MACQUARIE University SYDNEY · AUSTRALIA

Accountability in Cloud Services

by

Jun Zou

A thesis submitted in fulfillment

of the requirements for the degree of

Doctor of Philosophy

in the

Department of Computing

Faculty of Science and Engineering

Macquarie University

Supervisor: A/Prof. Yan Wang

Associate Supervisor: Prof. Jian Yang

2016

Statement of Candidate

I certify that the work in this thesis entitled **"Accountability in Cloud Services"** has not previously been submitted for a degree nor has it been submitted as part of requirements for a degree to any other university or institution other than Macquarie University.

I also certify that the thesis is an original piece of research and it has been written by me. Any help and assistance that I have received in my research work and the preparation of the thesis itself have been appropriately acknowledged.

In addition, I certify that all information sources and literature used are indicated in the thesis.

[illegible]

Jun Zou

1 November 2016

To my mother Lifeng Xiang,

and my wife Annie Zou,

who make me understand the true meaning of love

Abstract

Service computing has become the main theme in IT while Web services and cloud services are widely adopted by the industry. An important characteristic of service computing is that it turns IT capability into a service, and facilitates automatic service delivery and consumption through the ubiquitous Internet. As such, it closely aligns IT with business, which brings up the important issue of accountability that is largely overlooked in traditional IT.

In a business context, accountability encompasses transparency, responsibility, responsiveness and willingness for assuming liability. Applying that concept in a service computing context, it should mean a clear disclosure of service obligations; faithfully honoring disclosed obligations, or otherwise assuming liability for the unsatisfactory performance of the obligations.

A comprehensive study of the accountability literature reveals that the vast majority of researchers in the IT community share a quite different view on accountability than their counterparts in the business community. Most researchers in IT tend to take some aspects of quality of service (QoS) or architectural concerns, such as security, provenance and auditability, as accountability, missing the crucial components of disclosure, obligation fulfilment monitoring and liability assignment that concern business more frequently.

This discrepancy in understanding can cause a significant gap between business's expectations of accountability and the actual accountability capability of an IT system. While this gap may not manifest as a major concern in the traditional IT environment, it will exert a serious negative impact on the development of service computing, since a major theme of service computing is the close alignment of IT and business.

Three main objectives are set in this thesis. The first is to raise awareness of the

accountability gap from both the conceptual perspective and the actual architectural implementation perspective. The second is to clarify the confusion on the topic of accountability in the service computing industry and, more importantly, to lay down a foundation to strengthen accountability in service computing. The third is to describe approaches for building an advanced service accountability mechanism, aimed at automating the accountability processes in a truly service-oriented environment.

Accordingly, the thesis is structured in a way that progressively meets the above objectives. The Introduction chapter focuses on achieving the first objective, and sets the theme for the rest of the thesis.

The second objective, is achieved through two steps. The first step is to clarify the confusion around the topic of accountability through a thorough analysis of the existing accountability literature. The second step involves a series of tasks for laying down the service accountability foundation by proposing a service accountability framework, addressing the accountability weaknesses of the current Service-Oriented Architecture.

The third objective, is achieved through three different approaches. The first approach is to represent a service contract using semantic web technology. The second approach is to represent a service contract using dynamic logic and process algebra techniques. The third approach provides a decentralised service contract management scheme based on the blockchain technology, taking advantage of the irreversibility and tamper-proof features of the blockchain, and presents a scheme for service contract disclosure and obligation tracking.

We hope that through achieving these three objectives, a solid foundation can be laid for strengthening accountability in service-oriented environments, which addresses business's concerns on service accountability and boosts business's confidence in fully embracing cloud services.

Acknowledgments

First of all, I would like to express my sincere appreciation to my principal supervisor A/Prof. Yan Wang, Prof. Mehmet A. Orgun and my associate supervisor Prof. Jian Yang for their invaluable work in helping me to complete an eight-year journey of (part-time) PhD study. They have consistently provided me with guidance and encouragement, which are critical for me to overcome the challenges encountered during the long study journey. Their dedication spirit and rigorous work attitude set a great role model for me to follow, and helped my growth in research capability. Without their rigorous supervision work, I would have not been able to reach the destination of my research work.

I would like to also thank Prof. Kwei-Jay Lin, who provided insightful guidance on the topic of accountability, and taught me some valuable research skills from which I am still benefiting today.

In addition, my colleagues have helped me develop this work.

I wish to thank my former colleagues Dr. Christopher J. Pavlovski, Dr. Jing Mei and Mr. Christopher De Vaney for their valuable contributions on the collaboration of the service accountability topic, and their great co-authoring work in some of the published papers.

I also wish to express my thanks to my peers in PhD study, Dr. Guanfeng Liu, Dr. Lei Li, Dr. Haibing Zhang, Mr. Lie Qu, Dr. Xiaoming Zheng and Mr. Bin Ye, for their kind support and the friendships during these years.

Many thanks to the staff in the Department of Computing for their help. I would like to thank Donna Hua, Camille Hoffman, Sylvian Chow, Melina Chan, Fiona Yang and Jackie Walsh for their warm support.

Most important of all, I would like to thank my family. My mother, Lifeng Xiang, my wife, Annie Zou, and my lovely daughter and son, Beverley Zou and Skyler Zou. Their unconditional love and unwavering support are the inexhaustible source of energy, propelling me to always achieve the next milestone in my life. I wish to thank them for all the support that they have given me during my eight-year PhD study.

Publications

This thesis is based on the research work I have performed with the help of my supervisor and associate supervisor and other colleagues during my PhD program at the Department of Computing, Macquarie University between 2008 and 2016. Some parts of my research have been published in the following papers:

[1] **Jun Zou**, Yan Wang and Mehmet A. Orgun, A Dispute Arbitration Protocol Based on a Peer-to-Peer Service Contract Management Scheme, 23th IEEE International Conference on Web Services (IEEE ICWS 2016, research track, acceptance rate = 13% **Received the Best Student Paper Award**), pages 41-48, June 27-July 2, California, USA (**CORE2014 Rank A**).

[2] **Jun Zou**, Yan Wang and Mehmet A. Orgun, Modeling Accountable Cloud Services Based on Dynamic Logic for Accountability, International Journal of Web Services Research (IJWSR), 12(3):48-77, 2015.

[3] **Jun Zou**, Yan Wang and Mehmet A. Orgun, Modeling Accountable Cloud Services, 21th IEEE International Conference on Web Services (IEEE ICWS 2014, application track), pages 353-360, June 27-July 2, New York, USA (**CORE2014 Rank A**).

[4] **Jun Zou**, Christopher J. Pavlovski, Accountability in Enterprise Mashup Services, Advances in Software Engineering, 2013(8):1-13, 2013.

[5] **Jun Zou**, Mei Jing and Yan Wang, From Representational State Transfer to Accountable State Transfer Architecture, 17th IEEE International Conference on Web Services (IEEE ICWS 2010, research track, acceptance rate = 18%), pages 299-306, July 5-10, Miami, USA (**CORE2014 Rank A**).

[6] **Jun Zou**, Yan Wang and Kwei-Jay Lin, A Formal Service Contract Model for SaaS and Cloud Services, IEEE International Conference on Service Computing (IEEE SCC 2010, research track, acceptance rate = 18%), pages 73-80, July 5-10, Miami, USA (**CORE2014 Rank A**).

[7] Kwei-Jay Lin, **Jun Zou** and Yan Wang, Accountability Computing for E-society, Advanced Information Networking and Applications (AINA 2010), pages 34-41, April 20-23, Perth, Australia (**CORE2014 Rank B**).

[8] **Jun Zou**, Christopher De Vaney and Yan Wang, A Metamodeling Framework to Support Accountability in Business Process Modeling, United Information Systems Conference (UNISCON 2009), pages 539-550, April 21-24 Sydney, Australia.

[9] **Jun Zou**, Christopher J. Pavlovski and Yan Wang, A Quantitative Service Accountability Model, IEEE International Conference on e-Business Engineering (IEEE ICEBE 2009), pages 391-396, October 21-23, Macau, China (**CORE2014 Rank B**).

[10] **Jun Zou**, Christopher J. Pavlovski and Yan Wang, A Disclosure Framework for Service Accountability in SOA, IEEE International Conference on e-Business Engineering (IEEE ICEBE 2008), pages 437-442, October 22-24, Xian, China (**CORE2014 Rank B**).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As early as the twentieth century B.C.E., the Code of Hammurabi already held the builder of a house accountable to the house owner [143]. Since then, written history indicates that accountability is the cornerstone for ensuring the proper functioning of a business and a society. In cases where accountability is compromised, scandals and loss would inevitably occur, as was witnessed in the first decade of the twenty-first century when the likes of Enron and Worldcom made the headline news, and later on with the overwhelming loss caused by the global financial crisis. There is a major theme that is permeating the early twenty-first century, which is the strong call for accountability in business from the general public, as people start to demand answers to questions such as "what are the truths that have not been told?", "who are accountable?" and "how do we prevent those incidents from happening again?"

Traditionally, accountability is mainly a business or social concern and fewer people pay attention to the accountability issues associated with technology. As information technology (IT) development increasingly becomes a major driver for economic growth and social transformation, the impact of accountability in IT becomes increasingly significant. Since accountability in IT has not been systematically investigated in the literature, it is chosen as the research direction of this thesis. More specifically, this thesis will focus on the accountability mechanism for service computing, which is the direction of the future IT.

First, we need to understand what *accountability* really means, and what problems exist for accountability in IT. Then we need to explore what needs to be done in order to build accountability mechanisms in a service computing environment and the associated challenges that they may have.

1.1 Problem Statements

1.1.1 What is Accountability?

Although *accountability* has become a focal concern of the general public, there is no commonly agreed definition for the term “accountability”. Oakes and Young note that the *accountability* concept is broad and difficult to characterise precisely [145]. They find that while some researchers define *accountability* as “answerability” or an “obligation to account for how well resources used to meet specified outcomes”, others argue that *accountability* is either credit or blame or corporate scapegoating [145]. In this thesis, the definition of accountability at Wikipedia is adopted as the basic definition, which references Schedler, who defines accountability as the “obligation to inform other parties about actions and decisions, or justify them and to be punished in the case of misconduct” [166].

1.1.2 Problems Exist for Accountability in IT

Starting from the latter half of the twentieth century, the Digital Revolution, also known as the Third Industry Revolution, has brought sweeping changes to the economy and our society, and marked the beginning of the Information Age. As the key driver of the Digital Revolution, Information Technology (IT) becomes a key differential tool that most people use to gain competitive advantage. One worrying sign is that accountability in IT has been largely overlooked by society, as most people are willing to forgo accountability in favour of early deployment of drastic new technologies. In 1994, Nissenbaum voiced her concerns on accountability in an increasingly computerised society and asserted that accountability is systematically undermined [143]. She suggests that a combination of factors working in unison leads to the erosion of accountability in IT, such as an extremely narrow understanding of the accountability concept, a set of misplaced assumptions about the capabilities and shortcomings of IT systems, and a willingness to accept that the vendors of computer systems do not need to be answerable for the impacts of their products [143].

Nissenbaum's findings highlight two problems: the first is that the accountability topic has not been treated rigorously in the IT literature, and as a result there is a general misconception about accountability in IT; the other is that the accountability concern has not been widely accepted as a formal architectural concern in IT literature, contrasting to other concerns like security, integrity, performance and scalability. Consequently, people have no alternative other than accepting IT products that may compromise accountability.

Since then, the pace of IT technology development has further accelerated. Now we are in an era where IT is truly ubiquitous. It is evident that people are increasingly connected through social networking sites; billions of sensors are embedded in objects, making the world more interconnected, instrumented and intelligent; and computing resources can be virtualised as services and then be massively delivered and consumed on the Internet. IT is entering a cloud computing era, where the boundary of business and IT is blurring, i.e., business service is delivering through the IT platform, whereas IT service becomes a kind of business service. IT is the key enabler for a global integrated economy. Driven by the rapid development of IT technology, the world economy is moving towards a service-centric economy. [29]

Unfortunately, in contrast to the rapid pace of technological advancement, accountability consideration in IT rarely changes and largely stays the same as a decade ago. The consequence is quite obvious as the ever increasing fraud, spam and security attack incidents in the Internet [91], and the huge financial loss caused by a large number of IT project failures in the industries [98]. As IT plays an increasingly vital role in today's society and becomes deeply embedded into the fabric of business, the practice of purely focusing on technology, while overlooking accountability, in the IT industry will inevitably become unsustainable. The call for establishing an accountability mechanism in cloud services is becoming more compelling in the cloud computing era.

Traditionally, accountability in service is achieved through the enforcement of a legal and paper-based contract. In a cloud service context, using a paper-based contract is no longer effective. The current practice is for service providers to publish a terms and conditions page and a text-based SLA for their offerings on their web-site like Amazon's S3 does. In its plain-text form, a web-enabled paper-based contract can neither be interpreted by software agents, nor be used as a basis for monitoring the execution of a contract. Although policy management tools, such as Ponder and KAoS, can be adopted to represent obligations in a formal manner, their strengths lie in specifying policies on fine-grained objects, lacking generality [153] and an overarching process view that is required in dealing with service participants' obligations in service collaborations. While Service Level Agreement (SLA) is an extensively researched topic and it can be represented formally by existing approaches, in essence, the service level only covers non-functional requirements, missing the crucial functional requirements for business. Thus, existing approaches neither enable the disclosure of service obligations, nor allow software agents to decide which party is responsible for what action, and which party is liable for what result. This is evident in today's cloud market, where there are no formal policy-based or SLA languages used in representing service contracts. Hence the consumer has no effective means to detect service obligation violation, and the service provider can hardly be held accountable. As such, currently the accountability of cloud services on the market is a serious concern. This may become a major obstacle for enterprise customers to take up those cloud services.

1.1.3 A Motivating Example in Cloud Service

We here use a traditional storage service and a typical cloud service - Amazon S3 storage service as an example to compare and contrast different accountability issues. Suppose that a service provider offers a data storage managed service to enterprise consumers. The regular process for this kind of off-line service provision is described as follows:

1. The provider prepares a statement of work (SOW) and a service-level agreement (SLA) based on the requirements submitted by the consumer. The SOW describes the rights and responsibilities of the provider and the consumer respectively, specifying each party's permitted activities, obligated activities including deliverables, and acceptance criteria. For instance, the provider's permitted activities may include delivering data in either plain or compressed format; obligated activities can be storage and retrieval of data objects while making sure that the data are not lost, damaged or leaked to other people. The SLA stipulates the committed service levels for the service activities and deliverables, i.e., a 99.9% availability of the overall storage service. Sometimes penalty clauses may be attached if an obligation is violated, for example, not achieving the 99.9% of the availability target.

2. The provider negotiates with the consumer on the SOW and SLA. If a deal is reached, normally an overarching contract is signed between the provider and the consumer, specifying the general terms and legal clauses, plus company specific conditions. The SOW and SLA are normally attachments of the contract.

3. The provider starts the delivery of the service once the contract is in effect. When issues arise during the execution of a contract, the provider and consumer can meet face-to-face to resolve the issues, with legal proceedings as the last resort for dispute resolution.

However, in a cloud service setting, accountability in general is much harder to maintain. In the S3 storage cloud service case, Amazon provides only two web-based contract documents. The first one is the "AWS Customer Service Agreement", which is effectively the overarching legal contract between Amazon and its customers [5]. The other contract document is the S3 SLA, which is a text document specifying the

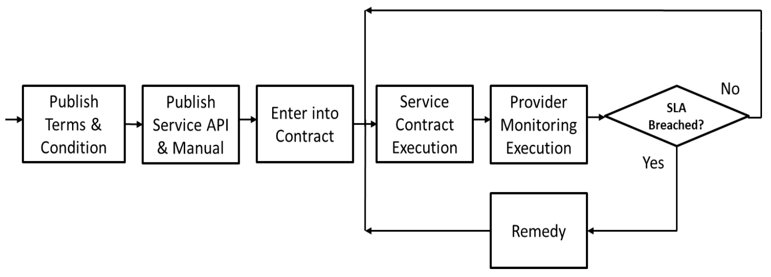

Publish Terms & Condition
Publish Service API & Manual
Enter into Contract
Service Contract Execution
Provider Monitoring Execution
SLA Breached?
No
Yes
Remedy

Figure 1.1: Current Cloud Service Contract Execution Process

committed availability target of 99.9%, and the service credit offering to the customer if the target is not achieved [4].

The contract specification and execution process of S3 can be generalised and is described in Fig. 1.1. Initially, a service provider publishes a text-based service terms and conditions; then publishes the service APIs and the text-based manuals. When a consumer accepts the terms and conditions, the provider and the consumer enter into a legal contract. The next stage is contract execution. Normally, only the service provider can monitor the service execution (in the AWS' case, CloudWatch is used as a monitoring tool). If the service does not meet the SLA (in S3's case, 99.9% availability is not achieved), the service provider provides a remedy (in AWS's case, offering the consumer some service credits) and the contract execution cycle goes on.

The major accountability problems in the S3 service in particular and other cloud services in general are listed below:

Prob1: The cloud service contract does not have an SOW component as traditional services do. Consumers need to thoroughly check service provider's technical documents such as *User Guides* or *API References* in order to understand what functions the service offers. Without an SOW, there is no clear service obligation statement disclosed by the provider. For instance, Amazon's customer agreement only defines the customer's responsibilities, plus indemnifications and disclaimers that are mainly protecting the provider's own interests rather than those of customers;

Prob2: There is no formal representation of the contract terms and conditions, SLA and obligations in the current cloud service contracts. This prevents customers from using a software agent to search and match services based on accountability requirements;

Prob3: There is no precise process defined for service execution and thus it is not possible for customers to monitor the service behaviour automatically during service consumption; and

Prob4: There is no mechanism to detect obligation violation and identify which party is at fault during service execution.

1.1.4 What Needs to be done for Building Accountability Mechanisms in Cloud Service

In order to establish an accountability mechanism in cloud service, the following tasks need to be accomplished:

1. Defining a cloud service model that encompasses the accountability properties, in particular, the service contract property that is ignored in the current literature. Most cloud service models in the current literature only focus on modelling the interface and implementation aspects of a service, missing the critical component of the contractual relationship between the service provider and the service consumer. Without the service contract component, there is in no way to hold service participants accountable for their service contractual obligations.

2. Presenting a formal representation of a service contract that facilitates disclosure of service obligations, service auto-discovery and matching, and assignment of liability. One important aspect of accountability is to ensure transparency of the obligations of service participants. In a cloud computing environment, services are delivered and consumed automatically with virtually no human intervention. Therefore, it is important that the service contract representation can be

interpreted by machines, rather than only by humans. So service discovery and matching can be done automatically based on a disclosed service contract. Furthermore, the service contract representation should allow easy identification of which party fails to meet its obligations.

3. Providing an approach for tracking and monitoring of the execution of a service contract that is objective, fair and accurate. After a service contract is established, a service contract can be executed multiple times. Current practice is for each party to monitor the execution in their own environment, which can lead to subjective and inconsistent results. An accountable service execution environment should provide a means to avoid the conflict of interest when it comes to tracking and monitoring of the obligation fulfilment of each service participant.

4. Providing an arbitration protocol to resolve a dispute if it arises during the execution of a service contract. Currently, if a dispute arises during the execution of a service contract, there is no accurate way to judge which party is at fault, as each party may provide their version of logs that support their claims and arguments. Thus we need a protocol that can resolve disputes in a fair and accurate fashion.

While Fig. 1.1 depicts the current cloud service contract specification and execution process, Fig. 1.2 illustrates an accountable approach, where the first step is for the service provider to specify the obligations in a formal language; then validate them to make sure they are free of contradictions. Next, for each obligation, it is decomposed into a collaboration process that specifies the interaction behaviours of both the provider and the consumer. Then the provider can publish service APIs, manuals and, more importantly, the obligations and collaboration processes specified in formal languages. The published obligations and collaboration process are effectively the cloud service contract. Once the consumer accepts the obligations and the processes, the provider and the consumer enter into a legally binding contract. During the execution of the service contract, both the provider and the consumer can monitor the contract

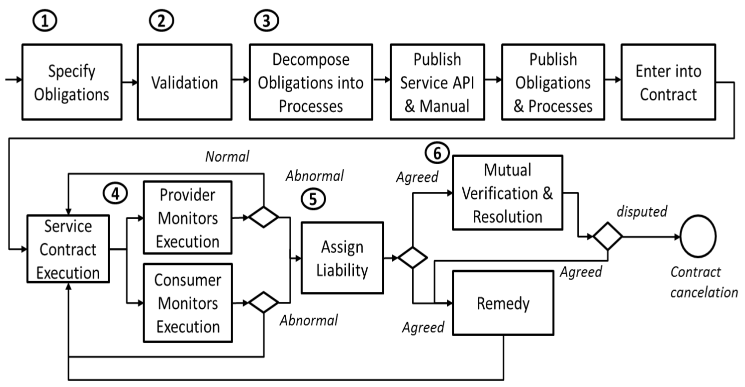

1
2
3
Specify Obligations
Validation
Decompose Obligations into Processes
Publish Service API & Manual
Publish Obligations & Processes
Enter into Contract
Normal
Abnormal
4
5
6
Agreed
Provider Monitors Execution
Consumer Monitors Execution
Service Contract Execution
Assign Liability
Mutual Verification & Resolution
disputed
Contract cancelation
Agreed
Remedy
Abnormal
Agreed

Figure 1.2: Accountable Cloud Service Contract Execution Process

execution respectively based on the published obligations and the collaboration process model. If a deviation to the contract in execution is spotted, then they can decide which party is liable, based on the published contract. In cases where the provider and the consumer do not agree with each other on the liability assignment, a mutual verification and resolution step will be needed. If the discrepancy cannot be reconciled at the end, the contract may be ended prematurely; otherwise a remedy will be applied by the liable party and the contract execution cycle continues.

1.1.5 Challenges for Accomplishing the Tasks

To accomplish the above tasks, the following challenges need to be overcome.

1. Traditionally, service modelling focuses on model elements in service interface and implementation, abstracting out other details like accountability concerns. Stemming from a business and management context, the concept of accountability tends to be qualitative in nature. Thus how to precisely define and quantify it in a cloud service context remains a challenge.

2. In order to meet the objectives of service contract obligation disclosure, automatically discovering and matching services based on service contracts, the representation of a service contract must be interpretable by machines, which

means that the representation language should provide both formal semantic as well as syntactic capabilities. Therefore, the language's underpinning logic system must have sufficient expressive power; in the meantime, it must be decidable with computation completeness. As obligation in a service has both static and dynamic properties, it is a challenge to find an existing language or logic to fully meet the representation requirements.

3. In a cloud environment, monitoring the execution of a service contract can be a challenge. Neither the service provider's monitoring log nor the service consumer's one can be taken as the source of truth. An obvious solution is to use a trusted-third party (TTP) to provide centralised monitoring. However, this model also has issues associated with centralisation, i.e., bias, performance bottleneck and single-point of failure. On the other hand, a decentralised monitoring solution also has challenges for establishing trust and consensus.

4. The same challenge as item [3] also exists in dispute arbitration. Apart from the issues associated with centralised and decentralised models, how to incentivise the arbiters and encourage them to make honest decisions is also a challenge.

1.2 Thesis Contributions and Roadmap

1.2.1 Contributions of the Thesis

Today, accountability is the most important concern in ethics, governance and business. However, the linkage between accountability and technology is still yet to be appreciated by most people, especially the technical community. This thesis aims at first raising the awareness of accountability in service computing, highlighting the need for an automated, intelligent way to build accountability mechanisms in the cloud era. Secondly, it aims at laying down a foundation for building accountability mechanisms in a cloud service environment. To achieve these two objectives, we not only need to overcome many technical challenges but, more importantly, we need to change the way we view a cloud service. The traditional view of a cloud service is a pre-compiled computer program that is static, reactive and lacks intelligence, whereas we argue that a cloud service is driven by a service contract, it is dynamic, proactive and may be armed with artificial intelligence. While this thesis focuses on the context of cloud services, the accountability principles can certainly be extended to other online services in general. Therefore, the contributions of the thesis can be summarised below:

1. The first contribution is to distil the essence of the accountability concept through investigation in both the management discipline as well as the IT discipline, and present a precise definition for accountability in IT. We have observed that there are three levels of accountability work in IT: the technical protocol level, the architectural level and the governance level. We have pointed out that, thus far, there are two crucial aspects of accountability that have not been addressed in the IT literature, which are the disclosure and service contract management mechanisms.

2. The second contribution is to propose a novel service accountability model, taking into consideration of the accountability requirements. The key differentiation of the service accountability model is that it incorporates the service contract concept into the service model and positions the service contract as the first class model element in a service.

   (a) We outline the service contract properties, and present a new approach that models a cloud service as a proactive system, rather than as a reactive system like that in the traditional modelling approach. The proactive system is concerned with the actors who conduct the activities and the exceptions which occur during the action execution, plus the causality behind these. These model elements are missing in the traditional reactive system modelling approach. Due to the rapid growth of cloud service adoptions and the emergence of more intelligent agent applications, the issue of accountability arising from those proactive systems is becoming ever more significant than before;

(b) We analyse the accountability management requirements for cloud services and define a formal construct for a pro-accountability service contract model, proposing unique concepts, such as service contract execution and action evidence, that are not seen in other service contract models;

(c) We present a semantic-web approach to modelling a service contract. The semantic model adopts the decidable OWL-DL, coupling with the enhanced action semantics and DL-Safe SWRL rules to represent the service contract construct, namely OWL-SC. We also propose a novel approach to map OWL-SC action semantics to a coloured Petri-nets model, namely SC-CPN, and thus enable visual modelling, validation and simulation of an action model in OWL-SC;

(d) We also present an alternative approach to modelling a service contract based on a refined version of dynamic logic - Dynamic Logic for Accountability (DLA). DLA can represent the deontic semantic of a contract, which is missing in the OWL-DL language;

(e) A graphical notation based on a reduced version of BPMN2.0 has been proposed so that contract obligations can be further decomposed into collaborative activities between the service provider and the consumer;

(f) An action dependency chain has been proposed for obligation violation detection, causality reasoning and liability assignment. In addition, an Obligation Flow Diagram (OFD) has been proposed to allow easy conflict resolution and consistency checking in the service contract model;

(g) Finally, we propose a novel Accountable Process Algebra (APA), which extends the traditional process algebra to a form of process algebra suitable for proactive systems like cloud services. APA allows analysis of the execution behaviour of a cloud service contract based on an algebraic approach, which produces a more concise model to reflect the dynamics of service accountability during the provision as well as the consumption of a cloud service.

3. The third contribution is that we have proposed a centralised architecture for monitoring the execution of a cloud service contract.

   (a) We outline the architectural principles and decisions for enabling accountability in a cloud service environment.

   (b) Secondly, guided by those principles and decisions, we propose a novel Accountable State Transfer (AST) architecture with an accountable state transfer protocol to enable service accountability, yet retain scalability of REST architecture. The new architecture seamlessly integrates service contract semantics into the traditional syntactic-based REST services.

   (c) Thirdly we apply the formal OWL-SC service contract model to design a Credit Check domain specific service contract with a hybrid reasoning mechanism that leverages strengths from formalisms like DL, Rules and traditional programming language. The hybrid reasoning mechanism provides capabilities like temporal reasoning and negation as failures that are not found in normal DL and SWRL. Moreover, it separates reasoning in the design-time stage and runtime stage, taking into account both the expressiveness and computational complexity of the underlying logic formalisms.

   (d) Lastly we provide a prototype implementation for a Credit Check service that demonstrates the practicality of AST architecture, proving that the new AST architecture can be implemented with existing products and technologies.

4. The fourth contribution is that we have proposed a distributed accountability mechanism that fits in the Internet environment and avoids the issues associated

with centralisation like cost, scalability, resilience, fairness and objectivity [58, 137];

(a) We present a service contract management scheme to enable automatic publication of a service contract, automatic service discovery and selection. This improves transparency and accountability in service computing by eliminating the information asymmetry.

(b) We propose a service blockchain that can be used as an anti-tamper public "activity ledger" for recording the interactions between the service provider and the consumer, enabling monitoring of contract obligation fulfilment in an open and objective environment.

(c) We propose a novel dispute arbitration protocol that uses anonymous Proof of Work (POW) miners acting as arbiters to arbitrate a service contract dispute. Coupling with the commitment scheme and the majority function techniques, the protocol is designed to be fair, accurate and free from the influence of any authorities.

(d) Finally, we study the dynamics of the key parameters of the arbitration protocol through three experiments and various case studies. The optimal parameter settings are studied, striking the balance of fairness, accuracy and sustainability of the arbitration protocol.

1.2.2 Roadmap of this Thesis

This thesis is structured as follows.

Chapter 2 presents a comprehensive literature review of accountability, starting from a management, governance perspective, then moving onto the IT technical and architecture perspective, and finally centring around the overlaid area of IT and business, i.e. accountability in cloud service as a result of the establishment of a service contract. This chapter examines the existing work and highlights the key aspects of service accountability that are missing in the current literature.

Chapter 3 presents a foundational accountability model, providing precise definitions for basic accountability concepts in IT service, outlining the service accountability processes, proposing a quantitative accountability measurement method and designing a service accountability framework to address the elementary accountability requirements. Part of this chapter is based on our papers published at ICEBE2008 and ICEBE2009 (please refer to [9] and [10] on page xii).

Chapter 4 proposes an advanced service accountability approach based on a semantic web approach. In order to automate the process of disclosure of a service contract, and service discovery and matching based on a disclosed contract, the contract specification language must have both semantic and syntactic capabilities. We use OWL-DL to formally specify a service contract, building a service contract model that is interpretable by machines. We also propose a graphical model SC-CPN based on a coloured Petri-net to allow model visualisation and validation. Finally, we apply the model to extend Representational State Transfer (REST) architecture to Accountable State Transfer (AST) architecture, augmenting the mainstream SOA architecture implementation with an accountability mechanism. The AST architecture is based on a centralised service contract management approach that facilitates service obligation disclosure, obligation tracking, and action justification in a stateless service environment. This chapter is based on our papers published at ICWS2010 and SCC2010 (please refer to [5] and [6] on page xi and page xii).

Chapter 5 presents an alternative advanced service accountability model based on an algebraic approach. The approach brings in deontic semantics and process algebra to the service contract model, enabling a contract representation that is more closely aligned to the contract in the real world, and allowing more efficient service contract disclosure and contract model validation. This chapter is based on our papers published at ICWS2014 and IJWSR 2015 (please refer to [2] and [3] on page xi).

Chapter 6 proposes a distributed contract monitoring approach and a peer-to-peer dispute resolution protocol based on the blockchain technology underlying the bitcoin network. This chapter is based on our paper published at ICWS2016 (please refer to

[1] on page xi).

Finally, Chapter 7 concludes the work in this thesis and discusses some directions for future research opportunities.

Chapter 2

Literature Review

Traditionally, accountability is a term that is mainly used in management and law literature, but has received considerably less attention in an IT context. In recent years, while IT exerts an increasingly important role in society and business, a number of scholars across different fields have studied accountability in IT and presented various thoughts and arguments in the literature.

In this chapter, the literature review on the above aspects is organised as follows:

- Section 2.1 introduces the general concept of accountability in literature.
- Section 2.2 introduces the existing accountability work in the IT literature.
- Section 2.3 reviews the existing work on service contract, which is a key enabler for accountability in service computing.
- Section 2.4 summarises the literature review on accountability.

2.1 Accountability in Management Literature

2.1.1 General Concept of Accountability

Accountability has been an extensively researched topic in the management literature. While Oakes and Young note that the accountability concept is broad and difficult to characterise precisely [145], Johnson and Mulvey find that at least four meanings can be distinguished: *fault*, i.e., what went wrong; *causality*, i.e., what caused the problem; *liability*, i.e., what is the cost of the fault; and *role assignment*, i.e, who is to blame [97].

Kaufman believes that technologies of accountability, from record keeping and simple reporting to auditing and oversight have become essential ingredients in the construction of any organised public or private endeavour where transparency, answerability and responsiveness are deemed necessary for the sake of efficiency and effectiveness [101, 102].

Dubnick and Justice provide a comprehensive review on the topic of accountability in [54]. They distinguish accountability as a word ($accountability_w$) versus accountability as a concept ($accountability_c$). The $accountability_w$ resembles the meaning defined in the Oxford English Dictionary, which is "the quality of being accountable; liability to give account of, and answer for, discharge of duties or conduct; responsibility, amenableness (to a person, for a thing) [53]". The $accountability_c$ concept that has emerged from the literature is meaningful in six interrelated contexts, as listed below.

1) **Cultural Frame**: Accountability is a reflection of legitimised "certainties" within a community [83];

2) **Institutional Frame**: Accountability is manifested as rules, norms and grammars through which authority is "controlled" in order to render it "appropriately" exercised [119, 103, 52];

3) **Social Transactions**: Accountability emerges as a way for individuals to relate to one another - an ongoing process of account-giving, excuse-making and account-taking that is fundamental to the development or maintenance of trust [19, 48];

4) **Organisational Frame**: Accountability is the formation ("enactment") of infor-

mal and formal mechanisms for dealing with expectations and uncertainty [39, 195];

5) **Task Environmental**: In a complex task environment, accountability is a means for managing an otherwise chaotic situation [162, 30]; and

6) **Social Psychological**: Accountability has emerged as a means by which we "construct ourselves" and develop identities [181, 81].

The research from Dubnick and Justice reveals that the accountability concept in literature can be complex with multifaceted meanings.

On the other hand, Schedler provides a definition that succinctly captures the essence of accountability: "A is accountable to B when A is obliged to inform B about A's (past or future) actions and decisions, or justify them and to be punished in the case of misconduct" [166]. Schedler situates accountability in a context of relationship, emphasising the obligation for disclosure and the liability for misconduct. The relationship is driven by social, contractual, hierarchical or other factors according to Gibbins and Newton [70].

In [158], the authors outline eight types of accountability developed based on the relationship formed in a specific context, namely: moral, administrative, political, managerial, market, legal/judicial, constituency relation, and professional. In each type of accountability, the accountable party is obliged to perform certain actions in order to meet the other party's expectation.

Koppell argues that the term "accountability" has five dimensions as listed below [107]:

1. Transparency: did the organisation reveal the facts of its performance?

2. Liability: did the organisation face consequences for its performance?

3. Controllability: did the organisation do what the principal desired?

4. Responsibility: did the organisation follow the rules?

5. Responsiveness: did the organisation fulfil the substantive expectation?

In [9], the authors conducted extensive research on the management literature, and provided a definition of accountability in the context of service performance. This definition is outlined in the table below.

Table 2.1: Accountability Definition for Service Performance

Accountability refers to the obligation a person, group, or organization assumes for the execution of authority and/or the fulfillment of responsibility. This obligation includes: • Answering – providing an explanation or justification – for the execution of that authority and/or fulfillment of that responsibility, • Reporting on the results of that execution and/or fulfillment, and assuming liability for those results. • Assuming liability for those results.

This definition resembles Schedler's definition in terms of obligation, transparency and liability. Together with Koppell's other accountability dimensions, i.e. controllability, responsibility and responsiveness, they form a basis for analysing the accountability attributes in business processes and practices.

2.1.2 Accountability through Quality Management

The quality movement led by Deming, Juran, Shewhart and Feigenbaum in the twentieth century reflected the public demand for accountability for the quality of goods and services. On the other hand, the increasing market competition also put intensive pressure on business to continuously improve quality. Prominent methods such as Total Quality Management (TQM), Quality Function Deployment (QFD) and Six Sigma have emerged, advocating customer focus, quality design and a statistical approach to quality control. These methods promote the controllability and responsiveness dimensions of accountability. According to Brandon, the cornerstones of TQM are continuous improvement, employee empowerment, and customer focus, whereas the focus on QFD is on comprehensive quality design and taking into account both the customer-stated and unstated requirements. On the other hand, the emphasis of Six Sigma is to adopt a statistical approach to quantify process quality improvement [27]. Insofar as these methods can be mandated in the organisation, they can be effective in improving process quality, customer satisfaction and, ultimately, accountability. However, evidence in recent years suggests that these methods gradually lose their effectiveness due to environmental change [17]. A notable example is Toyota's global recall of eight million cars that were affected by unintended acceleration issues during 2009-2010, despite the company having the reputation of being the role model for implementing quality management best practices in business [168]. One obvious reason is that in an environment where business has increasing dependency on out-sourcing, off-shoring or the global supply-chain, it is virtually impossible to mandate those quality management processes across different organisations with different cultures in multiple geographic regions.

2.1.3 Accountability through Regulatory Compliance

Traditionally, the areas that most demand accountability are in the public sector, such as governments and schools, or institutions that may have a critical impact on peoples' health and safety, such as hospitals and pharmaceutical companies. One example is the enactment of The Health Insurance Portability and Accountability Act of 1996 (HIPAA) in the United States.[1]

Recently, accountability has become increasingly important in the private sector since the fallouts of some big institutions in the early 2000s. Government and regulatory bodies have created regulations such as Basel II[2], Sarbanes-Oxley Act[3] and the Anti-money Laundering (AML)[4] to strengthen accountability in financial institutions

[1]https://www.ihs.gov/hipaa/
[2]http://www.bis.org/publ/bcbsca.htm
[3]http://www.soxlaw.com/
[4]http://www.finra.org/industry/aml

and private enterprises, in responding to several high-profile scandals and large scale accountability breakdowns in financial institutions and private enterprises. These regulations impose stringent accountability requirements on various aspects of the business operations. Regulatory compliance is an effective way to enforce accountability. The down side is that it is slow to respond to change, and normally the regulations are created after the event.

2.1.4 Summary of Accountability in Management Literature

In summary, the accountability concept in management literature has the following attributes: obligation, transparency, liability, controllability, responsibility and responsiveness. Quality management and regulatory compliance are the main approaches to strengthening accountability in business operations.

2.2 Accountability in IT Literature

2.2.1 Accountability Definitions

Accountability has received considerably less attention in the IT literature when compared to that in the field of management research. The meaning of the term accountability in IT appears to vary considerably and is dependent upon the context.

Eriksen comprehensively explores the notion of accountability for information and communication technologies [57], citing a general definition of the term accountability as: "responsible for giving an account (as of one's acts): answerable" or "capable of being accounted for".

In [202], based on the review of previous work from both the IT and the management literature, the authors propose an accountability meta-model, which has four elements: identity, roles, responsibilities and outcome. The authors also extend the accountability definition in Table 2.1 and provide a unified definition of service accountability, covering both business and technological perspectives as follows.

Table 2.2: Service Accountability Definition

Accountability in services refers to the obligation that several persons, groups, or organizations assume for the execution and fulfillment of a service. This obligation includes: • answering, providing an explanation or justification, for the execution of that authority and/or fulfillment of that responsibility; • full disclosure on the results of that execution and/or fulfillment; • undeniable liability for those results (non-repudiation); and • obtaining trusted agreement of accountability from all entities involved in the service who in turn are bound to the obligations set out above.

In this definition, answering, providing explanation, and full disclosure of results are all relating to disclosure, which corresponds to what Schedler refers to as the obligation of A informing B about actions and decisions, or justifying them, or to the term transparency in Koppel's accountability dimensions. Undeniable liability implies a technical aspect of non-repudiation in addition to the normal sense of liability, as in Koppel's accountability dimensions, which is interpreted as punishment for misconduct in Schedler's definition. One aspect of accountability not seen in definitions from Schedler, Koppel and Table 2.2 is illustrated by the term obtaining trusted agreement, which highlights the notions of trust and contract in service accountability. It also implies that service accountability in SOA is purely driven from a contractual relationship, rather than social or hierarchical relationships. In summary, this definition incorporates the key elements of service accountability: disclosure, obligation in trusted agreement and non-repudiation.

Generally, accountability in IT can be viewed from three levels. The first level is a technical protocol level accountability, which focuses on delivering certain accountability attributes by building the relevant technical capabilities in the technical protocols. The second level is an architectural level accountability which promotes automation of manual accountability tasks in business services and business processes. This may involve utilising the protocol level accountability capabilities to build up the accountability mechanisms on the IT architecture level. The third level is an IT governance level accountability, which emphasises the overall accountability of the IT organisation and its governance process. We briefly review accountability on each level. In addition, as service-oriented architecture (SOA) gradually becomes the mainstream IT architecture, as well as cloud computing emerging as the popular IT service model, we also review the accountability capabilities in the Web service protocol stack and existing work on accountability in cloud computing respectively.

2.2.2 Technical Protocol-Level Accountability

On the technical protocol level, traditionally researchers focused on accountability in eCommerce transaction. According to Kailar, accountability is "the property whereby the association of a unique originator with an object or action can be proved to a third party" [99]. The definition implies non-repudiation in an eCommerce transaction. Kailar also proposes a framework for the analysis of communication protocols that require accountability [99, 100].

In the current literature, many accountability protocols address the issue of non-repudiation and fairness in interactions amongst three parties: the sender, receiver and trusted third party (TTP) [108]. Examples of the fair non-repudiation protocols with online TTP are the Zhou-Gollman protocol [201] and the Certified Email Protocol [1]. There are also many fair exchange trusted-third party (TTP) protocols including on-line TTP [18, 171, 156] and off-line TTP [188] that can manage dispute resolution in e-Commerce. A further example that does not require a TTP is the Markowitch-Roggeman protocol [126]. The work on fair exchange is assessed based upon both the sender and receiver obtaining the expected items or neither receiving any additional information about the other's item [10]. Based on the Zhou-Gollman protocol, Robinson *et al.* propose WS-NRExchange to enable fair non-repudiable interactions with web services [161]. Fair non-repudiation protocols address the non-repudiation part of the accountability requirements on the message or communication level, but do not deal with the overall service accountability requirements.

In [43], Crispo and Ruffo propose a framework for the analysis of delegation protocols. Adopting the notion of "provability" from Kailar's definition of accountability, their framework allows analysis of how accountability is transferred (or kept) by the delegators when they transfer some of their rights to the delegate.

In [40], Corin *et al.* present a language that allows agents to distribute data with usage policies in a decentralised architecture. They propose two notions of accountability. The first notion, *agent accountability*, focuses on whether the actions of a given agent were authorised. The second notion, *data accountability*, expresses that a given piece of data was not misused. They design a logic underpinning the language that allows audited agents to prove their actions, and to prove their authorisation to possess particular data.

2.2.3 Architectural-Level Accountability

In a broad sense, the architectural-level accountability crosscuts a wide range of architectural concerns such as security and privacy, trust and reputation, quality-of-service (QoS), provenance, traceability, visibility, trustworthy, logging, auditability, service-level-agreement (SLA) performance measurement, reliability and recoverability, on the overall IT solution. Although some of the researchers tend to use the accountability term and some of the above terms interchangeably, the concepts behind them are quite different, even though they are interrelated. A study of the literature below examines the relationship between accountability and various concepts in IT architecture.

2.2.3.1 Accountability versus Security and Privacy

Accountability has frequently been treated as part of security and privacy. However, their meanings are quite different. In [25], the authors suggest that security can be de-

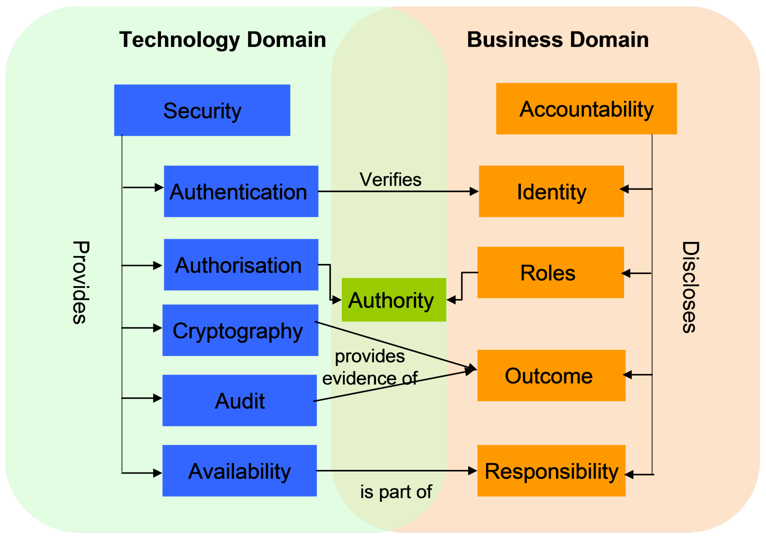


Figure 2.1: Accountability versus Security [116]

fined as a state that is free from danger and not exposed to damage from accidents or attack; or it can be defined as the process for achieving that desirable state. Privacy is about ensuring appropriateness in handling of customer's sensitive information [190]. On the other hand, accountability means obligation to inform other parties about actions and decisions, or justify them, and to be punished in the case of misconduct [166].

In the IT literature, the goal of security is to protect information assets. Security has three aspects: confidentiality, integrity and availability [167]. Conversely, the goal of accountability is to ensure justice in service or product consumption. It has both ethical and legal aspects. A system with poor security can undermine its accountability, however, a system with strong security does not necessarily have accountability. For example, an online illegal drug store may have strong security built in, but it does not have accountability, as it fails to disclose its licensing status for selling drugs [116].

Fig. 2.1 illustrates the relationship between accountability and security in the IT context. Security fits in the technology domain. The service provider provides the capabilities of authentication, authorisation, cryptography, audit and availability to protect information assets while ensuring legitimate access to those assets. On the other hand, accountability fits in the business domain. The service provider discloses identities, roles, responsibility and outcomes; and then delivers the service in accordance with what has been disclosed. Security and privacy supports accountability: authentication verifies identity; authorisation enforces authority of the roles; audit and cryptography support non-repudiation; and availability and privacy form part of the service provider's obligations under the service contract. Effectively, without security, the existence of accountability will be in doubt. However, security and privacy alone are not sufficient to guarantee accountability [116].

Ensuring security and privacy through information hiding and access control mechanisms becomes less effective while the whole industry is moving towards an Internet-based communication platform. In [190], Weitzner *et al.* raise the issue of information accountability. In contrast to most people who think that secrecy and upfront control of information will prevent the misuse of information, they promote instead that the use of information should be transparent so it is possible to hold individuals or institutions accountable for information misuse. The Wikileaks controversy in 2010 showed that most people were confused with the very topic of information accountability. Should the people who published classified information be punished? Or should the people who leaked the secret information to publishers be punished? Or should the people who misused the classified information be punished? People like US congress man William Delahunt even hold the view that "secrecy is the trademark of totalitarianism" whereas "Wikileaks provides the opportunity for ensuring transparency and openness", which is the essence of accountability [63]. Leading accountability researcher Simon Zadek calls this kind of accountability "disruptive accountability" and points out that disruptive accountability signals the failure of procedural accountability, in which rules of conducts were designed by power as the means to enforce accountability [199].

In [190], Weitzner *et al.* argue that debates over online privacy, copyright, and information policy questions have been overly dominated by the access restriction perspective. They believe that in a world where information is ever more easily copied and aggregated, and where automated correlations and inferences across multiple databases can uncover information even when it has not been explicitly revealed, accountability must become a primary means by which society addresses issues of appropriate use. They propose an alternative to the "hide it or lost it" approach, which is designing systems that are oriented toward information accountability and appropriate use, rather than information security and access restriction.

The goal of Weitzner *et al.* is to extend the web architecture to support transparency and accountability, so when information has been accessed, it is possible to identify who accesses what and determine whether the access is inappropriate. They name this extension Policy Awareness, which is a set of technical mechanisms that augment Web information with meta-data about provenance and usage policies, and provide automated means for maintaining that provenance and interpreting policies. Policy Awareness allows all participants with accessible and understandable views of the policies associated with information resources, provides machine-readable representations of policies in order to facilitate compliance with stated rules, and enables accountability when rules are intentionally or accidentally broken.

The Policy Awareness extension encompasses three capabilities: Policy Aware transaction logs, Policy Language Framework and Policy Reasoning Tools. A policy-aware transaction log will record data provenance and annotations about how the information was used, and what rules are known to be associated with that information, in addition to the traditional network and database transaction logs. A policy language framework is a common framework for describing policy rules and restrictions with respect to the information being used. A policy reasoning tool assists users to determine whether a piece of data is allowed to be used for a given purpose, providing a string of inferences permissible for use in a given context, depending on the provenance of the data and the applicable rules.

