc. In this case, we adopt the Jess rule [95] on top of our constraint policy aspect. The rules of the business policy are described as Jess rules in a separated rule file and executed by a Jess rule engine. From this case study, we can demonstrate that our policy framework can be extended with a policy language such as Jess rule for complex policies and reasoning. The following describes the policy aspect.

Listing 7.6: ControlledTransactionPolicy

```
1 package requestor1.policy.aspect;
2
3 import jess.Rete;
4 ...
5
6 @Aspect
7 public class ControlledTransactionPolicy {
8
9     // >> rule 1
10    @Pointcut("activity('order_inspection')")
11    @AdviceType(ActivityStateType.MANIPULATING_PRE_VALIDATING_PRE)
12    public Object constraintRule(WeavingRequestType context) throws Exception
13    {
14        // check conditions for approving transaction
15        if (!controlledTransactionConditions(context)) {
16            return new PaUndeterminedType();
17        } else {
18            return new PaValidateType();
19        }
20    }
21 }
```

```

19     }
20 }
21
22 // conditions
23 private boolean controlledTransactionConditions(WeavingRequestType context
24     ) throws Exception {
25
26     PurchaseOrder po = getPurchaseOrder(context);
27
28     Rete engine = new Rete();
29     // setup Jess rule engine with controlledTransaction jess rules
30     engine.batch(getRuleFile("controlledTransaction.clp"));
31     // add new facts
32     engine.add(po.getBuyer());
33     engine.add(po.getSeller());
34     engine.addAll(po.getItem());
35     engine.run();
36     // get result
37     Iterator ct = engine.getObjects(new Filter.ByClass(
38         ControlledTransaction.class));
39     if (ct.hasNext()) {
40         return true;
41     } else {
42         return false;
43     }
44 }

```

The *controlledTransactionPolicy* has one advice (lines 9-20), which applies the suitable offers on the *purchaseOrder* resource before the order inspection activity execution. It will check the buyer, seller and each item in the resource to decide if it is a controlled transaction as *conditions* of advices (*controlledTransactionConditions*, lines 22-42) to approve the transaction. The decision regarding the controlled transaction is determined by the Jess rule engine with the controlled transactions rule file (*controlledTransaction.clp*). The policy will validate the order if it is not a controlled transaction. The decision will be combined with other decisions from other policies defined on the process. Listing 7.7 shows a fragment of the Jess rule file.

Listing 7.7: *controlledTransaction.clp*

```

1 (import bo.*)
2 (deftemplate Item      (declare (from-class Item)))
3 ...
4
5 (defrule controlled-area-Ireland
6     (Seller {address.country == "Ireland"} (address.country ?area))
7     =>
8     (add (new ControlledArea ?area)

```

9)
10	...

7.5.3 Result and Discussion

The above policy aspects are deployed in a *PolicySet*, and are integrated with previous policies we defined in the policy chapter, so that the above policy aspects are integrated into and managed by our XML policy model. Any business policies already expressed in our XML policy model will still apply to business processes. The same test case based approach is used, and the result shows that all policies are enforced. All policy decisions or provider actions are combined as managed by our XML policies. From the case study, we can see that the AOP enhanced policy framework offers great extensibility, as other frameworks and policy models or rules can be adopted easily by means of aspects. Moreover, other policy models can be integrated on top of our policy model as united policies by defining policy aspects. Both XML policies and policy aspects are in a unified policy model.

AOP is a programming paradigm, and an advice language is a type of programming language. Developing policies with aspects in a programming language which is for system development, is certainly more complex than expressing business policies in our XML policy language which is for system configuration. Still, the AOP is only an extension of our enhanced policy framework and functions as an alternative to an XML based policy model for policy developers.

There is a limitation we have identified during the case study. Since policy aspects are Java classes, deploying aspects also requires to have the compiled *.class* files and the referred library files in the application deployed in the application container. We only need to update the XML policy file to apply a policy on business processes at runtime, where the class files are already deployed in the application container. However, with new policy aspects, where class files were not in the application container, we need to redeploy the application, which means that the process runtime governance is interrupted during the

redeployment stage. This problem also applies when the policy developers need to change the code of aspects.

7.5.4 Comparison with related work

In this section, we discuss work related to AOP in service computing and compare this with our AOP enhanced policy framework.

Firstly, we discuss the some of the joint work on policies and an AOP approach, as our work involves both domains. Afterwards, we discuss general AOP frameworks developed for Web services and business processes.

There is an amount of work [143] [144] [145] that discusses the combination of policy and AOP. However, we have a very different concept and aim. In the related work, aspects are used for the policy enforcement implementation of defined policy models, such as for WS-Policy [144] and business rule [147]. So the policy enforcement could be decoupled from the target program, i.e., Web service logic implementation. In our concept, aspects function as as an extension of XML policies, an alternative to XML *Policy* elements and coexisting in a policy assembly. It aims to provide a programming approach to define complex policy requirements, or integrate other policies or frameworks on top of our defined policy model. So our policy model can take advantage of other policies or systems, while also acting as a master policy model for top layer policies.

Both [146] and [147] have discussed the problems with the business rule approach by pointing to a lack of modularity. However, the problem we addressed is not only about adopting a business rule system or any another rule system as a separate module. Instead, we discuss adopting multiple rule or policy models that might be needed. The critical problem is how to provide an integrated approach that solves the conflicts between different policies defined in different rule or policy models for business processes.

In the following, we detail and compare a common Java AOP framework (SpringAOP [198]), two BPEL specialized AOP frameworks(AO4BPEL [141], A4B [58]) and our policy

AOP. The comparison is based on three views: the general concept, AOP specifications, and special features, to show the differences and advantages of our work.

General concept

	Spring AOP	AO4BPEL	A4B	Policy AOP
Target program	Java (OO language)	BPEL (Workflow language)	BPEL (Workflow language)	Business process

Spring AOP and AO4BPEL are generic AOP frameworks. A4B is influenced by the publish/subscribe system. An AOP broker is added between a BPEL engine and an ESB. It uses WS-Notification for publishing events to the broker, which is able to weave the aspect onto the BPEL processes. However, our work is built upon a policy based system.

Target program - SpringAOP is a commonly used AOP framework for the Java language. We consider some other AOP frameworks we have studied: AspectJ and JbossAOP [199] are in the same family as SpringAOP, as they target an OO programming language. In this study, we use SpringAOP to represent all of them. AO4BPEL and A4B are two frameworks that we have discovered for BPEL processes. Both works relate closely to our work as BPEL is the de-facto standard language for describing and executing business processes. In comparison, our work targets generic business processes without reference to any concrete workflow languages.

AOP specification

Join point model - SpringAOP and AO4BPEL derive from the target programming language perspective. A4B comes from the publish/subscribe system perspective, the join point model is expressed in terms of the events the BPEL engine needs to generate and notify, e.g., *ActivityReady*, *Link_Evaluated*, etc. Our work stems from a policy system perspective.

Pointcut - Joint point based PCDs or query based pointcut language have been used in these frameworks. The syntactic and semantic differences with pointcut languages are not

	Spring AOP	AO4BPEL	A4B	Policy AOP
Join point model	Method execution	Activity (Service invocation), internal (SOAP message in/out)	BPEL engine Event	Policy object
pointcut	Joint point based PCDs	Query based	A Joint point based PCD	Joint point based PCDs & query PCD
Advice type	Before Around After returning After throwing After	Before Around After	Before Instead After	Activity state types
Advice language	Java	BPEL	A4B XML schema	Java
Advice return	Exception Object	Message Object	Service reference	Policy actions
Fault handling	Via advice	Via advice	-	Via advice and policy combining algorithm
Aspect deployment	Spring Configuration XML	Deploy in BPEL engine	WS-Policy attachment XML	Policy XML
Aspect weaving	Static Dynamic	Dynamic	Dynamic	Dynamic

considered in this part of our work in the AOP research domain.

Advice type - Our work has the largest number of advice types, which are derived from the coordination protocol. Our work does not offer the *around* or *instead* advice, which is common in other frameworks. However, it could be achieved via $PA_{replace}$ and $PA_{manipulating}$ provider actions using our advice types.

Advice language - SpringAOP and AO4BPEL have target programming languages as the advice language. Advices in A4B only are service invocation notification events for the ESB. It simply uses its own schema (imports WS-Addressing) to describe the service reference. Java is used in our case.

Advice return - SpringAOP, AO4BPEL could return an object(BO) to a target program via the *around* or *instead* advice. It could throw an exception to prevent execution proceeding in SpringAOP. A4B returns an event which includes a service reference. In our case,

a provider action or policy decision is the returned object. A4B and our work both return business objects to the target program through event or provider actions execution. In our work, all thrown exceptions will be converted into a policy decision.

Fault handling - SpringAOP, AO4BPEL and our work all can handle exceptions via advices by utilizing the fault handler feature of the programming language. Moreover, our work also could handle exceptions that are not defined in fault handlers of advices by utilizing the combining algorithms. Fault handling is not discussed in A4B.

Aspect deployment - AO4BPEL deploys aspects as BPEL in an enhanced BPEL engine. The rest of the approaches use XML files to deploy aspects. In our case, the XML file itself is a policy file.

Aspect weaving - Spring AOP supports both static (compile time via the AspectJ compiler) and dynamic weaving. The other approaches only use dynamic weaving.

Special features

	Spring AOP	AO4BPEL	A4B	Policy AOP
Distributed aspect	Not available	Not available	Yes	Yes
Multi-tenant aspects	Not available	Not available	Not available	Yes
Multiple policies integrations	Not available	Not available	Not available	Yes

Distributed aspect - Spring AOP and AO4BPEL weave aspects into a target program is under the assumption that they are deployed on the same platform or application container. A4B uses the event notification mechanism to enable the deployment and execution of aspect in a remote location, rather than the target program execution platform. Our work advocates distributed aspects by utilizing the coordination protocol.

Multi-tenant aspect - Multi-tenancy in SpringAOP, AO4BPEL and A4B is not considered. Our work supports multi-tenant aspects by employing it on top of the coordination framework. Importantly, because of aspects could be deployed on the process consumers'

side, it is possible to hide policies from any other parties.

Multiple Policies integrations - SpringAOP and AO4BPEL are generic AOP and do not focus on policies. In A4B, aspects as assertions are specified in the WS-Policy attachment based on the grammar defined in WS-Policy. However, it does not consider or address the possible conflicts that might occur in different policies. And these conflicts could also occur between policy assertions defined in the WS-Policy itself. Our unified policy model clearly addresses this problem.

With the above comparisons, we can see that other AOP frameworks do not target or deal with the specific requirements needed for our problem. For the extensibility of our policy framework with the multi-tenancy requirement, we have a uniquely designed AOP framework.

7.6 Conclusion

In this chapter, we presented an AOP enhanced policy framework to offer a great extensibility to the original policy framework. Aspects are realized as policies of the XML policy model. As a result, other policy models and frameworks could be adopted, and under the management of our XML policy model, which acts as the master policy. The AOP framework provides policy aspect specification which specifies how to implement policy aspects for business processes, and aspect deployment and weaving specification specifies how aspects can be developed and weaved with XML policies. Our AOP framework design addresses the special features, such as multi-tenancy, which are not considered with other AOP frameworks. We effectively used two case studies to demonstrate our objective with the extensibility on the policy framework.