A critical consideration in enabling Policy Awareness is whether to re-architect the basic Internet and Web protocols, or build the capabilities on top of existing infrastructure. Weitzner *et al.* point out a possible latter approach, which uses a collection of accountability appliances that are distributed throughout the web and communicate using web-based protocols, both among themselves and with other web resources.

The accountability appliances would serve as proxies to data sources, mediating access to the data and maintaining provenance information and logs of data transfers, and presenting accountability reasoning in human readable ways through client facing services to browsers. The accountability appliances also allow annotation, editing, and publishing of the data and the details of the reasoning.

There are significant technological challenges in implementing the Policy Awareness extension at the moment. Some of the challenges are associated with reasoning over heterogeneous policy expressions, mainly in the dilemma of trading off the language expression power and the computation feasibility. In [77], Hanson *et al.* present an algebraic approach called Data-Purpose Algebra, which formalises the properties of the data, the organisations, and the trail of data transfers to guide automated inference in a privacy information accountability solution area.

Although the technology for implementing the Policy Awareness extension is still not mature, Weitzner *et al.* set out a new direction for greater accountability on the web.

2.2.3.2 Accountability versus Trust and Reputation

Like accountability, there are a lot of definitions of trust and reputation in the current literature. In [90], the authors have reviewed most of the published definitions of trust and reputation available. Here we adopt the widely cited definitions proposed in [140]:

Trust: a subjective expectation an agent has about another's future behaviour based on the history of their encounters.

Reputation: perception that an agent creates through past actions about its intentions and norms.

Based on these definitions, we can see that the trust and reputation concepts are related to the accountability concept, but with significant differences. Trust values and reputation ratings can be subjective; however, in some sense they can be the measurement of the degree of fulfilment and the quality of service. Hence they can be used as an indirect measurement of accountability [116].

In [36], Chun and Bavier suggest trust management in federated systems must also be accomplished with accountability. They present a layered architecture for addressing the end-to-end trust management and accountability problem. The accountability layer in their system provides two services, with one providing continuous monitoring on how trust relationships are being used by principals in the system, and the other providing periodic logging of monitoring data to create historical traces of how principals behave over time. The authors imply the accountability meaning as monitoring, logging on principals' behaviour.

2.2.3.3 Accountability versus QoS Properties

QoS properties, such as performance and reliability, are normally categorised as the non-functional requirements (NFR) of a system. These NFRs can be captured in a special form of contract called Service Level Agreement (SLA) as part of the responsibilities that the service provider needs to fulfil. The inconsistency between the quality of service in runtime and the disclosed SLA can contribute to the consumer's low expectation and poor perception of a service, and is therefore reflected in low trust values and reputation ratings. Hence monitoring SLAs is an important means that service providers use to manage accountability. However, SLA mainly targets non-functional requirements and normally does not have provisions on functional requirements. So similar to the security/accountability relationship, SLA management does not sufficiently address accountability [116].

In summary, integrity, security, privacy, trust, reputation and QoS monitoring all contribute to accountability. However, one important theme that is missing in those concepts is disclosure. Disclosure is the basis for accountability. Integrity, security, privacy and QoS are all service properties with specific target measures that need to be achieved by the service providers as part of their binding contracts with the service consumers. Service consumers will compare the metrics of those properties against what had been disclosed to them and form perceptions and expectations based on the result. These perceptions and expectations may feed the trust and reputation engines, which produce a trust value or reputation rating as indicators of the service providers accountability. While trust values or reputation ratings may put pressure on service providers to improve accountability, a systematic way to address the accountability issue in service is to build up accountability mechanisms in the service-oriented architecture.

2.2.3.4 Provenance and Traceability

Provenance means the process of tracing or recording the origin, derivation, history, ownership, or movement of a data object, also known as lineage [32, 33, 92]. In particular, Buneman *et al.* generalise the provenance problem as below.

Suppose a database (a view) $V = Q(D)$ is constructed by a query D applied to database D, and the question is finding the provenance of some piece of data d in $Q(D)$, *i.e.*, what parts of database D contributed to d?

Cui *et al.* [45] address the above problem in a data warehouse environment. They present a lineage tracing algorithm for relational views with aggregation. Based on their tracing algorithm, they propose a number of schemes for storing auxiliary views that enable consistent and efficient lineage tracing in a multi-source data warehouse.

Buneman *et al.* further classify provenance into two types: why-provenance and where-provenance. Why-provenance refers to the source data that had some influence on the existence of the data; whereas where-provenance refers to the location(s) of the databases from which the data was extracted [33]. They suggest that Cui has only addressed the why-provenance in a relational database setting, whereas they propose a Deterministic Query Language (DQL) to allow query of both why-provenance and where-provenance in a more general context, rather than limited to the relational database context. In the DQL data model, the location of any piece of data can be uniquely described by a path. They have described a system of rewrite rules in which why-provenance is preserved over the class of well-defined queries and where-provenance is preserved over the class of traceable queries.

In [92], Ikea *et al.* have conducted a survey on data lineage. They formalise

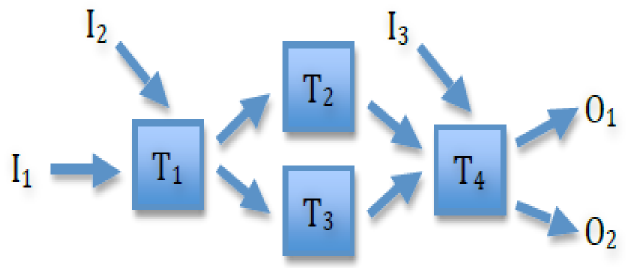


Figure 2.2: An Example of Transformation Graph [92]

provenance as below:

Input data sets $I_1, ..., I_k$ are fed into a graph of transformations $T_1, ..., T_n$ (an example is illustrated in Fig. 2.2) to produce output data sets $O_1, ..., O_m$.

Normally, the transformations form a directed acyclic graph(DAG) and the input and output data sets consist of individual items. Given a transformation graph, the provenance question can be described as:

Q1) Given some output, which inputs did the output come from?

Q2) Given some output, how were the inputs manipulated to produce the output?

These questions delineate two types of lineage: where-lineage (Q1) and how-lineage (Q2). Each type of lineage has two granularities:

1) Schema-level (coarse-grained)

2) Instance-level (fine-grained)

Schema-level where-lineage answers questions such as which data sets were used to produce a given output data set, while schema-level how-lineage answers questions such as which transformations were used to produce a given output data set. In contrast, instance-level lineage treats individual items within a data set separately, so more fine-grained questions can be asked such as which tuples from a set of base tables are responsible for the existence of a given tuple in a derived table (where-lineage).

Ikea *et al.* treat the why-lineage category of Buneman's definition as an instance-level where-provenance, whereas the where-provenance category of Buneman's definition is a more fine-grained of where-lineage.

Based on their categorization matrix, Ikea *et al.* survey eight papers, they find that in [31], Buneman *et al.* address how-lineage in both schema-level as well as instance-level; Heinis *et al.* [80] address schema-level for both where-lineage and how-lineage; whereas most majority papers [33, 46, 47, 89, 157, 191] address the instance-level of where-lineage.

The study of provenance mainly focuses on the source of the data and the transformation of the data. It is an important tool for ensuring the authenticity, integrity and traceability of the data, which are important aspects of the data accountability. However, the provenance topic does not address the actor and the obligation aspects of the accountability.

2.2.3.5 Logging, Monitoring and Auditing

While provenance mainly deals with the traceability of data in databases, the emphasis of service accountability is mainly on recording who does what from a collaboration process perspective. Therefore, tamper-evident logging, monitoring and authentic auditing are important aspects of service accountability.

Traditionally the mechanism to improve accountability in IT is by deploying IT monitoring (ITM) solutions which typically use a centralised monitoring server plus distributed monitoring agents or probes to monitor the health and Quality of Service (QoS) of the IT infrastructure, providing information on service level and alerting IT operators on abnormal events [173].

In [95], Jagadeesan *et al.* propose a logic for designing accountability-based distributed systems. Actors in the system are modelled as agents, including honest actors, dishonest actors, auditors and Trusted-third parties. Behaviours of all agents are described as process in process algebra with discrete time. Their approach supports both the design of accountability systems and the validation of auditors for finitary accountability systems.

In recent years, BPM has emerged as the latest trend in process management with the aim to improve process accountability. There are two interrelated BPM acronyms. One stands for business process management, which designates the automation and digitisation of processes. The other stands for business performance management, which expands on the performance of business process management. BPM systems typically include business process activity monitoring (BAM) and performance analysis capabilities [55]. For example, process is monitored in real time, with key performance indicators (KPIs) displayed on a dashboard. Business rules can be defined and automatically invoked when critical performance measures move out of control targets.

In [198], Yumerefendi and Chase promote accountability as a central design goal in network services by introducing a round processing framework. The authors suggest that an accountable system is undeniable, certifiable and tamper-evident. The accountability aspects are based upon the storage of digitally signed records or actions to detect inappropriate behaviour and to assign responsibility when things go wrong. While these techniques address the exchange of items, the work does not address accountability outside the transaction, such as full disclosure, maintaining confidentiality and the reputation of the parties.

In [182], Tseng *et al.* position accountability as the ownership of the responsibility to meet requirements in an end-to-end business process. The authors propose an Accountability Centered Approach (ACA) for business process engineering. The ACA approach suggests iterative decomposition of accountability to appropriate levels and mapping of sub-accountabilities into activities.

While ITM and BPM focus on proactive performance monitoring, they do not directly address other aspects of accountability, for example, disclosure, trust and reputation.

2.2.3.6 Defect Detection, Diagnosis and Recoverability

Some researchers treat accountability as the capabilities to detect system defects, diagnose the root cause of a service failure and recover automatically from service breakdown. One example is that Zhang *et al.* propose a 3-D approach (Detect, Diagnose, and Defuse) in their accountability model to discover and eliminate the root cause of problems when violations of service-level agreement (SLA) occur in business processes [200]. The approach adopts Bayesian Network reasoning for root cause analysis and a service reputation model to address problematic web services. In [115], Lin, Panahi and Zhang have created a prototype of Intelligent Accountability Middleware Architecture (LLAMA), which provides a dynamic and efficient service infrastructure to support service monitoring, root cause analysis and reconfiguration of service process after problem diagnosis.

Fig. 2.3 illustrates the LLAMA architecture. In the LLAMA-based system, two Trusted-Third Party (TTP) components are used: the Accountability Authority (AA) and Accountability Agents. They collaborate to perform service process monitoring, fault diagnosis, service process recovery, and service network optimisation. Multiple Agents are selected by AA to address scalability requirements. Each Agent is put in charge of monitoring a subset of services (as depicted by the circles in Fig. 2.3) during the execution of the service process. When undesirable process outcomes are detected during monitoring, Agents provide AA relevant service status information as evidence for AA to diagnose the run-time process problems. After diagnosis is confirmed, AA then follows with Defuse operations. The 3D approach emphasises the integrity, traceability, recoverability, trust and reputation aspects of accountability.

2.2.4 IT Governance-Level Accountability

On the IT governance level, the IT Governance Institute states that IT governance is concerned with IT's delivery of value to the business and mitigation of IT risks [93]. Weill and Woodham define IT governance as to specify the decision rights and accountability framework that encourages desirable behaviour in the use of IT [189]. COBIT (control objectives for information and related technology), developed and issued by Information Systems Audit and Control Association (ISACA) in 1996, has

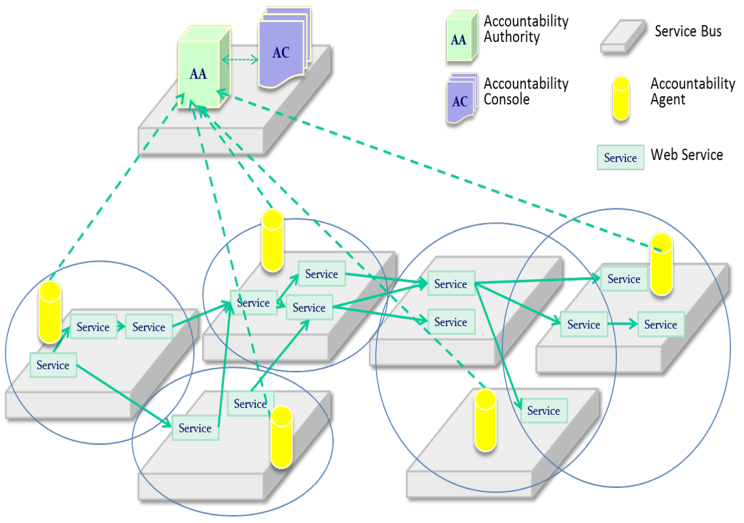


Figure 2.3: System Architecture of LLAMA Accountability Framework [115]

been adopted by corporations and governmental entities around the world and has become the de-facto standard for IT governance [184].

Since COBIT is largely used for governing the IT related processes, for business-focused SOA processes the term SOA governance has emerged. Moreover, SOA governance refers to the organisation, processes, policies, and metrics required to manage an SOA successfully [127]. SOA governance specifies and enforces conformance to SOA policies, which set the rules to govern the entire service lifecycle (*i.e.*, design, development, test, publish, discovery, runtime execution and sunset) and the behaviour of involved parties as well. SOA governance can be a challenging process for many organisations, as currently there is no standard model to evaluate accountability at the individual service level.

In the product development area, maturity models have been very successfully used in many different disciplines to improve process accountability [16]. Capability Maturity Models (CMMs) contain the essential elements of effective processes for one or more bodies of knowledge [178].

The Software Engineering Institute (SEI) at Carnegie Mellon University developed a Capability Maturity Model (CMM) [150], which defined key performance areas from an initial level of maturity to a level of optimisation. The CMM establishes a yard stick against which it is possible to judge in a repeatable way, the maturity of an organisation software process and compare it to the state of the practice of the industry [110]. It is developed to present sets of recommended practices in a number of key process areas that have been shown to enhance software development and maintenance capability. The CMM was designed to help developers select process-improvement strategies by determining their current process maturity and identifying the issues most critical to improving their software quality and process [110].

CMM defines five levels of maturity; each level is described below [150]:

Level 1 : Initial Maturity Level. In the first level of CMM, performance of an organisation is driven by the competence and heroics of the people doing the work. High quality and exceptional performance is possible so long as the best people can be hired. Unpredictability exists everywhere, for good or ill. The major problems faced by software organisations are managerial, not technical.

Level 2 : Repeatable Maturity Level. At this stage, the critical need is to establish effective software project management. Software project management processes are documented and followed. Organisational policies merely guide the projects in establishing management processes which are more project specific. Thus top management involvement is partial and does not drive the initiative. To be able to repeat best practices of the earlier successful projects requires these to be documented adequately. At this level , the focus is primarily on projects.

Level 3 : Defined Maturity Level. At level 3, the emphasis shifts to the organisation. Best practices are gathered across the organisation and Organisation Standard Software Processes are defined and tailored to projects, if required. The organisation now supports the projects by establishing common processes for software engineering and management, measurements and training. The process capability is based on a common, organisation wide understanding of the activities, roles and responsibilities. At this level, measurements have been defined and collected systematically.

Level 4 : Managed Maturity Level. At level 4, decisions are made based on data collected. The organisation sets quantitative goals for both software products and processes. The process performance and the project progress is controlled quantitatively. At this level, all organisational processes are mapped to a common measurement and assessed using a base line.

Level 5 : Optimising Maturity Level. At level 5, continuous process improvement is a way of life. The focus is on preventing the occurrence of defects and inducing innovations. In immature organisations, no one may be responsible for process improvement. Mature organisations usually have 70-80% participation in improvement activities at any given point in time- every one is involved. Continuous process improvement means controlled change and a measured improvement in process capability.

From what we can see, the progression of maturity level also reflects the level of accountability of the organisation.

Since 1991, CMMs have been developed for a myriad of disciplines. Some of the most notable include models for systems engineering, software engineering, software acquisition, workforce management and development, and Integrated Product and Process Development. The use of multiple models has been problematic. Many organisations would like to focus their improvement efforts across the disciplines within their organisations. Thus The CMM Integration^{SM} project was formed to sort out the problem of using multiple CMMs. [178]

The CMMI addresses some of the efficiency issues associated with CMM, changing the “waterfall” mentality to an “iterative” mentality of software development [163].

In [16], Baskarada *et al.* propose a TDQM based Capability Maturity Model (CMM) for Information Quality Management (IQM). They argue that the maturity model may assist organisations in assessing and enhancing their IQM capability, by addressing a wide range of Information Management (IM) and IQM process areas and organising those process areas into staged levels. This process ensures that every person is accountable for the quality of his or her work and does not send poor quality information to the next person in the process.

The maturity models are mainly used at the organisation level to govern the product development process in order to improve product quality and organisational performance, and thus improve accountability of both personnel and organisation level.

2.2.5 Accountability in Service-Oriented Architecture

Service-Oriented Architecture is suggested to be the most important software architecture in enterprise computing [22]. The early focus of SOA is on interoperability, reusability, flexibility and agility [149, 149]. In recent years the issue of governance has become a major concern in SOA [75]. Effective SOA governance is a precursor to promoting service accountability. However, the current focus of SOA governance is more on organisational alignment and service lifecycle management, but less on the accountability mechanism in the SOA architecture. Although some specifications have been developed in areas such as QoS management and security, and some of those specifications have even been implemented in the key SOA building blocks such as enterprise service bus (ESB), registry/repository and process engine, accountability as a whole has not been addressed by the current Web Services specifications and the current SOA implementations. In the following two subsections, we will examine the Web Services protocol stack and the accountability capabilities that the stack provides.

2.2.5.1 Web Services Protocol Stack

Since the release of the core Web Services protocols SOAP, WSDL and UDDI in 2000-2001, Web Services has rapidly become the technology of choice for distributed computing, replacing traditional technologies such as CORBA and DCOM [114]. These three basic specifications do not address enterprise concerns such as security, transactionality and reliability. As a result, ever more and more WS-* specifications have been developed by vendors to address those concerns.

In early 2006, HP, IBM, Intel and Microsoft jointly authored a Web Services road map white paper, clarifying their positions on various Web Services specifications for resources, events and management [37]. Based on [37], Fig. 2.4 shows the web services protocol stack. We will use this protocol stack as the basis to examine the accountability capability.

At the bottom of the stack, the standard internet transport protocols, like HTTP and SMTP, plus the XML family of specifications form the foundation. Building upon this foundation, the Web Services protocols like SOAP, ASAP, WS-Notification and WS-Addressing provide the messaging and encoding capabilities for service communication. The next layer is service description and discovery, enabled by WSDL, UDDI, WS-Policy and WS-MetadataExchange. Above that, the WS-Security family of specifications and WS-ReliableMessaging form the QoS layer. The next layer is business process, transaction and management, consisting of WS-DM, WS-Coordination, WS-ResouceFramework, WS-Transaction and BPEL. The top layer is the user experience, which at the time had only one specification: WS-RemotePortal.

Note that other alternative Web Services specifications can also fall onto the respective layer on the same stack. For example, WS-Reliability can replace WS-ReliableMessage in the QoS layer; WS-Eventing can replace WS-Notification in the Messaging and Encoding layer.

2.2.5.2 Accountability Capability of the Existing Web Services Protocol Stack

Accountability is directly related to the Description/Discovery and the QoS layers, and to some extent, the business process, transaction and management layer of the Web Services Protocol Stack. Although process management, transaction management, security and QoS properties all contribute to accountability, they are not sufficient to enable accountability. One missing critical capability is disclosure, which is the basis for accountability. Disclosure is for the accountable party to disclose identities, roles,

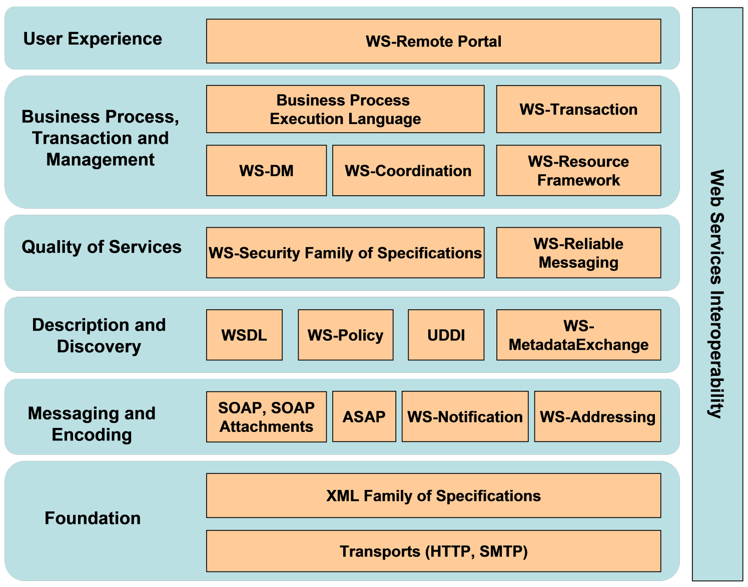


Figure 2.4: Web Service Protocol Stack

service responsibilities and the execution state and result of the service execution.

On the Description/Discovery layer, The WSDL is used to describe service definition, which includes service interface, location and binding details. WSDL document is normally treated as the service interface contract. The service descriptions disclosed by WSDL are technical in nature, without any accountability elements, such as roles, identities, responsibilities and service outcome. Thus WSDL does not contribute to accountability. UDDI is a registry standard that allows business to publish and discover Web services. It allows service providers to disclose some accountability information such as business entity, business services and taxonomy. However, due to the limitation of the UDDI data model, it does not support disclosure of roles, responsibility, execution state and outcome. WS-Policy provides a general purpose model and corresponding syntax to describe and communicate the policies of a Web service. However, WS-Policy is only a framework and does not define any specific policy. In theory, accountability information can be expressed in a policy assertion language and disclosed as service metadata through the WS-Policy framework and WS-MetadataExchange, which defines three request/response message pairs to retrieve three types of meta-data: WS-Policy, WSDL and XML schema. But there is no such assertion standard language yet.

On the QoS layer, WS-SecurityPolicy and WS-ReliableMessaging can be used as policy assertion languages in the WS-Policy framework to disclose non-functional requirements on security and reliable messaging. But other QoS policy assertion languages are yet to be formally defined.

On the Business, Transaction and Management layer, BPEL may be used to disclose the orchestration of the services, which is part of the process logic that falls into the functional aspect of responsibilities. But currently BPEL is used as a process modelling and execution language, not a disclosure language.

After examining the accountability capability of the relevant layers on the Web Services Protocol stack, we can see that the missing accountability capabilities are in the area of disclosure and service contract management. In particular, it is the ability to disclose roles, responsibilities (both functional and non-functional), service state and outcome, and the ability to track service obligation performance of the service contract. Hence it presents a capability gap in service accountability in Service-Oriented Architecture that is implemented using the WS-* stack.

2.2.5.3 Accountability in the Cloud Environment

Cloud computing brings new accountability challenges due to the characteristics of the cloud environment. The widely accepted definition of cloud computing is provided by the National Institute of Standards and Technology, which points out the five characteristics of cloud: on demand self-service, broad network access, resource pooling, rapid elasticity, measured service [132]. While these characteristics bring convenience, flexibility, high efficiency and low cost benefits to IT users, they also create new threats to accountability. In 2010, the Cloud Security Alliance (CSA) published the "Top Threats to Cloud Computing" report [44], which identifies the top seven threats as:

1. Abuse and Nefarious Use of Cloud Computing

2. Insecure Application Programming Interfaces
3. Malicious insiders
4. Shared Technology Vulnerabilities
5. Data Loss/Leakage
6. Account, Service & Traffic Hijacking
7. Unknown Risk Profile

In [105], Ko *et al.* discuss the complexities involved in achieving cloud accountability in a cloud environment. Some examples are (1) tracking of virtual-to-physical mapping and vice versa, (2) multiple operating systems, (3) logging from a file-centric perspective, (4) live and dynamic system, (5) the scale, size and scope of the logging. While acknowledging that the preventive controls through privacy protection and security measures such as encryption are either not sufficient, or are some-times impractical, they suggest augmenting preventive controls with detective controls that promote transparency, governance and accountability of the service providers. The detective controls of tracing data and file movement in the cloud are the focus in the paper. They propose the Cloud Accountability Life Cycle (CALC) and three abstraction layers (workflow, data and system) as a foundation to help researchers and practitioners to design tools and approaches which address all areas of cloud accountability.

Cloud computing is based on models like cluster computing, distributed computing, utility computing and grid computing in general [164]. In [112], Lee *et al.* propose an XML-based language for the specification of accountability policies for the grid computing system. They also design a distributed, agent-based system to enforce the policies expressed in this language. In addition, they provide security and privacy mechanisms for the storage of accountability data.

Commenced on October 1st 2012, A4Cloud (an Integrating Project in the EU's 7th Framework Programme (FP7)) focuses on accountability as the most critical prerequisite for effective governance and control of corporate and private data processed by cloud-based IT services [151]. A4Cloud sets out four project objectives [94]:

1. **control and transparency**: enable cloud service providers to give their users appropriate control and transparency over how their data is used;
2. **facilitate choice**: enable users to make choices about how cloud service providers may use and will protect data in the cloud;
3. **compliance**: monitor and check compliance with users expectations, business policies, and regulations;
4. **recommendations and guidelines**: implement accountability ethically and effectively.

In [144], Nunez *et al.* present a meta-model for defining metrics for accountability attributes, as part of the A4Cloud project. The goal of this meta-model is to act as a language for describing: accountability properties in terms of actions between entities and metrics for measuring the fulfilment of such properties. It also allows the recursive decomposition of properties and metrics, from a high-level and abstract world to a tangible and measurable one. Finally, they apply their metamodel in modelling the transparency property, and define the relevant metrics for it.

In [94], accountability in cloud computing is defined as:

"*Accountability for an organisation consists of accepting responsibility for data with which it is entrusted in a cloud environment, for its use of the data from the time it is collected until when the data is destroyed (including onward transfer to and from third parties). It involves the commitment to norms, explaining and demonstrating compliance to stakeholders and remedying any failure to act properly*"

The A4Cloud project also outlines a conceptual framework for accountability, which elaborates the definition above by means of a set of accountability attributes, accountability practices and accountability mechanisms. The core attributes of their accountability model are: transparency, responsiveness, remediability, responsibility, verifiability, appropriateness and effectiveness.

In [196], Yao *et al.* propose a separate accountability domain for cloud service providers to log the non-reputable evidences of their activities. The logging activities can be incorporated into existing business processes defined with process descriptive languages (*e.g.* BPEL[5]). A trusted-third party (TTP) will provide accountability as a service to enforce compliance based on the accountability domain. The authors further extend their approach in [197] and develop a quantitative model to represent the horizontal and vertical structures of the collaborations involving multiple un-trusted parties. Using this model, they classify four types of compliance and determine the logging needed for their verification.

In [174], a Cloud Information Accountability (CIA) framework is proposed. The CIA framework allows automatically logging any access to the data in the cloud together with an auditing mechanism. The authors of the CIA propose to extend the programmable capability of Java Archives (JARs) to automatically log the usage of the user's data by any entity in the cloud. Users will send their data along with any policies such as access control policies and logging policies that they want to enforce, enclosed in JAR files, to cloud service providers. Any access to the data will trigger an automated and authenticated logging mechanism local to the JARs. There are two major components of the CIA, the first being the logger, and the second being the log harmoniser. The logger is the component which is strongly coupled with the users' data, so that it is downloaded when the data are accessed, and is copied whenever the data are copied. The log harmoniser is responsible for auditing. It supports two auditing strategies: push and pull. Under the push strategy, the log file is pushed back to the data owner periodically in an automated fashion. The pull mode is an on-demand approach, whereby the log file is obtained by the data owner as often as requested. The CIA framework provides distributed logging and auditing capabilities that are suitable to the cloud environment. However, it is overly reliant on the Java environment which

[5]http://www.ibm.com/developerworks/library/wsbpelcol1/

may limit its applications.

2.2.6 Summary of Accountability in IT

Accountability is a term used loosely in the IT literature, referring to a variety of desired attributes in various aspects of IT. These attributes can be security, privacy, integrity, provenance, traceability, recoverability, trust and non-repudiation on the technical or architectural levels, or can be maturity on the governance level.

In [179, 180], Techapanupreeda *et al.* take an integrated view of accountability, proposing an accountability model and protocol in Internet transaction that satisfies essential security properties: confidentiality, integrity, authorisation, authentication, non-repudiation, liability and responsiveness. The protocol is designed using asymmetric cryptography and a hash function to ensure that it meets all the above accountability properties.

So far in this chapter, we have discussed the general concept of accountability, touching on the common practices of achieving accountability through quality management and regulatory compliance. Then, we briefly reviewed accountability from the management literature, outlining the accountability dimensions and accountability definition for service performance. Next, we described accountability related work in the IT literature. We observed that there are three levels of accountability work in IT: the technical protocol level, the architectural level and the governance level. We noticed that, thus far, there are two crucial aspects of accountability that have not been addressed in the IT literature: the disclosure and service contract management mechanisms. Following that, we clarified from an architectural concern perspective, how the accountability concern related to other architectural concerns, such as integrity, security, privacy, monitoring of QoS properties, trust and reputation. Next, we have examined the accountability capabilities in the current Web Service protocol stack and finally identified the gaps in accountability capabilities, which exist in the areas of disclosure and service obligation tracking based on the underlying service contract.

2.3 Service Contract-based Accountability

In a business environment, accountability is normally upheld through the enforcement of a legal contract. In a service arrangement, both the service provider and the service consumer are bound to the service contract, *i.e.* they are accountable to the fulfilment of the service contract. One major theme in current accountability research in IT is that it focuses too much on technical properties and rarely addresses the accountability concerns in the underlying business contract. While a Service Level Agreement (SLA) is a kind of contract, it normally only records the non-functional aspect of obligations and falls short on the functional aspect of service obligations, which is normally in the Statement-of-Work (SOW) scope. SOW deals with the issues of roles and responsibilities including obliged activities, deliverables and acceptance criteria, which are the key concerns for business.

2.3.1 Traditional Contract Concept in IT

Applying the contract concept in component and service design has been a widely adopted practice since the early days of the Object-Oriented movement. It was first implemented under the name of "design by contract" in the Eiffel language [134]. In [20], Beugnard, Jezequel *et al.* propose four levels of contracts for components: syntactical, behavioural, synchronisation and quality-of-service. The authors also presented a survey after ten years and found that contract-aware components are becoming mainstream in several domains such as embedded systems and Service-Oriented architecture [21]. These practices enhance integrity, reliability and, hence, accountability of components and services. The limitation of the existing approaches is that they only borrow the concept of contracts and apply technical constraints on fine-grained components, rather than treating contracts in a legal and business sense, which is a crucial requirement for accountable cloud services. They miss another level of abstraction that is required to deal with the Statement of Work (SOW), which is another prominent component of service contracts after the SLA.

In 1994, Szabo proposed the concept of the “smart contract”[6]. He suggested that many kinds of contractual clauses (such as collateral, bonding, delineation of property rights, etc.) can be embedded in the hardware and software, in such a way as to make breach of contract expensive for the breacher. In his view, a vending machine is an example of a smart contract. He further suggested that contracts can be embedded in all sorts of property that is valuable and controlled by digital means. Such contracts are smart contracts, which reference that property in a dynamic, often proactively enforced form, and provide much better observation and verification where proactive measures must fall short [175].

Although Szabo’s idea of the smart contract was proposed more than 20 years ago, it has largely been ignored by the industry, as there was no platform that could enforce them, until the emergence of the Bitcoin system in 2009. In the Bitcoin system, a distributed contract is a method of using Bitcoin to form agreements with people via the blockchain. It executes automatically, taking human judgements totally out of the loop [24].

In 2014, Buterin published a whitepaper on the Ethereum project. The intent of Ethereum is to create an alternative protocol to Bitcoin for building decentralised applications. Ethereum provides a blockchain with a built-in Turing-complete programming language, allowing anyone to write *smart contracts* and decentralised applications where they can create their own arbitrary rules for ownership, transaction formats and state transition functions [34].

In a strict sense, a smart contract is not a service contract. It is a set of rules that are embedded into a property. A smart contract can enforce a functional implementation of a particular requirement, and can show proof that certain conditions were met or not met [139].

[6]http://www.virtualschool.edu/mon/Economics/SmartContracts.html

2.3.2 Contract Representation

On the other hand, contract representation is an extensively researched area in the IT literature. IBM's Trading Partner Agreement (TPA) defines a service contract as an XML document (TPAml) that stipulates the general contract terms, conditions, participant roles, communication and security protocols, and business process [49]. However, like many other XML based approaches, the contract meta-model lacks the formal semantics to enable reasoning and verification which a logic-based model could afford.

Policy management frameworks such as Ponder and KAoS have been widely used in access control and security management. Although a policy can be treated as a form of rules in a contract, in a strict sense, it deals with action rules on a fine-grained object, rather than the high-level obligations of service participants. In contrast, service accountability requires a high-level abstraction that allows decomposition and delegation. Also, policy-based approaches only provide a rule view on contract obligations and miss the crucial process view. In [153], the authors conduct a survey on policy-based management approaches like Ponder, KAoS, Rei and WS-Policy. The survey concludes that, while these approaches can be partially adopted in SOA, none of them can be readily applied in service management. It is evident that none of these policy frameworks have been applied in today's cloud services.

In [194], Xu proposes a multi-party e-contract model that maps a paper-based contract into contract actions and contract commitments. An algorithm is outlined to detect contract violation based on the commitment graph. But it does not address the state of the commitment, i.e., how to test whether a commitment is successfully fulfilled or not. The only way to detect contract violation is to test whether there is any action missing. The action is defined as a tuple of name, sender, receiver and deadline, lacking other acceptance criteria such as state condition or QoS attributes like availability.

In [125], Marjanovic and Milosevic propose a formal service contract model based on deontic and temporal constraints. It provides verification of deontic consistency and temporal consistency of an e-contract, plus a visualisation tool for action verification and scheduling. However, the service contract model is still based on an "ought-to-do" model rather than an "ought-to-be" model, therefore it is not suitable for modelling accountability requirements.

In [82], Herrest and Krogh provide a logical analysis on directed obligation, prohibition and permission. The paper discusses directed obligation in light of the benefit theory and the claimant theory with its conclusion supporting the benefit theory. In [177], the authors refine Herrest and Krogh's counterparty definition and present formal definitions for directed obligations and permissions in trade contracts. These approaches do not address the paradoxes associated with the standard deontic logic.

In [50], a service contract model based on Modal Action Logic, Deontic Logic and Subjective Logic is presented. In [72], a Business Contract Language (BCL) and Formal Contract Language (FCL) are proposed using Defeasible Logic and Deontic Logic. Other approaches include applying Event Calculus to service contracts (ecXML) [60] and extending First Order Logic (FOL) to handle the dynamic aspect of service contracts [51].

Most of the above approaches use some variant of FOL to represent service contracts, which is not easy to seamlessly integrate into the SOA architecture and, moreover, most of these approaches favour expressiveness at the expense of decidability - as we can see that FOL is not decidable. Moreover, these models do not provide an execution process diagram and complementary methods for obligation decomposition, as well as lacking methods to detect contract violation.

In [118], the authors outline a logic framework that incorporates concrete domain and action theory into an expressive DL called $\mathcal{ALCQO}(\mathcal{Q}*)$. The logic framework has the expressive power to describe both the static information and dynamic behaviour aspects of a service contract But it does not show how to detect contract violation. Grosof and Poon propose a service contract model by combining RuleML and DAML+OIL in [74]. They use DAML+OIL to represent MIT Process Handbook's process ontology and also present a contract ontology for the process. Then they outline an approach to specifying RuleML rules "on top of" DAML+OIL ontology to enable the specification of more complex situations in the contract. While their approach has more expressiveness, the implication of the decidability issue was not discussed. How to apply the contract model in a service oriented architecture environment was not covered either.

In [187], Wang *et al.* propose a contract-based accountability service model. The model uses federated accountability services to audit interactions between service consumers and service providers so that misbehaviours can be detected with undeniable evidence. A contract is prepared by a service provider according to a pre-defined schema, which is not a formal representation of a service contract. The contract mainly defines how operations in the service interface may change data states and how to track these changes.

2.4 Conclusions on Literature Review

A general overview of the research on accountability has been provided in this chapter. We have first briefly reviewed accountability in the management literature, including the general concept of accountability and the accountability mechanisms built through the quality management and compliance management perspectives. Through the review, we have distilled the essences of the general accountability concept, being disclosure, obligation fulfilment and liability assignment.

Second, we have provided a more detailed review of accountability in the IT literature. We have examined most existing work entitled "accountability". One finding is that the technical community share a quite different view on accountability than their business counterparts in the business community. The accountability term has been loosely used in IT literature to represent a variety of desired attributes in various aspects of IT, for example, security, privacy, non-repudiation, trust and reputation, compliance, provenance, auditability, reliability and recoverability. While these attributes do contribute to the overall accountability of an IT system, they are not the core concerns of accountability. The core concerns are the aforementioned essential attributes distilled from the general accountability concept. This presents a significant gap on the understanding of the accountability concept between the business and the IT communities. One notable exception is the Information Accountability framework [190] proposed by Weitzner *et al.* The connotation of their accountability term is more aligned to the essences of the accountability concept, yet their focus is on the data level rather than on the whole service level. We have also paid special attention to the SOA Web Service protocol stack and analysed its accountability capabilities. As a result we have identified an existing gap in disclosure and service contract management. Moreover, we have reviewed existing accountability work in the cloud computing domain. We found that while some valuable accountability solutions are starting to emerge, there is a general lack of accountability frameworks that deal with service contract disclosure, contract obligation fulfilment and arbitration in liability assignment.

Third, as service contract is a key element in upholding service accountability, we have reviewed existing work in the service contract space, focusing on service contract representation and contract management schemes. We have found that while service contract is an extensive researched area where a large volume of quality research works have been produced, there is still lacking a formal contract language that can be interpreted by machines, facilitating service contract disclosure, service selection and matching based on the disclosed service obligations. The other important missing component is a service contract management scheme that automates service contract execution monitoring and provides fair dispute arbitration in liability assignment. We believe that the service contract representation and service contract management scheme are crucial to enable accountability in service computing, especially for the cloud environment. Therefore, we have chosen to focus our research on these two topics.

Chapter 3

Service Accountability Foundation

As introduced in Chapter 1, the accountability issue has been largely ignored by the IT industry. The literature review in Chapter 2 also revealed that there seems to be no common understanding of the concept of IT accountability in the research community. Rather, some concerns from either the technological, architectural, or the organisational governance aspects are frequently treated as accountability concerns by various IT researchers. The business community and the IT community hold quite different views on accountability, and the main accountability considerations from the business community, in the area of disclosure, obligation fulfilment and liability assignment, are largely missing in IT literature. As service-oriented computing is about aligning business and IT, it is imperative to bridge the gap in accountability between business and IT.

Given the current state of accountability in IT, we argue that the first step towards service accountability is to lay down a foundational accountability framework for service computing, which defines the basic concepts, models, processes and measuring approach for service accountability. The processes include contracting, disclosure, monitoring and dispute arbitration. These processes can be manual or semi-automated, as long as they capture the essence of accountability requirements.

This chapter is organised as follows. We first examine the fundamental questions about accountability in service computing in Section 3.1. Then through a mashup service example we demonstrate how an accountable service behaves in Section 3.2. Next, we progress into the definition of the basic concepts for service accountability in Section 3.3. Following that, we outline the accountability processes in service computing in Section 3.4. Next, we provide a quantitative measurement for accountability in Section 3.5. Then, in Section 3.6, we propose an accountable SOA architectural style to instil accountability in the foundational SOA architecture, and outline an accountability framework to assist in building accountability mechanisms in service computing. Finally, we summarise the work in this chapter in Section 3.7.

3.1 Essential Accountability Questions

In the IT community, there is general confusion about the term "accountability", as various meanings have been attributed to it. The following questions help us to clarify the confusion, and align our understanding of accountability to that of the business community.

(1) Accountable to whom and for what?

As mentioned in the management literature review, accountability arises from various relationships. In the context of service computing, the main relationships are contractual, social (moral) and legal/regulatory. These relationships come with obligations; some are explicitly defined while some are implicitly required. In such a relationship, typically a party (A) is accountable to the other party (B) when A owes an obligation to B. In a contractual relationship, both service providers and service consumers are accountable for obligations as defined in the terms and conditions in a service contract, *i.e.*, the service provider is accountable to the service consumer for providing sufficient disclosure, and dutifully carrying out service obligations; whereas the service consumer is accountable to the service provider for disclosing financial information and promptly paying the service fees. In a social relationship, they are accountable for obligations implied in the ethical standards and social responsibilities, *i.e.*, accountable to the general public, *e.g.*, accessibility to people with a disability, environmental friendliness. In a legal/regulatory relationship, they are accountable for obligations as set out in statute laws or regulatory rules. To establish accountability in service computing, the first key concern is to figure out the service obligations of each service party.

(2) Who is accountable?

In a legal sense, obligation and liability lie with a human being or a legal entity. So only a human being or a legal entity, rather than a system or a service, can be held accountable. In a service-oriented environment like cloud computing, a service provider is held accountable for the service that he or she provides, whereas a consumer is held accountable for how he or she consumes the service. However, the rapid development of artificial intelligent (AI) technology sees an increasing number of IT systems or services gaining cognitive capabilities. This will present a great challenge in determining the accountable party in some circumstances when things go wrong. Increasingly, cloud services may behave like a proactive system, not like the passive, reactive system that we usually perceive. This may trigger a series of legal and ethical questions.

Imagine a hypothetical scenario, where a taxi company uses unmanned vehicles to provide taxi services and one of their unmanned vehicles caused a severe accident. In this case, who is accountable? It seems unfair to blame the taxi company. It is also difficult to lay the blame on the vendor of the unmanned vehicle, as it may aggregate many components from other vendors, using a lot of data feed services like traffic, weather, geographic information system (GIS), *etc.*, plus deep learning technology to arrive at the action decision that caused the accident. To meet this type of challenge, a precise accountability traceability map may need to be established, and a clearly defined service contract for each component is critical to enable such traceability.

(3) How to disclose the service contract obligations and the status of obligation fulfillment?

A key theme for accountability is to provide transparency in service obligations and the obligation fulfillment status, so service participants are kept informed during the life-cycle of the service contract. In business, disclosure of the financial information of a public company is generally done through a central platform, such as the stock exchange. How a continuous disclosure in a service computing environment can be undertaken is, as yet, unexplored territory.

(4) How to detect a violation in service contract?

In order to detect a violation of obligation in a service contract, a real-time monitor of the execution of a service contract must be in place. The monitor needs to understand the obligations in a service and compare these against each party's actions to detect a violation of a service obligation.

(5) How to prove that someone should be accountable for a fault?

A fault occurs when an obligation is breached by some party. It is necessary to show the causality which proves beyond reasonable doubt that the party is at fault based on the collected evidence. The evidence must be tamper-proof, with a non-repudiation guaranteed.

(6) If a dispute arises during a contract execution, how will the dispute be resolved?

In traditional service space, if a dispute occurs during the contract execution and can not be resolved between the service provider and the service consumer, they tend to go to a court or a tribunal to get the matter resolved. In a service computing space, especially in the cloud computing environment, the geographic diversity of the providers and the consumers, plus the lengthy and costly manual process, will render such manual processes impractical. Therefore, an automated dispute arbitration mechanism that demonstrates fairness is required to enforce accountability in service computing.

(7) How do we measure accountability?

Accountability in general is a qualitative term; it is difficult to measure accountability quantitatively. As per Peter Drucker's famous quote, "What get measured, get managed." [152], a quantitative measurement method of accountability is important for managing accountability in a service computing environment. Traditionally, QoS, the degree of SLA comformance, trust scores or reputation ratings have been used as indirect measurements for accountability. However, the measurement of key accountability concerns, such as the level of transparency, the degree of obligation fulfilment and the fairness of liability assignment, have not been explored in the literature.

Now, we use an accountable service sample to illustrate how these questions relate to service computing.

3.2 An Accountable Service Sample

We now use a Web 2.0 [148] Mashup service example to demonstrate the essential accountability concerns in a service arrangement. The term mashup originates from the practice of mixing song samples from two or more sources to produce a new sound track. In the context of the Internet, mashups are web sites or applications that combine content from more than one source into an integrated application.

In our mashup service scenario, Entity B offers a trading platform to allow their customers to trade various securities globally. It may also provide an automatic trading rules engine that allows automated trading based on criteria set by the customer. Entity B has contracts with different real-time financial data providers to provide price data, which is fed into a charting application provided by a service provider to produce price charts. For a particular trading transaction, customer Alice initiates the trade request with Entity B. This is based on the pricing charts provided by Entity C's charting service, with the real-time price data input from Entity D.

Between consumer Alice and Entity B, an accountable service arrangement will see sufficient disclosure from both parties prior to forming the service contract. Entity B should fully disclose the functionality of the trading platform, *i.e.*, statement of work (SOW), the source of data, the service-level agreement (SLA), fees and charges, and other compliance requirements, such as "Know Your Customer (KYC)" regulatory requirements [141]. Alice should also disclose her financial situation and risk profile to Entity B, in accordance with the relevant regulatory requirements. The disclosure may be carried out through verbal or written communications. They may negotiate the terms and conditions before entering into a contract. Once the contract is formed, Entity B should continue to disclose the service obligation fulfilment status until the contract is ended, and promptly respond to any queries that Alice may have on the execution of the service contract. Whereas Alice should also make disclosure to B if her financial circumstance has changed. If some obligations were breached during the execution of the contract, there should be some kind of monitoring mechanism to detect that. In addition, there should be a fair arbitration mechanism in place to assign liability based on non-repudiable evidence.

Similar arrangements also exist amongst Entity B, C, and D as well. Note that in the case of Alice and Entity B, since the interaction has human interface involved, the contracting, disclosure, monitoring and arbitration processes can be manual or semi-automated, as long as the processes are repeatable and delivering accountable results.

In contrast to the traditional computing that focuses on functionality and/or QoS performance, we can see from this sample that accountable service computing emphasises transparency, fairness, honouring promised obligations, and the willingness to be punished if an obligation has failed to deliver. This is a quite different mindset to what most IT service providers or consumers traditionally use. Notwithstanding the low awareness of accountability in the IT industry at present, the demand for service accountability will inevitably gain momentum while the service industry is maturing.

3.3 Basic Concepts of Service Accountability

In traditional IT, the term “accountability” was assigned different meanings in different contexts. There is no common definition of it. There has been a certain degree of confusion around the topic of accountability in the IT industry. As the mainstream of IT moves towards service computing with a goal to align business and IT, it needs a common vocabulary and language to describe accountability requirements in service computing. We now use the mashup service example in Section 3.2 to identify the key elements in an accountable service.

3.3.1 Essential Elements for Accountability

The trading mashup service in Section 3.2 brings speed, flexibility and convenience to customers. However, when things go wrong, the accountability implications can be significant. For example, if a trade incurs a huge loss, and it is caused by the malfunctioning of the trading platform, then the liability assignment can be a challenging task, since the functioning of Entity B's trading platform has various levels of dependencies on Entity C and D, as well as B's own code.

In order to sort out the key question of "who is accountable for what?", we need to examine the roles and responsibility of each party. Refers to Table 3.1

Table 3.1: Accountability Elements in a Web 2.0 Mashup Trading Service

Identity	Role	Responsibilities	Outcome
Alice	Trade Requestor	Enter code and bid price. Provide funds for purchase.	The request accepted by Entity B.
Entity B	Trade Provider	Display result page with data from Entity C and D. Execute trade requested. Pay Entity C and D fees due.	The trade is executed. Funds transferred from Alice's account.
Entity C	Charting Service Provider	Provide correct charted pricing indicators.	Chart is displayed and the fee is received from Entity B.
Entity D	Real-time Price Provider	Provide real-time pricing with integrity.	Data feed is provided and received fee from B.

From Table 3.1, we observe that the essence of accountability involves four elements: the identity of the involved party, the role that the party plays, and the agreed responsibilities in the form of contract, agreement, or signed off requirements. The last element is the performance outcome, the evidence of who has done what. Fig. 3.1 depicts the relationships amongst these four elements.