Chapter 8

Conclusion

Automated business processes are important for organisations' operations. The SOA style, RAs and frameworks do not address the problem of business process or service process sharing between cross organisations consumers, which is significantly highlighted with the emergence and growing of cloud computing and BPO.

Our work is designed to share business processes as Web services. It is a distinct problem to be positioned in a different cloud layer compared with the closed work of business processes or BPEL in cloud computing. They provide BPEL processes as end user applications or shared BPEL engines, such as the Cafe project, but do not offer BPEL processes as software components, just as Web services. For business processes as software components in the form of Web services that are available to be shared on the Internet, the overall architectural design and development needs a solution, which we have provided. Our solution consists of an architectural style and a supported architecture framework, which are the two main parts of this thesis to address the problem. Furthermore, we divided the main parts into sub-problems, and addressed them in different solution chapters. We used a case study section in each of the solution chapters to demonstrate and evaluate our solution gradually.

In this chapter, we will provide a work summary (Section 8.1) and also discuss the potential for future research (Section 8.2).

8.1 Work summary

The following conclusions can be drawn from the experiences of this PhD work.

- With the concept of SaaS in cloud computing, the software is not only restricted to end user applications or simple application APIs, such as task Web services, but is also possible in other contexts, such as business process or process logic sharing with the concept of (Business) Process as a Service.
- Business processes are valued assets of enterprises. A shareable process would increase the reuse potential for various consumers and make profits from external consumers with developed business processes. This makes maximum long-term financial returns for process providers. Process consumers can quickly respond to different circumstances and continues process improvement, with better control of time, cost constraints, and investment protection on process development.
- The importance of adaptation and customization for external monitoring and control is critical for the Process as a Service concept. It is overlooked in the current SOA style and RAs. A separate architectural SPA style focuses on the issues, defining the process governability principle to extend the SOA style. This addresses the problem of process sharing, and can affect the design and development of orchestrated task services or processes in association with the original principles of the SOA style.
- Consumers' requirements as business policies regarding business processes could be expressed in a formal policy language, which acts as a customization metadata of business processes. We have provided an XML schema based policy specification to formalize four categories of rules of policies, which we identified from different aspects related to business process execution (flexibility, constraint, fault and utility rules).
- Process on-the-fly customization and adaptation can be achieved from process consumers by means of process runtime governance based on process element - business activity. A coordination framework and protocol could be used for activities within

processes or subprocesses from different providers to work together on process execution for business transitions requested by consumers for the multi-tenancy capability. We have provided the coordination framework and protocol correlated with our policy model, as well as the process design template for implementations.

- In some cases, it may be difficult to address the business policies or requirements of process consumers in a single policy model or framework, and is also possible for conflicts regarding policies defined in many policy models to arise. Aspects can be modelled as policies on top of our policy model, offering extensibility of our policy model while also complying with coordination protocols. We have provided an aspect specification for the extensibility of our policy as the master policy in the overall policy framework.

8.2 Future work

This section describes the ideas for future work that would extend the current work. Each idea proposes the manner in which the idea of each chapter in this research could be extended in the future.

- Measuring of runtime governability

The process runtime governability is the key for retaining customizability and adaptability offered to process consumers. A standard approach and specification on runtime governability measurement is important for evaluating and comparing between different process designs and service process architectures. Related work could be adapted from different fields, such as requirement analysis [165], and variability management of SPL [114]. Inevitably, the compensative metrics and approaches for measuring process runtime governability require more research.

- Highly level policy modelling and related algorithms

High level goal policies are intuitive, and can be easily used to express business policies or goals for business developers. Some policy frameworks offer goal policy modelling [200] [201] for directly expressing goals or assuming users without strong technical background. The models and algorithms inside the policy frameworks, such as cost model, scheduling algorithms, selection algorithms, guarantee the goals defined in the high level policy are met during the process executions.

- Enhancements of coordination protocol and framework

The current coordination protocol only involves the process governance at the process logic level, but could be extended to include other levels, such as governance at process engine level for consumers. A specific process execution framework is required to be developed to implement the protocols and with multi-tenancy capability [77]. The coordination frameworks might be redesigned to support generic BPEL processes without requiring specific process design using BPEL templates. It is especially important to offer process consumers more runtime governability, but less complexity on process development for process providers.

- Scalability of coordination framework

The governance component with a *coord*^c is at the centre of all coordination conversations for a process consumer. With growing of business processes and organisations' business transactions, an extensive amount of data needs to be transmitted and handled efficiently between coordinators. Thus, it is important to have a scalable data processing ability to accommodate the growth of businesses and their requirements. Approaches, such as prediction or an improved cache mechanism, could be utilized and developed to improve the scalability of the coordination framework.

- Enhancements of aspect specification

The aspect specification can be enhanced by supporting other advice types (around, etc.), or having a workflow like advice language, etc. This will enhance the power of the AOP framework, and will make it easier for developers who have experience with workflow

modelling, etc. The join point model and point cut language could also be extended with the enhanced coordination protocol.

Bibliography

- [1] Ali Arsanjani, Liang-Jie Zhang, Michael Ellis, Abdul Allam, and Kishore Channabasavaiah. S3: A service-oriented reference architecture. *IEEE IT Professional*, 9(3):10–17, 2007.
- [2] Liang-Jie Zhang and Qun Zhou. Ccoa: Cloud computing open architecture. In *IEEE International Conference on Web Services*, 2009.
- [3] Oasis web services coordination (ws-coordination), 2009. <http://docs.oasis-open.org/ws-tx/wscoor/2006/06>.
- [4] Thanh Thoa Pham Thi, Markus Helfert, Fakir Hossain, and Thang Le Dinh. Discovering business rules from business process models. In *International Conference on Computer Systems and Technologies*, pages 89–94, 2010.
- [5] Mike Havey. *Essential Business Process Modeling*. O’Reilly, 2005. pages 18.
- [6] Paul Grefen, Rik Eshuis, Nikolay Mehandjiev, Giorgos Kouvas, and Georg Weichhart. Internet-based support for process-oriented instant virtual enterprises. *IEEE Internet Computing*, 13(6):65–73, 2009.
- [7] Ralph Mietzner. *A Method and Implementation to Define and Provision Variable Composite Applications, and its Usage in Cloud Computing*. PhD thesis, 2009. Department of computer science, electrical engineering and information technology, Universitt Stuttgart.
- [8] Luis M. Vaquero, Luis Roderio-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: Towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1), 2009.
- [9] Yi Wei and M. Brian Blake. Service-oriented computing and cloud computing: Challenges and opportunities. *IEEE Internet Computing*, 14(6):72–75, 2010.
- [10] Rik Eshuis and Alex Norta. A framework for service outsourcing using process views. In *IEEE International Enterprise Distributed Object Computing Conference*, 2010.
- [11] Thomas Erl. *SOA Principles of Service Design*. Prentice Hall. 2008. pages 110, 113.

- [12] Marten van Sinderen. From service-oriented architecture to service-oriented enterprise. In *International Workshop on Enterprise Systems*, 2009.
- [13] Linang-jie Zhang, Jia Zhang, and Hong Cai. *Service computing*. Tsinghua university press and Springer, 2007. pages 29, 90, 108.
- [14] Jeff A. Estefan, Ken Laskey, Francis G. McCabe, and Danny Thornton. Oasis reference architecture foundation for service oriented architecture 1.0 draft 2, 2009. <http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-cd-02.pdf>.
- [15] W3c web services policy 1.2 - framework (ws-policy). <http://www.w3.org/Submission/WS-Policy>.
- [16] Abdelkarim Erradi. *Policy-Driven Framework for Manageable and Adaptive Service-Oriented Processes*. PhD thesis, 2008. Computer Science and Engineering, The University of New South Wales.
- [17] Luciano Baresi and Sam Guinea. Self-supervising bpm processes. *IEEE Transactions on Software Engineering*, 37(2):247 – 263, 2011.
- [18] Anis Charfi and Mira Mezini. Ao4bpm: An aspect-oriented extension to bpm. *World Wide Web Journal*, 10(3):309 – 344, 2007.
- [19] Chang Jie Guo, Wei Sun, Ying Huang, Zhi Hu Wang, and Bo Gao. A framework for native multi-tenancy application development and management. In *IEEE International Conference on E-Commerce Technology and IEEE International Conference on Enterprise Computing, E-Commerce and E-Services*, pages 551–558, 2007.
- [20] Frederick Chong and Gianpaolo Carraro. Architecture strategies for catching the long tail, 2006. <http://msdn.microsoft.com/en-us/library/aa479069.aspx>.
- [21] Oasis extensible access control markup language (xacml) 3.0, 2010. <http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.html>.
- [22] Anne H. Anderson. An introduction to the web services policy language (wspl). In *IEEE International Workshop on Policies for Distributed Systems and Networks*, 2004.
- [23] Thomas Erl, Anish Karmarkar, Priscilla Walmsley, Hugo Haas, L. Umit Yalcinalp, Kevin Liu, David Umit Orchard, Andre Tost, and James Pasley. *Web Service Contract Design and Versioning for SOA*. Prentice Hall, 2008. pages 486.
- [24] Neal Leavitt. Is cloud computing really ready for prime time? *IEEE Computer*, 42(1):15 – 20, 2009.

- [25] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS Quarterly*, 28(1):75 – 105, 2004.
- [26] Mary Shaw and Paul Clements. A field guide to boxology: Preliminary classification of architectural styles for software systems. In *International Computer Software and Applications Conference*, 1997.
- [27] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, 2000. Information and Computer Science, University of California, Irvine.
- [28] Ali Arsanjani. Service-oriented modeling and architecture, 2004. <http://www.ibm.com/developerworks/webservices/library/ws-soa-design1/>.
- [29] Justin Ryan Erenkrantz. *Computational REST: A New Model for Decentralized, Internet-Scale Applications*. PhD thesis, 2009. Information and Computer Science, University of California, Irvine.
- [30] Roy T. Fielding and Richard N. Taylor. Principled design of the modern web architecture. *ACM Transactions on Internet Technology*, 2(2):115–150, 2002.
- [31] Mary Shaw and Paul Clements. Toward boxology: Preliminary classification of architectural styles. In *International software architecture workshop and international workshop on multiple perspectives in software development*, 1996.
- [32] Jr. Rob High, Stephen Kinder, and Steve Graham. Ibm soa foundation: An architectural introduction and overview, 2005. <http://www.ibm.com/developerworks/webservices/library/ws-soa-whitepaper/>.
- [33] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall/Pearson PTR, 2005. pages 279-320.
- [34] Opengroup soa reference architecture, 2009. <https://www.opengroup.org/projects/soa-ref-arch/uploads/40/19713/soa-ra-public-050609.pdf>.
- [35] W3c web services architecture, 2004. <http://www.w3.org/TR/ws-arch/>.
- [36] Oasis open service component architecture (sca). <http://oasis-openca.org/sca>.
- [37] Mike Edwards. Relationship between sca and bpel, 2007. <http://osoa.org/display/Main/Relationship+between+SCA+and+BPEL>.
- [38] Clement Escoffier, Richard S. Hall, and Philippe Lalanda. ipojo: an extensible service-oriented component framework. In *IEEE International Conference on Services Computing*, 2007.