These four elements also correspond to the key points in the performance special interest group's definition of accountability [9] in Table 2.1 listed in Section 2.1.1. A person, group or organisation can be translated to the concept of identity, whereas "execution of authority" implies the concept of role. According to Certo, responsibility

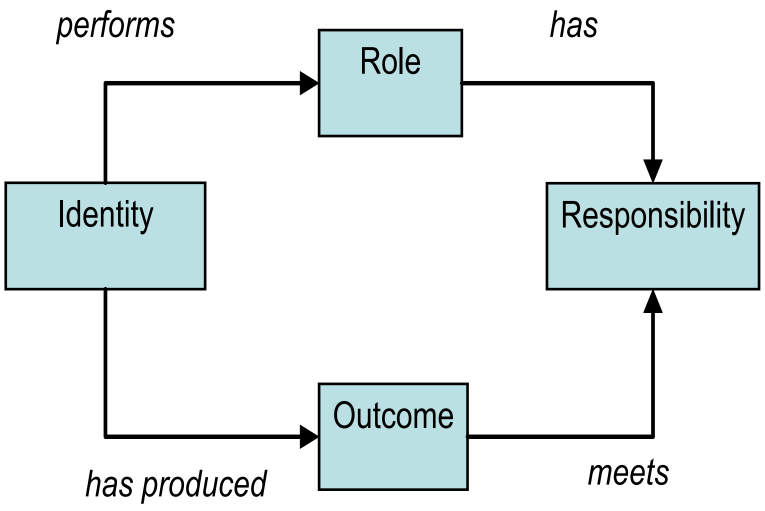


Figure 3.1: Essential Accountability Elements

is an obligation that someone "accepts" and is not allowed to delegate or pass on to someone else [35]. Accepting implies that there is some form of agreement in place. "Answering" and "reporting" relate to disclosure. "Assuming liability" for results requires a way to clearly demonstrate who has done what, which implies the requirement for non-deniable evidence of the outcome.

3.3.2 Definitions of Basic Concepts for Service Accountability

In moving towards building a common vocabulary and language for accountability, we now define the basic concepts for service accountability. We first define the commonly used term "service" in a service computing context.

Definition 1: (**service**): A service s is a set of collaboration process CP that exchanges economic values between the provision party (provider) and the receiving parties (consumers) via an interface i: $s = \langle CP, i \rangle$.

Service Properties:

1. A service interface i for service consumption;

2. A predefined set of processes CP for delivering expected effects to consumers;
3. A set of actors A to initiate or act on process activities.

Services Characteristics:

1. A service is intangible and is measured by service quality;
2. Service quality depends on actors' skills and the process properties;
3. A service is governed by implicit or explicit contracts. The later may include:
   a) General overarching contract
      The general contract that defines the legal terms and conditions;
   b) Statement of Work (SOW)
      SOW specifies the service's functional requirements. It defines agreed deliverables and the roles and responsibilities in agreed activities;
   c) Service Level Agreement (SLA)
      SLA specifies the service's non-functional requirements. It defines quality of service (QoS) and may include remedies if QoS is not met.

Now we give the definition of a cloud service below:

Definition 2: (**cloud service**): A cloud service cs is a service that is hosted on a cloud computing (refer to [132] for definition) infrastructure (identified by $CIid$) and can be consumed through a set of APIs defined by interface i through the Internet: $cs = \langle i, CP, CIid \rangle$. CP is a set of collaboration processes on which the service relies for delivering value. $CIid$ is the ID that identifies the underlying cloud infrastructure.

For example, Amazon's EC2 service can be represented as: $ecs = \langle ec2\_wsdl, acp, aws \rangle$, where $acp = \langle createVM, startVM, connectVM, stopVM, cancelVM \rangle$.

A general cloud service's characteristics:

1. The execution of a cloud service is largely automated or semi-automated. The service obligation fulfilment is not easily monitored in an objective manner by an independent third party.

2. The provision and consumption of a cloud service are both automatic or semi-automatic through the Internet. The consumer and provider do not meet face-to-face. It is difficult for a service provider and a consumer to determine and agree on the liable party when a service obligation is breached.

3. In contrast to an internal system or a software program which is programmed to respond to some external stimuli, a cloud service is normally initiated by human actors or, in some cases, by some intelligent agents. These human actors or intelligent agents can act proactively, based on their beliefs or knowledge. Hence a cloud service exhibits the behaviours of a proactive system rather than that of a reactive system.

For example, Amazon's EC2 service is largely executed automatically. Although Amazon provides a CloudWatch tool for monitoring the EC2 service, its objectivity is in doubt as it is a provider's tool. On the other hand, it is not easy for an independent third party to monitor the service. Also both the provision and consumption of Amazon's EC2 service are through the Internet. The service provider and the consumer do not meet face-to-face. If problems have occurred, it is difficult for the involved parties to agree on who caused the problems, *i.e.*, whether the consumer provided the wrong input or the provider did something wrong in the service operations. Moreover, both the service provider and the service consumer can act of their freewill during the service delivery and consumption. For example, the consumer may choose not to respond with a log-in request from the service provider, in which case, the execution of the service contract will stall.
Now we define the concept of "obligation", which is the most important concept of accountability.

Definition 3: (**obligation**): An obligation O is a finite set of action/evidence pair, $O = \{(a_1, e_1), (a_2, e_2), ..., (a_n, e_n)\}$, where action a is a quadruple: $a = \langle input, output, pre, post \rangle$, where $input$, $output$ are input and output of the action respectively; both pre and $post$ are binary condition expressions; evidence e is a finite set of evidence object,

$timestamp$ and condition triple: $e = \{\langle o_1, t_1, c_1\rangle, \langle o_2, t_2, c_2\rangle, ..., \langle o_n, t_n, c_n\rangle\}$, where o_i is an evidence object (typically should be signed with the actor's private key), t_i is the creation timestamp of the evidence object, c_i is a condition expression that is evaluated to true.

We denote the service provider's obligation as O_p and the service consumer's obligation as O_c. With the definition of the obligation, we now define the concept of service contract.

Definition 4: (**service contract**): A service contract sc is a formal representation of obligations in statement of work (SOW) and service level agreement (SLA) for a service.
$sc = \langle s, P, sow, sla, R, T\rangle$. It has the following properties:

- s is the service on which the contract is based;

- P is a pair of involved parties $P = \langle s_p, s_c\rangle$, where s_p is a provider and s_c is a consumer;

- sow is the statement of work,

  $$sow = \langle O_p, F_p, O_c, F_c\rangle$$

  where O_p is a set of provider obligations, F_p is a set of forbidden clauses for the provider, O_c is a set of consumer obligations and F_c is a set of forbidden clauses for the consumer. SOW specifies the service's functional requirements. It defines the agreed deliverables and the roles and responsibilities in those agreed activities;

- sla is the service level agreement, $sla = \langle O_p, F_p\rangle$, where O_p is a set of provider obligations and F_p is a set of forbidden clauses for the provider. SLA specifies the service's non-functional requirements that the provider must satisfy.

- Rules: R is a Horn clause [84] of the form $r \leftarrow r_1, r_2, \ldots, r_n$ where r is the consequent and $r_1, \ldots, r_n$ is the antecedent. A rule defines a remedy if an obligation is breached;

- Time Period: $T = \langle start\_time, end\_time \rangle$ is the contract's effective period.

A service contract can be executed many times. Each execution is defined as below:

Definition 5: (**service contract execution**): A service contract execution is a tuple $sce = \langle sc, I, O_p, O_c, se, R \rangle)$, where: sc is the service contract; I is execution information, $I = \langle start\_time, complete\_time, timeout\_value \rangle$; O_p is a set of obligations (see Definition 3) that are successfully completed by the provider; same applied to O_c as the completed obligations by the consumer; se is the contract execution state: $se \in SE$, $SE = \{se_1, se_2, ..., se_n\}$, where se_i is one of the user defined contract execution states, for example, $IN\_PROGRESS$, $COMPLETE$, $PENDING\_PROVIDER\_OBLIGATION$ *etc.*; and R is a set of Rules: $R = \{r_1, r_2, ..., r_n\}$, r_j is a horn clause: $consequent \leftarrow antecedent$.

We now give the definitions of "disclosure" and "service accountability".

Definition 6: (**disclosure**): Disclosure is the voluntary act to inform the service participants about information that may have a material impact on them. This may include, but is not limited to, identities involved, roles, responsibility, service status, and results of service execution.

Definition 7: (**service accountability**): The accountability of a service is its ability to disclose involved parties' obligations, fulfil obligations, disclose execution performance, detect obligation violation, determine fault causality and assign liability to the party at fault.

With the basic concepts for service accountability defined, now we examine the processes for service accountability.

3.4 Service Accountability Processes

Based on the essential accountability elements in Fig. 3.1 and the mashup services example in Table 3.1, we observe that four major processes are crucial in achieving service accountability:

- **disclosure**: process for disclosing of the roles, responsibilities and obligation fulfillment status by all parties;
- **contracting**: process for negotiating terms/conditions and establishing service contract with the other party;
- **monitoring**: process for monitoring the execution of a service contract, detecting contract violations and providing remedy when an obligation is breached;
- **liability assignment/arbitration**: process for assigning liability to a party by arbitrating a dispute during the execution of a service contract;
- **remedy**: process for rectifying a breach of obligation.

An accountable service should have repeatable disclosure, contracting, monitoring and arbitration processes in place in order to demonstrate transparency, fairness and honoring of obligation. Here we collectively name these processes "service accountability processes". We discuss each of the processes in the following sections.

3.4.1 The Disclosure Process

In business, an accountable public company is required to carry on continuous disclosure on any information that has a material impact on investors. [41]. The same principle applies to a service. An accountable service should provide a continuous disclosure process that enables transparency on service obligations in terms of roles and responsibilities, and the status of the obligation fulfilment.

Disclosure of roles and responsibility, to a large extent, can be enabled by rich service meta-data and facilitated by functions provided by a service broker; rich service meta-data means adding semantics to allow machine interpretation and reasoning. Currently, the registry (UDDI) provides service metadata in terms of business entities, taxonomy and reference to service information. The registry is a dynamic name binding service that is syntax based [122]. However, in mashup several sources require identification and these may need to be trusted sources in an accountability sense. We wish to enable semantic meaning, as in OWL-S [122], and suggest a more sophisticated role to facilitate trust by enabling richer metadata to capture aspects such as traceability of service composition and responsibilities for several parties.

We now explain the ways in which disclosure may be addressed in service computing and propose several elementary techniques that may be adopted in an SOA architecture.

3.4.1.1 Levels of Disclosure

We suggest that there are three fundamental levels of disclosure that may be adopted. The first level of disclosure is a manual-based, free text format disclosure, which is currently adopted by the industry. This form of disclosure is normally hosted on the service provider's web site, and can include a variety of content, such as "about us", "contact us", service "terms and conditions" and privacy policy. The Service Level Agreement (SLA) may be considered a form of disclosure on the QoS properties. To some extent, technical manuals and Application Programming Interface (API) documents can also be classified as some kinds of disclosure. While this approach may be sufficient for some business applications, there are several problems on this level of disclosure. For instance, this approach does not cater for machine interpretation. That is, it does not support dynamic service consumption in a Service-Oriented Architecture. A further issue is that the non-standard format of the disclosure creates difficulty for Trust and Reputation engines to source input from the service provider. In addition, the content for such disclosure is aimed at people with a legal background and not for general consumers.

The second level of disclosure is the emerging syntax-based XML format disclosure. There are several Web Services specifications that can be used for this purpose. WS-Policy provides a framework for general syntax for expressing capabilities and policies. WS-Agreement [6] can be used to disclose obligations. WSLA may also be used to express the Service Level Agreement [121]. While the syntactic level of disclosure holds a great promise in terms of supporting dynamic service consumption, it is nonetheless immature and yet to be widely accepted.

The third level of disclosure is the semantics-based disclosure. This level is more sophisticated, relying upon a commonly agreed ontology established amongst service providers and service consumers. Currently OWL is the most widely used language to create such an ontology. OWL-S allows a semantic description of a web service [128]. However, an ontology can become quite complex, especially when this delves into domain specific concepts. This approach is a more immature area when compared to the syntax-based disclosure.

3.4.1.2 Approaches to Disclosure

There are perhaps several approaches to implementing disclosure in an SOA architecture. The first approach is a registry based disclosure. This approach treats disclosure information as part of the service meta-data, and publishes the meta-data to a public registry to facilitate service discovery and matching. This approach is similar to publishing and discovery in SOA. Two key registry standards are currently available, UDDI and ebXML. Currently, UDDI is mainly used for the publishing of service description. Applied in this way, the data model makes it difficult to support the disclosure of roles, responsibility, contract execution state and outcome. On the other hand, ebXML allows business to first disclose its business profile information in an ebXML registry and then form an agreement based on the disclosed profiles of trading partners. The profile information includes roles, capabilities, constraints and implementation details, while the agreement may take the form of a Collaboration Protocol

Agreement [69]. Hence, the ebXML form provides stronger disclosure capability than UDDI. However, ebXML is normally applied to business-to-business (B2B) scenarios and may be considered too heavyweight for most business-to-consumer (B2C) scenarios.

The second disclosure approach is based on a *Publish and Subscribe* architectural pattern [79]. A service provider can publish the disclosure information, which can be subscribed by service consumers. *Publish-Subscribe* is a mature technology and it has been implemented in numerous commercial-off-the-shell middleware products. In a web services environment, WS-Notification is a family of related white papers and specifications that define a standard approach to develop *publish-subscribe* systems [73].

The third disclosure approach is to treat disclosure as a policy requirement to be enforced by a policy enforcer. In [153], the authors provide a survey on five policy frameworks: IETF, Ponder, KAoS, Rei and WS-Policy. They conclude that KAoS and Rei are more suitable for SOA systems due to their support for dynamic policy update. However, KAoS and Rei are ontology-based policy frameworks, which may not be easily applied in the normal syntactic web services environment. On the other hand, the authors also assert that WS-Policy is a low level policy language that is not suitable for managing an overall SOA system.

Currently, neither UDDI nor ebXML has been widely adopted in real SOA implementations due to their complexity and limitations. Hence, using the registry-based approach for disclosure will be a considerable task to achieve the objective of disclosure. On the other hand, the policy-based technology is still emerging and yet to mature. Thus it may not be a good choice for building disclosure mechanisms at present.

The four disclosure approach is to leverage the blockchain technology underpinning the bitcoin [165] system to provide a decentralised service disclosure solution. With this solution, service participants broadcast signed service contract information (including identities, roles and responsibilities) and the status of contract execution to a peer-to-peer service network. A miner [165] who has successfully mined a block will record the broadcasted messages into the blockchain, which forms a chronological, irreversible, tamper-proof transaction database that is accepted as consensus of the whole network. The blockchain technology provides transparency with the means to protect privacy. It is the ideal platform to enable disclosure for accountability. Refer to Chapter 6 for the details of the disclosure solution leveraging the blockchain technology.

3.4.2 Contracting Process

The contracting process includes three main steps. The first step varies for service provider and consumer. From a service provider perspective, the first step is contract preparation; whereas from a service consumer perspective, the first step is contract discovery and matching. The other two steps for both parties are the same, they are contract negotiation and contract establishment. We discuss the first step for each party respectively.

3.4.2.1 The Service Provider Perspective

From a service provider perspective, contract preparation involves preparing the service contract, normally including three parts of information. The first part is the overarching contract terms and conditions, including compliance requirements and the company's specific policies. The second part is the Statement of Work (SOW), detailing the roles and responsibilities, service functionality and deliverables. The third part is the Service-Level Agreement (SLA) that sets out the commitments on the quality-of-service (QoS) targets.

Contract preparation is a manual-based task, and it should be started during the service design stage. Generally, a commercial service would need to involve legal personnel to assist in formulating the service obligations, including SOW and SLA, in order to avoid the risks of over commitment. This is the current general practice for off-line service, like the motivating example outlined in Section 3.2. However, for on-line service like most cloud services, service providers' current practice falls short of providing a clearly defined SOW, like their off-line counterparts do. Instead, they tend to provide technical documents and user manuals of the service instead of committed obligations on the functionality that their service provides. This highlights the worrying sign of service accountability going backwards while the technology stack is moving upwards.

At the moment, most service contract is in free text format. Such a format of service contract can only be used in "first level disclosure", which inhibits automation in service discovery, matching, monitoring of the contract execution and dispute arbitration. Consequently, it may also prevent the realisation of service accountability in service-oriented computing. A solution to this should see the gradual evolution from a free-text format service contract to a syntax-based format one, and ultimately to a semantic-based format one.

After the service contract is prepared, the service provider can disclose the service contract information through the disclosure process.

3.4.2.2 The Service Consumer Perspective

On the other hand, from a service consumer perspective, the first step of the contracting process is to discover a service contract that matches its requirements. Currently this step is mostly manual-based. We envisage that when the representation of the service contract moves to a syntax-based/semantic-based format, then semi/full automatic service discovery and matching will become possible.

In a semi-automatic or fully-automatic discovery solution, the service consumer will deploy a software agent to search a service registry where the service providers publish their service contracts. The agent should be able to interpret the language representing the contract and select a service based on the service consumer's criteria.

3.4.2.3 The Contract Negotiation Process

Contract negotiation is a process of offering and counter offering on certain terms between the service provider and service consumer. For most services, currently the contract negotiation is carried out off-line manually. To the best of our knowledge, there are no commercial cloud services available that facilitate automatic service contract negotiation. Instead, most of the commercial cloud services available today use the provider's pre-written contracts, which means that there is no room for negotiation. The service consumer either chooses to enter the contract on the existing terms and conditions, or finds an alternative service from other service providers.

Automatic contract negotiation technically depends on the existence of a common machine-readable service contract representation. More importantly, its adoption also depends on the maturity level of service accountability in the overall service industry.

3.4.2.4 The Contract Establishing Process

Once the service consumer is satisfied with the service provider's terms and conditions of the service contract, the next step is establishing the contract and putting it into effect. The current practice for off-line service involves both the service provider and the service consumer signing legal contract documents. For on-line services, like the cloud service from Amazon AWS, the typical practice is for the service consumer to click an acceptance button on the terms and conditions page to enter into the service contract. After that, the service consumer can start to consume the service.

In an accountable service setting, the contract establishment process should include setting up the environment for monitoring of the execution of the service contract.

3.4.3 The Monitoring Process for Contract Execution

During the execution of a service contract, both service consumer and service provider need to monitor the status on contract execution, detecting any faults that may lead to a breach of the service contract. It should also provide remedy in case of an obligation breach. This kind of monitoring process defers to the monitoring process in the traditional IT space, where IT monitoring (ITM) solutions or Business Performance Management (BPM) solutions are deployed to monitor the health of the internal IT operations or business processes. These solutions generally only have abilities in detecting abnormal behaviours in IT system operations or business process execution, lacking the abilities in determining whether a service contract is breached or not.

Currently there is virtually no monitoring solution available for service contract execution in the industry. In Amazon AWS' case, Amazon offers a monitoring service called CloudWatch [11] to allow service consumers to monitor the health of their purchased services. CloudWatch is basically a cloud version of a traditional IT monitoring solution. Although it can provide system-wide visibility into resource utilisation, application performance, and operational health by track metrics, collecting and monitoring log files, the objectivity of the monitoring results is in doubt due to the potential conflict of interest as the monitor is provided by the service provider.

Ideally, a service contract execution monitoring service provided by a third-party can take a non-partisan stand when monitoring the obligation fulfillment status of both the service provider and the service consumer. Such a monitoring solution also has a technical dependency on the service contract representation language. Moreover, it also relies on the trust relationship established with both the service provider and service consumer, so the monitoring service provider can deploy monitoring agents or probes in the environments of both the service provider and the service consumer.

3.4.4 The Liability Assignment Process

Once an obligation breach occurs, the service participants would notice which party is at fault if they have deployed the service contract execution monitoring system. An accountable service party will admit their fault and provide a remedy voluntarily. However, in the case of both parties believing that is the other party's fault, then a dispute arises and needs a fair resolution mechanism to assign liability to one definite party. Typically, a trusted-third party (TTP) is needed to arbitrate based on the non-repudiable evidence provided by the involved parties.

A TTP solution usually comes with weaknesses like single-point of failure, performance bottlenecks and vulnerability to security attack due to its centralised nature. A decentralised arbitration protocol is introduced in Chapter 6.

3.4.5 Remedy Process

When a breach of obligation occurs, the party at fault can initiate the remedy process to rectify the fault. Depending on the terms defined in the service contract, sometimes the remedy process simply pays the fine, *i.e.*, assuming liability. Sometimes it may automatically fix the problem and continue with the execution of the service contract. The purpose of the remedy process is to resolve a contract breach situation and resume the execution of service contract.

3.4.6 Service Accountability Level

Corresponding to the classification of disclosure level in Section 3.4.1.1, if a service's majority accountability processes are manually based, we classify such service as "level 1 accountability" service. A "level 2 accountability" will have most accountability processes semi-automated, *i.e.*, syntax-based systems carry out procedural work, but need humans to make an interpretation on obligations, or to make decisions about obligation violation or liability assignment. A "level 3 accountability" will have most accountability processes fully automated, *i.e.*, the systems are capable of understanding the semantics of a contract, determining the causality of obligation violation and judging which party is at fault in a dispute situation.

In the mashup trading service example in Section 3.2, if Entity B has deployed systems to automatically publish service contract terms/conditions and service status, and to monitor the health of the service, like most of the current IT service management systems, then Entity B's trading service can be classified as a "level 2 accountability" service. On the other hand, if Entity B's systems can interpret a service contract, automatically detect contract violations and provide a fair dispute arbitration system, then it can be classified as a "level 3 accountability" service.

"Level 3 accountability" service is more in line with the future direction of service computing, especially more suitable to the cloud environment, as the virtue of cloud computing is achieving high efficiency and less human intervention.

3.5 Measuring Service Accountability

3.5.1 Introduction to Accountability Measurement

Traditionally accountability is assessed based on some qualitative measures setup in the governance framework, like COBIT, or regulatory compliance requirements, such as HIPAA, SOX or Basel II. Measuring accountability quantitatively is a challenging task. Dubnick and Justice take on the challenges of accounting for accountability from a governance perspective in [54]. They conclude that the feasible approach for measuring accountability is the continuing pursuit of measurement of the processes, conditions, and mechanisms through which accountability-based governance operates.

From a service computing perspective, qualitative assessment will not be sufficient in managing accountability risk, especially in the dynamic nature of an SOA environment. Other quantitative models, such as trust and reputation, can be subjective as they are mainly based upon consumer feedback [140]. We now outline a quantitative model to address the challenge of measuring accountability in a service computing perspective.

3.5.2 Principles and Theorems

We first outline the underlying theorems and tools that forms the basis for the proposed quantitative model.

3.5.2.1 Critical to Accountability Measurement

Many characteristics are critical to the quality of a service or product. The Six Sigma business management strategy defines these characteristics formally as Critical to Quality (CTQ), which represent the key measurable characteristics of a product or process, that must be met in order to satisfy a customer [117]. We borrow the CTQ concept and change the name to Critical-to-Accountability (CTA) in the service accountability context. Given a service provider that supplies a service, from the customer's perspective, there exists n number of CTA features associated with that service. This may be used later as the quantitative input to model accountability. A crucial indicator of the service provider's accountability is to measure out of these n number of CTAs, how many have been disclosed to the consumer; and further more, whether a compensation target has been disclosed if a particular CTA is not met.

3.5.2.2 Bernoulli Trial

A Bernoulli trial is an experiment where the result has only two outcomes, success or failure [61]. The probability of success is given as p and failure as $q = 1 - p$. The probability of n successes from N outcomes is given by a binomial distribution; this is referred to as a discrete probability distribution $P_p(n|N)$. The binomial distribution, probability of n success in N trials, is thus given by the following:

$$P_p(n|N) = \frac{N!}{n!(N-n)!}p^n(1-p)^{N-n} \tag{3.1}$$

The binomial distribution, *i.e.*, probability of n success in N trials in service contract execution, can be used to reflect the service accountability of a particular service.

3.5.2.3 Logistic Regression

Logistic regression [88] is a form of statistical modeling that is appropriate for describing the relationship between a dichotomous response variable and a set of explanatory variables. The logistic function is denoted as

$$f(t) = \frac{1}{1 + e^{-t}} \tag{3.2}$$

After a natural logarithm transformation, we have:

$$Logit(pi) = \beta_0 + \beta_1 x_1 + ... + \beta_k x_k \tag{3.3}$$

Where $Logit(pi) = ln(\frac{pi}{1-pi})$, $f(t) = pi$, and t = β_0 + $\beta_1 x_1$ + ... + $\beta_k x_k$

Logistic regression can be used to assess service accountability when there are multiple CTA features associated with that service.

3.5.2.4 Central Limit Theorem

A fundamental statistical theorem is the central limit theorem (CTL) [28]. The theorem states that as a sample size increases, the distribution of sample means approaches a normal distribution. In other words, the more sample data available the more likely the distribution of averages will resemble a normal distribution. In general, the sample size n must be sufficiently large to enable the approximation to function to work effectively. We discuss an appropriate sample size for approach later in Section 3.5.4.1.

3.5.3 Accountability Metrics

Similar to the contract concept in "design by contract" for Object-Oriented Programming [134], a service may be viewed as a contract established between the consumer and the service provider. Hence, a service can have preconditions, post-conditions, and invariants. The pre/post-conditions of the service can be viewed as the obligations of both parties to the committed service. Preconditions include valid input parameters, message format, and even fees and charges specified by the terms and conditions. Post-conditions are equivalent to the functional requirements that the service must deliver. Conversely, invariants are treated as the non-functional requirements in the service level agreement (SLA) that the service must support.

In light of the contract notion, service accountability may be assessed based on whether the contract has been breached; and if so, whether the compensation is duly honoured. Other aspects of service accountability that require consideration are:

- the degree of disclosure of the service information including preconditions, post-conditions and invariants by the service provider;
- the service's capability to provide non-repudiable evidence;
- the responsiveness of the service provider to service consumer's inquiry.

Thus from a quantitative perspective, service accountability may not be reduced to a single numeral. Rather, accountability is to be measured based on a set of indicators: *i.e.*, runtime accountability error, compensation, disclosure, responsiveness and evidence provision.

We first outline a general guiding approach for checking accountability breaches. This is given as follows:

General Service Accountability Breach Checking Approach

Let s denote a service and let $V = v_1, ..., v_k$ represent all the CTA *invariants* of s. The set $PR = \{pr_1, ..., pr_j\}$ represents all the *precondition* of s, and $PO = \{po_1, ..., po_k\}$ represents all the CTA *post-conditions*.

GIVEN: Consumer and service provider satisfy the set of all n preconditions, $\forall pr \in PR, pr = true$;

CHECK: That none of the *post-conditions* and *invariants* have been invalidated,
$\forall po \in PO$, ASSERT($po = true$);
$\forall v \in V$, ASSERT($v = true$);

If the check fails, there is an accountability breach. In particular, the check tests if the service provider s has breached non-functional requirements in set V or the functional requirements in set PO of the contract. This is an initial verifier and can be subsequently ratified if the compensation is honoured, otherwise the accountability breach remains.

3.5.3.1 Runtime Error Indicators

Definition 8: Assume $t1$ is the observation time interval, which can be monthly, daily or hourly depending on the desired measurement period. During time interval $t1$, the service has been executed n times, with i CTA *invariant* instances evaluated false and k CTA post-condition instances evaluated false, where $i \leq n$ and $k \leq n$. Then, we define the service respective runtime non-functional en and functional ef accountability error indicator as follows.

$$en = (i/n) * 100, n \geq 1 \wedge i \leq n \tag{3.4}$$

$$ef = (k/n) * 100, n \geq 1 \wedge k \leq n \tag{3.5}$$

Accountability also has a non-runtime aspect, which addresses compensation, disclosure, responsiveness on service inquiry and non-repudiable evidence. We define each of these below.

3.5.3.2 Compensation Indicators

For a particular service we assume that the service violates a CTA invariant p times and compensates q times. In addition, the service violates CTQ post-conditions on r occasions but only compensates s times. It follows that the non-functional (cn) and functional (cf) compensation indicator are given in the following.

$$cn = (q/p) * 100, p \geq 1 \wedge q \leq p \tag{3.6}$$

$$cf = (s/r) * 100, r \geq 1 \wedge s \leq r \tag{3.7}$$

Note that if either $p = 0$ or $r = 0$, then the respective compensation indicator is unknown. Also, assuming each CTA condition has an expected compensation target if the condition is not met, ce is the cumulated expected total compensation, and ca is the cumulated actual compensation value, then the cumulative compensation deficits are defined as:

$$cd = ce - ca \tag{3.8}$$

3.5.3.3 Disclosure Indicators

Assume a service S has i number of CTA *invariant* conditions, j numbers of *pre-conditions*, and k number of CTA *post-conditions*. For each CTA *invariant* and *post-condition*, we defined a full disclosure as disclosing all details associated with the conditions as well as the compensation target if the conditions are not met. If during the service history, the service provider only discloses s numbers of CTA *invariants* and the compensation targets if the condition is not met, k number of CTA *post-conditions* and the compensation targets if the condition is not met (i.e. a dispute arises, see 3.5.3.5). Then the respective non-functional (dn) and functional (df) disclosure indicators are given in the following.

$$dn = (s/i) * 100, i \geq 1 \wedge s \leq i \tag{3.9}$$

$$df = (k/b) * 100, (b \geq 1 \wedge k \leq b) \tag{3.10}$$

For preconditions dp, assume that the service provider discloses l number of conditions out of a, then

$$dp = (l/a) * 100, a \geq 1 \wedge l \leq a \tag{3.11}$$

As the level of disclosure is an important aspect of accountability, the disclosure indicators are the key metrics in assessing the service accountability.

3.5.3.4 Responsiveness Indicator

Assume in the service usage history that a service consumer has initiated n inquiry requests regarding the service status or service outcome, whereas the service provider only responded m times within a given time period. Hence, the responsiveness indicator re is defined as:

$$re = (m/n) * 100 \tag{3.12}$$

3.5.3.5 Evidence Capability Indicator

Assume in the service consumption history, that there are n times where a dispute arises that require the service provider to supply non-repudiable evidence. When sufficient evidence has been provided that resolves the dispute within a given time period, this is positive accountable outcome. Conversely, when the service provider is not able to resolve the dispute with the evidence provided, this is a negative accountable outcome. Given n disputes, with a subset of these disputes resolved m, the evidence provision indicator ep is defined as:

$$ep = (m/n) * 100 \tag{3.13}$$

In summary, the level of a service's accountability can be assessed based on these five metrics discussed: Runtime Error Indicator, Compensation Indicator, Disclosure Indicator, Responsiveness Indicator and Evidence Capability Indicator.

3.5.4 Accountability Assessment Model

We now outline a quantitative service accountability model. The model may be used to identify the scope of accountability requirements to be addressed, how service accountability may be measured, and for defining how they may be assessed for acceptance. We first however, outline a fundamental model that identifies the key attributes that may be measured. By extending this base model we then propose our final quantitative modelling approach.

Building upon the assertions in the previous section, accountability can be basically evaluated using the following criteria.

(1) Service provider's quality credentials (ISO 9001) [38].

(2) Service provider responsiveness on service inquiry.

(3) The runtime error indicators from both functional and non-functional perspective.

(4) If the disclosure includes both the target of CTA and the compensation target if the CTA is not met.

(5) Capability of providing non-deniable evidence.

(6) How the service provider honours compensation promises.

Using the criteria above, an indicator may be assigned to each category. An overall rating can then be calculated based on the assigned weight for each category.

3.5.4.1 Scorecard using Logistic Regression

The SOA model assumes that business processes consume services both internal and external to the organisation, and therefore there is a risk element in their use which requires quantifying. As an overall business management risk strategy, there is a requirement to facilitate service selection and comparison on all characteristics, including accountability metrics.

There are potentially a large number of service characteristics that may influence accountability, and their impact can be realistically assessed in terms of probability models. The variability factors created by those characteristics can be better dealt with by adopting a statistical rather than an engineering approach. We can explore the relationships between service accountability characteristics, and use these as population estimators, subject to validation through confidence interval and hypothesis testing techniques.

In general, the strategy to deal with this problem is to abstract out the service details and extract the common accountability metrics across services that are fundamental to service risk. Section 3.5.3 above provides the common quantitative metrics to enable assessing a service's current and historical accountability. However, part of the analysis of risk concerns the probability of a service breaching its accountability. It is therefore useful to have a model that computes a risk rating based on these accountability metrics. This rating can then be used by the service consumer to compare the service against a family of similar services and determine if a service is a higher or lower risk for accountability breaches.

The most pragmatic approach to developing such analyses would be through regression modelling, whose most recognised framework, linear regression, is used in exploring relationships between dependent and independent variables. The goal of the analysis is to know the probability of a service suffering an accountability breach, which in turns alters the overall accountability and trust of the business institution. There are, however, several factors in the accountability problem space that make selecting an appropriate regression model problematic. Linear Regression is not appropriate, because the predictor model is purely quantitative, and accountability characteristics may be both quantitative and qualitative. In addition, the value of linear regression can go beyond 1, which makes comparison of service accountability difficult. Linear regression is only useful for continuous dependent variable and not for dichotomous dependent variables, which is the case in service accountability, as we want to know the probability of a service breaching accountability. (Dichotomous dependent variables will invalidate the normality and homoscedasticity, equal variance of residuals across independent variable values, which are the assumptions in linear regression [26]).

In selecting a model, consideration has been given to the need to support precisely those constraints which preclude the strictly linear regression model. Given those constraints and that each observation on service accountability is an independent event; logistic regression is ideally suited for this situation. The main advantage of this approach is its flexibility. As long as each observation is an independent event, it does not need to assume a strictly linear relationship amongst the variables or the existence of a normal distribution.

Logistic regression enables the generation of service scores based on service accountability metrics and other predictor variables. In this model, the service score is the probability that a service suffers a breach of accountability. The criteria for a breach can be specific to each service consumer's risk profile, and can be defined using an upper limit of runtime error ratios or a financial threshold in compensation deficit as the criteria. A model based on such criteria can enable the service consumer to evaluate the risk involved in different service providers services. Furthermore, it is possible to standardise the criteria for a particular sector or industry, thus the same model can be used by different service consumers to assess different services provided by different service providers.

Selecting the relevant variables as the initial predictor variables is important for an effective regression process. We here define the scope for the independent variables in the accountability context. The selected initial variables should be from the accountability metrics.

Assume M represents the accountability metrics in Section 3.5.3, then the variables x_k in the logit formula below will satisfy the constraint: $x_k \in M$; where β_i is the respective weighting of the independent variable.

$$Logit(p_i) = \beta_0 + \beta_1 x_1 + ... + \beta_k x_k \tag{3.14}$$

The regression technique selected within the logistic regression is the stepwise which combines both forward selection and backward elimination to add and remove variables dynamically in each step until the best combination is reached. For example, assuming a sample of 500 observations that classified breached and non-breached services are available, we can choose *functional error indicator*, *non-functional error indicator*, *disclosure indicator* and *responsiveness indicator* as the independent variables and start the regression with the forward selection method. In each step of the regression, the variable with the highest predictive power (e.g Rao's efficient score [155]) will be added to calculate the weight for each variable. At the end of the regression, it will work out all the significant predictive variables which may be limited to the independent variables and their respective weights. Each of these weights is determined by a Maximum Likelihood Estimation (MLE) [76] method which seeks to maximise the log likelihood that reflects the probability that the observed dependent variables may be predicted from the observed independent variables.

Logistic regression needs a large sample size to reach a stable predictive model, with minimal sample sizes ranging between 100 in a worst-case to 500 in a best case scenario [176, 120]. This is not a problem for quantifying service accountability, as it is relatively easy to acquire a large sample of observations using service monitoring systems.

In summary, logistic regression appears to be appropriate in modelling service accountability, providing a relatively simple framework which can be applied both to individual services and groups of services. It also opens the possibility of using related techniques, such as multidimensional regression, once a body of indicators and set of series values have been established. The results obtained are also likely to be much more robust than simpler regression frameworks.

3.6 Service Accountability Framework

3.6.0.2 SOA Architectural Style

Currently the mechanisms to manage disclosure and service contract are missing in both IT literature and SOA solutions. We propose a framework for enhance accountability for Service Oriented Architecture (SOA), and hence briefly visit the fundamentals of this archetype. It is commonly agreed that SOA is an architectural style that involves a triangular relationship amongst three entities: service requester, service provider and the service registry [172, 124], see Fig. 3.2.

While the model captures the essence of the service-oriented architectural model, it may fall short on enabling accountability in service oriented-architectures, espe-

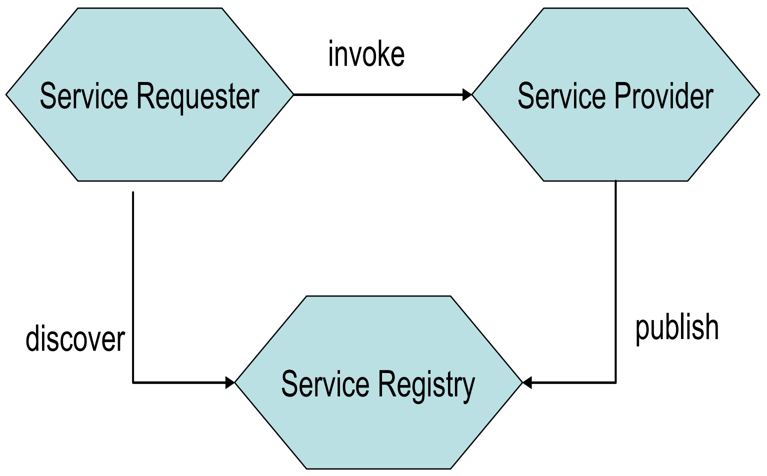


Figure 3.2: SOA Architectural Style

cially in the mashup service or cloud service environment. For instance, this does not address the roles of multiple parties and the associated responsibilities of disclosure, non-repudiation logging and obligation fulfillment monitoring.

In a mashup context, it is important to note that there are multiple service providers involved. There is also the introduction of the service source as a separate entity to the provider; although, in some cases the service provider is the same entity as the service source. In practice, the service provider may engage several external content source parties to participate in constructing the service. In this situation the service provider relies upon the source for accuracies of supplied content. As suggested in [68], there are a number of legal issues that need to be considered prior to developing mashup applications. As such, both the service provider and source are required to assume responsibility to ensure the mashup service complies with the intended application (and defined terms and conditions).

3.6.0.3 Accountable SOA Architectural Style

Based upon the previous discussions, we argue that the two identified roles, service requester and service provider, do not adequately represent all the roles involved in accountable service interactions. We now propose a model to depict these relation-

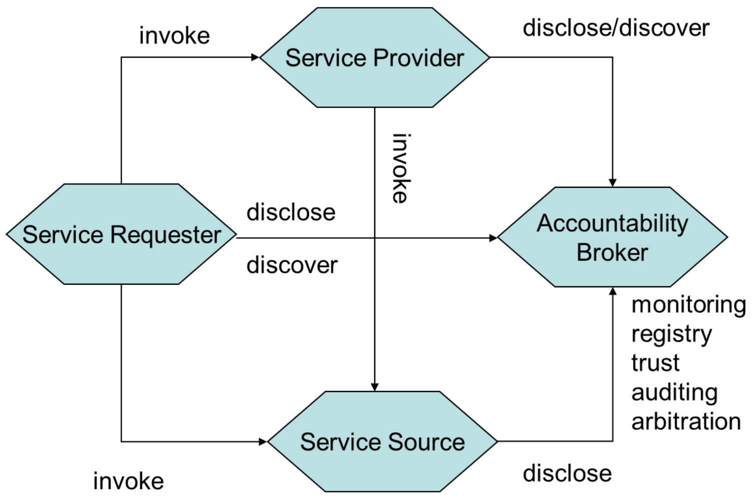


Figure 3.3: Accountable SOA Architectural Style

ships. The revised model is shown below, referring to Fig. 3.3. This is composed of the additional roles: service source and accountability broker. The service registry is still implied in this model, residing with the accountability broker.

Note that the service provider is a special type of role in the mashup environment which plays both requester and provider at the same time. The service provider will draw upon several internal and external sources and provide a resultant mashup page to the service requester. When sourcing content from a service source, the provider acts as the requester. The service source publishes a single or discrete set of (common) content sources that may be accessed directly by the service requester, or can be built upon and merged with other content source by a mashup service provider.

The accountability broker provides several additional benefits: as a trusted brokering agent (notary for unknown sources), monitoring, auditing (audit trail and evidence verification) to address the disclosure and non-repudiation requirements, rating and arbitration functions, and manages a combined registry and repository for multiple sources. Hence, the accountability broker role can be further refined into detailed roles based on these intermediary functions performed, see Fig. 3.4.

The Accountability Broker parent class is further extended to several children

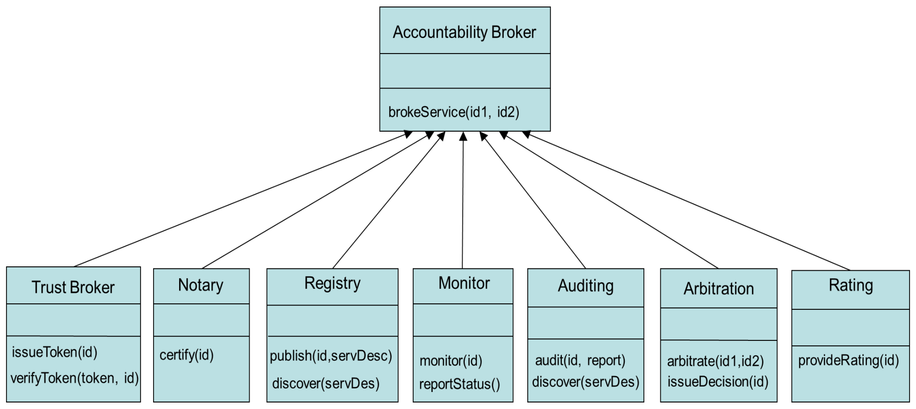


Figure 3.4: Expanded Accountability Broker Roles

classes. Their accountability functions are explained below:

- **Trust Broker**: providing trust by issuing trust tokens and verifying tokens.
- **Notary Broker**: providing notary services to service participants.
- **Registry/Repository**: providing publishing and discovery services to enable disclosure from service participants. It allows publishing of the service contract as well as the service contract execution status.
- **Monitoring Broker**: providing monitoring services to service participants.
- **Auditing Broker**: providing audit services to service participants.
- **Arbitration Broker**: providing arbitration services to service participants. It arbitrates a dispute and decides which party is at fault.
- **Rating Broker**: providing accountability rating services to service participants.

The enhanced role interaction model caters for both mashup and cloud services in a service oriented architectures. This helps to understand and define the accountability requirements in a service-oriented environment, and set out an accountability framework for service computing.

3.7 Conclusion

Despite the fact that the accountability issue increasingly becomes a major business concern, it is still largely ignored in IT, particularly in the service computing environment. One of the main reasons is the confusion around the accountability concept existing in the IT literature, and the large gap between what business understands about accountability and what the technical community thinks about accountability.

In this chapter, we took a business view of accountability, starting with a series of questions in order to clarify the essential meaning of accountability. In addition we used an example to demonstrate how an accountable service behaves from a business perspective. Then we translated the business meaning into IT terms and defined the basic concepts for service accountability. We also described the accountability processes for service computing and furthermore delved into the challenging topic of measuring accountability in the service computing environment. Finally, we presented an accountable SOA architectural style and outlined an accountability framework for building accountability mechanisms into service-oriented architecture.

To the best of our knowledge, our work is the first comprehensive accountability study in service computing and our proposed foundation for service accountability is novel and unique. We hope that it can raise the awareness of accountability in the IT service industry, bridging the gap between business to IT in the area of accountability and ultimately realising the goal of aligning business and IT.

Chapter 4

Advanced Service Accountability – A Semantic Approach

In Chapter 3, we laid down a foundational framework for service accountability. From the discussion in the last chapter we notice that the level of service accountability has a critical dependency on the availability of a machine-readable service contract language. Also the absence of such language creates an accountability gap in the current service-oriented architecture.

In this chapter, we adopt a semantic approach to design a formal service accountability model. The model consists of two components: one is a language component called OWL-SC, which defines the semantic representation of a service contract; and the other is a graphical component called SC-CPN, which allows verification of a service contract. Together these two components provide a formal representation language for specifying a valid, machine-readable service contract and facilitate the automation of the accountability processes such as disclosure, contracting, monitoring and arbitration. We first apply the model to describe a general web service example (Congo Book), then we further propose an Accountable State Transfer (AST) architecture to augment the mainstream SOA architecture implementation – the Representational State Transfer (REST), demonstrating the potential of such a model. We wish, through this semantic model, to narrow the accountability gap in service computing while increasing the level of service accountability.

The rest of this chapter is organised as follows. The next section further explains the motivation for creating a semantic-based contract representation language. Section 4.2 discusses the OWL semantic contract model OWL-SC with a graphical representation SC-CPN based on coloured Petri-net. This is followed by Section 4.2.4 that presents an example that uses the Congo Book service to illustrate the application of OWL-SC and SC-CPN. Section 4.3 further applies the model to design an accountable service architecture – Accountable State Transfer (AST), and explains how AST addresses the key accountability concerns in the popular REST architecture.

4.1 Motivation for Advanced Service Accountability

Traditionally, accountability is achieved through the enforcement of a legal, paper-based contract. In the cloud service environment, service providers generally publish a terms and conditions page for their offerings on their website. Upon clicking on the acceptance link, the consumer enters a binding contract with the service provider. Essentially, this is a web-enabled version of the paper-based contract, which presents serious accountability challenges in cyberspace. In its plain-text form, a web-enabled paper-based contract can neither be interpreted by software agents, nor be used as a basis for contract execution monitoring and state reasoning. Thus it does not enable service obligation disclosure, nor allow software agents to decide which party is responsible for what action, and which party is liable for what result. Moreover, the virtualised nature of the cloud service makes it even more difficult for liability settlement. This may become a major obstacle for the take-up of these cloud services.

The criticality of these issues motivates the need for a formal construct that maps the obligations in a paper-based contract to machine-interpretable capability statements. We call this formal construct a service contract model. Current SOA standards, such as SOAP-based WS-* and REST, do not provide a service contract model. REST in particular has a large accountability gap compared to SOAP-based services, as it was not originally designed to address enterprise requirements such as security, reliability, transaction ability and manageability. As REST is increasingly becoming a popular choice for implementing cloud services, it is imperative to build accountability mechanisms into the REST architecture. Without that, the obligation monitoring and liability assignment would be baseless. This presents an accountability gap in the current implementations of SOA that underpins the SaaS and Cloud platforms.

4.2 A Semantic Service Contract Model

In moving towards a detailed discussion on semantic service contract, we first examine the requirements for accountability management.

4.2.1 Requirements for Accountability Management

For SaaS and Cloud in particular, we can summarise the core requirements for managing accountability as below:

AR 1 : Obligations for both service provider and consumer can be specified unambiguously and are interpretable by software agents.

AR 2 : Obligations can be readily disclosed and accessible in a Web-based environment.

AR 3 : Obligations can be monitored and breaches can be immediately tracked; the status of contract execution can be reasoned by software agents.

AR 4 : Evidence of obligation fulfilment can be easily examined and reported.

4.2.2 A Semantic Service Contract Model – OWL-SC

4.2.2.1 Service Contract Definition

In order to meet the above requirements, we propose a service contract model as a formal construct for cloud service. The service contract specifies the obligations of both service provider and service consumer, which can be used as a basis for service participants to disclose contract terms and justify or explain their actions. Moreover, it can also be used as a baseline for obligation tracking and breach determination.

Recall the service contract definition 4 in Section 3.3.2 of Chapter 3, a service contract is defined as: $sc = \langle s, P, sow, sla, R, T\rangle$. This definition provides a generic two-party service contract structure that captures the key accountability elements. We do not deal with multi-party service contract models in this thesis, mainly because most contracts in cloud services only involve two parties. Even if multiple parties are involved in the underlying contract, the service contract model can always be decomposed into multiple two-party service contracts. Note that a cloud service normally is not an atomic service that only involves one interaction between a service consumer and a service provider. Instead, it is a composite service that may involve a series of conversations between a service consumer and a service provider. Each conversation is an instance of service contract execution that involves a series of actions performed by both parties. Also, during the valid contact period, the service can be executed multiple times, i.e. multiple conversations.