- [39] Clement Escoffier and Richard S. Hall. Dynamically adaptable applications with ipojo service components. In *International conference on Software composition*, 2007.
- [40] Longji Tang, Jing Dong, Yajing Zhao, and Liang-Jie Zhang. Enterprise cloud service architecture. In *IEEE International Conference on Cloud Computing*, 2010.
- [41] Longji Tang, Jing Dong, Tu Peng, and Wei-Tek Tsai. Modeling enterprise service-oriented architectural styles. *Service Oriented Computing and Applications*, 4(2):81–107, 2010.
- [42] Chris Peltz. Web services orchestration and choreography. *IEEE Computer*, 30(10):46–52, 2003.
- [43] Claus Pahl and Yaoling Zhu. A semantical framework for the orchestration and choreography of web services. In *International Workshop on Web Languages and Formal Methods*, 2005.
- [44] Boris Lublinsky. Service composition, 2007. <http://www.infoq.com/articles/lublinsky-soa-composition>.
- [45] Oasis web services composite application framework. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-caf.
- [46] Sheila McIlraith and Tran Cao Son. Adapting golog for composition of semantic web services. In *International Conference on Principles of Knowledge Representation and Reasoning*, 2002.
- [47] Dan Wu, Evren Sirin, James A. Hendler, Dana S. Nau, and Bijan Parsia. Htn planning for web service composition using shop2. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(4):377–396, 2004.
- [48] Evren Sirin, James Hendler, and Bijan Parsia. Semi-automatic composition of web services using semantic descriptions. In *Web Services: Modeling, Architecture and Infrastructure workshop in conjunction with ICEIS*, 2003.
- [49] Huiyuan Zheng, Jian Yang, and Weiliang Zhao. Qos analysis and service selection for composite services. In *International Conference on Services Computing*, 2010.
- [50] Tao Yu, Yue Zhang, and Kwei-Jay Lin. Efficient algorithms for web services selection with end-to-end qos constraints. *ACM Transactions on the Web*, 1(1), 2007.
- [51] Danilo Ardagna and Barbara Pernici. Adaptive service composition in flexible processes. *IEEE Transactions on Software Engineering*, 33(6):369 – 384, 2007.
- [52] Oasis web services business process execution language (ws-bpel) 2.0, 2007. <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>.

- [53] Rania Khalaf, A Keller, and Frank Leymann. Business processes for web services: Principles and applications. *IBM Systems Journal*, 45(2):425 – 446, 2006.
- [54] W.M.P. van der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Database*, 14(1):5–51, 2003.
- [55] Peyman Oreizy, Nenad Medvidovic, and Richard N. Taylor. Runtime software adaptation: Framework, approaches, and styles. In *International Conference on Software Engineering*, 2008.
- [56] Abdelkarim Erradi, Piyush Maheshwari, and Vladimir Tomic. Policy-driven middleware for self-adaptation of web services compositions. In *ACM/IFIP/USENIX International Middleware Conference*, 2006.
- [57] Yunzhou Wu and Prashant Doshi. Making bpel flexible adapting in the context of coordination constraints using ws-bpel. In *IEEE International Conference on Services Computing*, 2008.
- [58] Dimka Karastoyanova and Frank Leymann. Bpel’n’ aspects: Adapting service orchestration logic. In *IEEE International Conference on Web Services*, 2009.
- [59] Kareliotis Christos, Vassilakis Costas, and Georgiadis Panayiotis. Enhancing bpel scenarios with dynamic relevance-based exception handling. In *IEEE International Conference on Web Services*, 2007.
- [60] Danilo Ardagna, Marco Comuzzi, Enrico Mussi, Barbara Pernici, and Pierluigi Plebani. Paws: A framework for executing adaptive web-service processes. *IEEE Software*, 24(6):39–46, 2007.
- [61] Girish Chafle, Koustuv Dasgupta, Arun Kumar, Sumit Mittal, and Biplav Srivastava. Adaptation in web service composition and execution. In *IEEE International Conference on Web Services*, 2006.
- [62] Kareliotis Christos, Costas Vassilakis, Efstathios Rouvas, and Panayiotis Georgiadis. Qos-aware exception resolution for bpel processes: A middleware-based framework and performance evaluation. *International Journal on Web and Grid Services*, 5(3):284 – 320, 2009.
- [63] Liangzhao Zeng, Hui Lei, Jun-jang Jeng, Jen-Yao Chung, and Boualem Benatallah. Policy-driven exception-management for composite web services. In *IEEE International Conference on E-Commerce Technology*, 2005.
- [64] Kareliotis Christos, Costas Vassilakis, Efstathios Rouvas, and Panayiotis Georgiadis. Exception resolution for bpel processes: a middleware-based framework and performance evaluation. In *International Conference on Information Integration and Web-based Applications and Services*, 2008.

- [65] Peter Hrastnik and Werner Winiwarter. Twso - transactional web service orchestrations. In *International Conference on Next Generation Web Services Practices*, 2005.
- [66] Adina Mosincat and Walter Binder. Transparent runtime adaptability for bpm processes. In *International Conference on Service-Oriented Computing*, 2008.
- [67] Gerhard Friedrich, Mariagrazia Fugini, Enrico Mussi, Barbara Pernici, and Gaston Tagni. Exception handling for repair in service-based processes. *IEEE Transactions on Software Engineering*, 36(2):198 – 215, 2010.
- [68] Sattanathan Subramanian, Philippe Thiran, Nanjangud C. Narendra, Ghita Kouadri Mostefaoui, and Zakaria Maamar. On the enhancement of bpm engines for self-healing composite web services. In *International Symposium on Applications and the Internet*, pages 33–39, 2008.
- [69] Tobias Anstett, Frank Leymann, Ralph Mietzner, and Steve Strauch. Towards bpm in the cloud: Exploiting different delivery models for the execution of business processes. In *IEEE Congress on Services - I*, pages 670–677, 2009.
- [70] Tobias Anstett, Dimka Karastoyanova, Frank Leymann, Ralph Mietzner, Ganna Monakova, Daniel Schleicher, and Steve Strauch. Mc-cube: Mastering customizable compliance in the cloud. In *International Joint Conference on Service Oriented Computing*, 2009.
- [71] Christoph Fehling, Frank Leymann, and Ralph Mietzner. A framework for optimized distribution of tenants in cloud applications. In *IEEE International Conference on Cloud Computing*, 2010.
- [72] Ralph Mietzner, Frank Leymann, and Mike P. Papazoglou. Defining composite configurable saas application packages using sca, variability descriptors and multi-tenancy patterns. In *International Conference on Internet and Web Applications and Services*, 2008.
- [73] Ralph Mietzner, Frank Leymann, and Tobias Unger. Horizontal and vertical combination of multi-tenancy patterns in service-oriented applications. *Enterprise Information Systems*, 4(3), 2010.
- [74] Tobias Unger, Ralph Mietzner, and Frank Leymann. Customer-defined service level agreements for composite applications. *Enterprise Information Systems*, 3(3):369–391, 2009.
- [75] Frederick Chong, Gianpaolo Carraro, and Roger Wolter. Multi-tenant data architecture, 2006. <http://msdn.microsoft.com/en-us/library/aa479086.aspx>.
- [76] George Reese. *Cloud Application Architectures - Building Applications and Infrastructure in the Cloud*. O’ Reilly, 2009. pages 3.

- [77] Milinda Pathirage, Srinath Perera, Indika Kumara, and Sanjiva Weerawarana. A multi-tenant architecture for business process executions. In *IEEE International Conference on Web service*, 2011.
- [78] Afkham Azeez, Srinath Perera, Dimuthu Gamage, Ruwan Linton, Prabath Siriwardana, Dimuthu Leelaratne, Sanjiva Weerawarana, and Paul Fremantle. Multi-tenant soa middleware for cloud computing. In *IEEE International Conference on Cloud Computing*, 2010.
- [79] Henri Naccache, Gerald C. Gannod, and Kevin A. Gary. A self-healing web server using differentiated services. In *International Conference on Service Oriented Computing*, pages 203–214, 2006.
- [80] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, 2003.
- [81] Steffen Lamparter. *Policy-based Contracting in Semantic Web Service Markets*. PhD thesis, 2007. Karlsruhe Service Research Institute, University of Karlsruhe.
- [82] Morris Sloman. Policy driven management for distributed systems. *Journal of Network and Systems Management*, 2(4):333–360, 1994.
- [83] Wohl Associates. Soa governance - an ibm white paper, 2006. http://www-01.ibm.com/software/solutions/soa/Amy_Wohl_SOA_Governance_Analyst_White_Paper.pdf.
- [84] Opengroup: Service oriented architecture (soa). <http://www.opengroup.org/projects/soa/>.
- [85] Opengroup soa governance framework, 2009. http://www.opengroup.org/projects/soa-governance/uploads/40/19263/SOA_Governance_Architecture_v2.4.pdf.
- [86] T. G. J. Schepers, M. E. Iacob, and P. A. T. Van Eck. A lifecycle approach to soa governance. In *ACM symposium on Applied computing*, 2008.
- [87] Jeffrey O. Kephart and William E. Walsh. An artificial intelligence perspective on autonomic computing policies. In *IEEE International Workshop on Policies for Distributed Systems and Networks*, 2004.
- [88] Ravi S. Sandhu, Edward J. Coynek, Hal L. Feinsteink, and Charles E. Youmank. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.
- [89] Federica Paci, Elisa Bertino, and Jason Crampton. An access-control framework for ws-bpel. *International Journal of Web Services Research*, 5(4):20–43, 2008.
- [90] Barbara von Halle. *Business Rules Applied - Business Better Systems Using the Business Rules Approach*. John Wiley and Sons, Inc., New York, 2001. pages 33-35, 15.

- [91] David Luckham. The beginnings of it insight: Business activity monitoring, 2004. <http://www.ebizq.net/topics/cep/features/4689.html>.
- [92] Ian Graham. *Business Rules Management and Service Oriented Architecture: A Pattern Language*. Wiley, 2006. pages 22, 63-65, 50.
- [93] Marwane El Kharbili and Tobias Keil. Bringing agility to business process management: Rules deployment in an soa. In *IEEE European Conference on Web Services*, 2008.
- [94] BusinessRulesGroup. Defining business rules what are they really? http://www.businessrulesgroup.org/first_paper/br01c0.htm.
- [95] Jess, the rule engine for the java platform. <http://www.jessrules.com/>.
- [96] Omg semantics of business vocabulary and business rules (sbvr) v1.0, 2008. <http://www.omg.org/spec/SBVR/>.
- [97] Ibm ilog jrules. <http://www.ilog.com/products/jrules/>.
- [98] Stijn Goedertier and Jan Vanthienen. Compliant and flexible business processes with business rules. In *Workshop on Business Process Modeling, Development and Support*, 2006.
- [99] Florian Rosenberg, Christoph Nagl, and Schahram Dustdar. Applying distributed business rules - the vidre approach. *IEEE International Conference on Services Computing*, 2006.
- [100] Florian Rosenberg and Schahram Dustdar. Business rules integration in bpm a service-oriented approach. In *IEEE International Conference on E-Commerce Technology*, 2005.
- [101] Harish Gaur and Markus Zirn. *BPEL Cookbook Best Practices for SOA-based integration and composite applications development*. Packt Publishing, 2006. pages 67.
- [102] Michiharu Kudo and Satoshi Hada. Xml document security based on provisional authorization. In *ACM conference on Computer and communications security*, 2000.
- [103] Oasis core and hierarchical role based access control (rbac) profile of xacml v2.0. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-rbac-profile1-spec-os.pdf.
- [104] Markus Lorch, Seth Proctor, and Rebekah Lepro. First experiences using xacml for access control in distributed systems. In *ACM Workshop on XML Security*, 2003.
- [105] Tuncay Namli and Asuman Dogac. Using saml and xacml for web service security and privacy. *Securing Web Services: Practical Usage of Standards and Specifications*, pages 182–205, 2008.

- [106] Tim Moses, Anne Anderson, Frank Siebenlist, Frederick Hirsch, Ron Monzillo, and Simon Godik. Oasis web-services policy language use cases and requirements, 2003. <http://www.oasis-open.org/committees/download.php/1608/wd-xacml-wspl-use-cases-04.pdf>.
- [107] Wei Tan, Liana Fong, and Norman Bobroff. Bpel4job: A fault-handling design for job flow management. In *International Conference on Service Oriented Computing*, 2007.
- [108] Oracle soa suite 10g (10.1.3.3) - fault management framework, 2008. <http://www.oracle.com/technology/products/ias/bpel/pdf/10133technotes.pdf>.
- [109] Luciano Baresi and Sam Guinea. A dynamic and reactive approach to the supervision of bpel processes. In *India Software Engineering Conference*, 2008.
- [110] Luciano Baresi, Sam Guinea, and Pierluigi Plebani. Policies and aspects for the supervision of bpel processes. In *International conference on Advanced information systems engineering*, 2007.
- [111] Abdelkarim Erradi, Piyush Maheshwari, and Vladimir Tosic. Recovery policies for enhancing web services reliability. In *IEEE International Conference on Web Services*, 2006.
- [112] Klaus Pohl, Gnter Bckle, and Frank van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 2005. pages 10.
- [113] Charles W. Krueger. Easing the transition to software mass customization. *Lecture Notes in Computer Science*, 2290:178–184, 2002.
- [114] Charles W. Krueger. Variation management for software production lines. *Lecture Notes in Computer Science*, 2379:37–48, 2002.
- [115] Michiel Koning, Chang-ai Sun, Marco Sinnema, and Paris Avgeriou. Vxbpel: Supporting variability for web services in bpel. *Information and Software Technology*, 51(2):258269, 2009.
- [116] Chang-Ai Sun and Marco Aiello. Towards variable service compositions using vxbpel. In *international conference on Software Reuse*, 2008.
- [117] Ralph Mietzner and Frank Leymann. Generation of bpel customization processes for saas applications from variability descriptors. In *IEEE International Conference on Service Computing*, 2008.
- [118] Osoa service component architecture specifications. <http://www.osea.org/display/Main/Service+Component+Architecture+Specifications>.

- [119] Alain Andrieux, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Toshiyuki Nakata, Jim Pruyne, John Rofrano, Steve Tuecke, and Ming Xu. Open grid forum web services agreement specification (ws-agreement). <http://www.ogf.org/documents/GFD.107.pdf>.
- [120] Joe Zou, Yan Wang, and Kwei-Jay Lin. A formal service contract model for accountable saas and cloud services. In *IEEE International Conference on Services Computing*, 2010.
- [121] Doug Bunting, Martin Chapman, Oisin Hurley, Mark Little, Jeff Mischkinsky, Eric Newcomer, Jim Webber, and Keith Swenson. Web services coordination framework (ws-cf), 2003. <http://www.jboss.org/dms/jbosstm/resources/standards/WS-CF.pdf>.
- [122] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services Concepts, Architectures and Applications*. Springer, 2004. pages 216, 225.
- [123] Sanjay Dalal, Sazi Temel, Mark Little, Mark Potts, and Jim Webber. Coordinating business transactions on the web. *IEEE Internet Computing*, 7(1), 2003.
- [124] Michael von Riegen, Martin Husemann, Stefan Fink, and Norbert Ritter. Rule-based coordination of distributed web service transactions. *IEEE Transactions on Service Computing*, 3(1):60–70, 2010.
- [125] Frank Leymann and Stefan Pottinger. Rethinking the coordination models of ws-coordination and ws-cf. In *IEEE European Conference on Web Services*, 2005.
- [126] Oasis web services business activity (ws-businessactivity), 2009. <http://docs.oasis-open.org/ws-tx/wsba/2006/06>.
- [127] Deepak Alur, John Crupi, and Dan Malks. *Core J2EE Patterns: Best Practices and Design Strategies 2nd Edition*. Prentice Hall / Sun Microsystems Press, 2003. pages 181.
- [128] Oasis web services atomic transaction (ws-atomictransaction), 2009. <http://docs.oasis-open.org/ws-tx/wsac/2006/06>.
- [129] Michael P. Papazoglou. Web services and business transactions. *World Wide Web: Internet and Web Information Systems*, 6(1):49–91, 2003.
- [130] Haerder Theo and Reuter Andreas. Principles of transaction-oriented database recovery. *ACM Computing Surveys*, 15(4):287–317, 1983.
- [131] Patrick Sauter and Ingo Melzer. A comparison of ws-businessactivity and bpel4ws long-running transaction. In Paul Miller, Reinhard Gotzhein, and Jens B. Schmitt, editors, *Kommunikation in Verteilten Systemen (KiVS)*, pages 115–125. Springer, 2005.
- [132] Oasis web services transaction (ws-tx) 1.2, 2009. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-tx.

- [133] Oasis business transaction protocol (btp), 2004. <http://www.oasis-open.org/committees/download.php/9836/>.
- [134] Johann Eder and Walter Liebhart. Workflow recovery. In *International Conference on Cooperative Information Systems*, 1996.
- [135] Stefan Pottinger, Ralph Mietzner, and Frank Leymann. Coordinate bpel scopes and processes by extending the ws-business activity framework. In *International Conference on Cooperative Information Systems*, 2007.
- [136] Ramnivas Laddad. *AspectJ in Action - Enterprise AOP with Spring Applications*. Manning Publications, second edition, 2010. pages 5.
- [137] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An overview of aspectj. In *European Conference on Object-Oriented Programming*, 2001.
- [138] Carine Courbis and Anthony Finkelstein. Weaving aspects into web service orchestrations. In *IEEE International Conference on Web Services*, 2005.
- [139] Andrei Popovici, Thomas Gross, and Gustavo Alonso. Dynamic weaving for aspect-oriented programming. In *International conference on Aspect-oriented software development*, 2002.
- [140] Siobhn Clarke and Elisa Baniassad. *Aspect-Oriented Analysis and Design: The Theme Approach*. Addison-Wesley, 2005. pages 3.
- [141] Anis Charfi. *Aspect-Oriented Workflow Languages: AO4BPEL and Applications*. Phd thesis, 2007. Department of computer science, Technischen University at Darmstadt.
- [142] Mehdi Ben Hmida, Ricardo Ferraz Tomaz, and Valerie Monfort. Applying aop concepts to increase web services flexibility. *Journal of Digital Information Management*, 4(1):37–43, 2006.
- [143] Fabien Baligand and Valrie Monfort. A concrete solution for web services adaptability using policies and aspects. In *International Conference on Service Oriented Computing*, 2004.
- [144] Guadalupe Ortiz and Frank Leymann. Combining ws-policy and aspect-oriented programming. In *Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services*, 2006.
- [145] Semih Cetin, N. Ilker Altintas, and Remzi Solmaz. Business rules segregation for dynamic process management with an aspect-oriented framework. In *BPM Workshop on Dynamic Process Management*, 2006.

- [146] Maria Agustina Cibran, Maja D' Hondt, and Viviane Jonckers. Aspect-oriented programming for connecting business rules. In *International Conference on Business Information Systems*, 2003.
- [147] Anis Charfi and Mira Mezini. Hybrid web service composition: Business processes meet business rules. In *International Conference on Service Oriented Computing*, 2004.
- [148] Bart Verheecke, Mara Agustina Cibran, Wim Vanderperren, Davy Suvee, and Viviane Jonckers. Aop for dynamic configuration and management of web services. *International Journal of Web Services Research*, 1(3):25–41, 2004.
- [149] Mara Agustina Cibran, Bart Verheecke, Wim Vanderperren, Davy Suvee, and Viviane Jonckers. Aspect-oriented programming for dynamic web service selection, integration and management. *World Wide Web Journal*, 10(3):211–242, 2007.
- [150] Khoulood Boukadi, Chirine Ghedira, and Lucien Vincent. An aspect oriented approach for context-aware service domain adapted to e-business. In *International conference on Advanced Information Systems Engineering*, 2008.
- [151] Selim Aissi, Pallavi Malu, and Krishnamurthy Srinivasan. E-business process modeling: the next big step. *IEEE Computer*, 35(5):55 – 62, 2002.
- [152] Reidar Conradi and Alfonso Fuggetta. Improving software process improvement. *IEEE Software*, 19(4):92–99, 2002.
- [153] Tariq Ellahi, Benoit Hudzia, Hui Li, Maik A. Lindner, and Philip Robinson. The enterprise cloud computing paradigm. In Rajkumar Buyya, James Broberg, and Andrzej Goscinski, editors, *Cloud Computing: Principles and Paradigms*, pages 97–118. Wiley, 2011.
- [154] ebxml (electronic business using extensible markup language). <http://www.ebxml.org/>.
- [155] Shuying Wang and Miriam A. M. Capretz. A policy driven approach for service-oriented business rule management. In *IEEE International Conference on Industrial Informatics*, 2007.
- [156] Ralph Mietzner, Tobias Unger, and Frank Leymann. Cafe: A generic configurable customizable composite cloud application framework. In *The Confederated International Conferences, CoopIS, DOA, IS, and ODBASE 2009 on On the Move to Meaningful Internet Systems: Part I*, pages 357–364, 2009.
- [157] Eyhab Al-Masri and Qusay H. Mahmoud. A framework for efficient discovery of web services across heterogeneous registries. In *Consumer Communications and Networking Conference*, 2007.

- [158] Chad Berndtson. Interop: Cloud computing adopters ready to 'trust, but verify', 2009. <http://www.crn.com/software/221900379;jsessionid=R1HY3YANN5EL1QE1GHOSKHWATMY32JVN>.
- [159] Aaron Weiss. Computing in the clouds. *netWorker - Cloud computing: PC functions move onto the web*, 11(4):16–25, 2007.
- [160] Architectural patterns and styles. <http://msdn.microsoft.com/en-us/library/ee658117.aspx>.
- [161] Apache axis2. <http://axis.apache.org/axis2/java/core/>.
- [162] Jboss ws. <http://www.jboss.org/jbossws>.
- [163] Alfonso Fuggetta, Gian Pietro Picco, and Giovanni Vigna. Understanding code mobility. *IEEE Transactions on Software Engineering*, 24(5), 1998.
- [164] Ralph Mietzner, Tobias Unger, Robert Titze, and Frank Leymann. Combining different multi-tenancy patterns in service-oriented applications. In *IEEE International Enterprise Distributed Object Computing Conference*, 2009.
- [165] David C. Hay. *Requirements analysis: from business views to architecture*. Pearson, 2003. pages 143.
- [166] Bashar Nuseibeh and Steve Easterbrook. Requirements engineering: a roadmap. In *Conference on The Future of Software Engineering*, 2000.
- [167] Gargi Dasgupta, Onyeka Ezenwoye, Liana Fong, Selim Kalayci, S. Masoud Sadjadi, and Balaji Viswanathan. Design of a fault-tolerant job-flow manager for grid environments using standard technologies, job-flow patterns, and a transparent proxy. In *International Conference on Software Engineering and Knowledge Engineering*, 2008.
- [168] Rik Eshuis and Akhil Kumar. An integer programming based approach for verification and diagnosis of workflows. *Data and Knowledge Engineering*, 69(8):816–835, 2010.
- [169] Rik Eshuis. Symbolic model checking of uml activity diagrams. *ACM Transactions on Software Engineering and Methodology*, 15(1):1–38, 2006.
- [170] Piero Corte and Debora Desideri. Nessi open framework reference architecture - definition of an architectural framework and principles, 2008. <http://www.nexof-ra.eu/sites/default/files/D7%20%20Definition%20of%20an%20architectural%20framework%20and%20principles.zip>.
- [171] John Domingue, Dieter Fensel, and Rafael Gonzalez-Cabero. Soa4all, enabling the soa revolution on a world wide scale. In *IEEE International Conference on Semantic Computing*, 2008.