Using the Congo Book service [169, 7] as an example, the FullCongoBuy composite service can be offered as a SaaS or Cloud service. The consumer can invoke FullCongoBuy many times to buy different books during the valid period of the underlying contract. Each execution may involve a series of actions performed by either the consumer or the provider. An example of a consumers action can be inputting the book name, whereas a provider's action can be executing the atomic LocateBook service for locating the book. Considering this characteristic, Definition 4 does not capture detailed information in each execution. Thus we need Definition 5 for service contract execution: $sce = \langle sc_i, I, O_p, O_c, se, R\rangle$.

4.2.2.2 Service Contract Representation

The definitions above define the structure of our service contract model. However, to satisfy the requirements listed in Section 4.2.1, we need to first have a representation mechanism. As suggested in our studies in Section 2.3.2 of Chapter 2, most of the existing service contract models in the literature use some logic models with strong expressiveness power to represent a legal contract at the expense of computation decidability. Moreover, most of the service contract models are theory based, lacking tooling support and implementation. Conversely, our approach to a service contract model is to make trade-offs amongst expressiveness, decidability and existing tooling support. The ultimate choice should satisfy the requirements in Section 4.2.1, yet retain computation decidability and can be implemented with existing products and technologies.

As services may vary in different domains, a service contract model needs to capture the domain concepts and their relationships, and have the reasoning capability to ensure consistency. An ontology provides exactly these required capabilities; thus a semantic service contract model should be based on ontology. In the meantime, it should allow capturing of service contract execution information in a knowledge base (KB) so that the service contract execution can be tracked and execution states can be reasoned.

An ontology can be specified using Web Ontology Language (OWL[1]), which is recommended by W3C as the standard for representing ontologies on the Web. As a revision of DAML+OIL, OWL provides three sub-languages with increasing levels of expressiveness: OWL-Lite (corresponding to $\mathcal{SHIF}\ (\mathcal{D})$ [87]; OWL-DL (corresponding to $\mathcal{SHOIN}(\mathcal{D})$ [87]; and OWL-Full, which is an extension to Resource Definition Framework (RDF). Both OWL-Lite and OWL-DL provide computation completeness and decidability [185], whereas OWL-Full has maximum expressiveness but no computational guarantee.

We have chosen OWL-DL to represent our service contract model since it has the better trade-off between expressiveness and decidability, and it also has mature tooling support. But OWL-DL has its limitations. OWL 1 does not support role chaining. For example, given hasParent and hasBrother roles, OWL 1 ontology can not entail the "hasUncle" role. OWL 2 partially solves this through property chains [186].

[1]www.w3.org/2004/OWL/

To address this limitation, we augment OWL-DL with Semantic Web Rule Language (SWRL[2]). SWRL is a W3C submission, extending OWL-DL axioms with a set of horn clause rules. It is basically a combination of OWL-DL/OWL-Lite with the unary/binary Datalog sublanguages of the Rule Markup Language (RuleML) [86]. Therefore, in our case, OWL-DL can be used to define the concepts and roles in our service contract ontology while SWRL can be used to define rules for contract execution state reasoning. With this in mind, we here define:

Definition 9: (**semantic service contract model**): A semantic service contract model SCM can be represented as a KB that is a triple $K_{scm} = (\mathcal{T}, \mathcal{A}, \mathcal{H})$, where:

- A TBox $\mathcal{T}$ consists of a finite set of concept inclusion axioms of the form $C \sqsubseteq D$, a finite set of role inclusion axioms of the form $R \sqsubseteq S$ and transitivity axioms $Trans(R)$, where C and D are concepts, R and S are roles;

- An ABox $\mathcal{A}$ consists of a finite set of concept and role assertions and individual equalities/inequalities $C(a)$, $R(a, b)$, $a = b$, and $a \neq b$ respectively;

- A horn rule set $\mathcal{H}$ consists of a finite set of horn clause axioms. A horn rule axiom consists of an antecedent (body) and a consequent (head) in the form of: $r \leftarrow r_1 \wedge r_2 \wedge ... \wedge r_n$, where r, r_i ($0 \leq i \leq$ n) are atoms in rules that can be of the form $C(x)$, $P(x, y)$, $Q(x, z)$, $sameAs(x, y)$ or $differentFrom(x, y)$, and C is a concept; P is an individual-valued property; Q is a data-valued property; x, y are either variables or individuals; and z is either a variable or a data value. Variables x, y, z must be bound to named individuals in the KB to satisfy the DL-Safe criteria.

The other challenge that DL has is that it lacks an action semantic to describe the dynamic world. In [13], Baader and *et al.* integrate action theory into DL $\mathcal{ALCQIO}(\mathcal{D})$ and explore the computation properties of such an extension. Based on their approach, the authors in [118] propose a service contract model based on $\mathcal{ALCQO}(\mathcal{Q}*)$, with

[2]http://www.w3.org/Submission/SWRL/

a slightly different action semantics. While executability and projection are the major concerns in [13], the key concern in our model is determining if an obligation is fulfilled.

We thus introduce an evidence concept and use SWRL rules to simplify action state reasoning. Intuitively, the evidence concept reflects how a particular action's fulfilment is verified in real life. An evidence object, created as a result of the action can be used as a record to prove the occurrence of that particular action. For example in a real life scenario, a receipt can be used as an evidence object to prove that a book selling action has occurred. We adopt a similar action structure as [13, 142] with simplified semantics:

Definition 10: (**action**): An action is a quadruple $act = (input, pre, output, post)$ where: $input$ is the input of the action, which is a finite set of individuals in K_{scm}; pre (precondition) is a finite set of assertions in $\mathcal{A}$, $output$ is the action output, which is a finite set of individuals in K_{scm}; and $post$ (post-condition) consists of a set of finite set of conditional expressions in the form of φ/χ, where φ is a set of assertions in $\mathcal{A}$, χ is a set of assertions of primitive literals for $\mathcal{T}$.

For example, the *LocateBook* action has the following attributes:
$input$ = a book name individual: "*Twin Cities*",
pre = $\top(b)$, $output \in \{ISBN, OutofStock, \text{"}NotFound\text{"}\}$,
$post = \{\exists inStock.Book(b)/LocatedBook(b)$,
$\neg\exists Exists.Book(b)/NotFoundBook(b)$,
$\exists Exists.Book(b) \sqcap \neg\exists inStock.Book(b)/OutofStock(b)\}$
where b is an individual of the book "*Twin Cities*".

Definition 11: (**evidence**): An evidence object is a triple $ev = (obj, timestamp, cond)$ where: obj is an individual in K_{scm}; $timestamp$ is a data property of obj representing the time stamp that obj is created; and $cond$ is a set of assertions with regard to obj.

An example of evidence can be:
obj = an acknowledgement message ack, $timestamp = Timestamp(ack)$;

$cond = \exists Log.Msg(ack) \sqcap \exists Header.Label(\text{“}Ack\_LocateBook\text{”})$
$\sqcap \exists ValidSignature.Msg(ack).$

As we use the evidence object as a proxy for the occurrence of an action, we thus have the obligation fulfilment axiom:

Axiom 1: $Obligation(?a) \wedge mustDo(?a, ?b) \wedge verifiedBy(?b, ?c) \rightarrow fullfilled(?a)$

This axiom semantic is equivalent to a trigger rule [12] semantic $C \Rightarrow D$ where C, D are concepts. The trigger rule can be translated into the inclusion axiom with epistemic operator $\mathbb{K}$ [12]: $\mathbb{K}C \sqsubseteq D$. Intuitively, the $\mathbb{K}$ operator denotes that the rule only applies to those individuals that KB “knows” to be the instance of concept C, not to arbitrary domain elements. In our case, the rule only applies to those known instances of evidence objects.

In our service contract model, a predefined list of the valid sequence of actions will regulate the action performing order from both the provider and the consumer. We can use the List class as in [129] to represent the action sequence. However, we do not include other control constructs such as if-else or split, as the actions in the service contract are more coarse-grained than the atomic process in [129]. From a business perspective, the main concern is on the correct performing sequence of the high-level obligations, not on the low level logic of atomic tasks as handled by the traditional workflow.

Based on the above definitions, a service contract ontology can be defined. Fig. 4.1 shows a simplified version of the ontology. The highlighted *scContract* class has contract term definitions (Definitions class), and it links to a predefined service $p1 : Service$ which can be semantically described by OWL-S [129]. The contract class involves *ServiceProvider* and *ServiceConsumer*, both of which have a super class *Party*. Each party has Obligation which consists of multiple *Action* and *Evidence* pairs. *scContract* also has an *ActionSequence* class, which defines the valid action sequences. An execution instance of *scContract* will be defined by the highlighted *scContractExecution* class. The *scContractExecution* keeps track of the fulfilled obligations from both the service provider and the service consumer.

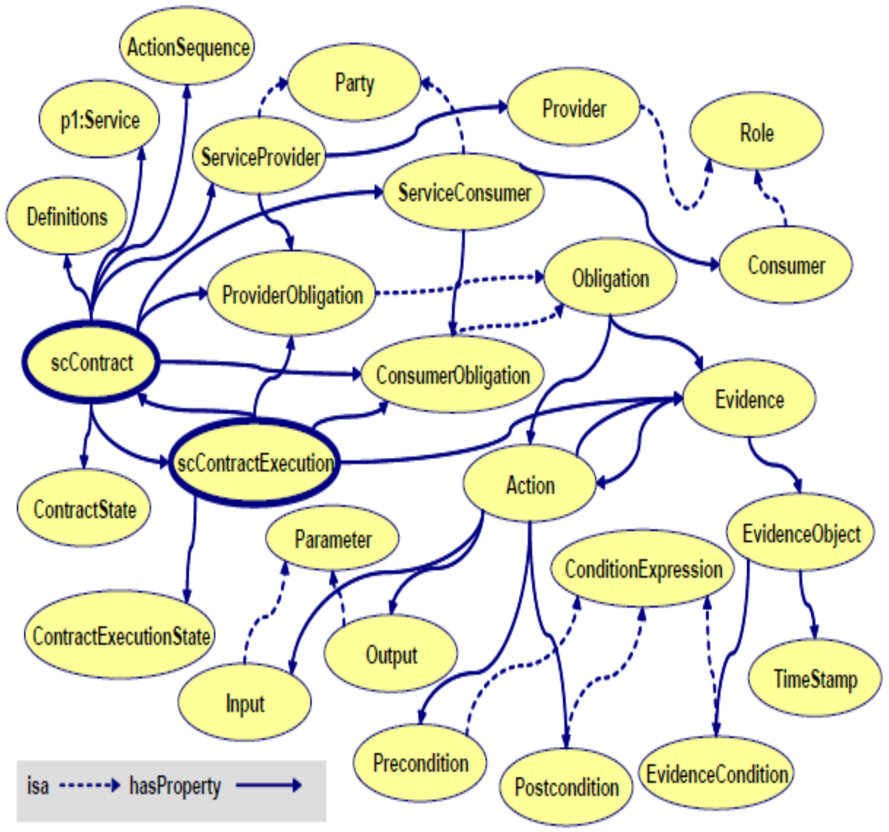


Figure 4.1: Service Contract Ontology

We name the service contract model built on top of this ontology OWL-SC, which complements a service defined by OWL-S with clear obligations spelled out for the service participants. An example is that for the *FullCongoBuy* service [129], a domain specific contract class *CongoBookContract* can inherit from the generic *scContract* class. Such domain specific contract instances may be executed multiple times. Each execution can be an instance of the *CongBookContractExecution* class, which inherits from the *scContractExecution* class. A KB can monitor the obligation fulfillment situation and moreover, reason the contract state and execution state based on the defined rules in the KB.

4.2.2.3 Properties of Service Contract Model OWL-SC

1) Expressiveness: The underlying DL in OWL-SC is currently $\mathcal{SHOIN}(\mathcal{D})$, with intention to move to OWL 2 $\mathcal{SROIQ}(\mathcal{D})$ when tooling support for OWL 2 is mature. $\mathcal{SHOIN}(\mathcal{D})$'s expressiveness is constrained by its syntax and semantics, which is listed in Fig. 1 in [87]. As mentioned earlier, OWL-DL is limited in role chaining expressiveness. Moreover, the built-in data type in OWL 1 is limited to $xsd : string$ and $xsd : integer$. In OWL-SC, we augment OWL-DL with SWRL, which extends OWL DL's expressiveness power at two fronts: first, it allows the reasoning of role chaining; second, SWRL built-ins can increase expressiveness significantly on $datatypes$ and the operations on them. These extensions allow us to bring in reasoning power in action semantics in our model. SWRL's limitation is that it does not allow disjunction and negation in the rules; moreover, explicit qualification over rules is also not supported. However, a combined OWL-DL and SWRL can leverage both strengths and provide the expressiveness to satisfy the requirements in Section 4.2.1.

2) Computational Properties: While OWL-DL is a decidable logic, SWRL is proven not decidable [138]. As the authors suggest, this is because that DL algorithm can always reach a finite tree model for the satisfiability check, but adding the rules breaks the tree model and it therefore becomes undecidable. To avoid the problem, the authors propose a so called DL-Safe rule. A rule r is called DL-Safe if each variable in r occurs in a non-DL-atom in the rule body. A program Pro is DL-Safe if all its rules are DL-Safe (see [138] for more details). The DL-Safe restriction is only exposed to ensure that the variables in the rule body are bound to only explicitly existing individuals in the KB. In our model, anonymous individuals are disregarded, as we apply the DL-Safe rule restriction.

From a computational complexity perspective, reasoning in $\mathcal{SHOIN}(\mathcal{D})$ has a worst-case nondeterministic exponential time (NExpTime) [85]. Research on sound and complete reasoning algorithm for OWL DL and rules are still an ongoing effort. Various approaches [113, 71, 130] have been proposed but each has limitations. In particular, we are interested in [71, 130] for our service contract model as the reasoning is based on an efficient production rule algorithm Rete. According to Forgy, Rete's worst complexity for the set of satisfied rules is linear in the number of rules, and polynomial in the number of objects [65].

3) Tooling Support: The basic requirements for tooling in our service contract model are the reasoning engine, ontology editor and rule editor. We choose Protege[3] 3.4.3 as our ontology editor and rule editor. Protege 3.4.3 supports OWL-DL, it bundles with Pellet and also provides a DIG interface for other reasoners like KAON2 and RACER. Moreover, it bundles with SWRLtab, which allows SWRL rules editing. There is also a SWRLJessTab [71] plug-in available for Protege 3.4.3, which can translate OWL facts to Jess facts and SWRL rules to Jess rules. It then allows invocation of the Jess rule engine that implements the RETE algorithm to do reasoning on the translated rules and facts.

4) Meeting the Requirements in Section 4.2.1: Compared to other service contract modelling approaches, the differentiation of our service contract model is that it is not just a theoretical model, but can be practically implemented with existing standards and tools; moreover, the action semantics and evidence concept closely mimic how accountability is treated in real situations in the business domain. In summary, with OWL-SC, obligations can be clearly specified, and interpreted by software agents. This satisfies **AR 1** in Section 4.2.1. Secondly, obligations specified in OWL-SC can also be referred through URI on the web, which meets **AR 2** and suits the SaaS and Cloud environment. Finally the action semantic and the concept of evidence in OWL-SC allow accountability solutions to be built to satisfy **AR 3** and **AR 4** respectively.

[3]http://protege.stanford.edu/

4.2.3 A Graphical Model – SC-CPN

4.2.3.1 Coloured Petri-nets

OWL-SC provides a structure and semantics for modelling obligations and actions in a service contract. But neither OWL-DL nor Protege provides an easy way for modelling the sequence of the obligations. On the other hand, coloured Petri-nets (CP-Nets) provide an intuitive graphical representation underpinned by a rigorous mathematics foundation; and more importantly, CP-Nets have an explicit semantic to describe both actions and states whereas most other formalisms can only focus on one aspect. Furthermore, tools like CPN-Tools[4] are available to assist in net editing, simulation and state space analysis. Thus CP-Nets are ideal for visual modelling, analysis and validation of our service contract model. We first outline the definition of CP-Nets from [96]:

Definition 12: (**Coloured Petri-net**): A CP-Net is a tuple $CPN = (\Sigma, PL, TR, AR, ND, CF, GF, EX, IF)$ where: Σ is a finite set of non-empty types, also called colour sets; PL is a finite set of places; TR is a finite set of transitions; AR is a finite set of arcs such that: $PL \bigcap TR = Pl \bigcap AR = TR \bigcap AR = \emptyset$; ND is a node function. It is defined from AR into $PL \times TR \cup TR \times PL$; CF is a colour function. It is defined from PL into Σ; GF is a guard function. It is defined from TR into expressions such that: $forallt \in TR$: $[Type(GF(t)) = \mathbf{B} \bigwedge Type(Var(GF(t))) \subseteq \Sigma]$; EX is an arc expression function. It is defined from AR into expressions such that: $\forall a \in AR$: $[Type(EX(a)) = CF(p)_{MS} \bigwedge Type(Var(EX(a))) \subseteq \Sigma]$ where p is the place of $ND(a)$; IF is an initialisation function. It is defined from PL into closed expressions, such that: $\forall p \in PL : [Type(IF(p)) = CF(p)_{MS}]$.

In the above definition, $\mathbf{B}$ denotes the boolean type. The type of a variable v is denoted by $Type(v)$, and the type of an expression $expr$ is denoted by $Type(expr)$; whereas the set of variables in an expression $expr$ is denoted by $Var(expr)$.

[4]http://cpntools.org/

4.2.3.2 CP-Nets Model for Service Contract – SC-CPN

Definition 13: (**SC-CPN**): Given a KB defined in OWL-SC K_{scm} = ($\mathcal{T}$, $\mathcal{A}$, $\mathcal{H}$), we map each concept in $\mathcal{T}$ to a colour set ∂ in a place p, $\partial \in \Sigma$, $p \in PL$; each assertion in $\mathcal{A}$ to a constant or variable declaration in the CP-Net, therefore each model of $\mathcal{T}$ and $\mathcal{A}$ in $\mathcal{I}$ corresponds to an initial marking in M_0. We also map each consumer action in A_c to a consumer transition in T_c, each provider action in A_p to a provider transition in T_p, where $T_c \bigcap T_p = \emptyset$, $T_c \bigcup T_p = TR$; input in and output out to token colours in P_i, where $P_i = T_p \bigcap T_c \bigcup Tc \bigcap Tp$, $P_i \subseteq PL$; precondition pre to $i \wedge g$, where i is a set of token colours in T that satisfies with input inscription E_i, $E_i \subseteq EX$, $g \in GF$, g is a transition guard in TR; also map post-condition $post$ to E_o , where E_o is output arc inscription, $E_o \subseteq EX$, $E_i \bigcap E_o = \emptyset$.

Based on Definition 13, we can construct a CP-Net from OWL-SC; we call it service contract CP-Net SC-CPN. Thus the standard CP-Net analysis techniques can be applied to validate the correctness of the service contract model with regard to the consumer and provider behaviour. In OWL-SC, we are concerned about two problems. One is the action executable problem [12, 159], *i.e.*, whether all pre-conditions are satisfied in the states of the world considered possible. The second problem is the projection problem [12, 159], *i.e.*, if the action is executable, we want to know whether applying it achieves the desired effect.

Intuitively, we can obtain two theorems on executability and a projection for their definitions) based on the mapping rules in Definition 13.

Theorem 1: The action executability problem in OWL-SC can be translated to a transition fireability problem in SC-CPN.

Proof: Based on the mapping rules, it is obvious that the theorem holds for one action. Assume it holds for $(a_1, ..., a_k)$ where $1 \leq i \leq k$, *i.e.*, for transitions $(t_1, ..., t_k)$, if $t_1, ..., t_k$ are fireable, then for all models $\mathcal{I}$ of $\mathcal{T}$ and $\mathcal{A}$, all interpretations $\mathcal{I}$ with $\mathcal{I} \Rightarrow Ta_1, ..., a_i\, \mathcal{I}$, we have $\mathcal{I} \vDash pre_{i+1}$. Now assume transition $t_{k+1} in(t_1, ..., t_{k+1})$ sequence is also fireable. As defined in the mapping rules, all initial markings M_0 correspond to all models $\mathcal{I}$ of $\mathcal{T}$ and $\mathcal{A}$. Prior to transition t_{k+1}, the marking is m_k, which is fireable, i.e. tokens at t_{k+1} satisfy input inscription e_i, as well as guard g at t_{k+1}, this implies that pre_{k+1} holds. Since m_k corresponds to $\mathcal{I}''$, where $\mathcal{I} \Rightarrow Ta_i, ..., a_j\ \mathcal{I}$, $i \leq j < k+1$, $\mathcal{I} \vDash pre_{j+1}$, and therefore, $\mathcal{I} \Rightarrow Ta_1, ..., a_i, ..., a_j\ \mathcal{I}$, $i \leq j < k+1$, which suggests action series $(a_1, ..., a_{k+1})$ is executable.

Theorem 2: The action projection problem in OWL-SC can be translated to reachability problem in SC-CPN.

Proof: Assume at marking m_i and m_r, there is at least a sequence of transitions $u = (t_1, t_2, ..., t_k)$, so $m_i[u\rangle m_r$, which means marking m_r is reachable from m_i through at least one sequence of transitions. Let action a_1 correspond to transition t_1, a_2 to t_2, and so on till a_k to t_k. As the post condition of a_k is mapped to E_t, which is the output arc inscription of transition t_k. So for all models $\mathcal{I}$ of $\mathcal{A}$ and $\mathcal{T}$, $\mathcal{I}$ $\Rightarrow \mathcal{T}a_1, a_2, ..., a_k\ \mathcal{I}'$, and $\mathcal{I}' \vDash E_t$, therefore, the effect of the action sequence can be projected through reachability.

The *ActionSequence* in *scContract* contains the valid action sequences to regulate the consumer and provider interactions. Therefore, to check whether or not a service contract is breached in an execution instance, it is simply a matter of checking the actual action sequence in *scContractExecution*, to see if it matches any predefined action sequence in *ActionSequence* in *scContract*. If not matched, by locating which party's action causes the discrepancy we can identify the responsible party.

4.2.3.3 SC-CPN Properties

The desired properties of SC-CPN with regard to our service contract model are listed below:

Structural Boundedness: Assume place p in
$\langle\, \mathcal{N}, m_0\rangle$, $\mathbf{b}(p) = sup\{m[p] | minRS(N, m_0)\}$, a net is structural boundedness iff $\forall p$ $\mathbf{b}(p) < \infty$, where sup is the token upper bound of p, RS is a reachability set.

Liveness Properties: We always expect that the net can be terminated at some state, *i.e.*, there exists at least one dead marking for any initial marking of the net.

Also, each transaction should have the possibility of firing, *i.e.*, no dead transition exists. The SC-CPN liveness property can be described as: $\exists m \in RS(\mathcal{N}, m_0)$, for $\forall t \in TR$ such that t is not fireable at m, and for $\forall t \in TR$, $\exists m \in RS(\mathcal{N}, m_0)$, $\exists \sigma$ such that $m^{\sigma t} \rightarrow m$.

Reversibility: We expect that the execution of service will lead to a new state. Thus non-reversibility is a preferred property for SC-CPN, which is defined as: $\exists m \in RS(\mathcal{N}, m_0)$, $\exists \sigma$ such that $m^{\sigma t} \rightarrow m_0$.

4.2.4 Using OWL-SC and SC-CPN to Model Service Contract for Congo Book Service

SC-CPN provides an intuitive graphical model for modelling the action sequence for OWL-SC. Simulation can be used to identify errors in the service contract's action model. Standard Petri-nets state and reachability analysis can be used to analyse the executability and projection in OWL-SC.

Now we illustrate how to use OWL-SC and SC-CPN to build a service contract model for the Congo Book service, which is a widely used example of a semantic web service. We will first define the basic concepts and SWRL rules in OWL-SC; then we use SC-CPN to model the obligation and action sequence; next we will use Pallet reasoner to classify and check the consistency of the ontology; then we will use Protege's SWRLJessTab to translate the OWL individuals to Jess facts, SWRL rules to Jess rules and use Jess to entail new facts; and finally we will use SQWRLQuerytab to query the service contract knowledge base.

4.2.4.1 Defining Contract Service Ontology for Congo Book

We now use Protege 3.4.3 to create the ontology based on the model in Fig. 4.1 Firstly we extend *scContract* class to create *CongoBookServiceContract* as the service contract class for Congo Book service, then subclass *scContractExecution* to create *CongoBookServiceExecution* for the contract execution class. After that we can enter some individuals. We use some candidate actions based on the Congo Book semantic service *CongoProcess.owl* [7] as a starting point: *P_LocateBook*, *P_SignInUser*, *P_PutInCart*, *etc*.

4.2.4.2 Modelling Obligations using SC-CPN

Now we use SC-CPN to model the actions and verify the correct sequence of actions. We leverage CPN-Tools' hierarchical net feature [42] to create a hierarchy of service contract nets across multiple pages for ease of modeling. Fig. 4.2 shows the hierarchy page. Fig. 4.3 shows the top page of *CongoBookServiceContract*, which can be used as a generic high-level SC-CPN model for SaaS or Cloud services. In the model, *ServiceConsumer* interacts with *ServiceProvider* through the *ServiceInput* and *ServiceOutput* places; each party maintains session information such as shopping cart, credentials in *ServiceSession* place; and *ServiceConsumer* pays for service with considerations and *ServiceProvider* delivers service effect.

Fig. 4.4 shows the *ServiceConsumer* page, which further breaks down the transitions to another level. Fig. 4.5 shows the *DeliverService* page, which models the lowest granularity of a transition; in this case, the Service Provider's obligation to ship the book to the consumer.

Once all of the net pages have been constructed, we run simulations to see the interplay between service consumer and service provider. We next use CPN-Tools' state space analysis capability to analyse the SC-CPN properties. We can continue to refine the SC-CPN model based on the desired properties discussed in Section 4.2.3.3. Then from the reachability graph, we produce the valid action sequence for OWL-SC. For example, one normal action sequence is:

C_ProvBookName, *P_LocateBook*, *C_PutInCart*, *P_GetSignOn*, *C_ProvSignOn*, *P_SignUserIn*, *P_PutInCart*, *C_Checkout*, *P_GetPaymentInfo*, *C_ProvPaymentInfo*, *P_ProcPayment*, *P_GetAddInfo*, *C_ProvAddInfo*, *P_ReqAuthPay*, *C_AuthorizePay*, *P_ShipBook*, *C_AcceptBook*.

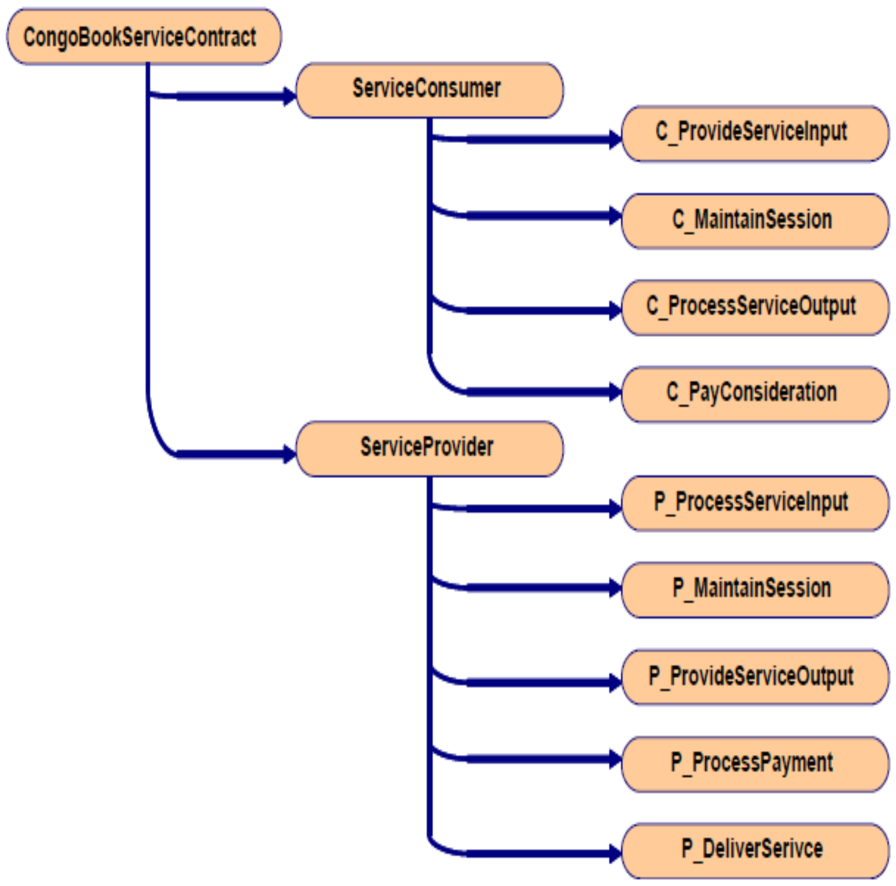


Figure 4.2: CongoBookServiceContract Hierarchy Page

Another sequence could be just *C_ProvideBookName*, *P_LocateBook*, where the consumer either just wants to search without intention to pursue the purchase; or the book may be out of stock. Other valid action sequences are omitted due to the page limit.

4.2.4.3 Reasoning of Congo Book Service Contract Model

With the valid action sequences identified, we now add the action sequence instances into the ontology. Then we input some evidence instances to simulate the collection of action evidences in the service contract KB. Finally, we create rules to reason the com-

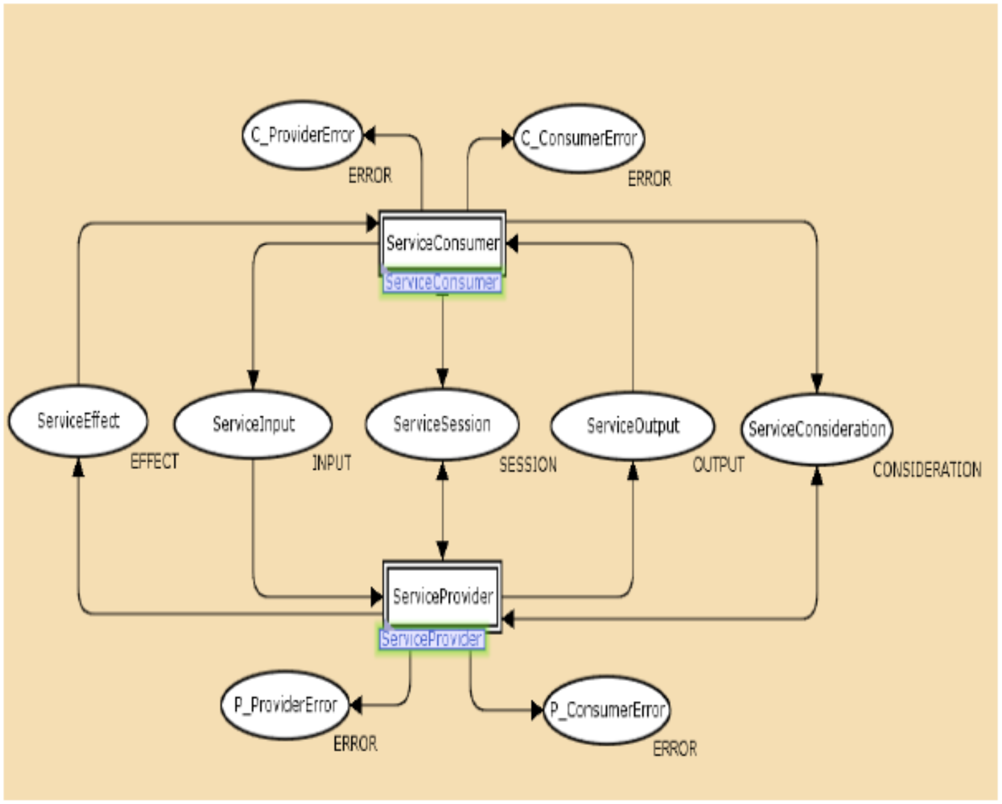


Figure 4.3: CongoBookServiceContract Top Page

pletion of obligations and the states of the contract execution. For example, axiom 1 for *CongoBookServiceContract* can be input in SWRLTab as:

$ServiceContractExecution(?x) \wedge execute(?x, ?y) \wedge specifiedObligation(?y, ?z) \wedge mustDo(?z, ?a) \wedge verifiedBy(?a, ?b) \wedge produceEvidence(?x, ?b) \rightarrow fulfilledObligations(?x, ?z)$

Once the ontology and rules are finally built, we invoke Pellet reasoner to classify the TBox and check consistency of the ABox. Then through the SWRLJessTab, we convert OWL individuals/SWRL rules to Jess facts/rules respectively; and use Jess engines to do reasoning which can entail new Jess facts. Fig. 4.6 shows an example of the reasoning result. As we can see, the Jess engine inferred a lot of facts about fulfilled obligations based on evidence produced by contract execution instance

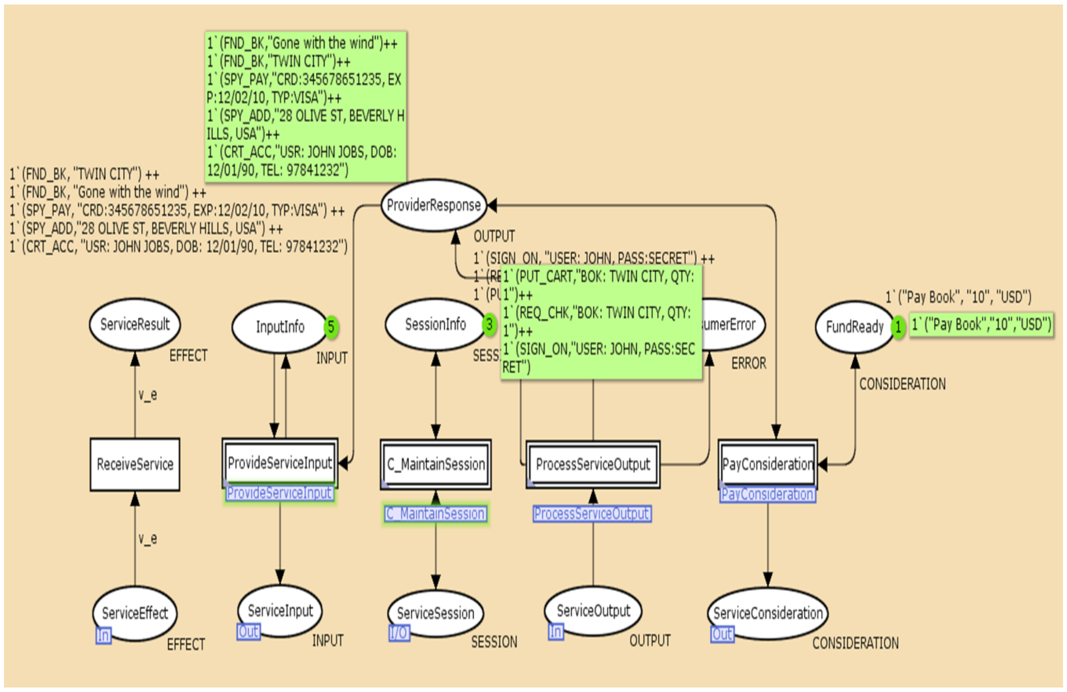

1`(FND_BK,"Gone with the wind")++
1`(FND_BK,"TWIN CITY")++
1`(SPY_PAY,"CRD:345678651235, EXP:12/02/10, TYP:VISA")++
1`(SPY_ADD,"28 OLIVE ST, BEVERLY HILLS, USA")++
1`(CRT_ACC,"USR: JOHN JOBS, DOB: 12/01/90, TEL: 97841232")
1`(FND_BK, "TWIN CITY") ++
1`(FND_BK, "Gone with the wind") ++
1`(SPY_PAY, "CRD:345678651235, EXP:12/02/10, TYP:VISA") ++
1`(SPY_ADD,"28 OLIVE ST, BEVERLY HILLS, USA") ++
1`(CRT_ACC, "USR: JOHN JOBS, DOB: 12/01/90, TEL: 97841232")
ProviderResponse
OUTPUT
1`(SIGN_ON, "USER: JOHN, PASS:SECRET") ++
1`(PUT_CART,"BOK: TWIN CITY, QTY: 1")++
1`(REQ_CHK,"BOK: TWIN CITY, QTY: 1")++
1`(SIGN_ON,"USER: JOHN, PASS:SECRET")
1`("Pay Book", "10", "USD")
1`("Pay Book","10","USD")
ServiceResult
InputInfo
5
SessionInfo
3
umerError
FundReady
1
EFFECT
INPUT
SESS
ERROR
CONSIDERATION
v_e
ReceiveService
ProvideServiceInput
C_MaintainSession
ProcessServiceOutput
PayConsideration
v_e
ServiceEffect
In
EFFECT
ServiceInput
Out
INPUT
ServiceSession
I/O
SESSION
ServiceOutput
In
OUTPUT
ServiceConsideration
Out
CONSIDERATION

Figure 4.4: ServiceConsumer Page

$CongoBookServiceExecution\_1$. It also concluded that the $CongoBookServiceExecution\_1$ is in completion status based on the highlighted rule. After reasoning, we can use SPARQL or SQWRLQueryTab to query the KB.

4.2.4.4 Discussion

So far we have demonstrated the modelling of a semantic service contract model for an online service using existing tools. Clearly this service contract model can be applied to other SaaS and Cloud services to support service obligation disclosure, monitoring and tracking of non-compliance. The strengths of our model are that, firstly, it addresses the accountability concerns through a formal construct, which is firmly grounded on the intuitive concept of a service contract in the real world. Secondly, the construct is based on rigorous formalisms, like OWL-DL/DL-Safe SWRL and CP-Nets. The former allows machine interpretation and web accessibility, yet provides computation decidability. The later facilitates visual modelling and simulation, yet is

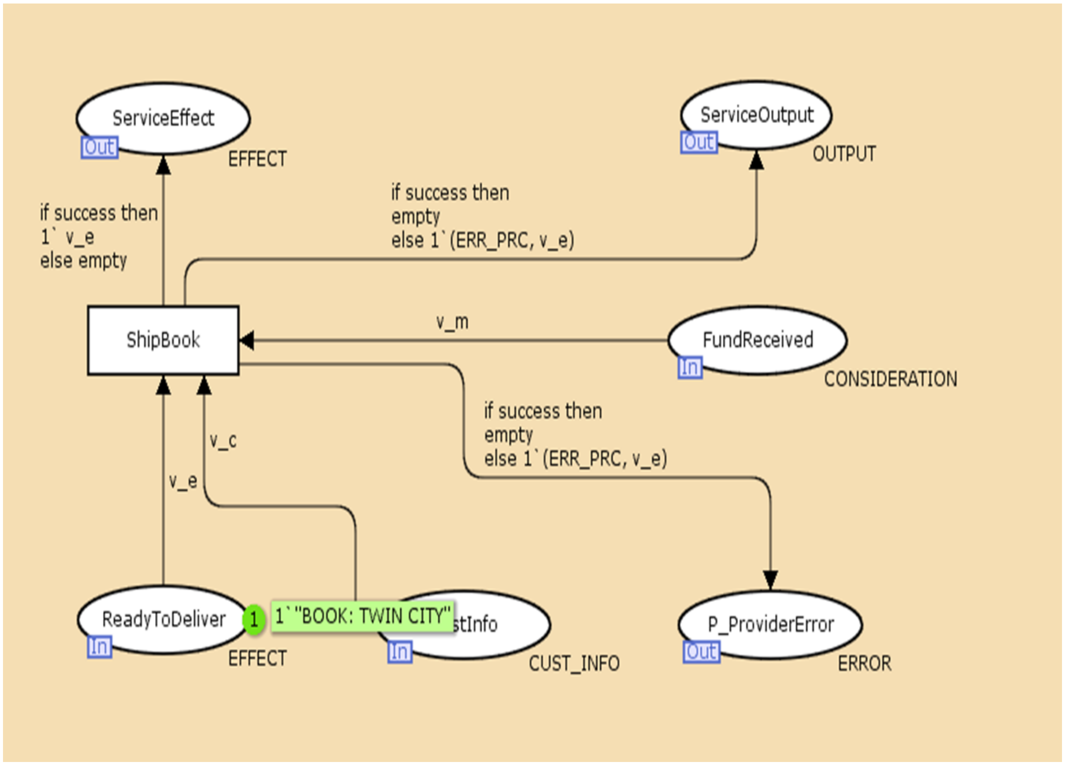
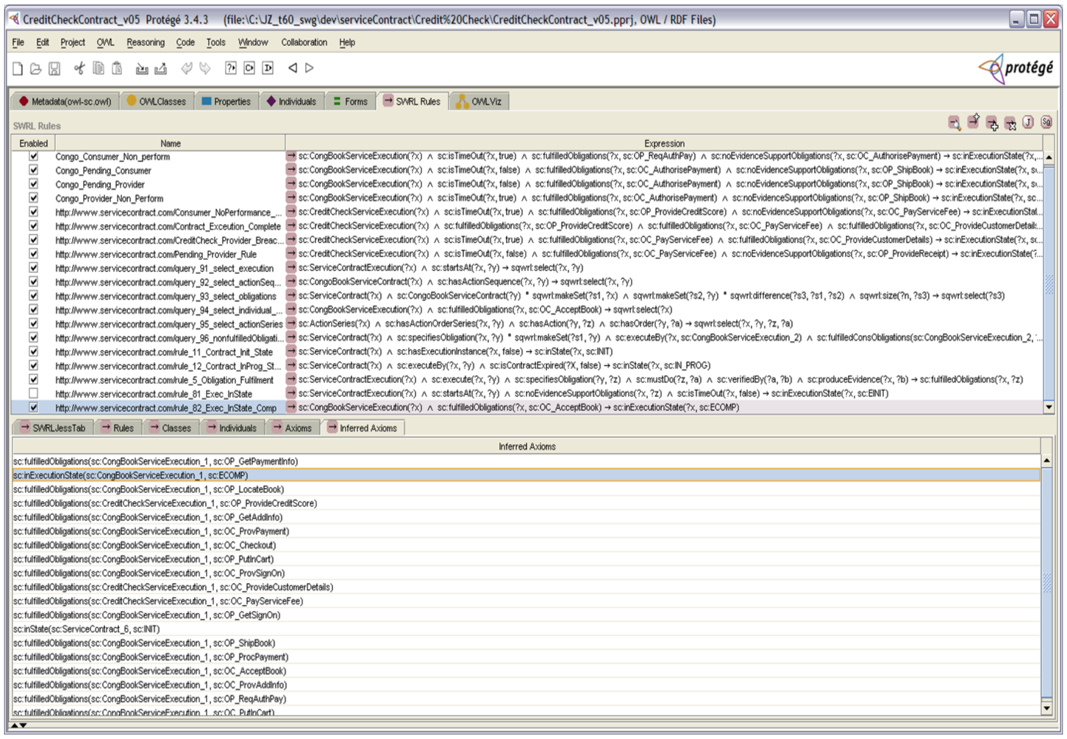


Figure 4.5: CongoBookServiceContract Deliver Service

backed by solid mathematics. Lastly, our approach can be supported by existing tools and readily applied in practical applications, rather than being just a theoretical model. We also notice that one weakness of the model is in its expressiveness and reasoning power. For example, it is quite difficult to reason whether a contract has expired, since SWRL's Temporal built-in [154] currently does not provide the support for current time (*e.g.* now). Another issue is that the translation from OWL-DL to Jess is not complete; the anonymous individuals will not be translated to Jess facts [130]. However this limitation does not impact the completeness of our model because anonymous individual information is not relevant to our model. The last issue is that the inferred facts from Jess reasoning can potentially make the initial OWL-DL ontology inconsistent [130]. We suggest that a hybrid approach that leverages strengths from different formalisms can best address the weaknesses. We separate our service contract reasoning to design-time reasoning and run-time reasoning stages. We only use DL reasoner


CreditCheckContract_v05 Protégé 3.4.3 (file:\C:\JZ_t60_swg\dev\serviceContract\Credit%20Check\CreditCheckContract_v05.pprj, OWL / RDF Files)
File Edit Project OWL Reasoning Code Tools Window Collaboration Help
protégé
Metadata(owl-sc.owl) OWLClasses Properties Individuals Forms SWRL Rules OWLViz
SWRL Rules
SWRLJessTab Rules Classes Individuals Axioms Inferred Axioms

Enabled	Name	Expression
✓	Congo_Consumer_Non_perform	sc:CongBookServiceExecution(?x) ∧ sc:isTimeOut(?x, true) ∧ sc:fulfilledObligations(?x, sc:OP_ReqAuthPay) ∧ sc:noEvidenceSupportObligations(?x, sc:OC_AuthorisePayment) → sc:inExecutionState(?x,...
✓	Congo_Pending_Consumer	sc:CongBookServiceExecution(?x) ∧ sc:isTimeOut(?x, false) ∧ sc:fulfilledObligations(?x, sc:OC_AuthorisePayment) ∧ sc:noEvidenceSupportObligations(?x, sc:OP_ShipBook) → sc:inExecutionState(?x, s...
✓	Congo_Pending_Provider	sc:CongBookServiceExecution(?x) ∧ sc:isTimeOut(?x, false) ∧ sc:fulfilledObligations(?x, sc:OC_AuthorisePayment) ∧ sc:noEvidenceSupportObligations(?x, sc:OP_ShipBook) → sc:inExecutionState(?x, s...
✓	Congo_Provider_Non_Perform	sc:CongBookServiceExecution(?x) ∧ sc:isTimeOut(?x, true) ∧ sc:fulfilledObligations(?x, sc:OC_AuthorisePayment) ∧ sc:noEvidenceSupportObligations(?x, sc:OP_ShipBook) → sc:inExecutionState(?x, sc...
✓	http://www.servicecontract.com/Consumer_NoPerformance_...	sc:CreditCheckServiceExecution(?x) ∧ sc:isTimeOut(?x, true) ∧ sc:fulfilledObligations(?x, sc:OP_ProvideCreditScore) ∧ sc:noEvidenceSupportObligations(?x, sc:OC_PayServiceFee) → sc:inExecutionStat...
✓	http://www.servicecontract.com/Contract_Exceution_Complete	sc:CreditCheckServiceExecution(?x) ∧ sc:fulfilledObligations(?x, sc:OP_ProvideCreditScore) ∧ sc:fulfilledObligations(?x, sc:OC_PayServiceFee) ∧ sc:fulfilledObligations(?x, sc:OC_ProvideCustomerDetail...
✓	http://www.servicecontract.com/CreditCheck_Provider_Breac...	sc:CreditCheckServiceExecution(?x) ∧ sc:isTimeOut(?x, true) ∧ sc:fulfilledObligations(?x, sc:OC_PayServiceFee) ∧ sc:fulfilledObligations(?x, sc:OC_ProvideCustomerDetails) → sc:inExecutionState(?x, s...
✓	http://www.servicecontract.com/Pending_Provider_Rule	sc:CreditCheckServiceExecution(?x) ∧ sc:isTimeOut(?x, false) ∧ sc:fulfilledObligations(?x, sc:OC_PayServiceFee) ∧ sc:noEvidenceSupportObligations(?x, sc:OP_ProvideReceipt) → sc:inExecutionState(?...
✓	http://www.servicecontract.com/query_91_select_execution	sc:ServiceContractExecution(?x) ∧ sc:startsAt(?x, ?y) → sqwrl:select(?x, ?y)
✓	http://www.servicecontract.com/query_92_select_actionSeq...	sc:CongoBookServiceContract(?x) ∧ sc:hasActionSequence(?x, ?y) → sqwrl:select(?x, ?y)
✓	http://www.servicecontract.com/query_93_select_obligations	sc:ServiceContract(?x) ∧ sc:CongoBookServiceContract(?y) ° sqwrl:makeSet(?s1, ?x) ∧ sqwrl:makeSet(?s2, ?y) ° sqwrl:difference(?s3, ?s1, ?s2) ∧ sqwrl:size(?n, ?s3) → sqwrl:select(?s3)
✓	http://www.servicecontract.com/query_94_select_individual_...	sc:CongBookServiceExecution(?x) ∧ sc:fulfilledObligations(?x, sc:OC_AcceptBook) → sqwrl:select(?x)
✓	http://www.servicecontract.com/query_95_select_actionSeries	sc:ActionSeries(?x) ∧ sc:hasActionOrderSeries(?x, ?y) ∧ sc:hasAction(?y, ?z) ∧ sc:hasOrder(?y, ?a) → sqwrl:select(?x, ?y, ?z, ?a)
✓	http://www.servicecontract.com/query_96_nonfulfilledObligati...	sc:ServiceContract(?x) ∧ sc:specifiesObligation(?x, ?y) ° sqwrl:makeSet(?s1, ?y) ∧ sc:executeBy(?x, sc:CongBookServiceExecution_2) ∧ sc:fulfilledConsObligations(sc:CongBookServiceExecution_2, ...
✓	http://www.servicecontract.com/rule_11_Contract_Init_State	sc:ServiceContract(?x) ∧ sc:hasExecutionInstance(?x, false) → sc:inState(?x, sc:INIT)
✓	http://www.servicecontract.com/rule_12_Contract_InProg_St...	sc:ServiceContract(?x) ∧ sc:executeBy(?x, ?y) ∧ sc:isContractExpired(?X, false) → sc:inState(?x, sc:IN_PROG)
✓	http://www.servicecontract.com/rule_5_Obligation_Fulfilment	sc:ServiceContractExecution(?x) ∧ sc:execute(?x, ?y) ∧ sc:specifiesObligation(?y, ?z) ∧ sc:mustDo(?z, ?a) ∧ sc:verifiedBy(?a, ?b) ∧ sc:produceEvidence(?x, ?b) → sc:fulfilledObligations(?x, ?z)
☐	http://www.servicecontract.com/rule_81_Exec_InState	sc:ServiceContractExecution(?x) ∧ sc:startsAt(?x, ?y) ∧ sc:noEvidenceSupportObligations(?x, ?z) ∧ sc:isTimeOut(?x, false) → sc:inExecutionState(?x, sc:EINIT)
✓	http://www.servicecontract.com/rule_82_Exec_InState_Comp	sc:CongBookServiceExecution(?x) ∧ sc:fulfilledObligations(?x, sc:OC_AcceptBook) → sc:inExecutionState(?x, sc:ECOMP)

```
Inferred Axioms
sc:fulfilledObligations(sc:CongBookServiceExecution_1, sc:OP_GetPaymentInfo)
sc:inExecutionState(sc:CongBookServiceExecution_1, sc:ECOMP)
sc:fulfilledObligations(sc:CongBookServiceExecution_1, sc:OP_LocateBook)
sc:fulfilledObligations(sc:CreditCheckServiceExecution_1, sc:OP_ProvideCreditScore)
sc:fulfilledObligations(sc:CongBookServiceExecution_1, sc:OP_GetAddInfo)
sc:fulfilledObligations(sc:CongBookServiceExecution_1, sc:OC_ProvPayment)
sc:fulfilledObligations(sc:CongBookServiceExecution_1, sc:OC_Checkout)
sc:fulfilledObligations(sc:CongBookServiceExecution_1, sc:OP_PutInCart)
sc:fulfilledObligations(sc:CongBookServiceExecution_1, sc:OC_ProvSignOn)
sc:fulfilledObligations(sc:CreditCheckServiceExecution_1, sc:OC_ProvideCustomerDetails)
sc:fulfilledObligations(sc:CreditCheckServiceExecution_1, sc:OC_PayServiceFee)
sc:fulfilledObligations(sc:CongBookServiceExecution_1, sc:OP_GetSignOn)
sc:inState(sc:ServiceContract_6, sc:INIT)
sc:fulfilledObligations(sc:CongBookServiceExecution_1, sc:OP_ShipBook)
sc:fulfilledObligations(sc:CongBookServiceExecution_1, sc:OP_ProcPayment)
sc:fulfilledObligations(sc:CongBookServiceExecution_1, sc:OC_AcceptBook)
sc:fulfilledObligations(sc:CongBookServiceExecution_1, sc:OC_ProvAddInfo)
sc:fulfilledObligations(sc:CongBookServiceExecution_1, sc:OP_ReqAuthPay)
sc:fulfilledObligations(sc:CongBookServiceExecution_1, sc:OC_PutInCart)
```

Figure 4.6: Congo Book Service Contract Reasoning Results

for concept consistency checking at design-time, while the Jess rule engine is used to maintain the KB at run-time. Other tasks, such as comparison and computation, can be better handled via a combination of traditional programming model and KB query. For example, when checking whether or not a contract has expired, it is easy to query the KB, get the end time, then compare it to the current time in a traditional program.