- [172] Abdelkarim Erradi, Piyush Maheshwari, and Vladimir Tosic. Ws-policy based monitoring of composite web services. In *IEEE International Conference on Services Computing*, 2007.
- [173] Yu Chen Zhou, Xin Peng Liu, Xi Ning Wang, Liang Xue, Chen Tian, and Xiao Xing Liang. Context model based soa policy framework. In *IEEE International Conference on Web Services*, 2010.
- [174] Yu Chen Zhou, Xin Peng Liu, Eduardo Kahan, Xi Ning Wang, Liang Xue, and Ke Xin Zhou. Context aware service policy orchestration. In *IEEE International Conference on Web Services*, 2007.
- [175] W3c xml schema, 2005. <http://www.w3.org/2001/XMLSchema>.
- [176] Basic xml schema patterns for databinding version 1.0, 2009. <http://www.w3.org/TR/xmlschema-patterns/>.
- [177] W3c xsl transformations (xslt) 2.0. <http://www.w3.org/TR/xslt20/>.
- [178] Levenshtein VI. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10:707, 1966.
- [179] Emil Lupu and Morris Sloman. Conflicts in policy-based distributed systems management. *IEEE Transactions on Software Engineering*, 25(6):852 – 869, 1999.
- [180] Luciano Baresi, Carlo Ghezzi, and Sam Guinea. Towards self-healing service compositions. In *First Conference on the Principles of Software Engineering*, 2004.
- [181] W3c ws-addressing 1.0, 2006. <http://www.w3.org/2005/08/addressing/>.
- [182] Frank Leymann and Dieter Roller. Business processes in a web services world - a quick overview of bpel4ws, 2002. <http://www.ibm.com/developerworks/webservices/library/ws-bpelwp/>.
- [183] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995. pages 305.
- [184] Eyhab Al-Masri and Qusay H. Mahmoud. Discovering the best web service. In *International World Wild Web conference*, 2007.
- [185] Jiri Kolar. Business activity monitoring. Master’s thesis, 2009. Faculty of informatics, Masaryk University.
- [186] Swen Ausmann and Michael Haupt. Axon - dynamic aop through runtime inspection and monitoring. In *Workshop on Advancing the State of the Art in Run-Time Inspection*, 2003.

- [187] Jonas Bonr. Aspectwerkz dynamic aop for java. In *Invited talk at International Conference on Aspect-Oriented Software Development*, 2004.
- [188] The aspectj programming guide, 2003. <http://www.eclipse.org/aspectj/doc/released/progguide/starting-aspectj.html>.
- [189] Aspectj. <http://www.eclipse.org/aspectj/>.
- [190] Spring aop. <http://www.springsource.org/documentation>.
- [191] Michael Eichberg, Mira Mezini, and Klaus Ostermann. Pointcuts as functional queries. In *Asian Symposium on Programming Languages and Systems*, pages 366–381, 2004.
- [192] W3c xml query (xquery). <http://www.w3.org/XML/Query/>.
- [193] W3c xml path language (xpath) 2.0, 2010. <http://www.w3.org/TR/xpath20/>.
- [194] Roberto Lopez-Herrejon, Don Batory, and Christian Lengauer. A disciplined approach to aspect composition. In *ACM SIGPLAN Workshop on Partial Evaluation and Semantics-based Program Manipulation*, 2006.
- [195] Istvn Nagy, Lodewijk Bergmans, and Mehmet Aksit. Declarative aspect composition. In *ASOD workshop on Software Engineering Properties of Languages and Aspect Technologies*, 2004.
- [196] Esper : Event processing for java. <http://www.espertech.com/products/esper.php>.
- [197] Esper reference documentation. <http://esper.codehaus.org/esper/documentation/documentation.html>.
- [198] Rod Johnson, Juergen Hoeller, Keith Donald, and Colin Sampaleanu. Spring framework. <http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/>.
- [199] Jboss aop framework. <http://www.jboss.org/jbossaop>.
- [200] Vinod Muthusamy, Hans-Arno Jacobsen, Tony Chau, Allen Chan, and Phil Coulthard. Sla-driven business process management in soa. In *International Conference of the (IBM) Centre for Advanced Studies on Collaborative Research*, pages 86–100, 2009.
- [201] Vinod Muthusamy and Hans-Arno Jacobsen. Bpm in cloud architectures: Business process management with slas and events. In *International Conference on Business Process Management*, 2010.

Appendix A

Schema of policy model

xmlns : spap = http : //www.computing.dcu.ie/mwang/spap

```
1 <?xml version="1.0"?>
2 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:spap="http://
  www.computing.dcu.ie/mwang/spap" targetNamespace="http://www.computing.dcu
  .ie/mwang/spap">
3   <xsd:import namespace="http://www.w3.org/XML/1998/namespace" schemaLocation=
    "http://www.w3.org/2001/03/xml.xsd"/>
4
5   <xsd:element name="PolicySet" type="spap:PolicySetType"/>
6   <xsd:complexType name="PolicySetType">
7     <xsd:sequence>
8       <xsd:element ref="spap:Description" minOccurs="0"/>
9       <xsd:element ref="spap:Objects"/>
10      <xsd:element ref="spap:ActivityStates"/>
11      <xsd:choice maxOccurs="unbounded" minOccurs="0">
12        <xsd:element ref="spap:PolicySet"/>
13        <xsd:element ref="spap:Policy"/>
14        <xsd:element ref="spap:Aspect"/>
15        <xsd:element name="PolicySetIdReference" type="xsd:anyURI"/>
16        <xsd:element name="PolicyIdReference" type="xsd:anyURI"/>
17      </xsd:choice>
18      <xsd:element ref="spap:Obligations" minOccurs="0"/>
19      <xsd:element ref="spap:ConstraintCombiningAlgorithm"/>
20      <xsd:element ref="spap:RemedyCombiningAlgorithm"/>
21      <xsd:element ref="spap:SequencingAlgorithm"/>
22    </xsd:sequence>
23    <xsd:attribute name="policySetId" type="xsd:anyURI" use="required"/>
24    <xsd:attribute name="priority" type="spap:PriorityType" default="0"/>
25  </xsd:complexType>
26
27
28  <xsd:simpleType name="PriorityType">
29    <xsd:restriction base="xsd:integer">
30      <xsd:minInclusive value="0"/>
31    </xsd:restriction>
32  </xsd:simpleType>
33
34  <!-- objects -->
35
36  <xsd:element name="Objects" type="spap:Objects"/>
37  <xsd:complexType name="Objects">
38    <xsd:sequence minOccurs="0" maxOccurs="unbounded">
39      <xsd:element ref="spap:ObjectsAnyOf"/>
40    </xsd:sequence>
41  </xsd:complexType>
42
43  <xsd:element name="ObjectsAnyOf" type="spap:ObjectsAnyOfType"/>
44  <xsd:complexType name="ObjectsAnyOfType">
45    <xsd:sequence minOccurs="1" maxOccurs="unbounded">
46      <xsd:element ref="spap:ObjectsAllOf"/>
47    </xsd:sequence>
48  </xsd:complexType>
49
50  <xsd:element name="ObjectsAllOf" type="spap:ObjectsAllOfType"/>
```

```

51 <xsd:complexType name="ObjectsAllOfType">
52   <xsd:sequence minOccurs="1" maxOccurs="unbounded">
53     <xsd:element ref="spap:Object"/>
54   </xsd:sequence>
55 </xsd:complexType>
56
57 <xsd:element name="Object" type="spap:ObjectType" abstract="true"/>
58 <xsd:complexType name="ObjectType">
59   <xsd:sequence>
60     <xsd:element ref="spap:SemanticMatchingAlgorithm" minOccurs="0"/>
61   </xsd:sequence>
62 </xsd:complexType>
63
64 <xsd:element name="Process" type="spap:ProcessType" substitutionGroup="
    spap:Object"/>
65 <xsd:complexType name="ProcessType">
66   <xsd:complexContent>
67     <xsd:extension base="spap:ObjectType">
68       <xsd:sequence>
69         <xsd:element name="Name" type="xsd:string"/>
70       </xsd:sequence>
71     </xsd:extension>
72   </xsd:complexContent>
73 </xsd:complexType>
74
75 <xsd:element name="Activity" type="spap:ActivityType" substitutionGroup="
    spap:Object"/>
76 <xsd:complexType name="ActivityType">
77   <xsd:complexContent>
78     <xsd:extension base="spap:ObjectType">
79       <xsd:sequence>
80         <xsd:element name="Name" type="xsd:string"/>
81       </xsd:sequence>
82     </xsd:extension>
83   </xsd:complexContent>
84 </xsd:complexType>
85
86 <xsd:element name="WS-Operation" type="xsd:string"/>
87 <xsd:element name="WS-Address" type="xsd:anyURI"/>
88
89 <xsd:element name="Resource" type="spap:ResourceType" substitutionGroup="
    spap:Object"/>
90 <xsd:complexType name="ResourceType">
91   <xsd:complexContent>
92     <xsd:extension base="spap:ObjectType">
93       <xsd:sequence>
94         <xsd:element name="Name" type="xsd:string"/>
95       </xsd:sequence>
96     </xsd:extension>
97   </xsd:complexContent>
98 </xsd:complexType>
99
100 <xsd:element name="Violation" type="spap:ViolationType" substitutionGroup="
    spap:Object"/>
101 <xsd:complexType name="ViolationType">
102   <xsd:complexContent>
103     <xsd:extension base="spap:ObjectType">
104       <xsd:sequence>
105         <xsd:element name="Type">
106           <xsd:simpleType>
107             <xsd:union memberTypes="spap:ViolationTypeType_
                spap:ExtendViolationStringPatternTypeType"/>