4.3 Accountable State Transfer Architecture

In this section, we are going to apply our semantic service contract model to augment a mainstream SOA implementation architecture – Representational State Transfer (REST) with accountability mechanism. The extended architecture is called Accountable State Transfer (AST) architecture. We first briefly introduce the preliminary of REST in the section below.

4.3.1 Preliminary on Representational State Transfer

4.3.1.1 REST Architecture and the Previous Extension Efforts

According to Fielding, REST behaves like a virtual state machine, where the state transition happens when the user selects links, resulting in the next state of the application being transferred to the user [62]. REST lays down the foundation for the Web architecture. There are a number of efforts to extend REST to address certain aspects of requirements. In [56], the authors suggest the concept of "Computational Transfer" and propose Computational REST architecture (CREST). The idea is to use AJAX and mashups as mechanisms for framing responses as interactive computations or for "synthetic redirection" and service composition. CREST is essentially the Web2.0 style of Web architecture. To strengthen RESTs capability in supporting enterprise requirements, the authors in [104] extend REST to induce four properties: events, routes, locks and estimates. They derive four new REST styles (ARREST, ARREST+E, ARREST+D and ARRESTED) optimised for each of the above four types of resources. However, currently REST and its extensions do not address the accountability requirements. In particular, they do not support the concept of service contract and also do not have any ability to provide justification on the action of state transfer.

4.3.1.2 REST Guiding Architectural Principles

The key characteristic of the REST architecture is that it takes a "resource view" of the world. The RESTful principles described in [62] and elaborated in [56] are:

RESTFul Principles

RP1 : Resource can be identified by an URI;

RP2 : Separation of the abstract resource and its concrete representations;

RP3 : Stateless interaction, each interaction contains all the necessary context information and meta-data;

RP4 : Small number of operations, with distinct semantics based on HTTP methods: safe operations (Get, Head, Options, Trace); non-safe, idempotent operations (Put, Delete); and non-safe, non-idempotent operation (Post);

RP5 : Idempotent operations and representation metadata support cache;

RP6 : Promote the presence of intermediaries such as proxies, gateways or filters to alter or restrict request and response based on metadata.

Following these principles ensures that our AST architecture retains REST's scalability and performance.

4.3.2 Service Contract as Foundation for Enabling Accountability

The key accountability concerns addressed in this thesis are: obligations disclosure; execution status tracking based on evidence; and the ability to provide justification and explanation of actions in relation to a pre-established contract. Compared to the existing accountability models in the IT literature, a key differentiation of our approach is that we position the service contract as the foundation underpinning the accountability concerns in an SOA environment.

4.3.3 Architectural Decisions in AST Implementation

In contrast to a lot of existing service contract models in the literature that only provide theoretical models, a key principle of our approach is to ensure the practicality of model implementation. To achieve that, trade-offs need to be made in implementation decisions. Followings are the architectural decisions for our AST architecture:

AD1 : Implement service contract as an ontology and store service contract execution instances in a knowledge base (KB); in addition, add rules to allow reasoning of the contract execution state in the KB. An ontology describes the concepts in the domain and the relationships held between those concepts. Building upon an ontology, our service contract model further needs rule capability for contract state reasoning.

AD2 : Take a resource view on service contracts and use URI to uniquely identify elements in the service contract. Thus the elements in a service contract ontology can be referred via URIs during service invocation.

AD3 : Separate the service contract and service contract execution concepts. The rationale behind decision 3 is that in a SaaS or Cloud computing environment, a service can be executed for multiple times during the valid period of the underlying business contract. For example, a Credit Check service contract may last for one year; during the year the service can be executed for multiple times. Each execution is an execution instance of the service contract. The separation of contract and contract execution concepts allows contract execution tracking, which is not seen in most of the existing service contract models.

AD4 : Adopt a hybrid reasoning approach that leverages strengths from different formalisms and technologies.

The last decision applies in the area of designing the reasoning mechanism for our service contract KB. We need to consider the expressiveness power and computational complexity of the underlying formalisms such as OWL-DL and SWRL; also take into account the availability of the tooling support to make the optimal design decision.

4.3.4 Accountable State Transfer Architecture

4.3.4.1 Extending REST to Support Accountability

In a traditional REST service, both the consumer and the provider can not be held accountable for their actions during the representational state transfer. At the client side, the client consumes the provider's services by following the URL links to get representational states from some resources under the provider's control. But the client does not know precisely the linkage between the state transfer and the provider's obligations. The provider also does not know exactly why the client makes a particular request. In a service-oriented environment like cloud, fundamentally each service is linked to a pre-established business contract between the service provider and the service consumer. Therefore, to establish accountability in the REST interaction, it is important to link the interaction to a particular contract context, so both the consumer and the provider can track the performance of the contract, understand the reasons behind each request and respond with regard to that very contract. For example, suppose a SaaS provider provides a credit check service by exposing some REST interfaces. A service consumer needs to establish a binding contract with the service provider before he/she can consume the service. Once the contract is established, the service consumer can invoke the REST service to check a particular customer's credit score. However, under the traditional REST, both the service consumer and service provider have no ways to know which contract the service is related to in runtime, let along track the progression status and determine which party breaches the contract.

In order to address the above problem, we extend the REST architecture by bringing in the service contract context as the meta-data during the interaction between the consumer and provider. The contract context information includes the name of the service contract, the current contract execution instance status and the overall contract progression status. In REST, each element of the service contract information can be treated as a resource, identified by an URI. Therefore the contract context meta-data can be simply referred to by URIs in HTTP headers. Also the contract progression and performance can be monitored by a trusted-third party (TTP). We call this style of REST extension Accountable State Transfer architecture.

4.3.4.2 The Accountable State Transfer (AST) Architecture

AST architecture introduces two extra components called *sContractMonitor* and *sContractManager* in addition to the traditional REST architecture components. *sContractMonitor* monitors the interactions between the consumer and the provider, and then feeds events to *sContractManager* through a low-coupling queuing mechanism. *sContractManager* determines the current contract execution instance's status based on the rules prescribed in the service contract and the events fed from *sContractMonitor*. It also maintains a service contract KB for all the contract execution instances so it can reason the overall contract status.

AST architecture can be classified into two categories. One is a centralised AST and the other is a peer-to-peer AST. In a centralised AST, it can be further categorised to two styles: one is an in-line TTP AST and the other is an on-line TTP AST. An in-line TTP AST's *sContractMonitor* acts as an HTTP proxy, seeing through all the interactions between the consumer and the provider. Based on the contract meta-data on the HTTP headers and the body message, it can verify whether the obligations have been met by checking the prescribed evidence. Then it generates the assertion events to *sContractManager*. *sContractManager*'s reasoner component determines the service contract execution status and maintains an up-to-date service contract execution KB based on the rules defined in the service contract. Fig. 4.7 illustrates the in-line TTP AST model.

In an on-line AST Model, the service contract monitor remotely monitors the consumer and the provider separately. In a peer-to-peer AST model, each party will have its own service contract monitor and manager, and needs an arbitrator reasoner for dispute resolution.

4.3.4.3 AST Protocols

In order to bring in service contract context information during REST interactions, the following items are defined for communicating the contract meta-data in the basic AST protocol:

Accountable State Transfer Protocol

AP1 : Refer to service contract during service invocation:

HTTP Header: sContract: sContract_URI;

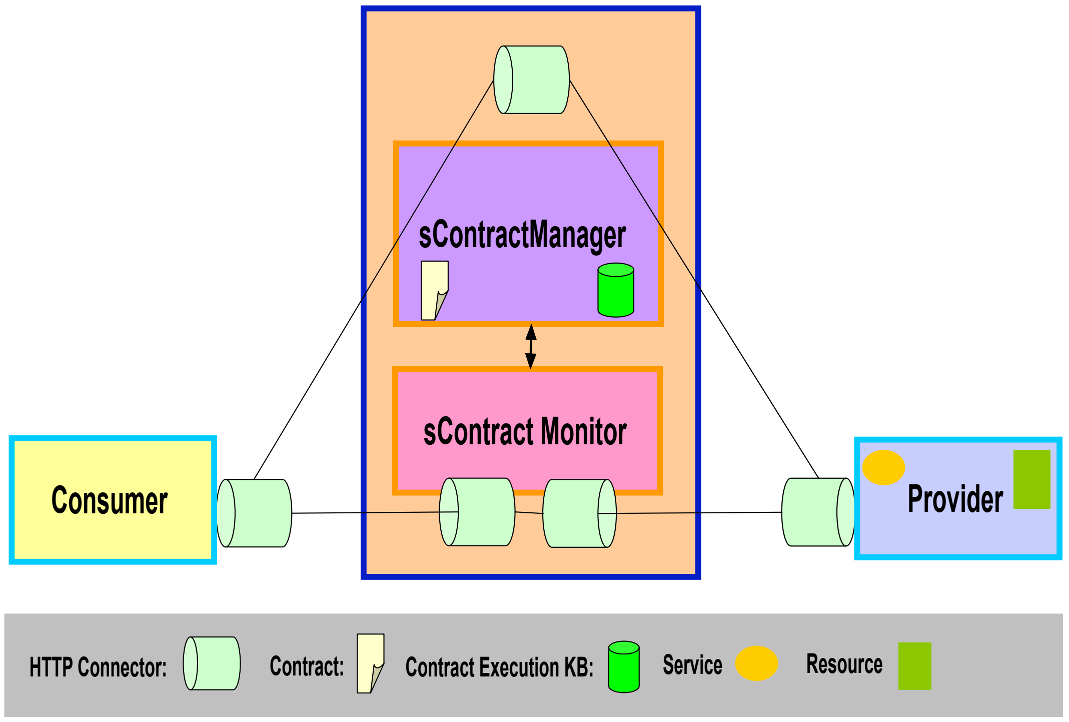


Figure 4.7: In-Line TTP AST Architecture

AP2 : Add service contract meta-data to a service request:

HTTP Header: Required-Obligations: action_URI list;

AP3 : Add service contract meta-data to a service response:

HTTP Header: Met-Obligations: action_URI list:

AP4 : Query on contract execution status:

GET sContract_KB_URI?queryString;

AP5 : Notify contract breach or execution abnormality:

POST involvedParty_URI with XML payload indicating *sContract* or *sContractExecution* status.

The HTTP header in **AP1** can be used in each REST interaction to establish linkage to a service contract. **AP2**'s HTTP header can be used when the consumer sends out a request, indicating the request is relating to the provider's obligation as specified in the service contract. **AP3** can be used when the provider replies with a response, indicating that the response is relating to the fulfilled obliged actions. **AP4** enables REST client and server to query the *sContractManager* for contract execution status. **AP5** allows *sContractManager* to notify a client or the server on the contract execution status.

Now we use an example to illustrate how the AST works. Suppose a service provider p_r $(p_r \in P)$ has signed a contract sc with a consumer p_c $(p_c \in P)$ to provide a Credit Check service defined by s for a period of T. The contract defines the terms relating to Credit Check in a definition set D. The contract prescribes the provider's obligation as O_p and the consumer's obligation as O_c. They are:
$O_p = \{(P\_checkCredit, E\_creditEvidence), (P\_returnError, E\_ErrorEvidence)\}$.
$O_c = \{(C\_provideInput, E\_inputEvidence), (C\_payFee, E\_feeEvidence)\}$.
The valid action sequences are defined as either:
"*C_provideInput*, *P_checkCredit*, *C_payFee*" or "*C_provideInput*, *P_returnError*".
Also the contract defines a set of rules R to determine the contract status st $(st \in S)$, based on the evidence of the fulfilled obligations.

There is no concept of service contract in traditional REST. Both service provider and service consumer rely on other means (mostly off-line and manual) to know the performance status of the contract. With AST, the contract performance can be tracked while executing the service. Moreover, both client and server understand the "why" behind the request and response (representational state transfer) from a service contract perspective. For example, when the consumer invokes the Credit Check service, he/she issues the following request with the Http headers below:

GET /credit_chk.jsp?fname=jon&lname=bond&id=102435 HTTP 1.1 Host www.creditcheck.com

HTTP headers:

sContract: http://sContractManager.com/creditcheck.owl

Required-Obligations: #P_checkCredit, #P_returnError

With these HTTP headers, the consumer links the request to a pre-established contract, also states that the request is related to the provider's obligations *P_checkCredit* and *P_returnError* as prescribed in the contract. When the server responds, it adds the

HTTP headers to further explain the response in relation to the contract:

```
HTTP/1.1 200 OK
```

HTTP headers:

```
sContract: http://sContractManager.com/creditcheck.owl

Met-Obligations: #P_checkCredit

Required-Obligations: #C_payFee
```

4.3.5 Implementation Of AST Architecture

4.3.5.1 Languages for Specifying the Service Contract Model

The ontology underpinning our service contract model can be specified using Web Ontology Language (OWL), which is recommended by W3C as the standard for representing ontologies on the Web. As per the guiding principles in Section 4.3.1.2, OWL-DL is chosen to specify our service contract model since it has the better tradeoff between expressiveness and decidability. The other benefit of using OWL-DL is that the consistency of the service contract can be validated using proven DL reasoners, such as RACER, KAON2, PELLET, etc. However, OWL-DL has limitations. In particular, it has the well known "hasUncle" problem; *i.e.*, it is impossible for OWL-DL to describe the role chain of *hasParent* and *hasBrother* leading to the *hasUncle* role. To address this limitation, we leverage Semantic Web Rule Language (SWRL) for defining rules to do contract state reasoning on top of OWL-DL.

4.3.5.2 Service Contract Ontology and Axioms

We apply the OWL-SC model to capture the fundamental aspect of a service contract. The service contract ontology of Fig. 4.1 can be represented using OWL-SC. An *sc-Contract* class has contract term definitions (*Definitions* class); it involves *Party* class, which has subclasses of *Provider* and *Consumer*. Each party has *Obligation* which consists of multiple *Action* and *Evidence* pairs. A domain specific contract class like

CreditCheckContract inherits from the generic *scContract* class. Such domain specific contract instances may be executed multiple times. Each execution is an instance of *scContractExecution* class. The *scContractExecution* instance executes *Obligations* as defined in the *scContract* and produces *Evidence* instances. If each *Action* instance can be proven by the respective *Evidence* instance, then the obligation is fulfilled. Otherwise either *Provider* or *Consumer* may breach the contract depending on the specific contract rules, which can be defined as the axioms for a domain specific service contract model.

We here list some axioms that determine the contract state for our generic service contract model. For domain specific service contracts, these axioms can be extended, overwritten, or new axioms can be developed, based on the specific terms and conditions of the underlying contract.

Generic Contract Axioms

Axiom 2: $scContract(?x) \wedge hasExecutionInstance(?x, false) \rightarrow inState(?x, INIT)$

Axiom 3: $scContract(?x) \wedge executedBy(?x, ?y) \wedge isContractExpired(?x, false) \rightarrow$
$inState(?x, IN\_PROG)$

Axiom 4: $scContractExecution(?x) \wedge startsAt(?x, ?y) \wedge$
$noEvidenceSupportObligations(?x, ?z) \wedge$
$isTimeOut(?x, false) \rightarrow inExecutionState(?x, EINIT)$

Axiom 5: $scContractExecution(?x) \wedge execute(?x, ?y) \wedge specifiesObligation(?y, ?z) \wedge$
$mustDo(?z, ?a) \wedge verifiedBy(?a, ?b) \wedge produceEvidence(?x, ?b) \rightarrow fulfilledObligations(?x, ?z)$

Axiom 2 states that if an *scContract* instance does not have any execution instance, then the *scContract* is in the initial (*INIT*) state. Axiom 3 states that if the *scContract* instance is executed by some execution instances and the contract has not expired, then the service contract state is in-progress (*IN_PROG*). Axiom 4 says if the instance of *scContractExecution* starts at a particular time, but no evidence is produced to prove the fulfilment of obligations, and the execution is not time out yet, then the contract execution instance is in initial (*EINIT*) state. Axiom 5 determines whether a particular obligation is fulfilled based on the collected evidence.

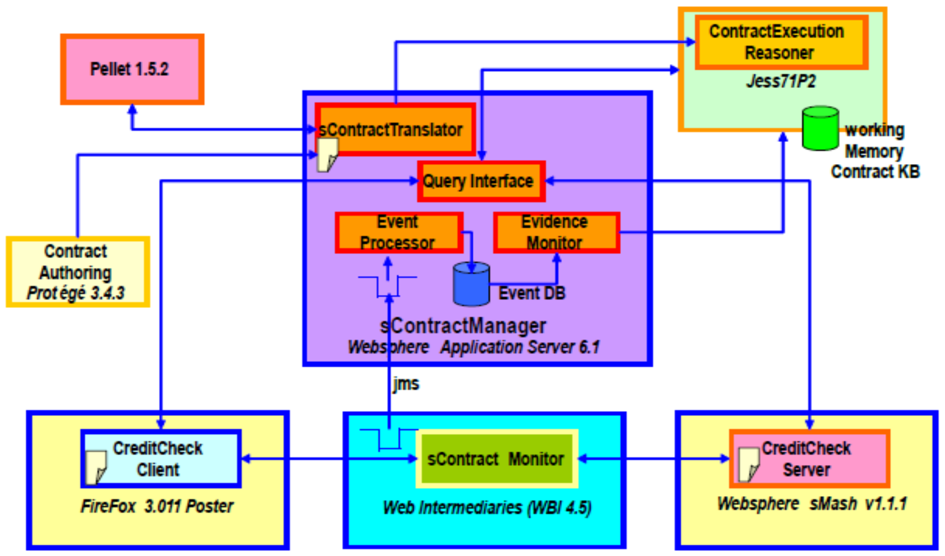


Figure 4.8: Prototype in-line TTP AST Implementation

4.3.5.3 Prototype Implementation

Overall Prototype Architecture

Fig. 4.8 depicts the Credit Check service prototype that implements the in-line TTP AST in Fig. 4.7. The main components are described below:

CreditCheck Client: We used Firefox Poster to simulate a Credit Check client. Firefox Poster provides an intuitive interface for sending REST requests with user defined HTTP Headers.

sContractMonitor: a contract monitor that is built on a RESTful component Web intermediary. We used IBM's Web Intermediaries Development Kit 4.5 (WBI DK) [15] as the underlying Web intermediary platform, creating a Monitor plug-in to track the HTTP messages. The messages will be sent to *sContractManager* through JMS.

CreditCheck Server: A Credit Check server is developed and hosted in Websphere sMash [123], which provides an environment for developing and hosting REST applications.

Contract Authoring: Protege 3.4.3 is used as the service contract authoring tool.

SWRLtab in Protege is used for SWRL rule authoring.

Pellet 1.5.2: Pellet[5] 1.5.2 is the DL reasoner that is used to classify the terms in service contract and check contract consistency at design time.

sContractManager: the contract monitor is implemented in a Websphere Application Server (WAS). It consists of *sContractTranslator*, *QueryInterface*, *EventProcessor* and *EvidenceMonitor*. *sContractTranslator* converts the service contract from OWL-DL/SWRL to Jess facts/rules using XSLT, then sends them to *ContractExecutionReasoner*. *EventProcessor* picks up the raw monitoring data, storing evidence data into an event database. Then *EvidenceMonitor* checks if the evidence is valid, if so, it sends assertions to *ContractExecutionReasoner*. Finally the *QueryInterface* allows contract status query from both the client and the server.

ContractExecutionReasoner: This component receives Jess facts or rules, and then invokes Jess71p2[6] to do reasoning, maintaining the contract execution KB in Jess's working memory.

Implementation of Hybrid Reasoning Mechanism

Based on architectural decision 4 outlined in Section 4.3.3, we adopt a hybrid approach to reasoning. In the contract authoring stage, a DL reasoner like Pellet will be used for normal TBox and ABox reasoning in design time. After the service contract is developed, the OWL-DL ontology will be translated to Jess facts via one XSLT file, while the SWRL rules will be translated to Jess user-defined rules via another XSLT file. Additionally, we need to import pre-defined Jess rules, which are transformational implementations for OWL semantics [131]. Then the Jess facts and (pre-defined and user-defined) rules will be fed into the Jess engine, taking advantage of the fast Rete algorithm for contract state reasoning at runtime. In our prototype, we created a *CreditCheckServiceContract* based on *scContract* in Fig. 4.8. The contract is between service provider *CreditBureau* and consumer *MortgageBank*. *CreditBureau*'s obligation is to complete actions *P_checkCredit* or *P_returnError* if exception occurs.

[5]https://github.com/complexible/pellet

[6]http://www.jessrules.com/

MortgageBank's obligation is to complete actions *C_provideInput* and *C_payFee*. The actions need to be proven by evidence which is also defined in the service contract ontology. This contract instance will be executed multiple times during the valid contract period. Each execution instance is an instance of *CreditCheckExecution* class. Two example rules used to determine if the service participants breach the obligation are listed below.

Axiom 6: $CreditCheckServiceExecution(?x) \wedge isTimeOut(?x, true) \wedge fulfilledObligations(?x, OC\_ProvideCustomerDetails) \wedge noEvidenceSupportObligations(?x, OP\_ProvideCreditScore) \wedge noEvidenceSupportObligations(?x, OP\_ReturnError) \rightarrow inExecutionState(?x, EP\_NOPF)$

Axiom 7: $CreditCheckServiceExecution(?x) \wedge isTimeOut(?x, true) \wedge fulfilledObligations(?x, OP\_ProvideCreditScore) \wedge noEvidenceSupportObligations(?x, OC\_PayServiceFee) \rightarrow inExecutionState(?x, EC\_NOPF)$

Axiom 6 states that if *MortgageBank* has provided input for credit check, but *CreditBueau* has not provided a credit score nor returned an error; and the execution is timeout, mark the current contract execution instance as status *EP_NOPF* (Service Provider Non-Performing obligations). A similar rule defined in Axiom 7 is used to determine consumer non-performing obligations. Note that the reasoning power is limited by the expressiveness of OWL DL and SWRL, so normal programming logic is still needed to address the limits of DL reasoner and rules engine. For example, the predicate *isTimeOut* in Axiom 6 is very difficult for reasoners to decide because there is no current time concept in OWL-DL, nor is it provided in the SWRL's temporal built-in. However, it can be easily done in a Java program by checking the current time, and producing an assertion triple to the Jess engine. So our hybrid approach can be simply described as: DL reasoning at design time, Jess rule reasoning at runtime, with input assertions produced by a Java program.

Results and Discussion

The test environment is based on a PC with a duo-core 2.4 GHz Intel CPU, 2GB RAM running on Windows XP. After completing the design of the Credit Check service contract in Protege, the Pellet reasoner is invoked to check the consistency of the ontology. It took 1.68 seconds to classify the taxonomy and 3.81 seconds to check the consistency of the service contract model. Then through *sContractTranslator*, both OWL-DL and SWRL rules are translated into Jess facts and rules. It took 1562 milliseconds for running *sContractTranslator* to translate OWL facts to Jess facts, generating 1948 asserted triples. Jess took less than one second to reason the input Jess facts and Jess rules, generating 2621 inferred triples in its working memory. As the Jess's Rete algorithm is linear to the number of rules and polynomial to the number of objects [65], when the KB grows, we need to scale up the underlying environment to cater for the load. Once the translation is done, the *sContractManager* is waiting for the event collected by the *sContractMonitor*. Once *evidenceMonitor* picks up an event, it validates if it is an evidence for a particular obligation.

There is no noticeable performance impact on both client and server, mainly due to the decoupling of *sContractManager* and *sContractMonitor*. The *sContractMonitor* is just a read-only plugin installed in a Web proxy; which is a widely adopted pattern in today's internet environment.

The limitation of translating OWL-DL to Jess facts is documented in [130]. In our model, since we only use OWL-DL reasoner at design time to verify the consistency of concepts in the service contract, in addition we apply DL-Safe restrictions in our model; and we use the Jess rule engine for run time reasoning, hence our reasoning is sound and complete in each reasoning stage. Theoretically, we acknowledge the loss of information when combining the two reasoning paradigms with interfaces for translating OWL-DL to Jess facts. However, the loss of information in our model is about reasoning on anonymous individuals, and such anonymous individuals in rules are disregarded due to our adoption of the DL-safe restriction.

Another interesting issue is about negation. OWL-DL is based on an open world assumption and thus can not reason "negation as failure". In our model, we work around this problem by defining properties like *noEvidenceSupportObligations*. The evidence monitor *EvidenceMonitor* is responsible for generating a Jess assertion on this property if no evidence is found. Therefore, we can use the Jess rule engine to reason non-fulfilled obligations. This demonstrates the strengths of our hybrid reasoning approach.

4.4 Conclusion

In this chapter, we address the accountability gap in service computing by proposing a formal service contract model and extending the Representational State Transfer (REST) architecture to Accountable State Transfer (AST) architecture.

High accountability standards not only benefit service consumers as a whole, but also can be a differentiator for service providers. As such, it is paramount to enable accountability in SaaS and Cloud services. In contrast with existing approaches that address accountability issues at a technical protocol level, we address them at an architectural level through a pro-accountability service contract formalism, which closely mimics the contract concept in the commercial world. Furthermore, we apply our service accountability model to solve the accountability issues in the REST architecture.

REST is increasingly becoming a key architectural style, thanks to the growing popularity of the Web 2.0 technology. REST services also form a major part of the services offered through SaaS or Cloud computing. Thus, building an accountability mechanism into the REST architecture is crucial for the long-term viability of these new business models.

Our contribution from a service contract perspective can be therefore summarised as: firstly, we analyse the accountability management requirements for SaaS and Cloud services and define a formal construct for a pro-accountability service contract model, proposing unique concepts, such as service contract execution and action evidence, that are not seen in other service contract models. We adopt the decidable OWL-DL, coupling it with the enhanced action semantics and DL-Safe SWRL rules to represent the service contract construct, namely OWL-SC. We also propose a novel approach to map OWL-SC action semantics to a colored Petri-nets model, namely SC-CPN, and thus enable visual modelling, validation and simulation of the action model in

OWL-SC. We have used the Congo Book service as an example to demonstrate how to use existing tools to build a service contract model, and discussed the strengths and weaknesses of the current model, plus the recommended approach to address the weaknesses.

From the accountable service-oriented architecture perspective, our contributions can be summarised as: firstly, we outline the architectural principles and decisions for enabling accountability in an e-Services environment. Secondly, guided by those principles and decisions, we propose a novel AST architecture with an accountable state transfer protocol to enable service accountability, yet retain the scalability of the REST architecture. The new architecture seamlessly integrates service contract semantics into the traditional syntactic-based REST services. Thirdly, we apply our semantic service contract model to design a Credit Check domain specific service contract with a hybrid reasoning mechanism that leverages strengths from formalisms like DL, Rules and traditional programming language. The hybrid reasoning mechanism provides capabilities like temporal reasoning and negation as failure, that are not found in normal DL and SWRL. Moreover, it separates reasoning in the design-time stage and the runtime stage, taking into account both the expressiveness and the computational complexity of the underlying logic formalisms. Lastly, we provide a prototype implementation for a Credit Check service that demonstrates the practicality of AST architecture, proving that the new AST architecture can be implemented with existing products and technologies.

The new architecture allows service obligation disclosure, obligation tracking, and action justification in a stateless service environment. With such capabilities provided at the architectural level, effectively, service participants can be held accountable for each representational state transfer during service consumption.

Finally, we observe that the future work entails applying the service contract model to SOAP-based Web Services models and Enterprise Service Bus (ESB) solutions.

Chapter 5

Advanced Service Accountability – An Algebraic Approach

Chapter 4 presents a service contract model based on OWL-DL and coloured Petri-nets. However like some other service contract models (i.e., [125]) it can only represent an "ought-to-do" model, rather than representing the deontic semantics of an "ought-to-be" model, which means it can not distinguish whether a fault scenario is due to inaction or improper action. Moreover, the semantic approach lacks an effective way to describe a process. Although SC-CPN presents a Petri-net based process view, its main use is in contract model validation, and it is not suitable for using as a formalism for obligation process disclosure.

In this chapter, we present an alternative approach to building the advanced service accountability capabilities. We first extend dynamic logic to Dynamic Logic for Accountability (DLA) and use it as a basis for cloud service contract specification. In addition, we argue that, from an accountability perspective a cloud service is a proactive system that needs to be modelled differently from the traditional reactive systems. We extend traditional structural operational semantics to cater for modelling of actors as well as scenarios of inaction and exception in state transitions. This leads to the creation of a new form of a process algebra called Accountable Process Algebra (APA). We also propose an Obligation Flow Diagram (OFD) as a simple method for conflict resolution and verification for the contract specification. The accountable process model enables service obligation specification, validation, decomposition, machine-interpretation, monitoring and liability assignment, and ultimately facilitates accountability in cloud service consumption. Using the Amazon S3 service as a case study, we show how to address those known accountability problems by using our process model. Finally we discuss the applicability of our process model to cloud services in general.

The remainder of this chapter is structured as follows. The next section provides a comparison between our work and existing studies. This is followed in Section 5.2 by setting the theme for designing an accountable process model for cloud contract specification and execution. Next in Section 5.3, we extend the dynamic logic to dynamic logic for accountability (DLA) so we can represent contract obligations. Then in Section 5.4, by taking a proactive system view, we propose a novel approach to modelling cloud services in Section 5.5. In Section 5.6, we show how to apply the process model to the motivating example in Section 1.1.3 in Chapter 1 and thereafter extend it to other cloud services. Finally, we summarise our contributions and discuss future work in Section 5.7.

5.1 Preliminary

While Service Level Agreement (SLA) is an extensively researched topic and it can be represented by existing approaches such as WSLA, WS-Agreement, SLAng [170], in essence, the service level only covers non-functional requirements, missing the crucial functional requirements for business. Thus, the existing approaches neither enable the disclosure of service obligations, nor allow software agents to decide which party is responsible for what action, and which party is liable for what result. This is evident in today's cloud market, where there are no formal policy-based or SLA languages used in representing service contracts. Hence the consumer has no effective means to detect the violation of service obligations, and the service provider can hardly be held accountable. As such, currently the accountability of cloud services on the market is a serious concern [105]. This may become a major obstacle for enterprise customers to take up those cloud services. Therefore it is critical to build an accountable process model for cloud service contract specification and execution.

Such a model should provide:

1. A representation language that is declarative and machine interpretable so that the obligation statements in a cloud service contract can be checked and interpreted automatically;

2. A collaboration process and diagram notation that clearly illustrate the interaction behaviour between the service provider and the consumer, therefore facilitating reasoning about obligation fulfilment and liability assignment; and

3. A tool for enabling the validation of the consistency of the model.

With these objectives in mind, we propose an algebraic service contract model to achieve advanced service accountability. Compared to the related works discussed above, the distinguishing characteristics of our approach are highlighted as follows:

1. Our model supports SOW modeling, which is not yet seen in existing service contract models;

2. Our model is based on a formal, hybrid logic system – Dynamic Logic for Accountability;

3. The deontic semantics in our model is refined to suit accountability requirements. The obligatory operator does not qualify the action; rather it qualifies the status after performing the action. Also the permission operator is omitted, since it is not a real concern in accountability;

4. We extend structural operational semantics (SOS) and propose an Accountable Process Algebra (APA) to model cloud services based on a proactive system view rather than the traditional reactive system view;

5. Our approach combines a formal model with visual tools like OFD and BPMN2.0 based graphical collaboration diagram notation to allow verification, conflict resolution, obligation decomposition and liability tracking; and

6. We treat a cloud service as a proactive system and propose a new form of a process algebra – APA to model the service contract execution behaviour, addressing the crucial accountability concerns of the “5Ws” (*Who* has done *what* action, *what* went wrong and *who* should be liable to *whom*).

5.2 Towards an Accountable Process Model for Cloud Service

5.2.1 Defining Accountable Cloud Service

We first look at what accountability properties a service contract should have. A contract consists of normative statements that can be placed into four categories given below [50]:

1) The obligation statements, which are the actions that the actor must carry out;

2) The forbiddance statements, which are the actions that are prohibited;

3) The remedy statements, which are the remedies when obligation statements or forbiddance statements are violated; and

4) The permission statements, which are the actions that are allowed to do but not mandatory.

As illustrated in Fig 5.1, a service contract involves normative statements for the service provider and the service consumer respectively. From service definition in Definition 1, values are exchanged between the service provider and the service consumer. The obligations in the service contract tend to create the value whereas the forbiddance statements normally protect values. The remedy statements compensate the value and lastly the permission statements let the party enjoy the values. Out of the four categories, *obligation*, *forbiddance* and *remedy* statements are accountability concerns, because breaching those normative statements contributes to the violation of the contract. Therefore, the service providers and consumers need to be accountable for those normative statements in the contract. On the other hand, for the *permission* statements, whether the actions have been carried out or not does not impact the performance of the contract, thus the permission statements in the service contract are indifferent to service accountability.

In the example outlined in Section 1.1.3 in Chapter 1, the service contract of the traditional storage service comprises SOW and SLA. The obligated activities specified in the SOW such as storage and retrieval of data objects are examples of obligation statements in the service contract. The 99.9% availability requirement specified in the SLA is also an example of an obligation statement. Obligation statements create the values for service exchange. On the other hand, making sure the data are not lost, damaged or leaked to other people are examples of the forbiddance statements in the service contract. Forbiddance statements protect value in service execution. If not meeting the 99.9% availability, the service provider needs to offer one month service credit to the service consumer, which is a typical remedy statement, which in turn compensates the lost values if a contract obligation is violated. A permission statement example is that the service provider can choose either plain or compression format for data delivery. Permission statements offer added benefits to the involved party. Typically, one party's obligation implies another party's right or permission, as stipulated in [50].

In contrast to a traditional service, a cloud service like Amazon's S3 does not provide a central place like SOW to outline its obligation statements, forbiddance statements, remedies and permission statements. Users must walk through Amazon's Customer Agreements, SLA on their web site, plus dig through Amazon's technical references and API documents in order to comprehend what exactly the committed

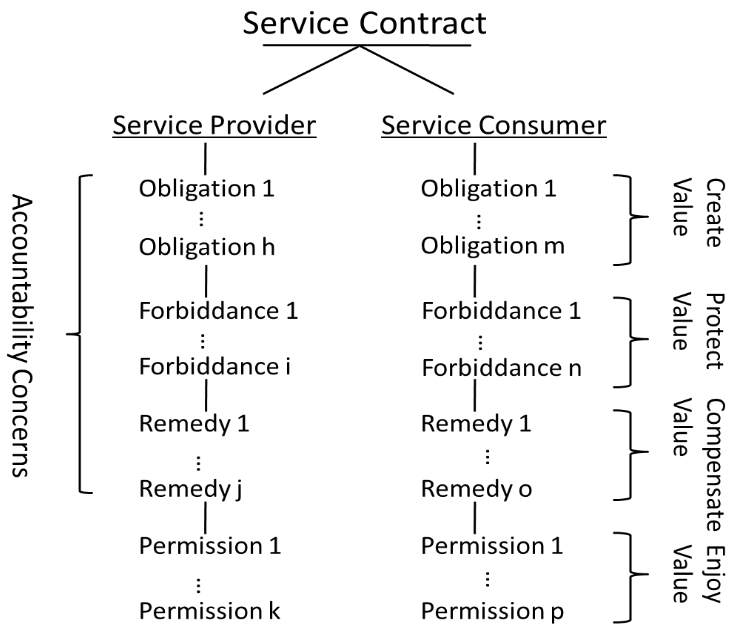


Figure 5.1: Service Contract Properties

offerings from the service provider are.

As per discussion in Section 2.3 in Chapter 2, most cloud services are not accountable due to the absence of a formal contract that stipulates the SOW and other obligations. We now define an accountable cloud service to rectify this problem.

Definition 14: (**accountable cloud service**): An accountable cloud service $acs = \langle i, CP, CIid, sc\rangle$ is a cloud service that is bound to a formal contract sc that defines the statement of work (SOW) and the service level agreement (SLA). Also it provides a formal collaboration process specification so that the violation of obligations can be detected, the causes of faults can be determined and liability can be assigned to the party at fault.

Compared to a traditional cloud service, an accountable cloud service has a formal service contract $sc = \langle P, sow, sla, R, T\rangle$. It has the following new properties:

- P is a pair of involved parties $P = \langle s_p, s_c\rangle$, where s_p is a provider and s_c is a consumer;

- sow is the statement of work,

  $$sow = \langle O_p, F_p, O_c, F_c \rangle$$

  where O_p is a set of provider obligations, F_p is a set of forbidden clauses for the provider, O_c is a set of consumer obligations and F_c is a set of forbidden clauses for the consumer. SOW specifies the service's functional requirements. It defines agreed deliverables and the roles and of the parties responsibilities in those agreed activities;

- sla is the service level agreement, $sla = \langle O_p, F_p \rangle$, where O_p is a set of provider obligations and F_p is a set of forbidden clauses for the provider. SLA specifies the service's non-functional requirements that the provider must satisfy.

- Rules: R is a Horn clause [40] of the form $r \leftarrow r_1, r_2, \ldots, r_n$ where r is the consequent and $r_1, \ldots, r_n$ is the antecedent. A rule defines a remedy if an obligation is breached;

- Time Period: $T = [start\_time, end\_time]$ is the contract's effective period.

5.2.2 An Accountable Process Model for Cloud Contract Specification and Execution

Fig. 1.2 in Chapter 1 describes an accountable process for cloud contract specification and execution. The key stages marked on the figure from 1 to 6 are: *obligation specification*, *obligation validation*, *obligation decomposition to collaboration processes*, *monitoring of contract execution by each party*, *assigning liability*; and lastly, *mutual verification and resolution*.

An accountable process model for cloud contract specification and execution should have the following characteristics:

1) Allow modelling of each service participant's high-level obligation unambiguously;
2) Allow specification of penalties as a new obligation;
3) Support conflict resolution and consistency verification of the service obligation;
4) Support obligation decomposition into a collaboration process that specifies interaction behaviours of the provider and consumer; and
5) Allow monitoring of contract execution, enabling detection of obligation violation and liability assignment;

The subsequent sections will discuss methods that facilitate up to the fifth stage. The sixth stage on Fig. 1.2, *mutual verification* and *resolution* is concerned with dispute resolution, which will be addressed in Chapter 6.

5.3 Dynamic Logic for Accountability (DLA)

Obligation is a kind of a normative statement that defines a desired state derived from the current state. Classical logics like propositional or first-order logic (FOL) cannot naturally be used to represent the normative semantics of obligations.

In [193], Wright proposes Standard Deontic Logic (SDL) to represent obligatory, permissible and forbidden actions. The use of deontic logic to represent legal contracts is discussed in [50]. However, traditionally deontic logic has been "plagued by" numerous paradoxes, *e.g.*, *Ross's Paradox* and *Free Choice Paradox* [135]. In [136], Meyer reduces deontic logic to dynamic logic [78] which is a weak modal logic that strictly separates axioms for actions from assertions. He names the new logic system as Propositional Deontic Logic(PD$_e$L). It avoids most of the paradoxes encountered in SDL. In [192], Wieringa and Meyer further extend PD$_e$L from propositional dynamic logic to Language of Dynamic Logic($L(Sig_{Dyn})$) which supports FOL and process algebra. Note that Sig_{Dyn} stands for a dynamic logic signature (see definition 22 for the meaning of *signature*). In this chapter, we propose Dynamic Logic for Accountability (DLA) to suit the cloud service accountability requirements based on the core concepts of $L(Sig_{Dyn})$.

We first present the core definitions of $L(Sig_{Dyn})$ below.

5.3.0.1 $L(Sig_{Dyn})$ Syntax

We first introduce the set of assertion formulas Asn that can be built from the following Backus-Naur Form (BNF) rule as defined in [192]:

$\phi ::= t_1 = t_2 | P(t_1, \ldots, t_n) | \neg\phi | \phi \wedge \psi | \phi \vee \psi | \phi \rightarrow \psi | \phi \leftrightarrow \psi | \forall x(\phi) | \exists x(\phi)$

where $t_1 ... t_n$ are first-order terms and P is a predicate symbol, $\wedge$ and $\vee$ are the conjunctive and disjunctive operators.

The dynamic logic language $L(Sig_{Dyn})$ in [192] is defined by the BNF rule given below:

$$\Phi ::= \phi | \Phi_1 \vee \Phi_2 | \neg\Phi | \Phi_1 \wedge \Phi_2 | \Phi_1 \rightarrow \Phi_2 | \Phi_1 \leftrightarrow \Phi_2 | [\beta]\Phi$$

where β is a process term (See [192] for its definition) and the intuitive semantics of $[\beta]\Phi$ is: "after execution of β, Φ holds necessarily". In order to represent the deontic constraints, the authors in [192] first introduce violation states $V : a : \alpha$ and $V : a : \neg\alpha$, with the intuitive meanings of "a illegally performed α" and "a illegally failed to perform α", respectively. Then they define the deontic modalities as below (actor a is omitted by the authors but can be modeled in the actions as given above):

- $P(\alpha) = \neg[\alpha]V : \alpha$ ($P(\alpha)$ means "α is permitted"),

- $O(\alpha) = [\neg\alpha]V : \alpha$ ($O(\alpha)$ means "α is obligatory"),

- $F(\alpha) = \neg P(\alpha)$ ($F(\alpha)$ means "α is forbidden").

5.3.0.2 DLA Syntax and Informal Semantics

We here reiterate the action concept and introduce the inaction concepts below:

Definition 15: (**action/inaction**): An action α has the form of $\alpha ::= a : act$, which represents an actor a performing an atomic activity act; $\neg\alpha$ means not to perform action α.

The obligatory function meaning in [136, 192] means that a party "*ought-to-do*" an action. However, in the service contract context, an obligation generally means that the outcome of an action "*ought-to-be*" in a certain state. For example, the *retrieveObject* obligation of a storage service provider is to ensure that "the retrieved data object must be the same as the stored data object". So, service accountability not only obligates the service participant to perform an action, but also stipulates that a certain outcome of the action must be achieved. The obligation is deemed to be breached even if the action has been conducted but failed to deliver the obligated outcome. This can be further explained as follows: assuming a is obligated to perform act, then there are two breach scenarios: the first one is that a did not perform act at all where he ought to; the second one is that a did perform act, however, the action is not terminated successfully; or if terminated, the service level is not met. We use the notation $V : \neg\alpha$ to denote the first scenario and $V : \emptyset_\alpha$ to denote the second one. Here $\emptyset_\alpha$ is a special action (failed action) which means that action α has been conducted but fails to achieve the required outcome. So, the obligation violation condition is $V : \neg\alpha \vee V : \emptyset_\alpha$. Recall that in our definition, an action includes an actor. Hence we define the obligatory modality in DLA as follows:

Definition 16: (**obligatory function**):

$$O([\alpha]\phi) ::= [\alpha]\phi \wedge [\neg\alpha](V : \neg\alpha) \wedge [\emptyset_\alpha](V : \emptyset_\alpha),$$

where $O([\alpha]\phi) \in Asn$, α is an action, $\phi \in Asn$ and $O([\alpha]\phi)$ means that there is an obligation for an agent to perform action α, and the outcome of α must satisfy ϕ; moreover, failing to perform action α (does not perform α or α does not terminate successfully) will lead to violation states.

Note that the obligation function in Definition 16 qualifies the $[\alpha]\phi$ assertion ("***ought-to-be***" approach) rather than the action as $O(\alpha)$ in [136, 192] ("***ought-to-do***" approach). The forbidden function in DLA is still defined as the same as in [192].

Definition 17: (**forbidden function**):

$$F(\alpha) ::= [\alpha]V : \alpha,$$

where $F(\alpha) \in Asn$, $F(\alpha)$ means that performing action α will lead to a violation state.

The permissible function $P()$ is not included in DLA as a permissible action is not a concern for service accountability. This is because that the service party cannot be held accountable for actions that are permissible, i.e., it is acceptable to either perform or not perform the action. We further define the breach function as follows:

Definition 18: (**breach function**):

$$B(O([\alpha]\phi)) = V : \neg\alpha \vee V : \emptyset_\alpha, B(F(\alpha)) = V : \alpha,$$

where $B(t) \in Asn$, $t \in Asn$. $B(O([\alpha]\phi))$ means that obligation $O([\alpha]\phi)$ is breached, $B(F(\alpha))$ means that forbiddance action $F(\alpha)$ is breached.