```

```

108         </xsd:simpleType>
109     </xsd:element>
110 </xsd:sequence>
111 </xsd:extension>
112 </xsd:complexContent>
113 </xsd:complexType>
114
115 <xsd:element name="Description" type="xsd:string"/>
116
117 <xsd:element name="Policy" type="spap:PolicyType"/>
118 <xsd:complexType name="PolicyType">
119     <xsd:sequence>
120         <xsd:element ref="spap:Description" minOccurs="0"/>
121         <xsd:element ref="spap:Objects"/>
122         <xsd:element ref="spap:ActivityStates"/>
123         <xsd:choice maxOccurs="unbounded">
124             <xsd:element ref="spap:Rule"/>
125             <xsd:element name="RuleIdReference" type="xsd:anyURI"/>
126         </xsd:choice>
127         <xsd:element ref="spap:FaultHandler" minOccurs="0"/>
128         <xsd:element ref="spap:Obligations" minOccurs="0"/>
129         <xsd:element ref="spap:ConstraintCombiningAlgorithm"/>
130         <xsd:element ref="spap:RemedyCombiningAlgorithm"/>
131         <xsd:element ref="spap:SequencingAlgorithm"/>
132     </xsd:sequence>
133     <xsd:attribute name="policyId" type="xsd:anyURI" use="required"/>
134     <xsd:attribute name="priority" type="spap:PriorityType" default="0"/>
135 </xsd:complexType>
136
137 <xsd:element name="Rule" type="spap:RuleType"/>
138 <xsd:complexType name="RuleType">
139     <xsd:sequence>
140         <xsd:element ref="spap:Description" minOccurs="0"/>
141         <xsd:element ref="spap:Objects" minOccurs="0"/>
142         <xsd:element ref="spap:ActivityStates" minOccurs="0"/>
143         <xsd:element ref="spap:Conditions" minOccurs="1"/>
144         <xsd:element ref="spap:Actions" minOccurs="1"/>
145         <xsd:element ref="spap:FaultHandler" minOccurs="0"/>
146         <xsd:element ref="spap:Obligations" minOccurs="0"/>
147     </xsd:sequence>
148     <xsd:attribute name="ruleId" type="xsd:anyURI" use="required"/>
149     <xsd:attribute name="priority" type="spap:PriorityType" default="0"/>
150 </xsd:complexType>
151
152 <xsd:element name="Aspect" type="spap:AspectType"/>
153 <xsd:complexType name="AspectType">
154     <xsd:sequence>
155         <xsd:element ref="spap:Description" minOccurs="0"/>
156     </xsd:sequence>
157     <xsd:attribute name="policyId" type="xsd:anyURI" use="required"/>
158     <xsd:attribute name="priority" type="spap:PriorityType" default="0"/>
159 </xsd:complexType>
160
161 <!-- Violation type type-->
162
163 <xsd:element name="ViolationType" type="spap:ViolationTypeType"/>
164 <xsd:simpleType name="ViolationTypeType">
165     <xsd:restriction base="xsd:string">
166         <xsd:enumeration value="Functional"/>
167         <xsd:enumeration value="Functional:Syntax"/>
168         <xsd:enumeration value="Functional:Effect"/>
169         <xsd:enumeration value="Functional:Protocol"/>

```



```

170     <xsd:enumeration value="QoS"/>
171     <xsd:enumeration value="QoS:Performance"/>
172     <xsd:enumeration value="QoS:Trust"/>
173     <xsd:enumeration value="QoS:Security"/>
174     <xsd:enumeration value="Financial"/>
175     <xsd:enumeration value="Security"/>
176     <xsd:enumeration value="Trust"/>
177     <xsd:enumeration value="Semantic"/>
178     <xsd:enumeration value="Linguistic"/>
179     <xsd:enumeration value="MeasuresAndStandard"/>
180     <xsd:enumeration value="Device"/>
181     <xsd:enumeration value="Connectivity"/>
182     <xsd:enumeration value="Unknown"/>
183 </xsd:restriction>
184 </xsd:simpleType>
185
186 <xsd:element name="ExtendViolationStringPatternType" type="
187     spap:ExtendViolationStringPatternTypeType"/>
188 <xsd:simpleType name="ExtendViolationStringPatternTypeType">
189     <xsd:restriction base="xsd:string">
190         <xsd:pattern value="Extend:\S.*"/>
191     </xsd:restriction>
192 </xsd:simpleType>
193
194 <!-- FaultHandler -->
195
196 <xsd:element name="FaultHandler" type="spap:FaultHandlerType"/>
197 <xsd:complexType name="FaultHandlerType">
198     <xsd:sequence>
199         <xsd:element ref="spap:Action" maxOccurs="unbounded"/>
200     </xsd:sequence>
201 </xsd:complexType>
202
203 <xsd:element name="Actions" type="spap:ActionsType"/>
204 <xsd:complexType name="ActionsType">
205     <xsd:sequence>
206         <xsd:element ref="spap:Action" maxOccurs="unbounded"/>
207     </xsd:sequence>
208 </xsd:complexType>
209
210 <xsd:element name="Action" type="spap:ActionType" abstract="true"/>
211 <xsd:complexType name="ActionType">
212     <xsd:sequence>
213 <!-- Conconsumer actions -->
214
215 <xsd:element name="ConsumerAction" type="spap:ConsumerActionType"
216     substitutionGroup="spap:Action"/>
217 <xsd:complexType name="ConsumerActionType">
218     <xsd:complexContent>
219         <xsd:extension base="spap:ActionType"/>
220     </xsd:extension>
221 </xsd:complexContent>
222 </xsd:complexType>
223
224 <xsd:element name="Ca-Log" type="spap:Ca-LogType" substitutionGroup="
225     spap:ConsumerAction"/>
226 <xsd:complexType name="Ca-LogType">
227     <xsd:complexContent>
228         <xsd:extension base="spap:ConsumerActionType">
229             <xsd:attribute name="level" type="xsd:string" use="required"/>
230         </xsd:extension>
231     </xsd:complexContent>
232 </xsd:complexType>

```

```

229     </xsd:complexContent>
230 </xsd:complexType>
231
232 <xsd:element name="Ca-Suspend" type="spap:Ca-SuspendType" substitutionGroup=
    "spap:ConsumerAction"/>
233 <xsd:complexType name="Ca-SuspendType">
234     <xsd:complexContent>
235         <xsd:extension base="spap:ConsumerActionType">
236             <xsd:attribute name="Time" type="xsd:duration" use="required"/>
237         </xsd:extension>
238     </xsd:complexContent>
239 </xsd:complexType>
240
241 <xsd:element name="Ca-Alert" type="spap:Ca-AlertType" substitutionGroup="
    spap:ConsumerAction"/>
242 <xsd:complexType name="Ca-AlertType">
243     <xsd:complexContent>
244         <xsd:extension base="spap:ConsumerActionType">
245             <xsd:attribute name="mailto" type="xsd:string" use="required"/>
246         </xsd:extension>
247     </xsd:complexContent>
248 </xsd:complexType>
249
250 <!-- Provider actions -->
251
252 <xsd:element name="ProviderAction" type="spap:ProviderActionType"
    substitutionGroup="spap:Action"/>
253 <xsd:complexType name="ProviderActionType">
254     <xsd:complexContent>
255         <xsd:extension base="spap:ActionType">
256         </xsd:extension>
257     </xsd:complexContent>
258 </xsd:complexType>
259
260 <xsd:element name="Pa-Validate" type="spap:Pa-ValidateType"
    substitutionGroup="spap:ProviderAction"/>
261 <xsd:complexType name="Pa-ValidateType">
262     <xsd:complexContent>
263         <xsd:extension base="spap:ProviderActionType">
264         </xsd:extension>
265     </xsd:complexContent>
266 </xsd:complexType>
267
268 <xsd:element name="Pa-Violate" type="spap:Pa-ViolateType" substitutionGroup=
    "spap:ProviderAction"/>
269 <xsd:complexType name="Pa-ViolateType">
270     <xsd:complexContent>
271         <xsd:extension base="spap:ProviderActionType">
272             <xsd:sequence maxOccurs="unbounded">
273                 <xsd:element ref="spap:Violation"/>
274             </xsd:sequence>
275         </xsd:extension>
276     </xsd:complexContent>
277 </xsd:complexType>
278
279 <xsd:element name="Pa-Ignore" type="spap:Pa-IgnoreType" substitutionGroup="
    spap:ProviderAction"/>
280 <xsd:complexType name="Pa-IgnoreType">
281     <xsd:complexContent>
282         <xsd:extension base="spap:ProviderActionType">
283         </xsd:extension>
284     </xsd:complexContent>

```

```

285 </xsd:complexType>
286
287 <xsd:element name="Pa-Skip" type="spap:Pa-SkipType" substitutionGroup="
    spap:ProviderAction"/>
288 <xsd:complexType name="Pa-SkipType">
289   <xsd:complexContent>
290     <xsd:extension base="spap:ProviderActionType">
291       </xsd:extension>
292     </xsd:complexContent>
293   </xsd:complexType>
294
295 <xsd:element name="Pa-Retry" type="spap:Pa-RetryType" substitutionGroup="
    spap:ProviderAction"/>
296 <xsd:complexType name="Pa-RetryType">
297   <xsd:complexContent>
298     <xsd:extension base="spap:ProviderActionType">
299       <xsd:attribute name="WaitFor" type="xsd:duration" use="required"/>
300     </xsd:extension>
301   </xsd:complexContent>
302 </xsd:complexType>
303
304 <xsd:element name="Pa-Replace" type="spap:Pa-ReplaceType" substitutionGroup="
    spap:ProviderAction"/>
305 <xsd:complexType name="Pa-ReplaceType">
306   <xsd:complexContent>
307     <xsd:extension base="spap:ProviderActionType">
308       <xsd:sequence>
309         <xsd:element ref="spap:ServiceConditions"/>
310       </xsd:sequence>
311       <xsd:attribute name="InstanceOnly" type="xsd:boolean" use="required"/>
312     </xsd:extension>
313   </xsd:complexContent>
314 </xsd:complexType>
315
316 <xsd:element name="Pa-Cancel" type="spap:Pa-CancelType" substitutionGroup="
    spap:ProviderAction"/>
317 <xsd:complexType name="Pa-CancelType">
318   <xsd:complexContent>
319     <xsd:extension base="spap:ProviderActionType">
320       </xsd:extension>
321     </xsd:complexContent>
322 </xsd:complexType>
323
324 <xsd:element name="Pa-Compensate" type="spap:Pa-CompensateType"
    substitutionGroup="spap:ProviderAction"/>
325 <xsd:complexType name="Pa-CompensateType">
326   <xsd:complexContent>
327     <xsd:extension base="spap:ProviderActionType">
328       <xsd:sequence>
329         <xsd:element ref="spap:ServiceConditions"/>
330       </xsd:sequence>
331     </xsd:extension>
332   </xsd:complexContent>
333 </xsd:complexType>
334
335 <xsd:element name="Pa-Manipulate" type="spap:Pa-ManipulateType"
    substitutionGroup="spap:ProviderAction"/>
336 <xsd:complexType name="Pa-ManipulateType">
337   <xsd:complexContent>
338     <xsd:extension base="spap:ProviderActionType">
339       <xsd:sequence>
340         <xsd:choice maxOccurs="unbounded">