Now we formally define the DLA syntax as follows:

Definition 19: (**DLA Syntax**): Let ϕ be an assertion, $\phi \in Asn$; α an action, $\alpha \in Act$; t_1, t_2 are accountability formulas, then t is defined as below:

$$t ::= O([\alpha]\phi)|F(\alpha)|B(t)|t_1 \rightarrow t_2|\neg t_1|t_1 \wedge t_2|t_1 \vee t_2,$$

where

- $t_1 \rightarrow t_2$ means that if t_1 holds, it implies that t_2 must be presented. It is used when an obligation is violated, and a new obligation as a remedy must be presented. This can be used for penalty specification;

- $\neg t_1$ means that it is not the case that accountability term t_1 must be presented.

5.3.0.3 DLA Formal Semantics

The central notion of the approach in [136, 192] is to use the modal operator associated with the execution of an action to represent the deontic modalities. Our DLA inherits this approach but changes the meaning of the *obligatory* function as well as removing the *permissible* function. The semantics of action modalities are defined below:

Definition 20: (**DLA model**): A DLA model is a *Kripke*-model [109] $M = (W, \Gamma, \mathcal{R})$, where

- W is the set of all possible states (or worlds);
- Γ is a function which associates each state with the condition it satisfies;
- $\mathcal{R}$ is a collection of binary relations on states where for each action α there is $\mathcal{R}_\alpha \subseteq W \times W$ in $\mathcal{R}$. $\mathcal{R}_\alpha(w, w')$ relates state w to w', that is, if action α is performed in state w, it says that we may end up at state w'.

Definition 21: (**satisfaction of a formula**): Given a model $M = (W, \Gamma, \mathcal{R})$, $\mathcal{R}_\alpha \in \mathcal{R}$ and a world $w \in W$,

$$(M, w) \models [\alpha]\phi \textit{ iff } \forall w' \in W(\textit{if } w\mathcal{R}_\alpha w' \textit{ then } (M, w') \models \phi)$$

M satisfies $\phi(M \models \phi)$ *iff* $\forall w \in W,\ (M, w) \models \phi$ and ϕ is valid ($\models \phi$) *iff* for all DLA models M, we have $M \models \phi$.

Now we use a diagram (Fig 5.2) to illustrate the semantics of the obligatory modality.

Assume $W = \{s_0, s_1, s_2, s_3\}$, relation $\mathcal{R}_\alpha$ gives: $s_0\mathcal{R}_\alpha s_1, s_0\mathcal{R}_\alpha s_2, s_0\mathcal{R}_\alpha s_3$; Γ assigns each state as: $\Gamma(s_0) = \varphi, \Gamma(s_1) = V : \neg\alpha, \Gamma(s_2) = \phi, \Gamma(s_3) = V : \emptyset_\alpha$. Note that $\mathcal{R}_\alpha$ is associated with the actions α, $\neg\alpha$ and $\emptyset_\alpha$. So, all the states s_1, s_2, s_3 are accessible after performing any of these actions. In any real scenario, only one of the actions would take place and the reached state would be uniquely determined by the

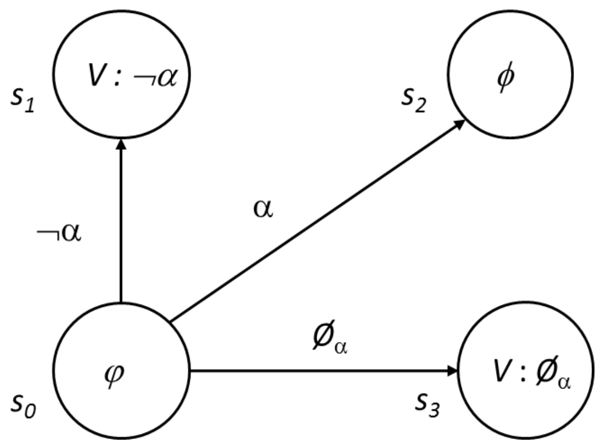


Figure 5.2: A Kripke Model Illustration

Table 5.1: DLA Rules

$$\frac{\vdash \phi}{\vdash [\alpha]\phi}, (1)$$

$$\frac{\vdash [\alpha]\phi}{\vdash O([\alpha]\phi)}, (2)$$

values of the violation state formulas. For instance, if action α is performed at state s_0, then $V : \neg\alpha \vee V : \emptyset_\alpha$ would be false at s_2.

Using *retrieveObject* as an example,

$$\varphi = Exist(oldobject),$$

$$\phi = Return(object) \wedge Same(object, oldobject) \wedge SLA(true).$$

$O([retrieveObject]\phi)$ means at s_0, by performing action $retrieveObject$, the world state can reach s_2 where ϕ holds; but by not performing $retrieveObject$, it will lead to s_1. If $retrieveObject$ fails for whatever reason, it will end up at s_3.

Based on the DLA semantics, we can derive the following rules:

Intuitively, Rule 1 in Table 5.1 indicates that if ϕ holds, then the action status $[\alpha]\phi$ must hold. Rule 2 indicates that if action status $[\alpha]\phi$ holds, then the obligation of the action status $O([\alpha]\phi)$ must hold.

5.3.0.4 DLA Theorems

Theorem 3:

$$O([\alpha]\phi) \equiv [\alpha]\phi \wedge [\emptyset_\alpha](V : \emptyset_\alpha) \wedge F(\neg\alpha);$$

Proof: Based on definition 17:
$F(\alpha) ::= [\alpha]V : \alpha, \therefore F(\neg\alpha) = [\neg\alpha]V : \neg\alpha,$
$\therefore O([\alpha]\phi) = [\alpha]\phi \wedge [\neg\alpha](V : \neg\alpha) \wedge [\emptyset_\alpha](V : \emptyset_\alpha)$
$= [\alpha]\phi \wedge [\emptyset_\alpha](V : \emptyset_\alpha) \wedge F(\neg\alpha).$

Theorem 4:

$$O([\alpha](\theta \wedge \varphi)) \equiv [\alpha]\theta \wedge [\alpha]\varphi \wedge [\emptyset_\alpha](V : \emptyset_\alpha) \wedge F(\neg\alpha);$$

Proof: As $O([\alpha]\phi) \equiv [\alpha]\phi \wedge [\emptyset_\alpha](V : \emptyset_\alpha) \wedge F(\neg\alpha),$
$\therefore O([\alpha](\theta \wedge \varphi)) \equiv [\alpha](\theta \wedge \varphi) \wedge [\emptyset_\alpha](V : \emptyset_\alpha) \wedge F(\neg\alpha);$
$\because [\alpha](\theta \wedge \varphi) = [\alpha]\theta \wedge [\alpha]\varphi$ (refer to theorem 3 in [78]),
$\therefore O([\alpha](\theta \wedge \varphi)) = [\alpha]\theta \wedge [\alpha]\varphi \wedge [\emptyset_\alpha](V : \emptyset_\alpha) \wedge F(\neg\alpha);$

Theorem 5: given $\vdash \theta \to \varphi$ then $\vdash O([\alpha](\theta \to \varphi))$;

Proof: $\because$ if $\vdash \theta \to \varphi$ then $[\alpha](\theta \to \varphi)$ (see table 1 rule 1), and if $\vdash [\alpha](\theta \to \varphi)$ then $O([\alpha](\varphi \to \varphi))$; (see table 1 rule 2)
$\therefore$ theorem 5 holds.

As we can see, in Theorem 3, an obligatory function can be substituted with a combination of modal action logic functions and forbidden functions. In Theorem 4, conjunctive conditions in an obligatory function can be further separated into a modal action logic function on each individual condition. Whereas in Theorem 5, if the implication relationship for two final states holds, then it implies that its obligation function also holds.

DLA is a reduced version of $L(Sig_{Dyn})$ in [192] with some key semantic changes to suit cloud service accountability requirements. $L(Sig_{Dyn})$ is proved to be sound and complete in [192] and hence DLA can also be proved to be sound and complete, while taking into account the semantic changes. $L(Sig_{Dyn})$ avoids most of the paradoxes associated with the SDL, for example, the *Ross Paradox* of "ought to mail a letter implies ought to mail a letter or burn it" [135]. DLA omits the *permissible* operator in deontic semantics, so it totally avoids those paradoxes, making the language more precise and consistent in describing the accountability terms in the cloud service contract. Therefore, DLA is suitable for using in stage 1 of the accountable process model in specifying the high-level obligations of the involved parties in a cloud service contract.

5.4 Cloud Service as a Proactive System

The service contract specifies the "legal" behaviour of service execution. Service execution is a run from an initial service contract status to a final contract status. On the surface, service execution looks similar to a process run going from one state to another, which typically can be modeled using automata theory, which was developed in the middle of the twentieth century [14]. However, the automata model lacks the capability to model the parallel behaviour of a process due to the absence of the notion of interaction. Since a system may interact with another system during its transition from the initial state to the final state, the process of distributed systems or parallel systems is typically modelled using concurrency theory after the 60s, as evident in the formalisms of Petri-nets, CCS, CSP, ACP and π-calculus [64]. In these modelling approaches, systems are treated as reactive systems rather than just state automata [14]. In the late 90s and early 2000s, service computing started to rise and service-oriented architecture gradually emerged as a mainstream architecture. However, despite the paradigm change, a service process is still largely modelled using the reactive system modelling approach.

When focusing on the service accountability aspect of a cloud service, we identify that some behaviours of a cloud service do not strictly follow that of a reactive system. A cloud service's behaviours are driven by the involved actors. Actors in a cloud service typically are the provider or the consumer, they can be human beings or computer agents. Actors can have goals, intentions or subjective bias, hence they can choose to act, or not act based on their beliefs and knowledge, rather than just purely responding to external events like a reactive system does. An established service contract constrains the involved actors' behaviours. Therefore, we argue that a cloud service should be modelled as a "proactive" system rather than a reactive system. To be exact, we cannot assume that a service action will always be carried out and a state transition will always happen. In some cases, the actor may deliberately not perform the action. In some cases, the actor may have performed the action, but due to abnormal events, i.e., human error or uncontrolled execution environment, the execution result is not delivered. In such cases, accountability issues may arise.

5.4.1 Modelling Concerns for Proactive Systems

A reactive system can be modeled using structural operational semantics [14], which defines a labelled transition system (LTS) over a term algebra.

We first define the concepts of signature, term, LTS and TSS from a *proactive* system view as below, referencing the relevant definitions in [64].

Definition 22: (**Signature**): A signature Σ consists of a finite set of function symbols (or operators) $f, g, ...$, where each function symbol f has an arity $ar(f)$, being its number of arguments. A function symbol $\alpha, \beta, \gamma, ...$ of arity zero is called a constant. We assume the presence of a countably infinite set of variables $x, y, z, ...$, disjoint from the signature.

Definition 23: (**Term**): Let Σ be a signature. The set $T(\Sigma)$ of (open) terms $s, t, u, ...$ over Σ is defined as the least set satisfying each variable is in $T(\Sigma)$;

• The variables are assumed to be members of $T(\Sigma)$.

• If $f \in \Sigma$ and $t_1, ..., t_{ar}(f) \in T(\Sigma)$, then $f(t_1, ..., t_{ar}(f)) \in T(\Sigma)$.

• A term is closed if it does not contain variables. The set of closed terms is denoted by $T'(\Sigma)$.

Definition 24: (**Labelled Transition System**): Let S be a non-empty set of states and L a finite, non-empty set of transition labels. More specifically, a transition label in our definition is equivalent to an action (refer to definition 1), $L = P \times A$, where P is a finite, non-empty set of actors and A is a finite, non-empty set of activities. A label (action) $\alpha = p : a$, where $p \in P$ and $a \in A$, denoting p performs a. A transition is a triple (s, α, s') with $\alpha \in L$, where $s, s' \in S$. A LTS is a (possibly infinite) set of transitions. A transition (s, α, s') is usually denoted as $s \xrightarrow{\alpha} s'$; it indicates that state s can evolve into state s' by the execution of action α. If at s, it can successfully execute action α, we say predicate $s \xrightarrow{\alpha} \surd$ holds true.

Definition 25: (**Transition System Specification**): A TSS is a set of transition rules. A transition rule ρ is an expression of the form $\frac{H}{\pi}$, with H a set of expressions $t \xrightarrow{\alpha} t'$, where $t, t' \in T(\Sigma)$, H is called the (positive) premises of ρ, and π is an expression $t \xrightarrow{\alpha} t'$ with $t, t' \in T(\Sigma)$, which is called the conclusion of ρ.

In modelling a reactive system, one assumption is that a transition will always terminate successfully, i.e., predicate $s \xrightarrow{\alpha} \surd$ holds true. A negative transition rule $s \overset{\alpha}{\nrightarrow}$ creates confusions and attracts different interpretations [183]. Moreover, in a reactive system, an actor is not a modelling concern. The system will automatically respond to external events. Actors and abnormal events have been abstracted away in the reactive system model. On the other hand, in a proactive system like a cloud service, the assumption may not hold true. An action specified in a service contract will end up in one of the three scenarios: terminate successfully; inaction (the actor did not initiate the transition); or terminate abnormally due to some uncontrollable events. In short, the main concerns of service accountability are: "who has done what, what went wrong and who should be liable". Therefore, to model a proactive system like a cloud service in an accountability context, we need to take actors and abnormal events into consideration. A label in a proactive transition should include an actor, i.e., $s \xrightarrow{p:a} s'$ where $s, s' \in T(\Sigma)$, $a \in A$, which is a finite non-empty set of activities. and p is an actor. An inaction is represented as $s \xrightarrow{p:\neg a} s'$, which means that the actor p did not initiate the action. Note that in this case, although the action was not carried out, the state of the contract execution actually transited to a new state s'. Finally, an abnormal terminate transition can be represented as $s \overset{p:a}{\nrightarrow}$ or $s \xrightarrow{p:\varepsilon(a)} s'$, where $\varepsilon(a)$ indicates that an exception is raised when actor p performs activity a.

5.5 Using Accountable Process Algebra to Model Cloud Services

Process algebra is a mainstream tool used to model the behaviour of a reactive system. It provides a formal approach to decompose a process into basic components, and then imposes an equational logic on process terms, so that people can reason about such systems using the algebra, i.e., equational reasoning to verify that a system satisfies a certain property and conforms to the desired external behaviour.

We can also take the process algebra approach to model service accountability so that accountability properties and accountable behaviour can be specified and verified. To do that, we need to first extend the traditional process algebra to cater for modelling a proactive system's concerns.

Based on the structural operational semantics, we extend the traditional process algebra to a new form of labelled transition system to address the accountability concerns in cloud services. We call it Accountable Process Algebra (APA), assigning new syntax and semantics based on the accountability requirements.

We first formally define the TSS for APA in an accountable cloud service context.

Definition 26: (**TSS for accountable process algebra**):
Let S be a non-empty set of a service contract execution states, A a finite, non-empty set of activities, P a finite, non-empty set of actors. $p : a$ is a closed term in $T'(\Sigma)$,

Table 5.2: APA Transition Rules

$$\frac{x \xrightarrow{p:a} \surd}{x \bullet y \xrightarrow{p:a} y}, \quad \frac{x \xrightarrow{p:a} x'}{x \bullet y \xrightarrow{p:a} x' \bullet y}, \quad \frac{x \xrightarrow{p:\neg a} x'}{x \bullet y \xrightarrow{p:\neg a} x'}, \quad \frac{x \xrightarrow{p:\varepsilon(a)} x'}{x \bullet y \xrightarrow{p:\varepsilon(a)} x'}$$

$$\frac{x \xrightarrow{p:a} \surd}{x \| y \xrightarrow{p:a} y}, \quad \frac{x \xrightarrow{p:a} x'}{x \| y \xrightarrow{p:a} x' \| y}, \quad \frac{x \xrightarrow{p:\neg a} x'}{x \| y \xrightarrow{p:\neg a} x' \| y}, \quad \frac{x \xrightarrow{p:\varepsilon(a)} x'}{x \| y \xrightarrow{p:\varepsilon(a)} x' \| y}$$

$$\frac{y \xrightarrow{p:a} \surd}{x \| y \xrightarrow{p:a} x}, \quad \frac{y \xrightarrow{p:a} y'}{x \| y \xrightarrow{p:a} x \| y'}, \quad \frac{y \xrightarrow{p:\neg a} y'}{x \| y \xrightarrow{p:\neg a} x \| y'}, \quad \frac{y \xrightarrow{p:\varepsilon(a)} y'}{x \| y \xrightarrow{p:\varepsilon(a)} x \| y'}$$

where $p \in P$ and $a \in A$. The variables x, x', y, and y' in the transition rules range over the terms $T(\Sigma)$, which represent process terms and have operators $\bullet$ for sequential composition, $\neg$ for inaction and $\|$ for parallel composition, plus a function ε that takes an activity as a parameter, indicating the action of raising an exception in the execution of that particular activity. Note that unlike traditional process algebra, the *choice* operator + is not used here. This is because *choice* is indifferent to accountability. The transition rules for APA are outlined in Table 5.2.

Note that in most process algebras such as CCS, ASP and ACP, the concepts of a state, an action and a process term are sometimes used interchangeably, which may cause confusion. In APA, we formally define their relationships through the definitions below:

Definition 27: (**collaboration process term**): Let Σ_1 be a collaboration process signature, $T(\Sigma)$ a set of terms over Σ_1, S a set of states, L a set of LTS labels, P a set of actors, A a set of activities, $\alpha \in L$, $\alpha = p : a$, where $p \in P, a \in A$. A collaboration term t is defined as below:

$$t := p : a | p : \neg a | \alpha | t_1 \bullet t_2 | t_1 \| t_2$$

, where t_1 and t_2 are also a collaboration process term.

$x, y, z, ...$ are variables in a collaboration process term, $\alpha, \beta, \gamma, \delta$ are constants in a collaboration process term.

Definition 28: (**state and collaboration process term mapping**): Let t, t_1, t_2 be terms of a collaboration process, S a set of states, $s, s' \in S$. A state s holds at a particular step of the collaboration process, and s will be transited to s' after the execution of process term t, i.e., after the execution of t, state s' holds. Thus a one-to-one relationship can be established between the state and the execution of a process term.

Based on the above definition, a collaboration process term execution can be mapped one-to-one to a state, therefore, a collaboration process term can be used to represent a state transition as well, for example, $x \xrightarrow{\alpha} x'$ represents that the state associated with collaboration process term x can transit to the state associated with the step after the execution of action α.

In Table 5.2, the first four transition rules are for sequential composition. $x \bullet y$ means that the process x will perform an activity a, if x terminates successfully, then y will be performed. Rule 2 indicates that if x is transited to x', then $x \bullet y$ will transit to $x' \bullet y$. Rule 3 specifies that if the actor did not perform activity a , from a service contract perspective, the contract state has still been transitioned from x to x', however, y will never have a chance to execute. The same applied in rule 4 when an exception arose while actor p performed activity a.

The next eight rules are for parallel composition. $x \| y$ means that x or y can be executed in parallel. In APA, unlike other process algebras, we do not consider the communication between x and y, and we just assume x and y will each execute independently and merge at the end. The eight rules are intuitive, without the need for further explanation.

For example, the transition rules in Table 5.2 provide the basic process term $(\alpha \bullet \beta) \| (\gamma \bullet \delta)$ with the process graph as depicted in Fig. 5.3.

Also the transition $(\alpha \bullet \beta) \| (\gamma \bullet \delta) \xrightarrow{p:a} \beta \| (\gamma \bullet \delta)$ can be proved using the rules in Table 5.2 as follows.

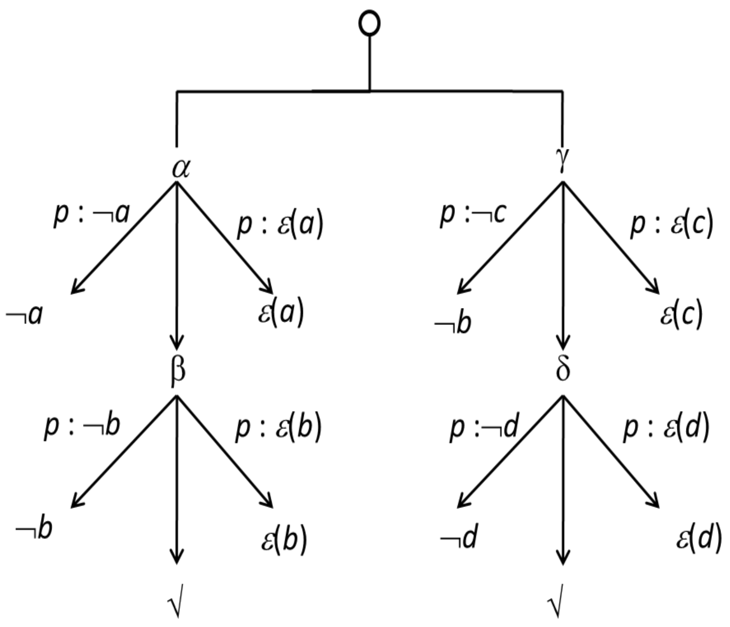


Figure 5.3: Process Graph for $(\alpha \bullet \beta)\|(\gamma \bullet \delta)$

Table 5.3: APA Axioms

$$x \bullet y \bullet z = (x \bullet y) \bullet z$$

$$x \bullet y \bullet z = x \bullet (y \bullet z)$$

$$x\|y = y\|x$$

$$x\|y\|z = (x\|y)\|z = x\|(y\|z)$$

$(\alpha \bullet \beta) \xrightarrow{p:a} \beta$ based on: $(\frac{x \xrightarrow{p:a} \surd}{x \bullet y \xrightarrow{p:a} y}, x := \alpha, y := \beta)$

$(\alpha \bullet \beta)\|(\gamma \bullet \delta) \xrightarrow{p:a} \beta\|(\gamma \bullet \delta)$ based on: $(\frac{x \xrightarrow{p:a} x'}{x\|y \xrightarrow{p:a} x'\|y}, x := \alpha \bullet \beta, x' := \beta, y := \gamma \bullet \delta)$

Next we list the axioms of APA in Table 5.3.

5.5.1 Obligation Specification Model Verification and Conflict Resolution

In traditional reactive systems, model checking is a widely used approach to verify the consistency of a model. Model checkers utilise software agents to analyse the state space of a model and confirm that certain properties hold or report that they are violated. In the case of a proactive system, model checking is less attractive, as an actor can act or not act based on its own discretion. Also as the DLA model is used to specify the normative statements of a contract obligation, the main concern is not the models that satisfy the statement, rather, it is whether there are conflicts amongst those normative statements. As such, we now introduce the verification and conflict resolution techniques for the obligations specified for the cloud contract.

We here first introduce the action dependency concept as in [160].

Definition 29: (**action dependency**): Action dependency is modelled by a relationship $\mathcal{D} \subseteq Act \times Act$ which is reflexive and symmetric. The inverse notion of independency is $I = Act \times Act \setminus \mathcal{D}$;

Intuitively, for a sequential execution of α_1 and α_2 we have $\alpha_1 \mathcal{D} \alpha_2$; for a parallel execution of α_1 and α_2 we have $\alpha_1 I \alpha_2$; if an obligated action O is decomposed into a series of actions, we can get a dependency chain:

$$\alpha_1 \mathcal{D} \alpha_2 \mathcal{D} \ldots \alpha_{n-1} \mathcal{D} \alpha_n.$$

Therefore, we can see that the fulfilment of the obligation O depends on the action chain. This enables reasoning about which party is at fault in an obligation violation situation. Action dependency chains are directly related to the transitions between the states in the Kripke models.

An *obligation specification* model should be consistent and free of conflicts. In our model, there are two situations where a conflict arises:

1) Obligation and forbiddance on the same action, i.e., action α is obligatory as

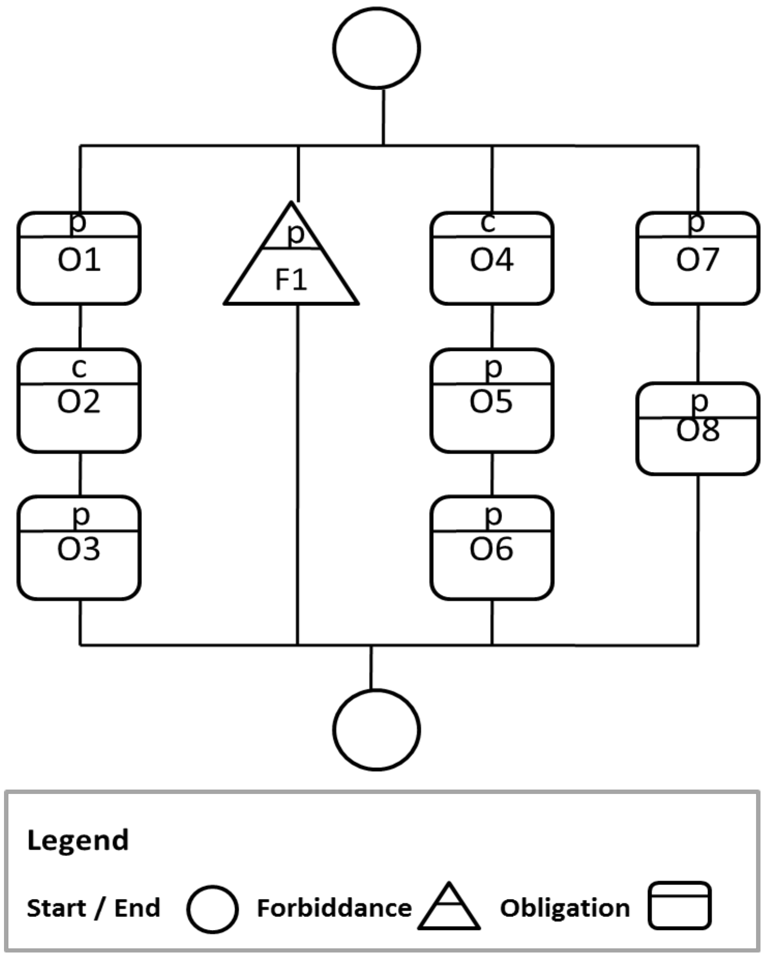


Figure 5.4: Obligation Flow Diagram

well as forbidden; and

2) Action interdependency: i.e., action α_1 depends on action α_2; in the mean time, action α_2 also depends on action α_1.

In order to uncover such conflicts, we design an Obligation Flow Diagram (OFD) based on the concept of state transition sequences and action dependency chains as illustrated in Fig 5.4.

In Fig 5.4, the obligations on the same horizontal level are independent obligations and they can be executed in parallel. The equivalent APA operator is the parallel composition operator $\|$. Obligations on the vertical line indicate the dependency relationship, meaning that the lower obligation depends on the completion of the upper one. The equivalent APA operator is the sequential composition operator $\bullet$. We can use APA to represent the OFD in Fig 5.4 as follows:

$$(O_1 \bullet O_2 \bullet O_3)\|F_1\|(O_4 \bullet O_5 \bullet O_6)\|(O_7 \bullet O_8)$$

We use two steps to verify an *obligation specification* model. The first step is to check the consistency of high-level obligations. This step involves the construction of an obligation flow diagram and validates that no obligation/forbiddance conflict exists; also there is no loop in the obligation dependency chain. The second step is to decompose each obligation into an action dependency chain. Then we can determine that there is no loop on the action dependency chain.

If there is a dependency chain from α_i to α_j, then α_i is a successor of α_j, and α_j is a predecessor of α_i. We combine all the actions and/or events from every involved party to form only one dependency chain.

We now illustrate the creation of an action dependency chain of a service contract, which starts from the starting event ε_s in OFD until the ending event ε_e or the abnormal terminal event ε_a. We here use the depth-first search algorithm to create the action dependency chain based on the input of the OFD graph and the starting event. The output is an array of the action dependency chain. The details of the construction of a dependency chain are presented in Algorithm 1.

After the action dependency chains are formed, we need to check that there is no contradiction in the model with the following two cases:
(1) The dependency chain is a directed graph without any loop.
(2) For any two actions in the dependency chain, they can co-exist, i.e., no action is both obligated and forbidden in the same chain.

If these two conditions hold, we can claim that there is no contradiction in the *obligation specification* model and trust that there is no contradiction in the service

Algorithm 1: The Creation of Action Dependency Chains for a Service Contract

```
Data: OFD graph $G$ and starting event $\varepsilon_e$
Result: Action Dependency Chain array $D[]$
begin
    Create a stack $s$;
    Create an array $D[]$;
    Create a vertex $v$;
    $n = 0$;
    $s.push\varepsilon_s$;
    while $s$ is not empty do
        $v$ =$s$.pop();
        $D[n] = v$;
        if $v$ is not marked as visited then
            Mark $v$ as visited;
            for all edges from $v$ to $w$ in $G.adjacentEdges(v)$ do
                $s.push(w)$;
                $D[n] = D[n]||w$;
            end
            $n++$;
        end
    end
    Return $D[]$;
end
```

contract. The details of contradiction checking in the service contract are illustrated in Algorithm 2.

The Obligation Flow Diagram and algorithms 1 & 2 provide a simple way to resolve conflicts and verify the specified obligations. This corresponds to the second stage of the accountable process model.

5.5.2 Service Collaboration Process Diagram

An obligation defined in a service contract is generally a high-level statement that stipulates the obligated outcome, which is achieved through the collaboration between the provider and the consumer during service execution. If an obligation is breached in a contract, we need to identify the responsible party. In some cases when the obligation is breached, the party who is responsible for performing the obligation cannot be blamed as he/she may have a dependency on another party's activities. Therefore, the obligation should be further decomposed into a series of atomic activities performed by the provider or the consumer. This is stage 3 of the accountable process model. Here we use a subset of BPMN2.0 [146] notations as listed in Fig 5.5.2 to illustrate the collaboration process. The only changes that we made to BPMN2.0 is to change

Algorithm 2: Contradiction Checking in service contract

```
Data: the dependency chain from the starting event $\varepsilon_s$ to the ending event $\varepsilon_e$.
Result: there is a contradiction in service contract or not
begin
    Create a stack $s_1$;
    Put the event $\varepsilon_s$ into stack $s_1$;
    while $s_1$ is not empty do
        Pop the vertex $V : v$ from the top of stack $s_1$ with all predecessors having been visited, and mark it as
          visited;
        if there is a loop starting from $v$ then
            Return there is a contradiction in service contract
        end
        for each successor $u$ of $v$ do
            if $u$ has no unvisited predecessors then
                Push $u$ into $s_1$;
            end
        end
    end
    Create stacks $s_2$ and $s_3$;
    Put the event $\varepsilon_s$ into stack $s_2$;
    Put the rest events in the dependency chain into stack $s_3$;
    while $s_2$ is not empty do
        Pop the vertex $V : v$ on the top of stack $s_2$, and mark it as visited;
        for every event $w$ in $s_3$ do
            if $u$ and $w$ cannot co-exist then
                Return there is a contradiction.
            end
        end
        for each successor $u$ of $v$ do
            if $u$ has no unvisited predecessors then
                Push $u$ into $s_2$;
                Pop $u$ from $s_3$;
            end
        end
    end
    Return there is no contradiction in service contract
end
```

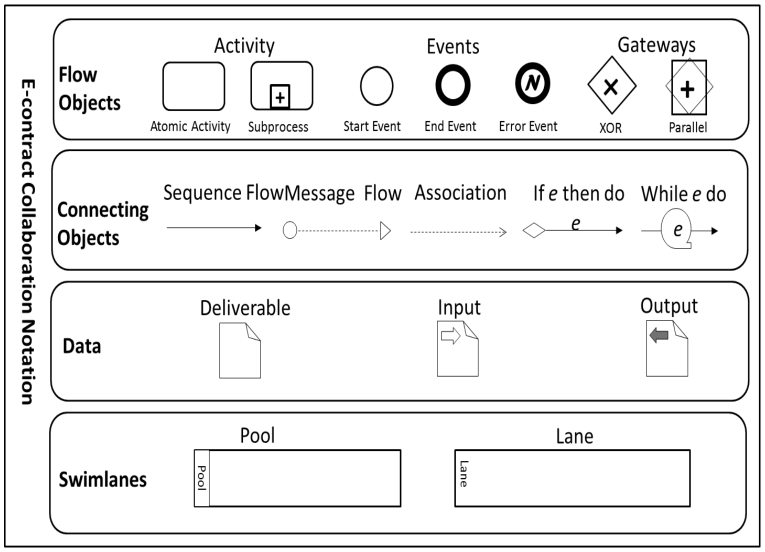


Figure 5.5: Collaboration Diagram Notation

the inclusive condition gateway from a flow object to a kind of connecting object to represent the "*if ... then ... else*" branching scenario, and also introducing a new connecting object to represent the *while condition loop*, which is only treated as a marker originally in BPMN2.0.

We now formalise the collaboration process using APA, adding conditional branching and looping operators.

Definition 30: (**service collaboration process**):
$CP = p : a|p : P_1 \bullet p : P_2|p : P_1||p : P_2|p : a(x \Leftarrow)|p : a(x \Rightarrow)|$
$?(e)p : P_1/p : P_2|\varpi(e)p : P|\varepsilon_s|\varepsilon_e|\varepsilon_a$,
where

- p is a party who performs the activity. $p : a$ means that p performs activity a;

- a is an atomic activity, represented by the atomic activity notation in Fig.5;

- Process P_1 or P_2 is composed by atomic activities, represented by the subprocess notation;

- $P_1 \bullet P_2$ is the sequential execution of process as P_1 and P_2, the sequence flow object in the BPMN2.0 notation represents the sequential execution.

- $P_1 \| P_2$ represented by the BPMN2.0 parallel gateway means the parallel execution of P_1 and P_2; however, it does not require the synchronisation of P_1 and P_2 like the traditional process algebra does.

- The synchronisation behaviour can be defined using the "*if ... then ... else*" conditional branching connecting object, which is represented by $?(e)P_1/P_2$; that is, if condition e holds, P_1 is executed otherwise P_2 is executed; the XOR gateway can also be represented using $?(e)P_1/P_2$ as well since the process is branching at the gateway based on a certain condition;

- Similarly, for the *while loop* notation, represented by $\varpi(e)P$, it means that while condition e holds, P will continue to be executed;

- ε_s is the starting event, ε_e is the ending event and ε_a is the abnormal termination event.

- The precedence of the operators is (), :, then ? /, ϖ , • and $\|$.

Compared to CCS or other process algebras, firstly we do not use the *choice* operator + in APA. This is because that non-deterministic choice is not a desired behaviour for accountability. An accountable collaboration should provide certainty with a clear outcome. Any non-determinism in the process reflects an issue in the service design, either some control is missing or the execution is not fully tested. The elimination of the *choice* operator also avoids the *Free Choice Paradox* as described in [136]. Secondly, we introduce a conditional branching operator, a *while loop* operator and *events* $(\varepsilon_s, \varepsilon_e, \varepsilon_a)$, making it more suitable for representing collaboration processes. We also do not use the *left merge*, *right merge* semantics of the parallel operator, as they are indifferent as far as accountability is concerned.

Algorithm 3: The Population of APA Representation of Contract Database

```
Data: the APA formula $P$
Result: the contract graph database $g$
begin
    for each obligation or action in $P$ do
        Create graph database $g$; Create node $n$ for the respective obligation or action;
        if $n$ is the first node then
            Create relationship $r$ from $n_0$ to $n$;
        end
    end
    for each operator op in $P$ do
        if op == “ • ” then
            Create relationship $r$ from op.leftaction to op.rightaction with property =“dependency” ;
        end
        if op == “||” then
            Identify the action $a$ prior to op.leftaction that is not followed by another op; Create relationship $r$
              from $a$ to op.rightaction with property = “independency”;
        end
    end
    Return $g$
end
```

5.5.3 Contract Execution Monitoring and Liability Assignment

The Obligation Flow Diagram (OFD) and the collaboration process decomposed from each obligation can be used as a basis for monitoring contract execution. As highlighted in stage 5 of the accountable process model depicted in Fig 1.2 in Chapter 1, either the service provider or the consumer can monitor the contract execution in their own environment.

There are three steps in monitoring. The first step is to create a contract database based on the input of the APA representation of OFD and the collaboration process of each obligation. Algorithm 3 outlines how to turn an APA representation into a graph database. The second step is to monitor the contract execution. This involves creating an interaction trace database for the actions of both the provider and the consumer for each obligation. The third step is to check if any obligation is breached. If so, it will compare the trace records against the associated collaboration process created in the contract database, and identify which party's interaction behaviour is deviating from the contract specifications, and thus assign the liability to that particular party.

5.6 A Case Study

We use the motivating example in Section 1 of the Amazon S3 service to illustrate the use of the accountable process model. From an accountability perspective, a cloud service should bind to a contract that can be published to disclose the obligations of the service participants, while in the mean time facilitate machine-interpretation, monitoring of executions and reasoning about obligation violations. The S3 service can be represented as a cloud service:

$$s3 = \langle s3\_wsdl, CP, aws, sc \rangle,$$

where CP is the collection of collaboration processes, which will be described using a collaboration diagram later; $s3\_wsdl$ is the S3 Web Services interface, see [3]; aws stands for Amazon's cloud infrastructure; sc is the service contract, which can be represented as:

$$sc = \langle P, sow, sla, R, T \rangle,$$

where P consists of the service participants, in this case, they are Amazon and the customer; $sow = \langle O_p, F_p, O_c, F_c \rangle$, where O_p is a set of the provider's obligations, F_p a set of the provider's forbiddance statements, O_c is a set of the consumer's obligations, F_c is a set of the customer's forbiddance statements. We also have that, $sla = \langle O_p, O_c \rangle$; R is the set of rules to determine the remedy for SOW and SLA violations; and T is the contract effective period.

5.6.1 S3 Contract Specification

Firstly, we need to identify the key collaboration processes in the S3 service. The sign-up process is a prerequisite for using the S3 service. The customer needs to provide valid credit card information for the service payment. Once the service provider accepts the customer's sign-up application, it will have an obligation to create an account for the customer. After the account is created, the customer can login and start

consuming the service. During the service consumption, the S3 service has four major collaboration processes. They are *createObject*, *retrieveObject*, *listBucket* and *deleteObject*. The service provider also has a billing process that charges the customer's credit card based on the service usage. Thus:
$CP = \{signUp, createAccount, createObject, retrieveObject,$
$listBucket, deleteObject, billCust\}.$

Then we specify the sow based on *DLA*. Again, we only need to specify the high-level, key obligations in the SOW. The meanings of the actions and predicate symbols can be interpreted intuitively. The high-level obligations can be further refined for automated reasoning purposes, but we omit the details of the process in this paper.

The obligations of the service provider O_p are:

1) $O([createAccount]Valid(account));$

2) $O([createBucket]Exist(bucket) \wedge (NameOf(bucket) == \text{``}mybucket\text{''}));$

3) $O([createObject]Contained(bucket, object) \wedge (Nameof(bucket) ==$
$\text{``}mybucket\text{''} \wedge Nameof(object) == \text{``}myobject\text{''}));$

4) $O([listBucket]ReturnObjectNames(mybucket)$
$== ObjectNames(mybucket));$

5) $O([retrieveObject]ReturnObj(\text{``}myobject\text{''}) ==$
$OriginObject(\text{``}myobject\text{''}) \wedge SLA(true));$

6) $O([deleteObject]Nonexist(object));$

7) $F(shareObject).$

The first obligation is that the service provider is obligated to create a unique account for the customer. The second obligation is to create a bucket at the customer's request. The third obligation is to create an object within the bucket; the fourth obligation is to be able to list the name of the objects in the bucket; the fifth obligation is to be able to retrieve the stored object, and the object content should be the same as the original object. Finally, the provider is forbidden to share the data object with other customers.

Similarly, the obligations of the service consumers O_c are:

1. $O_c([signUp]Valid(credit\_card) \wedge Unique(account))$;

2. $O_c([login]Valid(credential))$;

3. $O_c([iputValidOpt]OptionNameIn(\text{“}createBucket\text{”}, \text{“}listBucket\text{”}, \text{“}createObject\text{”}, \text{“}retrieveObject\text{”}, \text{“}deleteObject\text{”}))$.

This means that the consumer's obligations are to provide a valid credit card for payment purpose, login with valid credentials and input the valid operations.

To illustrate the provider's SLAs, we give an example of the availability obligation. The term $s3(c1)$ represents the action of Amazon S3 providing the contracted service. The sla can be simply represented as:

1. $O([s3]Availability(s3) \geq 99.9\%)$;

For the overall S3 service's service-level, the service provider has an obligation to maintain the availability SLA of 99.9%. If the availability SLA is not maintained, we will end up with a violation state where $V : \emptyset_{s3}$ will be true in which case the provider may be liable by the contract for giving credit back to the customer.

The contract rule R can be represented as:

1. $B(O([s3]Availability(s3) \geq 99.9\%)) \rightarrow O([giveCreditToCust](credit == Last\_Bill(s3) * (100 - Availability(s3))/100))$;

This rule means that if the SLA is breached, the service provider will have a new obligation to pay service credit to the consumer who experienced the poor service-level.

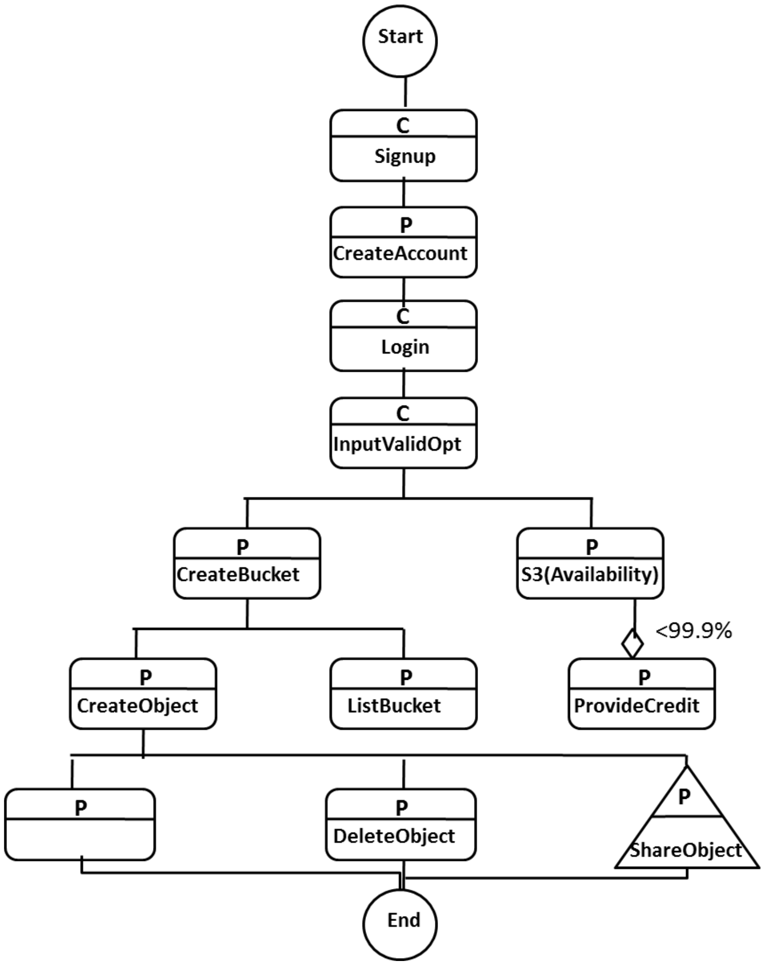


Figure 5.6: S3 Service Obligation Flow Diagram

We here use the obligation flow diagram as discussed at Section 5.5.1 to illustrate the S3 service contract as illustrated by Fig 5.6.

Fig. 5.6 illustrates the valid sequence for the interaction between the obligations of the provider and the consumer.

The APA representation of Fig. 5.6 is:

$C_{ofd} = O_c[Signup] \bullet O_p[CreateAccount] \bullet O_c[Login]$
$\bullet\, O_c[InputValidOpt] \bullet ((O_p[CreateBucket]$
$\bullet\, ((O_p[CreateObject] \bullet (O_p[RetrieveObject]$

$\|O_p[DeleteObject]\|F_p[ShareObject]))\|O_p[ListBucket]))$

$\|(O_p[Availability]\bullet?(< 99.9\%)O_p[ProvideCredit]/\varepsilon_e))$

5.6.2 S3 Contract Validation

Using Algorithm 1 in Section 5.5.1, we can derive six action dependency chains. Due to the space limit, we only list one below:

$$\begin{aligned}
&c : Signup\, \mathcal{D}\, p : CreateAccount\, \mathcal{D}\, c : Login\, \mathcal{D}\\
&c : inputValidOpt\, \mathcal{D}\, p : CreateBucket\, \mathcal{D}\\
&p : CreateObject\, \mathcal{D}\, p : RetrieveObject
\end{aligned} \tag{5.1}$$

C_{ofd} outlines the overall contract obligations and their flow sequence. Now we can validate the model using Algorithm 2. The validation process checks through the six dependency chains of C_{ofd} above and makes sure no conflicts exist in the current model.

5.6.3 S3 Obligation Decomposition

Each obligation needs to be decomposed to a collaboration process that involves atomic actions from both the provider and the consumer respectively. For example, we use the BPMN2.0 notation to decompose obligation *createObject* to the collaboration process between the service provider and the consumer as illustrated by Fig. 5.7

Using the accountable process algebra (APA), we can formalise this collaboration process as follows:

$CP = \varepsilon_s \bullet c : SignIn(user\_info \Rightarrow) \bullet p : Login(user\_info \Leftarrow)$

$\bullet\ (?(abnormal)\varepsilon_a/p : L\_Resp(status \Rightarrow)) \bullet c : Status\_Chk(status \Leftarrow) \bullet$

$(?(not\_found)\varepsilon_a/c : C\_CreateBucket(bucket\_name \Rightarrow)) \bullet$

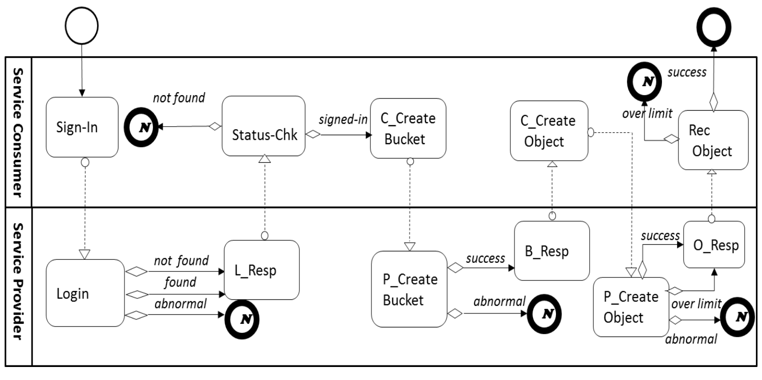


Figure 5.7: *createObject* Obligation Collaboration Process

$p : P\_CreateBucket(bucket\_name \Leftarrow) \bullet$

$(?(abnormal)\varepsilon_a / p : B\_Resp(bucket\_name(\Rightarrow)) \bullet$

$c : C\_CreateObject(object \Rightarrow) \bullet p : P\_CreateObject(object \Leftarrow) \bullet$

$(?(abnormal)\varepsilon_a / p : O\_Resp(object \Rightarrow)) \bullet$

$c : Rec\_Object(object \Leftarrow) \bullet (?(overlimit)\varepsilon_a / ?(success)\varepsilon_e)$

From the decomposed process, we can also get the *createObject* obligation dependency chain as:

$$c : Sign\_In\, \mathcal{D}\, p : Login\, \mathcal{D}\, p : L\_Resp\, \mathcal{D}$$
$$c : Status\_Chk\, \mathcal{D}\, c : C\_CreateBucket\, \mathcal{D}$$
$$p : P\_CreateBucket\, \mathcal{D}\, p : B\_Resp\, \mathcal{D}$$
$$c : C\_CreateObject\, \mathcal{D}\, p : P\_CreateObject$$
$$\mathcal{D}\, p : O\_Resp\, \mathcal{D}\, c : Rec\_Object$$

The collaboration diagram gives an intuitive illustration of each party's activities. Any abnormal event indicates the cause of a potential obligation breach for the *createObject* process. Also the pool in which the abnormal event falls indicates which party is at fault. For example, if the consumer creates an object with a size over the limit, then the *createObject* activity of the service provider will fail. However, in this case, the service provider does not violate her obligation; rather the failure is caused by the wrong input by the consumer. On the other hand, if the input is valid but *createObject* fails due to some internal error, then the service provider violates her obligation.

5.6.4 S3 Contract Execution Monitoring and Liability Assignment

After all the obligations are decomposed into collaboration processes, the contract execution can be monitored by either the provider or the consumer. The first step in monitoring is to populate the contract database as a reference point. Algorithm 3 in Section 5.5.3 can be used to store APA representation of OFD or collaboration processes into a graph database. A graph database system like Neo4j[1] enables easy query of properties of nodes and relationships in a directed graph. The second step of the monitoring is to watch the interactions from both the provider and the consumer, recording the trace of actions into a graph database. The third step is to check if any obligation is breached. If so, we can compare the reference collaboration process and the trace records and identify deviations, and subsequently assign the liability to the responsible party.

If an obligation is breached, the monitor can compare the action trace against the APA process CP to locate the deviation. For example, if the obligation *createObject* is breached, the monitor may find out that after the consumer sends the *createObject* command, the action trace does not record a response from the provider, whereas the reference process CP specifies the next action step should be O_Resp from the provider. Thus the monitor can identify that the provider is at fault and therefore assign liability to the provider.

5.6.5 Case Study Summary

We have demonstrated the accountable process model for cloud contract specification and execution using Amazon S3 service as an example. We first identified the key obligations in S3 service's SOW and SLA, then used DLA to specify them. This ad-

[1]http://www.neo4j.com

dresses **Prob1** (missing SOW) and **Prob2** (no obligation representation) as listed in Section 1.1.3 in Chapter 1. Then we used an Obligation Flow Diagram to describe the obligation sequence and generated the action dependency chain, ensuring the consistency of the contract model. Next we used APA and the collaboration diagram to visually decompose the obligations, enabling obligation decomposition and delegation. The OFD and APA also provide a basis for contract execution monitoring. This addresses **Prob3** (no precise process defined for contract execution monitoring). Finally, we have outlined how to detect the liable party when an obligation is breached. This addresses **Prob4** (no violation detection and liability assignment mechanism). By adopting this process model, S3 service's obligations will be clearly specified and validated, a violation can be easily detected and liability can be assigned without any doubt.

As **Prob1** – **Prob4** are general accountability problems existing in other cloud services, and Amazon S3 is a typical cloud service that exhibits the cloud service characteristics as presented in Definition 2, our accountable process model can be applied to general cloud services for enhancing accountability.

5.7 Conclusion

Since Service-Oriented Architecture (SOA) has emerged as a mainstream IT architecture together with cloud computing service (IaaS, PaaS, SaaS) gradually becoming popular, more and more cloud services are being offered by a variety of service providers. Service consumers require an effective way to evaluate the risk associated with those cloud services and manage this accordingly in their business operations. However, currently most cloud services on the market lack accountability. We argue that the first step towards addressing this issue is to establish a cloud service accountability mechanism, which is the main objective of this paper. Our contributions can be summarised as follows:

1. Our work addresses cloud service accountability by first proposing a cloud ser-

vice model that encompasses SOW, the most crucial component of a service contract. The cloud service model allows a clear service obligation specification so that the service provider can disclose their service obligations in a machine-interpretable way. This further enables service searching and matching based on accountability requirements;

2. A refined version of dynamic logic – DLA – has been proposed as the underlying formalism for the cloud service contract specification;

3. We present a proactive system view of a cloud service and extend structural operational semantics to create a new form of a process algebra – APA – which allows modelling of the service contract execution behaviour;

4. With regard to the characteristics of the proactive system, we propose an Obligation Flow Diagram (OFD) and Action Dependency Chain to allow easy conflict resolution and consistency checking of the cloud contract model. Algorithms have been presented for action dependency chain generation and consistency checking;

5. A graphical notation based on a reduced version of BPMN2.0 has been proposed so that obligations can be further decomposed into collaborative activities between the service provider and the consumer;

6. A contract execution monitoring approach has been presented that enables obligation fulfilment tracking and liability assignment.

Finally in Section 5.6, using Amazon's S3 service as a case study, we have demonstrated how to apply our model to address the four accountability problems (Prob1 – Prob4) listed in Section 1.1.3 in Chapter 1. As Amazon's S3 is a typical cloud service that exhibits general cloud services' characteristics, we can apply our model to general cloud services.

To the best of our knowledge, this is the first study that advocates a proactive system modelling approach for modelling a cloud service. Our approach is also the first that formally models SOW in a cloud service context. Furthermore, our approach is the first that addresses cloud service accountability from both formal logic and intuitive diagram notation perspectives. This provides the foundation for further modelling techniques that assess runtime contract negotiation and contact modification, which are the subjects of future work.

Chapter 6

Advanced Service Accountability – A Decentralised Approach

The service contract is the first-class citizen in the service accountability domain. The last two chapters address the crucial concern of service contract representation in a service-oriented architecture. A machine readable service contract representation lays down the critical stepping stone for advancing service accountability. Next, a robust service contract management scheme is needed to automate the disclosure, contracting, monitoring and dispute arbitration processes in an accountable service-oriented environment.

In Chapter 4, an in-line TTP service contract monitoring solution is proposed, extending REST, which is the mainstream SOA implementation architecture to an accountable state transfer (AST) architecture. The solution is a centralised approach that relies on a trusted-third party sitting in the middle of every transaction for monitoring and arbitrating. In a service computing environment, especially on a vast scale like the cloud environment, a centralised solution is prone to performance bottlenecks, single-point of failure and other issues, such as the lack of transparency, objectivity and fairness which, in themselves, create concerns in accountability.

In this chapter, we firstly propose a robust decentralised service contract management scheme based on the proven blockchain technology from the peer-to-peer network environment. Secondly, on top of the scheme, we present a novel dispute resolution protocol, based on the Byzantine Agreement and the commitment scheme.

Thirdly, we identify the optimal settings of the key parameters of the protocol through a set of experiments and scenario analysis, aiming to strike the balance of fairness, accuracy, and incentive maximisation for the honest arbiters and cost minimization for the overall arbitration process. With this approach, service participants can be held accountable in a truly distributed environment without the presence of a central authority, which provides strong accountability compared to a centralised solution.

The remainder of this chapter is structured as follows. Section 6.1 provides an introduction on the motivation of our work, plus a brief overview of the blockchain technology. Next, Section 6.2 proposes a service contract management scheme (SCMS) based on a peer-to-peer architecture. Section 6.3 presents a fair and accurate contract dispute arbitration protocol based on SCMS. This is followed in Section 6.4 by experiments, analysis and discussion of the experimental results. Finally, we summarise the key contributions and discuss future work in Section 6.5.

6.1 Preliminary

In recent years, an increasing number of businesses have adopted cloud services due to the agility and cost benefits that it offers. However, an important question is left unanswered, which is if the business uses cloud services in a mission critical area, how does the business ensure that the service provider will faithfully fulfill their service obligations?

One may think that the first step to answer this question is to deploy some tools to monitor the execution of the cloud service contract and check which party breaches their obligations. Then the question becomes, which monitoring tool is to be used and who should operate it? And if a dispute arises during the execution of the service contract, how does the dispute get resolved? Obviously the service consumer cannot rely on the service provider's provision of the monitor (*i.e.*, Amazon AWS's CloudWatch) and vice versa, because of the problem of conflict of interest.

6.1.1 Problems Associated with a Centralised Solution

One straightforward solution is to use a trusted-third party (TTP) acting as a central authority to monitor the execution of the service contract and arbitrate any disputes that may arise. However, the immediate problem with the solution is how to ensure the fairness and independence of the TTP. The second problem is the excessive cost in operating a centralised authority in the Internet environment. Other problems include poor scalability, poor performance and single-point of failure associated with a centralised model. These problems have been confirmed by Milojicic *et al.* [137], who have concluded that the centralised or client/server models have disadvantages compared to the peer-to-peer model, in terms of cost of ownership, scalability, aggregate performance, aggregate fault resilience, transparency and anonymity.

6.1.2 Challenges in a Peer-to-Peer Solution

Although a peer-to-peer solution has architectural strengths over a centralised one in the Internet environment, there are also a lot of technological challenges in providing service contract management and dispute arbitration using the peer-to-peer model. In particular, these challenges are listed as follows.

CH1: Without a centralised registry, how to enable service contract publication and discovery?

CH2: In a peer-to-peer network where trust is scarce, how to establish a source of truth for service contract specifications and contract execution logs?

CH3: When a dispute arises during contract execution, how to choose arbiters in order to ensure fairness and avoid collusion? Can the dispute be arbitrated in an environment where an authority is absent?

CH4: During the arbitration process, how to achieve a reasonable arbitration accuracy while keeping the overall arbitration cost low in a peer-to-peer environment?

CH5: How to make sure that the honest arbiters' incentives are higher than malicious arbiters' incentives in order to maintain an objective arbitration process?

6.1.3 Our Approach to Address the Challenges

In this chapter, we present a peer-to-peer service contract management scheme, together with a dispute arbitration protocol. Our approach adapts the blockchain technology from Bitcoin [165] to a cloud environment, avoiding the problems inherent to a centralised model and, at the same time, overcoming the aforementioned five challenges (**CH1-CH5**) in a peer-to-peer environment. With our approach, every service participant maintains a local service registry and a service blockchain that is used as a public service activity ledger, eliminating the **information asymmetry** that usually exists in a service society. More importantly, through the "proof of work" (POW) mining process [165], it guarantees that dishonest participants **cannot tamper** with the blockchain, which is the consensus of the honest participants as underpinned by the Byzantine Agreement [67]. Furthermore, by utilising the commitment scheme [147] and the majority function, together with the POW mining, it provides a **fair** and **accurate** arbitration protocol for a peer-to-peer environment.

6.1.4 Background on Decentralised Infrastructure

The most prominent example of a decentralised infrastructure on the Internet is the Bitcoin system. In 2009, Nakamoto published the Bitcoin paper [165], in which a peer-to-peer electronic cash system was proposed. A year later, Satoshi released the implementation of the Bitcoin system. The cornerstone of the Bitcoin system is the *blockchain* technology. The *blockchain* functions like a transaction ledger, together with the "proof of work" (POW) mining process, Satoshi's Bitcoin protocol sorts the critical double-spending problem (essentially, spending the same coin more than once) in crypto-currency without a central authority [59], with the assumption that a majority of CPU power in the Bitcoin network is controlled by honest nodes that are not cooperating to attack the network. As such, the longest blockchain represents the consensus of the transaction history. Without the presence of a centralised authority, the Bitcoin network has been successfully operating for over five years, handling crypto-currency issuing, circulation, payment, settlement and clearance that traditionally can only be performed by a highly regulated central bank [106].

The Bitcoin system broadcasts transactions to all the nodes in the network. Each node collects the new transactions into a block, then works on finding a difficult "proof-of-work" for its block, which is called the "mining" process. The mining process involves scanning for a value when it is hashed with SHA-256, and the result begins with a number of zero bits. Since the average scanning work required is exponential in the number of zero bits required but can be verified by simply executing a single hash, it is a perfect approach to present a "proof-of-work". The number of zero bits is used as a parameter to adjust the difficulty, i.e., the average time a block is found, which normally is around ten minutes. When a node finds a proof-of-work, it broadcasts the block to all the nodes. The other nodes accept the block only if all transactions in it are valid and not already spent. Once accepted, each node links the block with the previously accepted blocks by using the previous hash, thus forming a chain of blocks, which is called the blockchain in the Bitcoin community. The blockchain functions like a transaction ledger, together with the mining process, Satoshi's Bitcoin protocol sorts the critical double-spending problem in crypto-currency without a central authority, with the assumption that a majority of CPU power in the Bitcoin network is controlled by honest nodes that are not cooperating to attack the network. As such, the longest blockchain represents the consensus of the transaction history.

Since the release of the Bitcoin system, it has been successfully operating for over five years. The success of the Bitcoin system has triggered the flourish of Bitcoin-like crypto-currencies, called "alt coins" in [66]; the most notable ones are Litecoin, Dogecoin, Freicoin, Peercoin, NXT, just to name a few. In [8], the author categorises three different types of Bitcoin-like systems.

The first type is to alter the parameters of the "proof-of-work" generation, such as hashing algorithms, average time for mining a block, coin supply limit. Examples are LiteCoin, DogeCoin and Freicoin.

The second type is to alter the consensus forming approach, changing from “proof of work”(POW) to “proof of stake”(POS), or using a combination of POW and POS. Instead of requiring users to do a certain amount of power-intensive hashing “work”, POS requires users to own a certain stake of the currency in order for them to mine new coins and asks users to prove the ownership of a certain amount of currency to avoid the heavy power consumption of Bitcoin mining, as well as mitigating the 51% attack risks inherent from Satoshi’s original assumption. Examples are Peercoin, Blackcoin and NXT.

The third type of Bitcoin like systems are called “alt chain” [8], in which the crypto-currency is not the primary concern and the chain is altered for other purposes. The most notable examples are Colored Coins, Namecoin, Ethereum, CoinSpark, Counterparty and Mastercoin. Colored Coins uses the crypto-currency as a token to represent a physical tradable asset. Thus, the Colored Coins system is not just a crypto-currency system, but can also facilitate physical asset trading through the colouring process – associating a coin with an asset through meta-data. Namecoin is an another innovation that uses the blockchain technology as a naming registry, providing a DNS like service for the root-level .bit domain, free from the control of the centralised Internet naming authority – The Internet Corporation for Assigned Names and Numbers (ICANN). Ethereum is an opensource project that aims at building a decentralised application platform. On top of a Bitcoin like currency called *ether*, it provides a Turing-complete programming platform based on the blockchain ledger. Ethereum’s blockchain records *contracts*, which are coded in a byte-code like Turing-complete language. The term *contract* in Ethereum refers to an autonomous software agent that can send and receive *ether* payments, store data, and execute an infinite range of computable actions (hence Turing-complete). Ethereum brings in a new programming paradigm that enables distributed applications building on top of its blockchain infrastructure.

In [67], Garay *et al.* analysed Bitcoin's protocol in detail and proved two properties of the Bitcoin backbone protocol – "common prefix" and "chain quality". *Common prefix* means that the honest parties share the same blocks in their existing blockchain. *Chain quality* refers to the percentage of blocks in the blockchain contributed by the honest players. They show how the two properties can be used as a foundation for designing Byzantine Agreement (BA) and robust transaction ledger protocols. BA considers a set of n parties, connected by reliable and authenticated pair-wise communication links and with possible initial inputs, that wish to agree on a common output in the presence of the malicious behaviour of some others [67]. In essence, Byzantine Agreement is a protocol for honest players to get consensus in a peer-to-peer network where adversaries are present. It is designed to address the Byzantine Generals Problem [111]. Garay *et al.* find that for both the properties to hold, the honest majority, *i.e.*, the adversary's hashing power, should be strictly less than 50%.

Extending Bitcoin's distributed consensus mechanism to areas outside of cryptocurrency is gathering momentum. We observe that the blockchain technology can be used to address the accountability concerns in the execution of service contracts and dispute resolution. Furthermore, there have been no existing works on the distributed accountability infrastructure for service computing. Built upon Bitcoin's blockchain technology and the public ledger theory presented by Garay *et al.* in [67], our work irons out the design of a basic architecture for a distributed accountability infrastructure, which enables monitoring of the execution of a service contract and arbitrating disputes in a peer-to-peer fashion without the presence of a trusted-third party.

6.2 A Distributed Cloud Service Contract Management Scheme (SCMS)

A service contract management scheme (SCMS) consists of a static component and a dynamic component. The static component defines the basic constructs like service concept definitions, contract specifications, service registry and contract execution log. The dynamic component describes the service contract management process, which involves the publication of service offers, service discovery, contract negotiation, contract establishment, contract execution tracking, monitoring and dispute resolution. We leave the contract negotiation topic as future work.

In the static component, rather than using a centralised registry, our approach is to keep a local service registry for each participant in order to enable the publication of service offers and service discovery. In the dynamic component, the service provider will broadcast the service offer in the network, and each participant will receive the broadcast message and record the service offer in their local registry. Hence the service consumer can do service discovery in their own registry. Together with our previous work on the formal contract specifications in [203], this addresses the challenge **CH1** in Section 6.1. To address **CH2**, we extend Bitcoin's blockchain from a simple coin transaction ledger to a versatile service interaction public ledger, recording the activities from both the provider and the consumer during the execution of the service contract.

In the dynamic component, through the "proof of work" mining process, consensus can be established amongst the honest participants on the execution logs of the service contract as long as the collective computation power of the honest nodes exceeds 50% of that of the whole network. The dishonest participants cannot tamper with the honest participants' blockchains. Thus the source of truth for service contract execution can be established even in a peer-to-peer environment where trust is scarce.

Next, we first define the key concepts in the static component of SCMS as a basis for the discussion of the dynamic component.

6.2.1 Basic Concepts

Definition 31: (**Service Offer**) A service offer so is a service provider's solicitation to a service consumer for entering into a contract, where certain capabilities are guar-

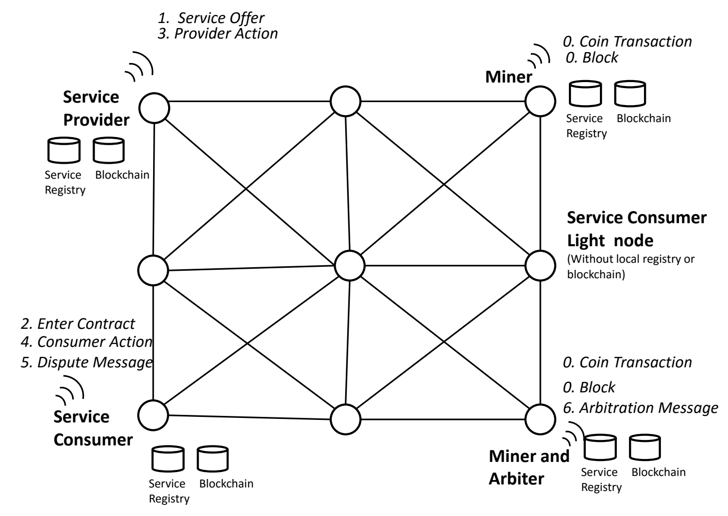


Figure 6.1: Service Network

anteed to be delivered to consumers if certain conditions are satisfied by the consumer. A service offer specifies service obligations and the associated processes for delivering the obligations.

Definition 32: (**Service Contract Establishment**) If a service offer is accepted by a service consumer, then the service offer becomes a service contract.

Note that we assume the existence of a formal specification language for a service offer/service contract. Our approach is oblivious to the underlying contract structure, as long as the contract representation can be interpreted by a machine. An example of such representation language of a service contract can be referred to [203], where a service offer can be represented as an obligation flow diagram (OFD) or an Accountable Process Algebra (APA) term (see Chapter 5 for details). An OFD is a directed acyclic graph (DAG) with obligations as nodes and the sequential or the parallel flow connectors as edges. A service offer is presented by $so = \langle Pr, O_{dag} \rangle$, where Pr is the service provider and O_{dag} is an obligation flow diagram that defines the obligations of both the service provider and the consumer.

Next we define a service network which is similar to the Bitcoin network, however,

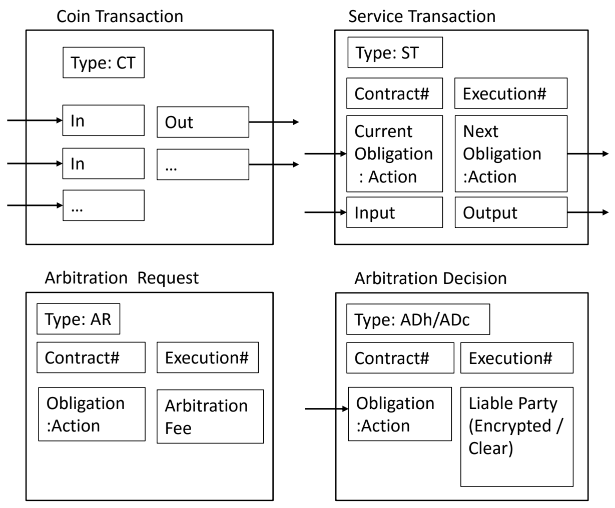


Figure 6.2: Transaction Types in Service Network

the prime focus of the network is to provide a distributed, accountable service contract management environment.

Definition 33: (**Service Network**) A service network sw is a peer-to-peer network, $sw = \langle N, CN \rangle$, where N is a set of nodes, and CN is a set of connections between pairs of nodes representing communication channels. For each node $n \in N$, n can play either one of, or a combination of four roles, being *PROVIDER*, *CONSUMER*, *MINER* and *ARBITER*. The *PROVIDER* role provides a service that allows the *CONSUMER* role to consume, whereas the *MINER* role mines the service coins while working on building the service blockchain. The *MINER* can choose to play the *ARBITER* role if a dispute request is raised in the network. A node can simultaneously play multiple roles at the same time.

The service coin and the mining process are exactly the same as those in Bitcoin. The service coins are used as incentives to get miners to find the "proof of work" and build up the blockchain.

Fig. 6.1 illustrates an overview of the service network. As shown in Fig. 6.1, each node has a local service registry database and a blockchain database. Each node broadcasts messages to their immediate neighbours.

Definition 34: (**Transaction**) A *transaction* tx in a *service network* is a record of the events in the network. There are four type of transactions: *Coin Transaction*, *Service Transaction*, *Arbitration Request* and *Arbitration Decision*. A *transaction* is generated by the current actor by digitally signing a hash of the previous transaction and the next actor's public key.

$$tx = Sign((Hash(tx_p, pubk_{nxt\_act}), prik_{cur\_act})$$

where $Sign$ is a signing function, $Hash$ is a hashing function, tx_p is the previous transaction, $pubk_{nxt\_act}$ is the public key of the next actor, and $prik_{cur\_act}$ is the private key of the current actor.

A Bitcoin network only has *Coin Transaction*, whereas the transactions in a service network are more versatile. Fig. 6.2 shows the four transaction types used in the service network.

The first transaction type *Coin Transaction* (CT) is exactly the same as the Bitcoin transaction. The owner transfers money by using his private key to sign a hash of the previous transaction and the public key of the new owner. The total amount in the *Out* field must be less than or equal to the total amount of the *In* field.

The second transaction type is *Service Transaction* (ST). It has the contract reference number from the service registry, and the contract execution instance number. It also records the current obligation/activity name pair and the next obligation/activity name pair as prescribed in the *service offer*. The *Input* field is the output generated in the last service transaction, whereas the *Output* field is the output generated in the current activity. The *Input* field must be signed by the previous transaction's actor, whereas the *Output* field must be signed by the current actor.

The third transaction type is *Arbitration Request* (AR). Apart from having the contract reference number and contract execution instance number, it also records the obligation and activity name pair in dispute, and announces an arbitration fee.

Algorithm 4: POW calculation

```
Data: Previous block hash $pb_h$, transaction root hash $r_h$, difficulty $D$ and starting nonce $non$
Result: $pow$
begin
    while $pow > D$ do
        $pow = Hash_{SHA-256}(pb_h, non, r_h);$
        $non$++;
    end
    Return $pow;$
end
```

The fourth transaction type is *Arbitration Decision* (AD), which contains a decision that records the address of the liable party in the transaction. It has two subtypes, one is ADh, which records the encrypted version, whereas the other one is ADc, which records the clear text version. This is the commitment scheme used to avoid collaboration amongst the arbiters.

Based on the "proof of work" (POW) concept from Bitcoin [165], here we define it more precisely for our *service network*.

Definition 35: (**Proof of Work (POW) mining**) A POW pow mining involves scanning for a value non, when hashed with an algorithm like SHA-256, the hash begins with a number of zero bits. The number of zero bits can be used to adjust the computation difficulty. pow can be calculated by using Algorithm 4.

In Algorithm 4, transactions in a block are hashed in a Merkle Tree [133]. The root hash r_h is included in the block hash. The difficulty D represents the predefined number that begins with a certain number of zero bits. The block hash only becomes a POW once it is equal to or less than D.

Built upon the above definitions, a *service public ledger* is defined as follows.

Definition 36: (**Service Public Ledger**) A *service public ledger* is a chain of blocks that contains service transactions. Each block is identified with a block hash b_h, *i.e.*, a POW. The block's hash is a hash of the previous block's hash pb_h, a nonce non and the transactions' root hash r_h. The blocks are chained to the genesis block [8], *i.e.*, the initial block, which contains the very first mined coins.

$$b_h = Hash_{SHA-256}(pb_h, non, r_h)$$

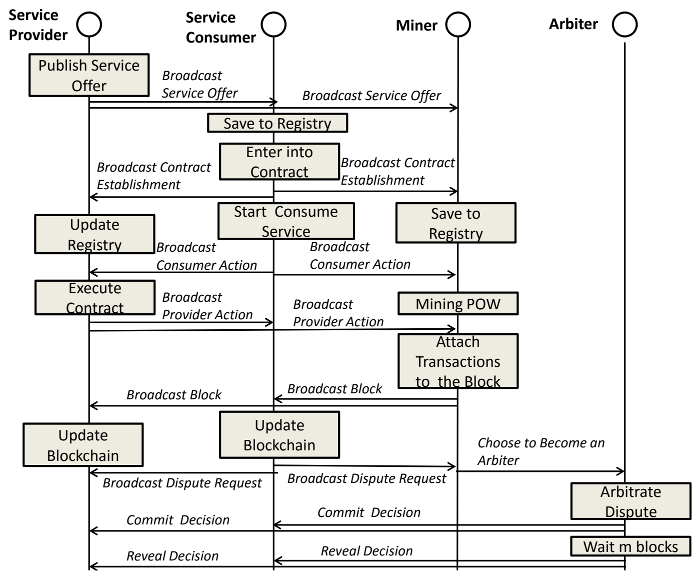


Figure 6.3: Contract Management Interactions

6.2.2 Service Contract Management Process

Now we discuss the dynamic component of the SCMS. The interaction sequence amongst various roles is shown in Fig. 6.3 and the typical scenario in a service network is described as follows.

a) A service provider broadcasts a *service offer* to the network. See the item tagged with "1" in Fig. 6.1. All the nodes will accept the *service offer* and save it to a local service registry;

b) A service consumer checks the capabilities of the *service offer*. If the consumer is satisfied with the service, it will broadcast a *service contract* establishment event to the network. See item tagged with "2" in Fig. 6.1. All the nodes will accept the contract establishment event and save it to their local service registries;

c) The service provider and the consumer will start to execute the service contract. He/she broadcasts the service transaction to the network after he/she completes an activity based on the service contract. See the items tagged with “3” and “4” in Fig. 6.1. Each node will receive the transaction and save it to a temporary storage;

d) In the background, there will be a lot of mining nodes engaged in mining service coins with exactly the same mechanism as that of the Bitcoin mining. If a mining node finds a block, he/she will put valid transactions from the temporary storage into the block, using the Merkle Tree [165] approach to hash transactions. Then it will broadcast the block to the network. See the item tagged with “0” in Fig. 6.1. The miner also receives service coins for the mining and transaction recording.

e) Each node will receive the block. It will first validate the block hash and then update its local version of the block chain. If a fork [8] occurs, the same principle as that of Bitcoin applies, that is, the longest chain will be taken as the consensus and treated as the public ledger of the service network, which reflects the source of truth for the execution log of the contract.

6.2.3 Theoretical Foundation of Public Ledger in Our Approach

While Bitcoin only uses blockchain as a coin transaction ledger to solve the double-spending problem in crypto-currency, our approach uses the blockchain as a public ledger for all interactions involved in a service contract execution. Our approach effectively needs to solve a distributed consensus problem, which is the main concern of Byzantine Agreement (BA).

After formally analysing Bitcoin’s backbone protocol, Garay *et al.* [67] present a “Public Transaction Ledgers and BA for honest majority” protocol (Public Ledger $\prod_{PL}$), and show that the *common prefix* and *chain quality* ensure two properties needed by the ledger, *i.e.*, *persistence* and *liveness*. *Persistence* indicates that once a transaction goes more than m blocks “deep” into the blockchain of one honest player, then it will be included in every honest player’s blockchain with an overwhelming probability, and it will be assigned a permanent position in the ledger. *liveness* states that all transactions originating from honest players will eventually end up at a depth of more than m blocks in an honest player's blockchain, and the adversary cannot alter those transactions in the honest player's blockchain. For both the properties to hold an honest majority is required, *i.e.*, the hashing power of the adversary is strictly less than 50%.

Therefore, the validity of our approach is backed by the (Public Ledger $\prod_{PL}$) protocol. The same principle can also be applied to arbitration in a peer-to-peer environment, as a consensus decision amongst arbiters formed by honest majority is a fair and accurate arbitration decision in most cases.

6.3 A Peer-to-Peer Dispute Arbitration Protocol (PPDAP)

Based on the SCMS, we further extend the dynamic model to include a dispute arbitration protocol. We first outline the objectives of the arbitration protocol as follows.

6.3.1 Objectives of the Arbitration Protocol

- **Fairness:** It should be impartial throughout the arbitration process, from the procedure of selection of arbiters to the decision making process.
- **High Accuracy:** It should ensure the final decision closely reflects the actual fact.
- **Sustainability:** It should exhibit the viability and durability of the arbitration system. It is the trade-off of enough incentives for honest arbiters to keep the system running while keeping the overall cost low.

With these objectives in mind, we design the dispute arbitration process in the following subsection.

6.3.2 Dispute Arbitration Process

Fig. 6.3 not only illustrates the typical interaction scenario in a service network as described in Subsection 6.2.2, it also shows the interaction sequence if a dispute is raised by either the service provider or the consumer. The arbitration process is described as follows.

1. During the execution of a service contract, if any party, for example, the service consumer, believes that the service provider violates an obligation, he/she can choose to broadcast a *dispute message*, with a fee attached for arbitration. See item tagged with "5" in Fig. 6.1.

2. The mining node which mined the POW block can also choose to perform the arbitration itself. The arbitration process involves checking the service transactions recorded in the blockchain against the service contract in the service registry, and determining which party is at fault.

3. After the arbiter makes the decision, a commitment scheme [147] will be used to ensure no collaboration amongst the arbiters during the arbitration, *i.e.*, the initial arbitration result will be encrypted (hiding) using the miner's public key so no one can see the decision, which will be included in the block as a transaction and be broadcast to the network. See the item tagged with "6" in Fig. 6.1.

4. The arbiter will later broadcast the clear text signed decision after a certain number of blocks (m) have been added to the blockchain. Each node can then verify that the hidden decision is the same as the clear text decision. This is the binding phase in a commitment scheme. Section 6.4 discusses the proper setting of m, *i.e.*, the number of blocks.

5. A majority function is applied to determine which party is liable for the dispute. The liable party should pay the arbitration fee to the arbiters whose decisions are aligned to the majority decisions through the service coin transaction. Failure to

pay the arbitration fee will be noticed by all the nodes and may be recorded in the blacklist in each node's service registry. The setting of the arbitration fee is discussed in Section 6.4.

Algorithm 5 implements the arbitration process.

A Peer-to-Peer arbitration protocol needs to overcome the challenges **CH3-CH4**, as listed in Section 6.1. In our protocol, we first take advantage of the randomness of the POW mining process, selecting arbiters from the successful miners who volunteer to be the arbiters. This meets **CH3**, as the selection of arbiters cannot be manipulated by either side of the dispute parties, yet the process needs no presence of an authority. We also use the commitment scheme to eliminate possible collusions amongst arbiters during the arbitration process. Achieving the high accuracy objective, *i.e.*, meeting the challenge **CH4** is the most difficult one in any arbitration process. We address it by leveraging the blockchain and the underlying Byzantine Agreement as a vehicle to build consensus amongst honest parties while keeping the overall cost low to the system, coupled with a majority function to achieve a reasonable level of accuracy in the final judgement. Furthermore, we need to work out the proper parameter settings for the protocol to address **CH5** and make sure the sustainability objective of the protocol is met. This is the main focus of Section 6.4.

6.3.3 Parameters of the Arbitration Protocol

Three main issues are left to be addressed in the arbitration protocol. One is to determine the percentage of honest nodes' collective computation power over the total power required in order to maintain a functioning service network and a reasonable arbitration accuracy. The second one is to determine the proper number m, *i.e.*, how "deep" in the blockchain that the block exhibits both "persistence" and "liveness" with an overwhelming probability [67]. This number influences fairness, accuracy and sustainability. Obviously, when m is larger, the final judgement is fairer and more accurate as it would have less chance for malicious miners or arbiters to fabricate their version

Algorithm 5: Boolean getArbitrationDecision()

```
Data: Block number $m$, Obligation Activity Pair $OA$, Contract ID $C_{id}$ and Contract Execution ID $CE_{id}$
Result: Boolean $providerLiable$
begin
    Create array $Arb[]$, $Decision[]$, $Msg[]$;
    $Provider\_Liable$ = false;
    $i$ = 0, int $proProvider$ = 0, int $proConsumer$ = 0;
    if ($Mod(m, 2) == 0$) /* $m$ needs to be an odd number */;
    $m$++ ;
    for $i$ from 1 to $m$ do
        /* return miners who mined a POW and willing to do arbitration */;
        $Arb[i] = getVoluntaryArbiter()$;
        /* call each arbiter to get decision based on contract spec and execution logs */
        $Decision[i] = makeArbDecision(Arb[i], OA, C_{id}, CE_{id})$;
        /*each arbiter commits (hides) the decision first*/;
        $Msg[i] = arbEncrypt(Decision[i], pubk_{Arb[i]})$;
        $Broadcast(Msg[i])$;
    end
    for $i$ from 1 to $m$ do
        /* each arbiter reveals decisions after all decisions committed */;
        $ArbBroadcast(Arb[i], Decision[i])$;
    end
    /* majority function, determine the final decision */;
    for $i$ from 1 to $m$ do
        /* Binding the commitment and the reveal decision */ ;
        if ($Binding(Msg[i], Decision[i], pubk_{Arb[i]})$ )
        if ( $Decision[i] == PROVIDER\_LIABLE$ ) $proConsumer$++;
        else $proProvider$++;
    end
    if ( $proConsumer == proProvider$ ) /* if the result is drawn due to binding failure, call the function one
      more time to get the decision */;
    Return ($getArbitrationDecision(1, OA, C_{id}, CE_{id}$);
    else if ($proConsumer > proProvider$) Return true;
    else Return false;
end
```

of decisions and the majority function would be more accurate as well. On the other hand, when m is larger, the overall arbitration cost would be higher, and the incentive for each honest arbiter will be lower. Therefore, a proper setting of m is a trade off amongst the three objectives. The third issue is to determine an ideal arbitration fee B, trading-off between the objectives of enough incentive to motivate honest arbiters and keeping the overall arbitration cost low.

We assume that given the opportunity to do arbitration, for an honest person who strictly follows the contract specification and checks the contract execution records to make judgment, the probability of making a correct judgement is p; the cost of going through the arbitration process is C, the probability of a correct judgement becoming a majority decision is p', the judgement fee received is B, and the probability of the provider being correct is q. Obviously, C can be treated as a constant.

In order to work out m and B, we need to first examine the different categories of arbiters' strategies adopted during arbitration and their respective expected incentives. There are five categories of arbiters: honest ones; rebels (*i.e.*, people who always give the opposite judgement); free riders (*i.e.*, people who randomly give judgement); pro-providers (*i.e.*, people who are bribed by the provider); and pro-consumer (*i.e.*, people who are bribed by the consumer). We here list these categories' strategies and work out their respective incentives as follows:

a) Expected incentives for an honest arbiter I_h:

$$I_h = p'(p(B-C)-(1-p)C)+(1-p')((1-p)(B-C)$$
$$-pC) = 2p'pB + B - C - pB - p'B$$

where $B = TotalDisputeFee/[(m/2+1)]$, m is the number of blocks, which is the same as the number of arbiters.

The expected incentives consist of two parts: one is when the majority function accurately reflects the fact (with probability p'), adding both incomes when the honest arbiter makes the right decision (with probability p), or otherwise (1-p); the other part is when the majority function is wrong (1-p'), also adding both incomes when the honest arbiter makes the right decision (with probability p), or otherwise (1-p);

b) Expected incentives for malicious people category one (rebels) I_r:

$$I_r = p'((1-p)(B-C)+p(-C))+(1-p')(p(B-C)+ \\ (1-p)(-C)) = p'B+pB-C-2p'pB$$

The rebels are always acting opposite to the honest arbiters. So, the incentive calculation can be adjusted accordingly.

c) Expected incentives of malicious people category two (free rider) I_f:

$$I_f = 50\% * B$$

Free rider will always have a 50% chance to get the arbitration income.

d) Expected incentives of malicious people category three (pro-provider) I_p:

$$I_p = p'qB+(1-p')(1-q)B = 2p'qB+BCqB-p'B$$

Similar to item a, we calculate the pro-provider's expected income in two cases, *i.e.*, when the majority function is right and wrong; and for each case, apply the probability (q), which is that the provider is actually right and the consumer is in fault.

e) Expected incentives of malicious people category four (pro-consumer) I_c:

$$I_c = p'(1-q)*B+(1-p')qB = p'B+qB-2p'qB$$

When the provider is right, then the consumer is wrong in a dispute situation. Therefore, we just reverse the calculation to get the expected incentives for the pro-consumer arbiters.

Note that we need to make sure that the honest arbiters' incentive is always greater than that of other categories. So the following constraint should be applied:

$$I_h > I_r, I_h > I_f, I_h > I_p, I_h > I_c$$

As we can see, there are many variables involved. Although intuitively we know that some variables are positively correlated (for example, p' and m, *i.e.*, when m becomes bigger, so does p'), there is no direct solution for establishing a precise and definite mathematical relationship for them. We therefore run simulations and analyse the dynamics of those variables and their impact on the objectives of the arbitration protocol in order to obtain the optimal settings of those variables.

6.4 Experiments and Analysis

We now conduct three experiments in this section. The first one is to analyse the malicious miners' probability of catching up an m block ahead blockchain, *i.e.*, the probability for them to tamper with the service public ledger. The second one is to analyse the arbitration accuracy (p') in relation to the number of blocks (m) and the percentage of computation power possessed by the honest arbiters versus that of various category of malicious arbiters. The third one is to analyse the average expected income for different categories of arbiter and the ideal benefit/cost ratio setting for the arbitration protocol.

6.4.1 Experiment 1: Malicious Miners Catching-up Probability

The first consideration of choosing a proper value of m must satisfy the fundamental requirements of a functioning service network, *i.e.*, after m blocks, the probabil-

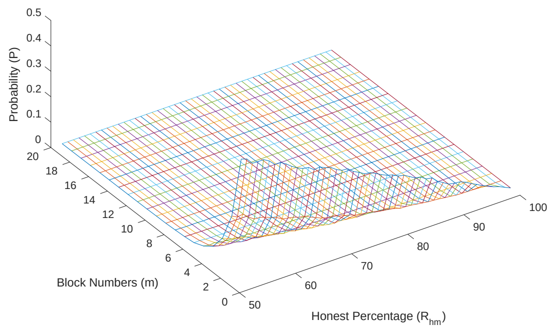


Figure 6.4: Experiment 1 - Malicious Miners Catchup Probability

ity of malicious miners catching-up and substituting their version of m blocks in the blockchain will be negligible. According to [165], the race between the honest chain and the malicious chain can be characterized as a Binomial Random Walk. Therefore, we can design an experiment, based on the probability of an honest miner mining a block and that of a malicious miner, to work out the probability of a malicious miner catching-up under different values of m. This experiment is repeated 10,000 times and the average results are shown in Fig. 6.4.

The results show that if $m > 6$, and the honest people's hashing power is higher than 50%, the chance of malicious people catching-up an m block blockchain is negligible. This experiment confirms the general practices in the Bitcoin network that only confirm a transaction after it is in six blocks deep of the blockchain, also the 50% honest hashing power threshold required by the Bitcoin backbone protocol as well as that of the Public Ledger $\prod_{PL}$ in [67].

6.4.2 Experiment 2 – scenarios analysis of different categories' strategies and the impact on arbitration accuracy

Based on the result of Experiment 1, we set $m > 6$, and honest people's hashing power greater than 50%. Now we investigate the relationships between the accuracy of the majority judgement function p', m and the honest/malicious hashing power ratio R_{hm}. Out of all the variables, the probability (p) of an honest arbiter making the objective decision and the probability (q) of a provider behaving rightfully according to the contract remain relatively stable, and they can be learned from training using the historical data set. Intuitively, p should be high (*i.e.*, close to 1) given a precisely defined contract specification and the execution logs. Also q should be higher than 50% to sustain a well-behaved service network. In order to simplify the discussion, we focus on m, p' and B/C ratio in our paper, and without loss of generality, we set p = 0.9 and q = 0.6 respectively.

The honest/malicious hashing power ratio R_{hm} can be further simplified to five scenarios: the first one is the honest/rebellion ratio R_{hr}; the second one is the honest/free rider ratio R_{hf}; the third one is the honest/pro-provider ratio R_{hp}; the fourth one is the honest/pro-consumer ratio R_{hc}; and the last one is the honest/even distribution of the above four various malicious groups R_{hem}.

Scenario 1: Relationships amongst p', m and R_{hr}

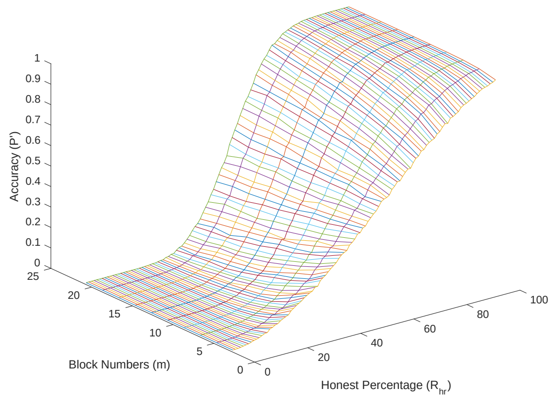


Figure 6.5: Experiment 2 Scenario 1 Honest/Rebellions Proportion

We design an experiment, with the assumption that there is a pre-known fact in terms of which party is liable in every run. Then, through a random walk based on a different range of probability values, we come up with a three dimensional diagram to illustrate the relationships amongst p', m and R_{hr}. For each combination of these three values, the experiment is repeated 10000 times and the average results are shown in Fig. 6.5. The results from Fig. 6.5 show that the accuracy is growing with the growing number of blocks and the honest percentage. When $R_{hr} \geq 81\%$, $m \geq 7$, the majority function accuracy $p' > 90\%$. This means that, with 81% hashing power from honest arbiters (the rest are rebels hashing power), the arbitration accuracy can achieve 90% at seven blocks.

Scenario 2: Relationships amongst p', m and R_{hf}

Similar to Scenario 1, we run through the simulation and plot Fig. 6.6. The results from Fig. 6.6 show that if the block number is seven, honest arbiters' hashing power is above 49% overall (the rest are the free riders' hashing power), then the majority function's accuracy can reach 90%.

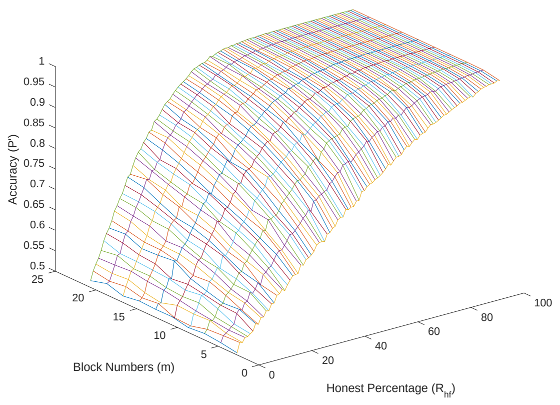


Figure 6.6: Experiment 2 Scenario 2 Honest/Free Riders Proportion

Scenario 3: Relationships amongst p', m, R_{hp}

Similar to Scenario 1, we run through the simulation and plot Fig. 6.7. The only difference is that all the arbiters are either honest arbiters or pro-providers. In Fig. 6.7 we can see that if the block number is seven, and the honest arbiters' hashing power is above 35% overall (the rest are the pro-providers' hashing power), then the majority function's accuracy of arbitration is above 90%.

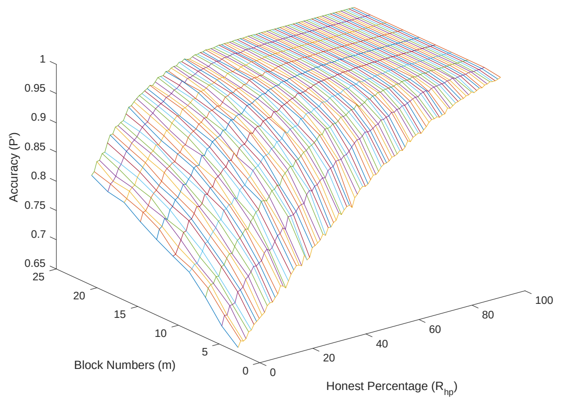


Figure 6.7: Experiment 2 Scenario 3 Honest/Pro-providers Proportion

Scenario 4: Relationships amongst p', m, R_{hc}

Similar to Scenario 1, we run through the simulation and plot Fig. 6.8, when all the dishonest arbiters are pro-customers.

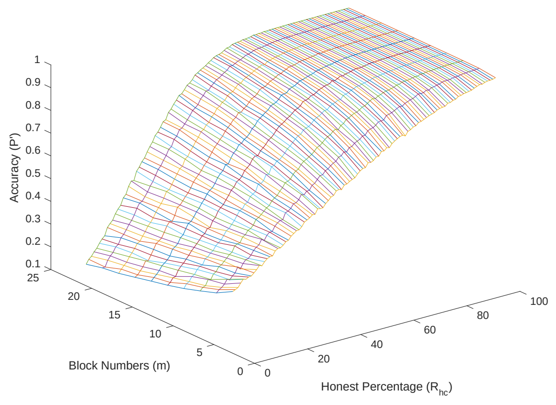


Figure 6.8: Experiment 2 Scenario 4 Honest/Pro-consumers Proportion

In Fig. 6.8, we can see that if the block number is 7, and the honest arbiters' hashing power is above 61% overall (the rest are the pro-consumers' hashing power), then the majority function's accuracy is above 90%.

Scenario 5: Relationships amongst p', m and R_{hc}.

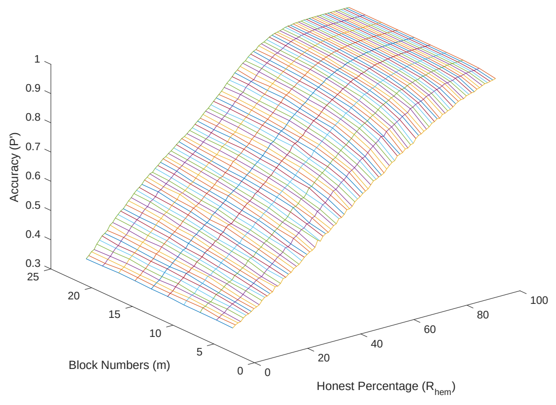


Figure 6.9: Experiment 2 Scenario 5 Honest/Even Distribution of Rest

Similar to Scenario 1, we run through the simulation and plot Fig. 6.9. However in this case, the dishonest arbiters consist of all the above four types, which are equally distributed.

In Fig. 6.9, we can see that if the block number is seven, and the honest arbiters' hashing power is above 68% overall (the rest are evenly distributed), then the majority function's accuracy is above 90%.

In comparison of the above five experiments shown in Figs. 6.5- 6.9, it can be concluded that with the use of seven blocks at arbitration, at least 81% of the honest arbiters are needed to guarantee 90% accuracy of the arbitration in the worst case. However, this case rarely happens, as the rebellious arbiters would not always take the lowest benefit expectation. In the normal case, only 68% of the honest arbiters are needed to guarantee a 90% accuracy.

6.4.3 Experiment 3

Scenario 1: Average Arbitration Benefit for Different Categories of Arbiters.

We calculate the average individual benefit of each type of arbiter when there are 81% honest arbiters and the dishonest arbiters consist of different types, as in scenario 5 of experiment 2. We set $p = 0.9$, $R_{hc} = 0.81$, $q = 0.6$.

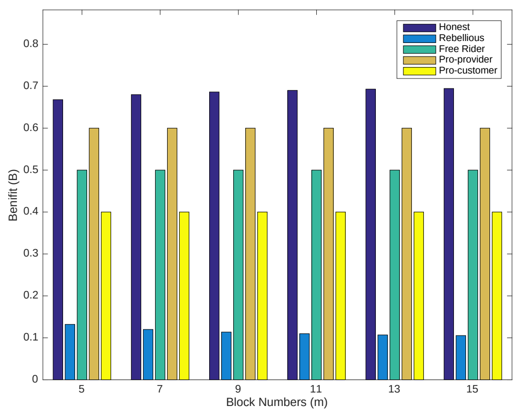


Figure 6.10: Experiment 3 Scenario 1 Average Incentive for Different Categories of Arbiters

The results from Fig. 6.10 show that, under the given parameter settings, we can ensure that the honest arbiters' incentive is greater than those of all other groups of arbiters.