```

```

341         <xsd:element ref="spap:Copy" minOccurs="1"/>
342     </xsd:choice>
343 </xsd:sequence>
344 </xsd:extension>
345 </xsd:complexContent>
346 </xsd:complexType>
347
348 <xsd:element name="Copy" type="spap:CopyType"/>
349 <xsd:complexType name="CopyType">
350     <xsd:sequence>
351         <xsd:element ref="spap:From" minOccurs="1"/>
352         <xsd:element ref="spap:To" minOccurs="1"/>
353     </xsd:sequence>
354 </xsd:complexType>
355
356 <xsd:element name="From" type="spap:FromType"/>
357 <xsd:complexType name="FromType">
358     <xsd:sequence>
359         <xsd:choice>
360             <xsd:element ref="spap:Literal"/>
361             <xsd:element ref="spap:XsltTrans"/>
362         </xsd:choice>
363     </xsd:sequence>
364 </xsd:complexType>
365
366 <xsd:element name="Literal" type="spap:LiteralType"/>
367 <xsd:complexType name="LiteralType" mixed="true">
368     <xsd:sequence>
369         <xsd:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs
370             = "1"/>
371     </xsd:sequence>
372 </xsd:complexType>
373
374 <xsd:element name="XsltTrans" type="spap:XsltTransType"/>
375 <xsd:complexType name="XsltTransType">
376     <xsd:attribute name="source" type="xsd:string"/>
377     <xsd:attribute name="xslt" type="xsd:anyURI"/>
378 </xsd:complexType>
379
380 <xsd:element name="To" type="spap:ToType"/>
381 <xsd:complexType name="ToType">
382     <xsd:attribute name="query" type="xsd:string" use="required"/>
383 </xsd:complexType>
384
385 <!-- following actions for code generation with policy weaver development
386     only -->
387
388 <xsd:element name="Pa-Undefined" type="spap:Pa-UndefinedType"
389     substitutionGroup="spap:ProviderAction"/>
390 <xsd:complexType name="Pa-UndefinedType">
391     <xsd:complexContent>
392         <xsd:extension base="spap:ProviderActionType"/>
393     </xsd:extension>
394 </xsd:complexContent>
395 </xsd:complexType>
396
397 <xsd:element name="Pa-Unexpected" type="spap:Pa-UnexpectedType"
398     substitutionGroup="spap:ProviderAction"/>
399 <xsd:complexType name="Pa-UnexpectedType">
400     <xsd:complexContent>
401         <xsd:extension base="spap:ProviderActionType"/>
402     </xsd:extension>

```

```

399     </xsd:complexContent>
400 </xsd:complexType>
401
402 <xsd:element name="Pa-Undetermined" type="spap:Pa-UndeterminedType"
      substitutionGroup="spap:ProviderAction"/>
403 <xsd:complexType name="Pa-UndeterminedType">
404   <xsd:complexContent>
405     <xsd:extension base="spap:ProviderActionType">
406       </xsd:extension>
407     </xsd:complexContent>
408   </xsd:complexType>
409
410 <xsd:element name="Pa-Compensate-Ignore" type="spap:Pa-Compensate-IgnoreType"
      substitutionGroup="spap:ProviderAction"/>
411 <xsd:complexType name="Pa-Compensate-IgnoreType">
412   <xsd:complexContent>
413     <xsd:extension base="spap:ProviderActionType">
414       <xsd:sequence>
415         <xsd:element ref="spap:ServiceConditions"/>
416       </xsd:sequence>
417     </xsd:extension>
418   </xsd:complexContent>
419 </xsd:complexType>
420
421 <xsd:element name="Pa-Compensate-Replace" type="spap:Pa-Compensate-
      ReplaceType" substitutionGroup="spap:ProviderAction"/>
422 <xsd:complexType name="Pa-Compensate-ReplaceType">
423   <xsd:complexContent>
424     <xsd:extension base="spap:ProviderActionType">
425       <xsd:sequence>
426         <xsd:element ref="spap:ServiceConditions" minOccurs="2" maxOccurs="2"
              "/>
427       </xsd:sequence>
428       <xsd:attribute name="InstanceOnly" type="xsd:boolean" use="required"/>
429     </xsd:extension>
430   </xsd:complexContent>
431 </xsd:complexType>
432
433 <!-- Obligations -->
434
435 <xsd:element name="Obligations" type="spap:ObligationsType"/>
436 <xsd:complexType name="ObligationsType">
437   <xsd:sequence maxOccurs="unbounded" minOccurs="1">
438     <xsd:element ref="spap:Obligation"/>
439   </xsd:sequence>
440 </xsd:complexType>
441
442 <xsd:element name="Obligation" type="spap:ObligationType"/>
443 <xsd:complexType name="ObligationType">
444   <xsd:sequence maxOccurs="unbounded" minOccurs="1">
445     <xsd:element ref="spap:ConsumerAction"/>
446   </xsd:sequence>
447   <xsd:attribute name="Type" use="required">
448     <xsd:simpleType>
449       <xsd:restriction base="xsd:string">
450         <xsd:enumeration value="Pa-Validate"/>
451         <xsd:enumeration value="Pa-Violate"/>
452         <xsd:enumeration value="Pa-Ignore"/>
453         <xsd:enumeration value="Pa-Replace-InstanceOnly"/>
454         <xsd:enumeration value="Pa-Replace"/>
455         <xsd:enumeration value="Pa-Compensate"/>
456         <xsd:enumeration value="Pa-Cancel"/>

```

```

457         <xsd:enumeration value="Pa-Retry"/>
458         <xsd:enumeration value="Pa-Undetermined"/>
459         <xsd:enumeration value="Pa-Undefined"/>
460         <xsd:enumeration value="Pa-Unexpected"/>
461     </xsd:restriction>
462 </xsd:simpleType>
463 </xsd:attribute>
464 </xsd:complexType>
465
466 <!-- Service conditions -->
467
468 <xsd:element name="ServiceConditions" type="spap:ServiceConditionsType"/>
469 <xsd:complexType name="ServiceConditionsType">
470     <xsd:sequence>
471         <xsd:element ref="spap:ServiceConditionExpression" minOccurs="0"
472             maxOccurs="unbounded"/>
473     </xsd:sequence>
474 </xsd:complexType>
475
476 <xsd:element name="ServiceConditionExpression" type="
477     spap:ServiceConditionExpressionType"/>
478 <xsd:complexType name="ServiceConditionExpressionType">
479     <xsd:attribute name="force" type="xsd:boolean" default="false"/>
480     <xsd:attribute name="expression" type="xsd:string" default="false"/>
481 </xsd:complexType>
482
483 <!-- Rule conditions -->
484
485 <xsd:element name="Conditions" type="spap:ConditionsType"/>
486 <xsd:complexType name="ConditionsType">
487     <xsd:sequence minOccurs="0" maxOccurs="unbounded">
488         <xsd:element ref="spap:ConditionExpression"/>
489     </xsd:sequence>
490 </xsd:complexType>
491
492 <xsd:element name="ConditionExpression" type="xsd:string"/>
493
494 <!-- Activity States -->
495
496 <xsd:element name="ActivityStates" type="spap:ActivityStatesType"/>
497 <xsd:complexType name="ActivityStatesType">
498     <xsd:sequence minOccurs="0" maxOccurs="unbounded">
499         <xsd:element ref="spap:ActivityState"/>
500     </xsd:sequence>
501 </xsd:complexType>
502
503 <xsd:element name="ActivityState" type="spap:ActivityStateType"/>
504 <xsd:simpleType name="ActivityStateType">
505     <xsd:restriction base="xsd:string">
506         <xsd:enumeration value="Validating-Pre"/>
507         <xsd:enumeration value="Validating-Post"/>
508         <xsd:enumeration value="Manipulating-Pre-Validating-Pre"/>
509         <xsd:enumeration value="Manipulating-Pre-Validating-Post"/>
510         <xsd:enumeration value="Manipulating-Post-Validating-Pre"/>
511         <xsd:enumeration value="Manipulating-Post-Validating-Post"/>
512         <xsd:enumeration value="Handling-Pre"/>
513         <xsd:enumeration value="Handling-Post"/>
514         <xsd:enumeration value="Cancelling"/>
515     </xsd:restriction>
516 </xsd:simpleType>

```

```

517 <!-- algorithms -->
518
519 <xsd:element name="ConstraintCombiningAlgorithm" type="
    spap:ConstraintCombiningAlgorithmType"/>
520 <xsd:complexType name="ConstraintCombiningAlgorithmType">
521   <xsd:attribute name="type" use="required">
522     <xsd:simpleType>
523       <xsd:restriction base="xsd:string">
524         <xsd:enumeration value="Pa-Violate-Override-Through-All"/>
525         <xsd:enumeration value="Pa-Validate-Override-Through-All"/>
526         <xsd:enumeration value="Pa-Violate-Unless-Pa-Validate-Through-All"/>
527         <xsd:enumeration value="Pa-Validate-Unless-Pa-Violate-Through-All"/>
528       </xsd:restriction>
529     </xsd:simpleType>
530   </xsd:attribute>
531 </xsd:complexType>
532
533 <xsd:element name="RemedyCombiningAlgorithm" type="
    spap:RemedyCombiningAlgorithmType"/>
534 <xsd:complexType name="RemedyCombiningAlgorithmType">
535   <xsd:sequence>
536     <xsd:element name="DefinedSequenceElement" maxOccurs="unbounded">
537       <xsd:simpleType>
538         <xsd:restriction base="xsd:string">
539           <xsd:enumeration value="Pa-Ignore"/>
540           <xsd:enumeration value="Pa-Retry"/>
541           <xsd:enumeration value="Pa-Replace"/>
542           <xsd:enumeration value="Pa-ReplaceInstanceOnly"/>
543           <xsd:enumeration value="Pa-Cancel"/>
544           <xsd:enumeration value="Pa-Skip"/>
545         </xsd:restriction>
546       </xsd:simpleType>
547     </xsd:element>
548   </xsd:sequence>
549
550   <xsd:attribute name="type" use="required">
551     <xsd:simpleType>
552       <xsd:restriction base="xsd:string">
553         <xsd:enumeration value="Defined-Sequence-Overrides-Through-All"/>
554         <xsd:enumeration value="Pa-Ignore-Unless-Defined-Sequence-Through-
            All"/>
555         <xsd:enumeration value="Pa-Cancel-Unless-Defined-Sequence-Through-
            All"/>
556       </xsd:restriction>
557     </xsd:simpleType>
558   </xsd:attribute>
559 </xsd:complexType>
560
561 <xsd:element name="SequencingAlgorithm" type="spap:SequencingAlgorithmType"/
    >
562 <xsd:complexType name="SequencingAlgorithmType">
563   <xsd:attribute name="type" use="required">
564     <xsd:simpleType>
565       <xsd:restriction base="xsd:string">
566         <xsd:enumeration value="Ordered"/>
567         <xsd:enumeration value="PriorityBased-QuickSort"/>
568       </xsd:restriction>
569     </xsd:simpleType>
570   </xsd:attribute>
571 </xsd:complexType>
572

```

```
573 <xsd:element name="SemanticMatchingAlgorithm" type="
      spap:SemanticMatchingAlgorithmType"/>
574 <xsd:complexType name="SemanticMatchingAlgorithmType">
575   <xsd:attribute name="type" use="required">
576     <xsd:simpleType>
577       <xsd:restriction base="xsd:string">
578         <xsd:enumeration value="LevenshteinDistance"/>
579       </xsd:restriction>
580     </xsd:simpleType>
581   </xsd:attribute>
582   <xsd:attribute name="matchingDegree" type="xsd:float"/>
583 </xsd:complexType>
584 </xsd:schema>
```


Appendix B

Schema of coordination context

xmlns : spac = http://www.computing.dcu.ie/mwang/spac

```
1 <?xml version="1.0"?>
2 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:spac="http://
  www.computing.dcu.ie/mwang/spac" targetNamespace="http://www.computing.dcu
  .ie/mwang/spac" xmlns:wsa="http://www.w3.org/2005/08/addressing">
3   <xsd:import namespace="http://www.w3.org/XML/1998/namespace" schemaLocation=
    "http://www.w3.org/2001/03/xml.xsd"/>
4   <xsd:import namespace="http://www.w3.org/2005/08/addressing" schemaLocation=
    "http://www.w3.org/2006/03/addressing/ws-addr.xsd"/>
5
6   <xsd:element name="CoordinationContext" type="spac:CoordinationContextType"/>
7
8   <xsd:complexType name="CoordinationContextType">
9     <xsd:sequence>
10      <xsd:element name="CId" type="xsd:string"/>
11      <xsd:element name="CoordinationType" type="xsd:anyURI"/>
12      <xsd:element ref="spac:ProtocolService"/>
13      <xsd:element ref="spac:Cache" minOccurs="0"/>
14    </xsd:sequence>
15  </xsd:complexType>
16
17  <xsd:element name="Cache" type="spac:CacheType"/>
18  <xsd:complexType name="CacheType">
19    <xsd:sequence>
20      <xsd:element name="StartDateTime" type="xsd:dateTime" />
21      <xsd:element name="EndDateTime" type="xsd:dateTime"/>
22    </xsd:sequence>
23    <xsd:attribute name="Scope" default="Global">
24      <xsd:simpleType>
25        <xsd:restriction base="xsd:string">
26          <xsd:enumeration value="Process"/>
27          <xsd:enumeration value="Global"/>
28        </xsd:restriction>
29      </xsd:simpleType>
30    </xsd:attribute>
31  </xsd:complexType>
32
33  <xsd:element name="ProtocolService" type="wsa:EndpointReferenceType"/>
34 </xsd:schema>
```

Appendix C

Schema of process activity protocol

xmlns : spaa = http://www.computing.dcu.ie/mwang/spaa

```
1 <?xml version="1.0"?>
2 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:spaa="http://
  www.computing.dcu.ie/mwang/spaa" targetNamespace="http://www.computing.dcu
  .ie/mwang/spaa" xmlns:spac="http://www.computing.dcu.ie/mwang/spac">
3   <xsd:import namespace="http://www.w3.org/XML/1998/namespace" schemaLocation=
    "http://www.w3.org/2001/03/xml.xsd"/>
4   <xsd:import namespace="http://www.computing.dcu.ie/mwang/spac"
    schemaLocation="coordination.xsd"/>
5
6   <xsd:element name="WeavingRequest" type="spaa:WeavingRequestType"/>
7   <xsd:complexType name="WeavingRequestType">
8     <xsd:sequence>
9       <xsd:element ref="spaa:Process"/>
10      <xsd:element ref="spaa:Activity"/>
11      <xsd:element ref="spaa:Resource"/>
12      <xsd:element ref="spaa:Violation" minOccurs="0" maxOccurs="unbounded"/>
13      <xsd:element ref="spaa:ActivityState"/>
14    </xsd:sequence>
15  </xsd:complexType>
16
17  <xsd:element name="WeavingResponse" type="spaa:WeavingResponseType"/>
18  <xsd:complexType name="WeavingResponseType">
19    <xsd:sequence>
20      <xsd:element ref="spaa:ProviderAction"/>
21    </xsd:sequence>
22  </xsd:complexType>
23
24  <xsd:element name="Process" type="spaa:ProcessType"/>
25  <xsd:complexType name="ProcessType">
26    <xsd:sequence>
27      <xsd:element ref="spaa:Name"/>
28      <xsd:element ref="spaa:ServiceReference"/>
29    </xsd:sequence>
30  </xsd:complexType>
31
32  <xsd:element name="Activity" type="spaa:ActivityType"/>
33  <xsd:complexType name="ActivityType">
34    <xsd:sequence>
35      <xsd:element ref="spaa:Name"/>
36      <xsd:element ref="spaa:ServiceReference"/>
37    </xsd:sequence>
38  </xsd:complexType>
39
40  <xsd:element name="Resource" type="spaa:ResourceType"/>
41  <xsd:complexType name="ResourceType">
42    <xsd:sequence>
43      <xsd:any namespace="##other" processContents="lax" minOccurs="0"
        maxOccurs="unbounded"/>
44    </xsd:sequence>
45  </xsd:complexType>
46
47  <xsd:element name="Name" type="xsd:string"/>
48
```

```

49 <xsd:element name="Violation" type="spaa:ViolationType"/>
50 <xsd:complexType name="ViolationType">
51   <xsd:sequence>
52     <xsd:element ref="spaa:Type"/>
53   </xsd:sequence>
54 </xsd:complexType>
55
56 <xsd:element name="Type" type="xsd:string"/>
57
58 <xsd:element name="ActivityState" type="spaa:ActivityStateType"/>
59 <xsd:simpleType name="ActivityStateType">
60   <xsd:restriction base="xsd:string">
61     <!-- provider part -->
62     <xsd:enumeration value="Manipulating-Validating-Pre"/>
63     <xsd:enumeration value="Manipulating-Validating-Post"/>
64     <xsd:enumeration value="Handling-Pre"/>
65     <xsd:enumeration value="Handling-Post"/>
66     <xsd:enumeration value="Cancelling"/>
67     <!-- consumer part -->
68     <xsd:enumeration value="Validating-Pre"/>
69     <xsd:enumeration value="Validating-Post"/>
70     <xsd:enumeration value="Manipulating-Pre-Validating-Pre"/>
71     <xsd:enumeration value="Manipulating-Pre-Validating-Post"/>
72     <xsd:enumeration value="Manipulating-Post-Validating-Pre"/>
73     <xsd:enumeration value="Manipulating-Post-Validating-Post"/>
74     <!-- for our code generate purpose only -->
75     <xsd:enumeration value="Executing"/>
76     <xsd:enumeration value="Completed"/>
77   </xsd:restriction>
78 </xsd:simpleType>
79
80 <xsd:element name="ServiceReference" type="spaa:ServiceReferenceType"/>
81 <xsd:complexType name="ServiceReferenceType">
82   <xsd:sequence>
83     <xsd:element ref="spaa:Address"/>
84     <xsd:element ref="spaa:Operation"/>
85     <xsd:element ref="spaa:ServiceName"/>
86     <xsd:element ref="spaa:PortName"/>
87     <xsd:element ref="spaa:SOAPAction"/>
88   </xsd:sequence>
89 </xsd:complexType>
90
91 <xsd:element name="Operation" type="xsd:string"/>
92 <xsd:element name="Address" type="xsd:anyURI"/>
93 <xsd:element name="ServiceName" type="xsd:QName"/>
94 <xsd:element name="PortName" type="xsd:QName"/>
95 <xsd:element name="SOAPAction" type="xsd:string"/>
96
97 <!-- provider action -->
98
99 <xsd:element name="ProviderAction" type="spaa:ProviderActionType" abstract="
   true"/>
100 <xsd:complexType name="ProviderActionType">
101 </xsd:complexType>
102
103
104 <xsd:element name="Pa-Validate" type="spaa:Pa-ValidateType"
   substitutionGroup="spaa:ProviderAction"/>
105 <xsd:complexType name="Pa-ValidateType">
106   <xsd:complexContent>
107     <xsd:extension base="spaa:ProviderActionType">
108       <xsd:sequence>

```

```

109         <xsd:element ref="spaa:Resource"/>
110     </xsd:sequence>
111 </xsd:extension>
112 </xsd:complexContent>
113 </xsd:complexType>
114
115 <xsd:element name="Pa-Violate" type="spaa:Pa-ViolateType" substitutionGroup=
    "spaa:ProviderAction"/>
116 <xsd:complexType name="Pa-ViolateType">
117     <xsd:complexContent>
118         <xsd:extension base="spaa:ProviderActionType">
119             <xsd:sequence maxOccurs="unbounded">
120                 <xsd:element ref="spaa:Violation"/>
121             </xsd:sequence>
122         </xsd:extension>
123     </xsd:complexContent>
124 </xsd:complexType>
125
126
127 <xsd:element name="Pa-Ignore" type="spaa:Pa-IgnoreType" substitutionGroup="
    spaa:ProviderAction"/>
128 <xsd:complexType name="Pa-IgnoreType">
129     <xsd:complexContent>
130         <xsd:extension base="spaa:ProviderActionType">
131             </xsd:extension>
132         </xsd:complexContent>
133 </xsd:complexType>
134
135 <xsd:element name="Pa-Skip" type="spaa:Pa-SkipType" substitutionGroup="
    spaa:ProviderAction"/>
136 <xsd:complexType name="Pa-SkipType">
137     <xsd:complexContent>
138         <xsd:extension base="spaa:ProviderActionType">
139             </xsd:extension>
140         </xsd:complexContent>
141 </xsd:complexType>
142
143 <xsd:element name="Pa-Retry" type="spaa:Pa-RetryType" substitutionGroup="
    spaa:ProviderAction"/>
144 <xsd:complexType name="Pa-RetryType">
145     <xsd:complexContent>
146         <xsd:extension base="spaa:ProviderActionType">
147             <xsd:attribute name="WaitFor" type="xsd:duration" use="required"/>
148         </xsd:extension>
149     </xsd:complexContent>
150 </xsd:complexType>
151
152 <xsd:element name="Pa-Replace" type="spaa:Pa-ReplaceType" substitutionGroup=
    "spaa:ProviderAction"/>
153 <xsd:complexType name="Pa-ReplaceType">
154     <xsd:complexContent>
155         <xsd:extension base="spaa:ProviderActionType">
156             <xsd:sequence>
157                 <xsd:element ref="spaa:ServiceReference"/>
158             </xsd:sequence>
159             <xsd:attribute name="InstanceOnly" type="xsd:boolean" use="required"/>
160         </xsd:extension>
161     </xsd:complexContent>
162 </xsd:complexType>
163
164

```

```

165 <xsd:element name="Pa-Cancel" type="spaa:Pa-CancelType" substitutionGroup="
      spaa:ProviderAction"/>
166 <xsd:complexType name="Pa-CancelType">
167   <xsd:complexContent>
168     <xsd:extension base="spaa:ProviderActionType">
169       </xsd:extension>
170     </xsd:complexContent>
171   </xsd:complexType>
172
173
174 <xsd:element name="Pa-Compensate" type="spaa:Pa-CompensateType"
      substitutionGroup="spaa:ProviderAction"/>
175 <xsd:complexType name="Pa-CompensateType">
176   <xsd:complexContent>
177     <xsd:extension base="spaa:ProviderActionType">
178       <xsd:sequence>
179         <xsd:element ref="spaa:ServiceReference"/>
180       </xsd:sequence>
181     </xsd:extension>
182   </xsd:complexContent>
183 </xsd:complexType>
184
185
186 <xsd:element name="Pa-Undefined" type="spaa:Pa-UndefinedType"
      substitutionGroup="spaa:ProviderAction"/>
187 <xsd:complexType name="Pa-UndefinedType">
188   <xsd:complexContent>
189     <xsd:extension base="spaa:ProviderActionType">
190       </xsd:extension>
191     </xsd:complexContent>
192 </xsd:complexType>
193
194 <xsd:element name="Pa-Unexpected" type="spaa:Pa-UnexpectedType"
      substitutionGroup="spaa:ProviderAction"/>
195 <xsd:complexType name="Pa-UnexpectedType">
196   <xsd:complexContent>
197     <xsd:extension base="spaa:ProviderActionType">
198       </xsd:extension>
199     </xsd:complexContent>
200 </xsd:complexType>
201
202 <xsd:element name="Pa-Undetermined" type="spaa:Pa-UndeterminedType"
      substitutionGroup="spaa:ProviderAction"/>
203 <xsd:complexType name="Pa-UndeterminedType">
204   <xsd:complexContent>
205     <xsd:extension base="spaa:ProviderActionType">
206       </xsd:extension>
207     </xsd:complexContent>
208 </xsd:complexType>
209 </xsd:schema>

```