Scenario 2: Average Incentive vs Cost/Benefit Ratio.

We now study the relationship between the average incentive and the cost/benefit ratio. Fig. 6.11 demonstrates the relationship of the average individual incentive and the cost/benefit ratio with 81$\%$ honest arbiters and 90$\%$ arbitration accuracy.

The results from Fig. 6.11 show that when the benefit of one single arbitration goes higher, the individual benefit of honest arbiters grows and gradually beats all the other types of arbiters, as the arbitration cost is constant. In the case with 81$\%$ honest arbiters achieving 90$\%$ arbitration accuracy, when the single arbitration benefit reaches eight times the arbitration cost, the individual benefit of honest arbiters is the highest.

Overall, the results of the three experiments indicate that when m=7, B/C = 8 and the honest arbiters' computation power is over 81$\%$ in the worst case or over 68$\%$ in the normal case, our arbitration protocol can achieve a better balance of fairness, accuracy and sustainability objectives.

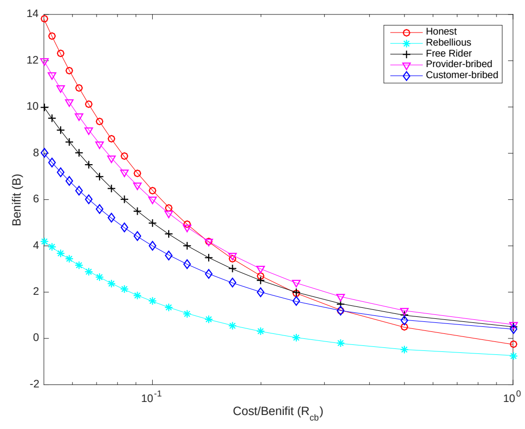


Figure 6.11: Experiment 3 Scenario 2 Average Incentive for Cost/Benefit Ratio

6.5 Summary and Further Work

As service computing is becoming mainstream, businesses gradually shift their emphasis from the functional aspect of service to the accountability aspect of service. Both the service providers and the consumers require an effective way to manage the risks associated with the execution of a service contract. However, viable solutions are currently lacking in the literature and the traditional centralised or client/server based solutions do not fit in the distributed Internet environment. In this paper, we have proposed a new peer-to-peer service contract management scheme (SCMS), and based on this, we designed a novel dispute arbitration protocol to address this problem.

The advantages of our approach can be emphasized as follows:

1. Our approach eliminates the need for a central authority and avoids its cost and other disadvantages.

2. Our distributed service registry and service blockchain concepts promote transparency and eliminate the information asymmetry in the service society.

3. We provide a service contract management scheme that tracks the execution of a service contract with tamper resistance quality, enabling monitoring of contract execution in a peer-to-peer fashion.

4. We present a distributed dispute arbitration protocol that exhibits fairness, impartiality and enough accuracy, even in an environment where malicious arbiters may exist;

5. We present the optimal parameter settings, striking the balance of fairness, accuracy and sustainability through thoughtful design of experiments/scenarios analysis.

In summary, our approach meets the five challenges (**CH1** – **CH5**) listed in Section 6.1. To the best of our knowledge, our approach is the first that provides a distributed service accountability infrastructure based on the blockchain technology. It is also the first that addresses the publication of cloud service contract specification, the tracking and monitoring of the execution of a service contract, and the arbitration of contract dispute in a peer-to-peer fashion. This provides the foundation for further enhancements in terms of contract negotiation, arbitration decision appeal and enforcement, which are the subjects of future work.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

Service-Oriented Architecture (SOA) is a new paradigm that turns a technology-focused software function to a business-specific representation of function that includes a notion of "contract" [23]. The notion of contract implies the accountability requirements for the service provider. Additionally, under SOA, a Web service becomes a commodity that can be bought, sold, and delivered in a similar manner to any other kind of service such as electricity or telecommunications [2]. The implication is that accountability will become a key consideration for the consumer when selecting a desired service. Also the boundary of IT and business is increasingly blurring, where business services are increasingly delivered through the IT service platform and, in the meantime, ever more IT services become important business services. Hence, it is imperative to establish a service accountability mechanism in the service-oriented architecture.

In this thesis, we have investigated the accountability literature in both the business and IT domains. A key finding of the investigation is that researchers in the IT community share a quite different view on the accountability concept than their counterparts in the business community. While in business accountability means transparency, responsibility, responsiveness and willingness to assume liability, the accountability concept usually gets mixed with QoS concepts or technical concerns, such as security, provenance and auditability, in IT. The gap in the conceptual understanding between

IT and business also leads to a gap between the business's expectation and the actual accountability capabilities in IT services. Both gaps present a worrying sign for the development of service computing.

Since nowadays cloud services have a much bigger business impact than that of other type of services, we have therefore focused our discussions on the context of cloud services. However, by no means can the accountability principles not be applied to other online services in general.

The first objective of this thesis is to raise awareness of the accountability gap between business and IT. The second objective is to provide a clearly defined service accountability foundation model that clarifies the confusion on the concept of accountability in service computing and highlights the notion of the "service contract" as the key concern in building service accountability and, more importantly, proposes a service accountability framework helping service providers to strength accountability in their service offering. The third objective is to provide practical approaches for building advanced service accountability mechanisms within the service-oriented architecture.

As a result of achieving the above three objectives, the contributions of this thesis are summarised below:

1. Presenting a unique view on service accountability that unifies the concerns from both business and the IT domains, which is crucial to bridge the accountability gap existing today.

2. Laying down a solid foundation model for service accountability, which includes:

    a) a precise definition of the basic service accountability concepts;

    b) clearly articulated accountability processes involving disclosure, contracting, monitoring and liability assignment;

    c) a quantitative accountability measurement approach; and

   d) a service accountability framework that addresses the weaknesses of current SOA architecture in accountability, helping service providers to build proper accountability capabilities into their service offerings.

3. Proposing a novel semantic service contract model that provides a machine-readable service contract representation, which enables advanced service accountability, such as automation of disclosure, monitoring and liability assignment processes.

4. Applying the semantic service contract model in service contract monitoring, extending the mainstream Representational State Transfer (REST) architecture to a novel Accountable State Transfer architecture and strengthening service accountability in SOA.

5. Proposing a novel algebraic service contract model that provides fine-grained obligation decomposition and efficient contract disclosure features. It provides:

   a) a formal logic called Dynamic Logic for Accountability (DLA) extended from Dynamic Logic, which addresses the unmanaged accountability issues such as the absence of SOW, lack of formal representations and validations of a contract in a cloud service;

   b) a simple tool called the Obligation Flow Diagram (OFD) for model validation and conflict resolution;

   c) a new approach that models a cloud service as a proactive system, rather than as a reactive system like that in the traditional modelling approach. The proactive system is concerned with the actors who conduct the activities and the exceptions which occur during the action execution, plus the causality behind them; and

   d) a novel Accountable Process Algebra (APA), which extends the traditional process algebra to a form of process algebra suitable for proactive systems like cloud services. APA allows analysis of the execution behaviour of a cloud service contract based on an algebraic approach, which produces a more concise model to reflect the dynamics of service accountability during the provision, as well as the consumption, of a cloud service.

6. Presenting a decentralised service contract management scheme that fits in the Internet environment and avoids the issues associated with centralisation, like cost, scalability, resilience, fairness and objectivity. It includes:

   a) a service contract management scheme to enable automatic publication of a service contract, automatic service discovery and selection. It improves transparency and accountability in service computing by eliminating the information asymmetry;

   b) a service blockchain that can be used as an anti-tamper public "activity ledger" for recording the interactions between the service provider and the consumer, enabling monitoring of contract obligation fulfilment in an open and objective environment;

   c) a novel dispute arbitration protocol that uses anonymous POW miners acting as arbiters to arbitrate a service contract dispute. Coupled with the commitment scheme and the majority function techniques, the protocol is designed to be fair, accurate and free from the influence of any authorities; and

   d) a study on the dynamics of the key parameters of the arbitration protocol through experiments and various scenarios analysis. The optimal parameter settings are studied, striking the balance of fairness, accuracy and accountability.

7.2 Future Work

Service accountability is a relatively new discipline that increasingly demonstrates its relevance in today's service computing environment, and it will become more important in the future while technologies in artificial intelligence (AI) and Internet of things (IOT) advance. This thesis lays down a foundation for service accountability in service-oriented architecture, paving the way for further research on accountability in IT in general. The following are some suggestions for future research in this direction.

From a service contract management life-cycle perspective, there is room for significant future work in each of the disclosure, contracting, monitoring, liability assignment and remedy processes.

A full semantic disclosure approach relies on a comprehensive set of ontologies covering all aspects of accountability, including domain knowledge concepts, especially for legal and regulation compliance and social ethics. Building and integrating ontologies for accountability is a huge undertaking, and thus is an ongoing, long term effort.

An important phase of contracting process is contract negotiation, an area that is predominantly conducted through manual process by humans at the moment. Using service agents to automate the negotiation process, based on a set of predefined rules, can provide efficiency, flexibility and elegance in forming a service contract.

In the monitoring process, research on reasoning and detection of obligation breach based on the evidence of the service contract execution is required to improve accuracy and efficiency in fault detection, while reducing the false-alarm rate.

The liability assignment process needs further research on the root cause diagnosis of an obligation breaches to ensure a high accuracy in arbitration decisions. It should also cater for an automatic arbitration decision appeal process.

The remedy process should provide capabilities in automatic rectifying of faults during service contract execution, as well as automatic means for the enforcing of a liability assignment.

As service computing is moving towards a proactive system-based paradigm, it is predicted that upholding service accountability will increasingly rely on artificial intelligence and computer cognition technologies. The interesting question of whether or not an intelligent agent should be accountable for the service obligation may become a research topic for future researchers.

Appendix A

Notations Used in This Thesis

Table A.1: Notations used in Chapter 3

Notation	Representation	First occurrence
s	a service	Section 3.3.2
CP	a set of collaboration process	Section 3.3.2
i	a service interface	Section 3.3.2
A	a set of actors	Section 3.3.2
cs	a cloud service	Section 3.3.2
$CIid$	the cloud infrastructure ID	Section 3.3.2
ecs	Amazon EC2 service	Section 3.3.2
O	the service obligation	Section 3.3.2
a	an action	Section 3.3.2
$input$	the input object	Section 3.3.2
$output$	the output object	Section 3.3.2
pre	the precondition	Section 3.3.2
$post$	the post condition	Section 3.3.2
e	the evidence	Section 3.3.2
$timestamp$	the timestamp for object creation	Section 3.3.2
O_p	the provider obligation	Section3.3.2
O_c	the consumer obligation	Section 3.3.2
sc	the service contract	Section 3.3.2
P	a pair of party	Section 3.3.2
sla	service level agreement	Section 3.3.2
sow	statement of work	Section 3.3.2
s_p	the service provider	Section 3.3.2
s_c	the service consumer	Section 3.3.2

Table A.2: Notations used in Chapter 3 (continued)

Notation	Representation	First occurrence
F_p	provider forbidden clause	Section 3.3.2
F_c	consumer forbidden clause	Section 3.3.2
R	the horn rule	Section 3.3.2
T	the contract period	Section. 3.3.2
$start\_time$	the started time	Section 3.3.2
end_time	the end time	Section 3.3.2
sce	the service contract execution	Section 3.3.2
I	contract execution information	Section 3.3.2
$complete\_time$	complete time	Section 3.3.2
se	the contract execution state	Section 3.3.2
SE	the set of contract execution state	Section 3.3.2
$consequent$	the consequent in a horn rule	Section 3.3.2
$antecedent$	the antecedent in a horn rule	Section 3.3.2
P_p	Burnoulli probability distribution	Eq. 3.1
p	the success probability	Eq. 3.1
q	the failure probability	Eq. 3.1
n	the number of success run	Eq. 3.1
N	the number of trials	Eq. 3.1
$f(t)$	the logistic function	Eq. 3.2
e	the base of the natural logarithm	Eq. 3.2
t	the logistic score	Eq. 3.2
p_i	value of logistic function	Eq. 3.2
V	the set of all CTA invariants	Section 3.5.3
v_k	an invariant	Section 3.5.3
PR	the set of preconditions	Section 3.5.3
PO	the set of postconditions	Section 3.5.3
pr_j	a precondition	Section 3.5.3
po_k	a post condition	Section 3.5.3
en	runtime nonfunctional error	Eq. 3.4
ef	runtime functional error	Eq. 3.5
cn	nonfunctional compensation indicator	Eq. 3.6
cf	functional compensation indicator	Eq. 3.7

Table A.3: Notations used in Chapter 3 (continued)

Notation	Representation	First occurrence
cd	the cumulated compensation deficits	Eq. 3.8
ce	the cumulated expected total compensation	Eq. 3.8
ca	the cumulated actual compensation	Eq. 3.8
dn	non functional disclosure indicator	Eq. 3.9
df	functional disclosure indicator	Eq. 3.10
dp	precondition disclosure indicator	Eq. 3.11
re	the responsiveness indicator	Eq. 3.12
ep	evidence provision indicator	Eq. 3.13
β_k	weight on kth variable	Eq. 3.14
x_k	kth independent variable	Eq. 3.14

Table A.4: Notations used in Chapter 4

Notation	Representation	First occurrence
SCM	a semantic service contract model	Section 4.2.2.2
K_{scm}	a service contract knowledge base	Section 4.2.2.2
$\mathcal{T}$	a Tbox consists of a finite set of concept inclusion axioms	Section 4.2.2.2
$\mathcal{A}$	an Abox consists of a finite set of individual or role assertions	Section 4.2.2.2
$\mathcal{H}$	a finite set of horn clause axioms	Section 4.2.2.2
$C \sqsubseteq D$	the concept inclusion axiom	Section 4.2.2.2
$R \sqsubseteq S$	the role inclusion axioms	Section 4.2.2.2
$Trans(R)$	the transitivity axiom	Section 4.2.2.2
C, D	OWL-DL concepts	Section 4.2.2.2
R, S	OWL-DL roles	Section 4.2.2.2
a, b	the individual objects in OWL-DL	Section 4.2.2.2
r, r_n	atoms in rules	Section 4.2.2.2
P	an individual value property	Section 4.2.2.2
Q	a data value property	Section 4.2.2.2
x, y	either individuals or variables	Section 4.2.2.2
z	either a variable or a data value	Section 4.2.2.2
act	an action	Section 4.2.2.2
$input$	input of the action	Section 4.2.2.2
$output$	output of the action	Section 4.2.2.2
pre	precondition of an action	Section 4.2.2.2
φ	a set of assertion in $\mathcal{A}$	Section 4.2.2.2
χ	a set of assertion of primitive literals for $\mathcal{T}$	Section 4.2.2.2
ev	an evidence object	Section 4.2.2.2
obj	an individual in K_{scm}	Section 4.2.2.2
$timestamp$	the timestamp that obj is created	Section 4.2.2.2
$cond$	a set of assertions with regard to obj	Section 4.2.2.2
ack	an acknowledgement object	Section 4.2.2.2
$\mathbb{K}$	apply the rule only to those known instances	Section 4.2.2.2

Table A.5: Notations used in Chapter 4 (continued)

Notation	Representation	First occurrence
$\mathcal{SHOIN}(\mathcal{D})$	OWL-DL language	Section 4.2.2.3
$\mathcal{SROIQ}(\mathcal{D})$	OWL-2 language	Section 4.2.2.3
Pro	a rule program	Section 4.2.2.3
CPN	a coloured Petri-net	Section 4.2.3.1
Σ	a finite set of colour sets	Section 4.2.3.1
PL	a finite set of places	Section 4.2.3.1
TR	a finite set of transitions	Section 4.2.3.1
AR	a finite set of arcs between place and transition	Section 4.2.3.1
ND	node function	Section 4.2.3.1
CF	a coloured function	Section 4.2.3.1
GF	a guard function	Section 4.2.3.1
EX	an arc express function	Section 4.2.3.1
IF	an initialization function	Section 4.2.3.1
v	a variable	Section 4.2.3.1
$Type(v)$	return the type of variable v	Section 4.2.3.1
$expr$	an expression	Section 4.2.3.1
$Var(expr)$	return the set of variables in expression $expr$	Section 4.2.3.1
$\mathbf{B}$	denotes the boolean type	Section 4.2.3.1
∂	a color set	Section 4.2.3.2
M_0	an initial marking	Section 4.2.3.2
$\mathcal{I}$	model of $\mathcal{T}$ and $\mathcal{A}$	Section 4.2.3.2
A_c	a set of consumer action	Section 4.2.3.2
T_c	a set of consumer transition	Section 4.2.3.2
A_p	a set of provider action	Section 4.2.3.2
T_p	a set of provider transition	Section 4.2.3.2
P_i	a set of token colours	Section 4.2.3.2
E_i	an input inscription	Section 4.2.3.2
E_o	an output inscription	Section 4.2.3.2
$\mathcal{I}'$	transition from $\mathcal{I}$	Section 4.2.3.2
$\mathcal{I}''$	transition from $\mathcal{I}'$	Section 4.2.3.2
$\mathcal{N}$	a Petri-net	Section 4.2.3.3
sup	the token upper bound	Section 4.2.3.3
RS	a reachability set	Section 4.2.3.3
E_o	an output inscription	Section 4.2.3.3

Table A.6: Notations used in Chapter 5

Notation	Representation	First occurrence
acs	an accountable cloud service	Section 5.2.1
i	a service interface	Section 5.2.1
CP	a set of collaboration process	Section 5.2.1
$CIid$	a cloud infrastructure id	Section 5.2.1
sc	a service contract	Section 5.2.1
P	a pair of party	Section 5.2.1
sow	a statement of work	Section 5.2.1
sla	service level agreement	Section 5.2.1
R	a set of horn rules	Section 5.2.1
T	the contract period	Section 5.2.1
s_p	the service provider	Section 5.2.1
s_c	the service consumer	Section 5.2.1
O_p	the service provider obligation	Section 5.2.1
O_c	the service consumer obligation	Section 5.2.1
F_p	service provider forbidden clauses	Section 5.2.1
F_o	service consumer forbidden clauses	Section 5.2.1
act	an action	Section 5.2.1
$input$	input of the action	Section 5.2.1
$output$	output of the action	Section 5.2.1
pre	precondition of an action	Section 5.2.1
r	an individual rule	Section 5.2.1
$start\_time$	the contract starting time	Section 5.2.1
end_time	a contract end time	Section 5.2.1
obj	an individual in K_{scm}	Section 5.2.1
$timestamp$	the timestamp that obj is created	Section 5.2.1
$L(Sig_{Dyn})$	dynamic logic	Section 5.3
Asn	a set of assertion formulas	Section 5.3
ϕ	an assertion	Section 5.3

Table A.7: Notations used in Chapter 5 (continued)

Notation	Representation	First occurrence
t_1, t_2, t_n	the first order logic terms	Section 5.3.0.1
ψ	an assertion	Section 5.3.0.1
$\forall x$	universal quantifier function	Section 5.3.0.1
$\exists x$	existential quantifier function	Section 5.3.0.1
P	a predicate symbol	Section 5.3.0.1
Φ, Φ_1, Φ_2	dynamic logic terms	Section 5.3.0.1
β	a process term	Section 5.3.0.1
a	an actor	Section 5.3.0.1
$V : a : \alpha$	a illegally performed or failed to perform α	Section 5.3.0.1
$\alpha)$	an action	Section 5.3.0.1
$P(\alpha)$	α is permitted	Section 5.3.0.1
$O(\alpha)$	α is obligatory	Section 5.3.0.1
$F(\alpha)$	α is forbidden	Section 5.3.0.1
$O([\alpha]\phi)$	an obligation to perform action α, and the outcome of α must satisfy ϕ	Section 5.3.0.2
$B(t)$	term t is breached	Section 5.3.0.2
v	a variable	Section 5.3.0.2
M	a DLA model	Section 5.3.0.3
W	the set of all possible states	Section 5.3.0.3
Γ	a function which associates each state with the condition it satisfies	Section 5.3.0.3
$\mathcal{R}$	a collection of binary relations on states	Section 5.3.0.3
w, w'	world state	Section 5.3.0.3
s_0, s_1, s_2, s_3	world states	Section 5.3.0.3
Σ	a signature	Section 5.4.1
f, g	function symbols	Section 5.4.1
$ar(f)$	arity of function f	Section 5.4.1
$x, y, z, ...$	a countable set of variables	Section 5.4.1
$T(\Sigma)$	a set of open term over Σ	Section 5.4.1
$T'(\Sigma)$	set of closed term over Σ	Section 5.4.1
S	a non-empty set of states	Section 5.4.1
L	a finite, non-empty set of transition labels	Section 5.4.1
$s \xrightarrow{\alpha} s'$	state s can evolve into state s' by the execution of action α	Section 5.4.1
$s \xrightarrow{\alpha} \surd$	s can execute α	Section 5.4.1
ρ	a transition rule	Section 5.4.1
$\frac{H}{\pi}$	positive premise H, conclusion π	Section 5.4.1
$\varepsilon(a)$	exception raised when perform a	Section 5.4.1
Σ_1	a collaboration process term	Section 5.4.1

Table A.8: Notations used in Chapter 6

Notation	Representation	First occurrence
so	a service offer	Section 6.2.1
Pr	a service provider	Section 6.2.1
O_{dag}	an obligation flow diagram	Section 6.2.1
sw	a service network	Section 6.2.1
N	a set of nodes in a service network	Section 6.2.1
CN	a set of connections between nodes	Section 6.2.1
n	a node in a service nework	Section 6.2.1
tx	a service transaction	Section 6.2.1
$Sign$	a signing function	Section 6.2.1
$Hash$	a hashing function	Section 6.2.1
tx_p	the previous service transaction	Section 6.2.1
$pubk_{nxt\_act}$	the public key of the counter-party nxt_act	Section 6.2.1
$prik_{cur\_act}$	the private key of the current actor cur_act	Section 6.2.1
pow	a proof of work	Section 6.2.1
pb_h	the previous block hash	Section 6.2.1
r_h	the transaction root hash	Section 6.2.1
D	the difficulty set in the Bitcoin network	Section 6.2.1
non	a nonce	Section 6.2.1
b_h	a block hash	Section 6.2.1
$\prod_{PL}$	a public ledger	Section 6.2.3
m	the number of blocks ahead of a transaction	Section 6.2.3
OA	an obligation-activity pair	Algorithm 5
C_{id}	a service contract id	Algorithm 5
CE_{id}	a contract execution id	Algorithm 5
$providerLiable$	provider liable boolean indicator	Algorithm 5
$Arb[]$	array for arbiters	Algorithm 5
$Decision[]$	decision array	Algorithm 5
$Msg[]$	message array	Algorithm 5

Table A.9: Notations used in Chapter 6 (continued)

Notation	Representation	First occurrence
B	the arbitration fee	Section 6.3.3
C	the cost for an arbitration	Section 6.3.3
p	the probability for an arbiter to make a correct judgement α	Section 6.3.3
p'	the probability of a correct judgment becoming a majority decisionα	Section 6.3.3
q	the probability of a provider being correctα	Section 6.3.3
I_h	expected incentives for a honest arbiter	Section 6.3.3
I_r	expected incentives for a rebel	Section 6.3.3
I_f	expected incentives for a free rider	Section 6.3.3
I_p	expected incentives for a pro-provider	Section 6.3.3
I_c	expected incentives for a pro-consumer	Section 6.3.3
R_{hm}	the honest/malicious hashing power ratio	Section 6.4.2
R_{hr}	the honest/rebellion hashing power ratio	Section 6.4.2
R_{hf}	the honest/freerider hashing power ratio	Section 6.4.2
R_{hc}	the honest/pro-consumer hashing power ratio	Section 6.4.2
R_{hp}	the honest/pro-provider hashing power ratio	Section 6.4.2
R_{hem}	the honest/even distribution of malicious hashing power ratio	Section 6.4.2

Bibliography

[1] M. Abadi and N. Glew. Certified email with a light on-line trusted third party: Design and implementation. *WWW*, pages 387–395, 2002.

[2] P. R. Allen and S. Higgins. *Service orientation winning strategies and best practices*. Cambridge University Press, 2006.

[3] Amazon. Amazon s3 interface. In *http://s3.amazonaws.com/doc/2006-03-01/AmazonS3.wsdl*, 2006.

[4] Amazon. Amazon s3 service level agreement. In *https://aws.amazon.com/cn/s3-sla*, 2007.

[5] Amazon. AWS customer agreement. In *https://aws.amazon.com/cn/agreement*, 2012.

[6] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web services agreement specification (WS-Agreement). In *Open Grid Forum*, volume 128, page 216, 2007.

[7] A. Ankolekar, M. Burstein, J. R. Hobbs, O. Lassila, D. Martin, D. McDermott, S. A. McIlraith, S. Narayanan, M. Paolucci, T. Payne, et al. DAML-S: Web service description for the semantic web. In *The Semantic WebSWC 2002*, pages 348–363. Springer, 2002.

[8] A. M. Antonopoulos. *Mastering Bitcoin*. OReilly, 2015.

[9] W. Artley. Establishing accountability for performance. *The Performance Based Management Handbook*, 3, 2001.

[10] N. Asokan, V. Shoup, and M. Waidner. Asynchronous protocols for optimistic fair exchange. In *Security and Privacy - 1998 IEEE Symposium on Security and Privacy, Oakland, CA, USA, May 3-6, 1998, Proceedings*, pages 86–99, 1998.

[11] AWS. AWS cloudwatch. In *https://aws.amazon.com/cloudwatch/*, 2016.

[12] F. Baader. *The description logic handbook: Theory, implementation and applications*. Cambridge University Press, 2003.

[13] F. Baader, C. Lutz, M. Milicic, U. Sattler, and F. Wolter. Integrating description logics and action formalisms: First results. In *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 572–577, 2005.

[14] J. C. Baeten. A brief history of process algebra. *Theoretical Computer Science*, 335(2):131–146, 2005.

[15] R. Barrett and P. P. Maglio. Intermediaries: New places for producing and manipulating web content. *Computer Networks*, 30(1-7):509–518, 1998.

[16] S. Baskarada, A. Koronios, and J. Gao. Towards a capability maturity model for information quality management: A TDQM approach. In *Proceedings of the 11th International Conference on Information Quality, MIT, Cambridge, MA, USA, November 10-12, 2006*, pages 499–510, 2006.

[17] R. Basu, N. Wright, R. Basu, and N. Wright. Quality beyond Six Sigma. *TQM Magazine*, 15(6):424–424, 2003.

[18] M. Ben-Or, O. Goldreich, S. Micali, and R. L. Rivest. A fair protocol for signing contracts. *IEEE Trans. Information Theory*, 36(1):40–46, 1990.

[19] W. L. Benoit. *Accounts, Excuses, and Apologies: A Theory of Image Restoration Strategies*. Marcombo, 1995.

[20] A. Beugnard, J. Jézéquel, and N. Plouzeau. Making components contract aware. *IEEE Computer*, 32(7):38–45, 1999.

[21] A. Beugnard, J. Jézéquel, and N. Plouzeau. Contract aware components, 10 years after. pages 1–11, 2010.

[22] M. Bichler and K. Lin. Service-oriented computing. *IEEE Computer*, 39(3):99–101, 2006.

[23] N. Bieberstein. *Service-oriented architecture compass: business value, planning, and enterprise roadmap*. FT Press, 2006.

[24] Bitcoinwiki. Contract. In *https://en.bitcoin.it/wiki/Contract*. Bitcoin wiki.

[25] S. Bosworth and M. E. Kabay. *Computer security handbook*. John Wiley & Sons, 2002.

[26] G. E. Box and D. R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 211–252, 1964.

[27] D. M. Brandon. Project management for modern information systems. *Electronic Library*, 25(3):378–379, 1983.

[28] G. A. Brosamler. An almost everywhere central limit theorem. *Math. Proc. Cambridge Philos. Soc*, 104(3):561–574, 1988.

[29] F. J. Buera and J. P. Kaboski. The rise of the service economy. Technical report, National Bureau of Economic Research, 2009.

[30] J. Bundt. Strategic stewards: Managing accountability, building trust. *Journal of Public Administration Research and Theory*, 10(4):757–778, 2000.

[31] P. Buneman, A. Chapman, and J. Cheney. Provenance management in curated databases. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 539–550. ACM, 2006.

[32] P. Buneman, S. Khanna, and W.-C. Tan. Data provenance: Some basic issues. In *FST TCS 2000: Foundations of software technology and theoretical computer science*, pages 87–93. Springer, 2000.

[33] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings.*, pages 316–330. 2001.

[34] V. Buterin. A next-generation smart contract and decentralized application platform. In *https://github.com/ethereum/wiki/wiki/White-Paper*, 2014.

[35] S. C. Certo. *Principles of modern management: Functions and systems*. WCB/McGraw-Hill, 1986.

[36] B. N. Chun and A. C. Bavier. Decentralized trust management and accountability in federated systems. In *37th Hawaii International Conference on System Sciences (HICSS-37 2004), CD-ROM / Abstracts Proceedings, 5-8 January 2004, Big Island, HI, USA*, 2004.

[37] K. Cline, J. Cohen, D. Davis, D. F. Ferguson, H. Kreger, R. McCollum, B. Murray, I. Robinson, J. Schlimmer, J. Shewchuk, et al. Toward converging web service standards for resources, events, and management. *A Joint White Paper from Hewlett Packard Corporation, IBM Corporation, Intel Corporation and Microsoft Corporation*, 2006.

[38] F. Coallier. How ISO 9001 fits into the software world. *IEEE Software*, 11(1):98–100, 1994.

[39] F. Collins. Managerial accounting systems and organizational control: a role perspective. *Accounting, Organizations and Society*, 7(2):107–122, 1982.

[40] R. Corin, S. Etalle, J. den Hartog, G. Lenzini, and I. Staicu. *A logic for auditing accountability in decentralized systems*. Springer, 2005.

[41] A. C. G. Council and A. S. Exchange. Corporate governance principles and recommendations, 2007.

[42] CPNtools. CPN-Tools. In *http://cpntools.org/*, 2015.

[43] B. Crispo and G. Ruffo. Reasoning about accountability within delegation. In *Information and Communications Security*, pages 251–260. Springer, 2001.

[44] CSA. Top threats to cloud computing v1.0. In *https://cloudsecurityalliance.org/topthreats/csathreats.v1.0.pdf*. Cloud Security Alliance, 2010.

[45] Y. Cui and J. Widom. Practical lineage tracing in data warehouses. In *ICDE*, pages 367–378, 2000.

[46] Y. Cui and J. Widom. Lineage tracing for general data warehouse transformations. *VLDB J.*, 12(1):41–58, 2003.

[47] Y. Cui, J. Widom, and J. L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Transactions on Database Systems (TODS)*, 25(2):179–227, 2000.

[48] S. R. Curran. Agency, accountability, and embedded relations: what's love got to do with it?. *Journal of Marriage and Family*, 64(3):577–584, 2002.

[49] A. Dan, D. M. Dias, R. Kearney, T. C. Lau, T. Nguyen, F. N. Parr, M. W. Sachs, and H. Shaikh. Business-to-business integration with tpaML and a business-to-business protocol framework. *IBM Systems Journal*, 40(1):68–90, 2001.

[50] A.-K. Daskalopulu. *Logic-Based Tools for the Analysis and Representation of Legal Contracts*. PhD thesis, Imperial College London (University of London), 1999.

[51] H. Davulcu, M. Kifer, and I. V. Ramakrishnan. CTR-S: a logic for specifying contracts in semantic web services. In *Proceedings of the 13th international*

conference on World Wide Web - Alternate Track Papers & Posters, WWW 2004, New York, NY, USA, May 17-20, 2004, pages 144–153, 2004.

[52] L. Diamond. Institutions of accountability. *Hoover Digest*, 3:87–91, 1999.

[53] O. E. Dictionary. Oxford english dictionary online. *Mount Royal College Lib., Calgary*, 14, 2004.

[54] M. J. Dubnick and J. B. Justice. Accounting for accountability. In *Annual Meeting of the American Political Science Association*, pages 2–5, 2004.

[55] W. W. Eckerson. *Performance Dashboards: Measuring, Monitoring, and Managing Your Business*. John Wiley & Sons, 2010.

[56] J. R. Erenkrantz, M. M. Gorlick, G. Suryanarayana, and R. N. Taylor. From representations to computations: the evolution of web architectures. In *Proceedings of The 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2007, Dubrovnik, Croatia, September 3-7, 2007*, pages 255–264, 2007.

[57] S. Eriksén. Designing for accountability. In *Proceedings of the Second Nordic Conference on Human-Computer Interaction 2002, Aarhus, Denmark, October 19-23, 2002*, pages 177–186, 2002.

[58] J. Famaey, J. Donders, T. Wauters, F. Iterbeke, N. Sluijs, B. De Vleeschauwer, F. De Turck, P. Demeester, and R. Stoop. Comparative study of peer-to-peer architectures for scalable resource discovery. In *2009 First International Conference on Advances in P2P Systems*, pages 27–33, 2009.

[59] R. Farell. An analysis of the cryptocurrency industry. *Wharton Research Scholars Journal, Paper 130*, 2015.

[60] A. D. Farrell, M. J. Sergot, M. Salle, C. Bartolini, D. Trastour, and A. Christodoulou. Performance monitoring of service-level agreements for utility computing using the event calculus. In *Proceedings of the 1st International Workshop on Electronic Contracting*, pages 17–24. IEEE, 2004.

[61] W. Feller. *An introduction to probability theory and its applications. Vol. I.* Wiley, 1950.

[62] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[63] D. Fisher. Lawmakers, scholars warn against reactionary anti-wikileaks legislation. In *https://threatpost.com/lawmakers-scholars-warn-against-reactionary-anti-wikileaks-legislation-121610/74785/*. Threatpost.com, 2010.

[64] W. Fokkink. *Introduction to Process Algebra*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2000.

[65] C. L. Forgy. *On the Efficient Implementation of Production Systems*. PhD thesis, Carnegie-Mellon University, 1979.

[66] P. Franco. Alt(ernative) coins - understanding bitcoin: Cryptography, engineering, and economics. 2014.

[67] J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. *Advances in Cryptology - EUROCRYPT 2015 Volume 9057 of the series Lecture Notes in Computer Science*, pages 281–310, 2015.

[68] R. S. Gerber. Mixing it up on the web: Legal issues arising from internet mashups. *Intellectual Property and Technology Law Journal*, 18(8):11–14, 2006.

[69] B. K. Gibb and S. Damodaran. *ebXML: Concepts and application*. John Wiley & Sons, 2002.

[70] M. Gibbins and J. D. Newton. An empirical exploration of complex accountability in public accounting. *Journal of Accounting Research*, 32(2):165–186, 2010.

[71] C. Golbreich and A. Imai. Combining SWRL rules and OWL ontologies with protégé OWL plugin, Jess, and Racer. In *7th International Protégé Conference, Maryland, USA*, 2004.

[72] G. Governatori and Z. Milosevic. A formal analysis of a business contract language. *International Journal of Cooperative Information Systems*, 15(04):659–685, 2006.

[73] S. Graham, P. Niblett, D. Chappell, A. Lewis, N. Nagaratnam, J. Parikh, S. Patil, S. Samdarshi, I. Sedukhin, D. Snelling, et al. Publish-subscribe notification for web services. 2004.

[74] B. N. Grosof and T. C. Poon. Sweetdeal: representing agent contracts with exceptions using XML rules, ontologies, and process descriptions. In *Proceedings of the Twelfth International World Wide Web Conference, WWW 2003, Budapest, Hungary, May 20-24, 2003*, pages 340–349, 2003.

[75] Q. Gu and P. Lago. A stakeholder-driven service life cycle model for SOA. In *2nd international workshop on Service oriented software engineering: in conjunction with the 6th ESEC/FSE joint meeting*, pages 1–7. ACM, 2007.

[76] J. D. Hamilton. *Time series analysis*, volume 2. Princeton University Press Princeton, 1994.

[77] C. Hanson, L. Kagal, T. Berners-Lee, G. J. Sussman, and D. Weitzner. Data-purpose algebra: Modeling data usage policies. In *Policies for Distributed Sys-*

tems and Networks, 2007. POLICY'07. Eighth IEEE International Workshop on, pages 173–177. IEEE, 2007.

[78] D. Harel. Dynamic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume 165 of *Synthese Library*, pages 497–604. Springer Netherlands, 1984.

[79] N. B. Harrison, P. Avgeriou, and U. Zdun. Using patterns to capture architectural decisions. *IEEE Software*, 24(4):38–45, 2007.

[80] T. Heinis and G. Alonso. Efficient lineage tracking for scientific workflows. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1007–1018. ACM, 2008.

[81] K. C. Herbst. Accountability as a deterrent to self-enhancement. *Journal of Personality and Social Psychology*, 83:592, 2002.

[82] H. Herrestad and C. Krogh. Obligations directed from bearers to counterparts. In *Proceedings of the 5th international conference on Artificial intelligence and law*, pages 210–218. ACM, 1995.

[83] C. Hood. *The art of the state: Culture, rhetoric, and public management*. Oxford University Press, 2000.

[84] A. Horn. On sentences which are true of direct unions of algebras. *The Journal of Symbolic Logic*, 16(01):14–21, 1951.

[85] I. Horrocks and P. F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In *The Semantic Web - ISWC 2003, Second International Semantic Web Conference, Sanibel Island, FL, USA, October 20-23, 2003, Proceedings*, pages 17–29. 2003.

[86] I. Horrocks and P. F. Patel-Schneider. A proposal for an OWL rules language. In *Proceedings of the 13th international conference on World Wide Web*, pages

723–731, 2004.

[87] I. Horrocks, P. F. Patel-Schneider, and F. Van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Web semantics: science, services and agents on the World Wide Web*, 1(1):7–26, 2003.

[88] D. W. Hosmer Jr and S. Lemeshow. *Applied logistic regression*. John Wiley & Sons, 2004.

[89] J. Huang, T. Chen, A. Doan, and J. F. Naughton. On the provenance of non-answers to queries over extracted data. *PVLDB*, 1(1):736–747, 2008.

[90] F. K. Hussain and E. Chang. An overview of the interpretations of trust and reputation. In *Third Advanced International Conference on Telecommunications (AICT 2007), May 13-19, 2007, Mauritius*, page 30, 2007.

[91] IC3. Internet crime complaint centre. In *http://www.ic3.gov/media/annualreport/2009_IC3Report.pdf*, 2009.

[92] R. Ikeda and J. Widom. Data lineage: A survey. *Stanford InfoLab*, 2009.

[93] ITGI. *Board briefing on IT governance (2nd ed.)*. The IT Governance Institute, 2003.

[94] M. G. Jaatun, S. Pearson, F. Gittler, and R. Leenes. Towards strong accountability for cloud service providers. In *CloudCom*, pages 1001–1006, 2014.

[95] R. Jagadeesan, A. Jeffrey, C. Pitcher, and J. Riely. Towards a theory of accountability and audit. In *Computer Security–ESORICS 2009*, pages 152–167. Springer, 2009.

[96] K. Jensen. An introduction to the theoretical aspects of coloured Petri nets. In *A Decade of Concurrency, Reflections and Perspectives, REX School/Symposium*, pages 230–272, 1993.

[97] D. G. Johnson and J. M. Mulvey. Accountability and computer decision systems. *Commun. ACM*, 38(12):58–64, 1995.

[98] J. Johnson. Chaos summary. *The Standish Group Report*, 2009.

[99] R. Kailar. Reasoning about accountability in protocols for electronic commerce. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy, Oakland, California, USA, May 8-10, 1995*, pages 236–250, 1995.

[100] R. Kailar. Accountability in electronic commerce protocols. *IEEE Transactions on Software Engineering*, 22(5):313–328, May 1996.

[101] H. Kaufman. *The forest ranger: A study in administrative behavior*. Resources for the Future, 1967.

[102] H. Kaufman and M. Couzens. Administrative feedback: Monitoring subordinates' behavior. 1973.

[103] A. M. Khademian. *Checking on banks: Autonomy and accountability in three federal agencies*. Jessica Kingsley Publishers, 1996.

[104] R. Khare and R. N. Taylor. Extending the representational state transfer (rest) architectural style for decentralized systems. In *Proceedings of the 26th International Conference on Software Engineering*, pages 428–437. IEEE Computer Society, 2004.

[105] R. K. Ko, B. S. Lee, and S. Pearson. Towards achieving accountability, auditability and trust in cloud computing. In *Advances in Computing and Communications*, pages 432–444. Springer, 2011.

[106] D. Kondor, M. Posfai, I. Csabai, and G. Vattay. Do the rich get richer? an empirical analysis of the Bitcoin transaction network. *PLOS ONE*, 9(2), 2013.

[107] J. G. Koppell. Pathologies of accountability: ICANN and the challenge of “multiple accountabilities disorder”. *Public Administration Review*, 65(1):94–108, 2005.

[108] S. Kremer, O. Markowitch, and J. Zhou. An intensive survey of fair non-repudiation protocols. *Computer Communications*, 25(17):1606–1621, 2002.

[109] S. A. Kripke. Semantical analysis of intuitionistic logic I. In *Formal Systems and Recuisive Functions*, pages 92–130. North Holland, Amsterdam, 1965.

[110] G. A. Kumta and M. D. Shah. Capability maturity model. *A Human Perspective, Delhi Business Review*, 3(1), 2002.

[111] L. Lamport, R. E. Shostak, and M. C. Pease. The Byzantine Generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[112] W. Lee, A. C. Squicciarini, and E. Bertino. The design and evaluation of accountable Grid computing system. In *29th IEEE International Conference on Distributed Computing Systems (ICDCS 2009), 22-26 June 2009, Montreal, Québec, Canada*, pages 145–154, 2009.

[113] A. Y. Levy and M.-C. Rousset. Combining horn rules and description logics in carin. *Artificial Intelligence*, 104(1):165–209, 1998.

[114] K. Liew, H. Shen, S. See, W. Cai, P. Fan, and S. Horiguchi. *Parallel and Distributed Computing: Applications and Technologies: 5th International Conference, PDCAT 2004, Singapore, December 8-10, 2004, Proceedings*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004.

[115] K. J. Lin, M. Panahi, and Y. Zhang. The design of an intelligent accountability architecture. In *IEEE International Conference on E-Business Engineering*, pages 157–164, 2007.

[116] K.-J. Lin, J. Zou, and Y. Wang. Accountability computing for e-society. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 34–41. IEEE, 2010.

[117] K. Linderman, R. G. Schroeder, S. Zaheer, and A. S. Choo. Six Sigma: a goal-theoretic perspective. *Journal of Operations Management*, 21(2):193–203, 2003.

[118] H. Liu, Q. Li, N. Gu, and A. Liu. Modeling and reasoning about semantic web services contract using description logic. In *The Ninth International Conference on Web-Age Information Management, WAIM 2008, July 20-22, 2008, Zhangjiajie, China*, pages 179–186, 2008.

[119] W. S. Livingston. Britain and america: The institutionalization of accountability. *Journal of Politics*, 38(4):878–894, 1976.

[120] J. S. Long and J. Freese. *Regression models for categorical dependent variables using Stata*. StataCorp LP, 2006.

[121] H. Ludwig, A. Keller, A. Dan, R. P. King, and R. Franck. Web service level agreement (WSLA) language specification. *IBM Corporation*, pages 815–824, 2003.

[122] J. Luo, B. E. Montrose, A. Kim, A. Khashnobish, and M. H. Kang. Adding OWL-S support to the existing UDDI infrastructure. In *2006 IEEE International Conference on Web Services (ICWS 2006), 18-22 September 2006, Chicago, Illinois, USA*, pages 153–162, 2006.

[123] R. Lynn, K. Bishop, et al. *Getting Started With IBM Websphere SMash*. Pearson Education India, 2010.

[124] K. J. Ma. Web services: what's real and what's not? *IT professional*, 7(2):14–21, 2005.

[125] O. Marjanovic and Z. Milosevic. Towards formal modeling of e-contracts. In *5th International Enterprise Distributed Object Computing Conference (EDOC 2001), 4-7 September 2001, Seattle, WA, USA, Proceedings*, pages 59–68, 2001.

[126] O. Markowitch and Y. Roggeman. Probabilistic non-repudiation without trusted third party. In *2nd Conference on Security in Communication Network*, 2001.

[127] E. A. Marks and M. Bell. *Service-Oriented Architecture (SOA): A Planning and Implementation Guide for Business and Technology*. John Wiley & Sons, Inc., 2006.

[128] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, et al. OWL-S: Semantic markup for web services. *W3C member submission*, 22:2007–04, 2004.

[129] D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, et al. Bringing semantics to web services: The OWL-S approach. In *Semantic Web Services and Web Process Composition*, pages 26–42. Springer, 2004.

[130] J. Mei and E. P. Bontas. Reasoning paradigms for SWRL-enabled ontologies. *Protege with Rules Workshop*, 2005.

[131] J. Mei, E. P. Bontas, and Z. Lin. OWL2Jess: A transformational implementation of the OWL semantics. In *Parallel and Distributed Processing and Applications-ISPA 2005 Workshops*, pages 599–608. Springer, 2005.

[132] P. Mell and T. Grance. The NIST definition of cloud computing. *Computer Security Division, Information Technology Laboratory*, 2011.

[133] R. C. Merkle. Protocols for public key cryptosystems. In *Symposium on Security and Privacy*, pages 122–133, 1980.

[134] B. Meyer. Applying “design by contract”. *IEEE Computer*, 25(10):40–51, 1992.

[135] J. J. C. Meyer, F. P. M. Dignum, and R. J. Wieringa. The paradoxes of deontic logic revisited: a computer science perspective. *University of Utrecht Department of Computer Science*, 1994.

[136] J.-J. C. Meyer et al. A different approach to deontic logic: Deontic logic viewed as a variant of dynamic logic. *Notre dame journal of formal logic*, 29(1):109–136, 1988.

[137] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-peer computing. *Technical Report HPL-2002-57, HP Labs*, 2002.

[138] B. Motik, U. Sattler, and R. Studer. Query answering for OWL-DL with rules. In *The Semantic Web–ISWC 2004*, pages 549–563. Springer, 2004.

[139] W. Mougayar. 9 myths surrounding blockchain smart contracts. In *http://www.coindesk.com/smart-contract-myths-blockchain/*. Coindesk, March 2016.

[140] L. Mui, M. Mohtashemi, and A. Halberstadt. A computational model of trust and reputation for e-businesses. In *35th Hawaii International Conference on System Sciences (HICSS-35 2002), CD-ROM / Abstracts Proceedings, 7-10 January 2002, Big Island, HI, USA*, page 188, 2002.

[141] D. Mulligan. Know your customer regulations and the international banking system: Towards a general self-regulatory regime. *Fordham International Law Journal*, 22(5):2324, 1998.

[142] S. Narayanan and S. A. McIlraith. Simulation, verification and automated composition of web services. In *Proceedings of the Eleventh International World*

Wide Web Conference, WWW 2002, May 7-11, 2002, Honolulu, Hawaii, pages 77–88, 2002.

[143] H. Nissenbaum. Computing and accountability. *Commun. ACM*, 37(1):72–80, 1994.

[144] D. Nuñez, M. C. F. Gago, S. Pearson, and M. Felici. A metamodel for measuring accountability attributes in the cloud. In *IEEE 5th International Conference on Cloud Computing Technology and Science, CloudCom 2013, Bristol, United Kingdom, December 2-5, 2013, Volume 1*, pages 355–362, 2013.

[145] L. S. Oakes and J. J. Young. Accountability re-examined: evidence from hull house. *Accounting, Auditing & Accountability Journal*, 21(6):765–790, 2008.

[146] OMG. Notation (bpmn) version 2.0 (2011). In *http://www.omg.org/spec/BPMN/2.0*. OMG,Business Process Model, 2011.

[147] S. J. Ong and S. Vadhan. An equivalence between zero knowledge and commitments. In *Theory of Cryptography*, pages 482–500. Springer, 2008.

[148] T. O'reilly. What is web 2.0: Design patterns and business models for the next generation of software. *Communications & strategies*, (1):17, 2007.

[149] M. P. Papazoglou. Service-oriented computing: Concepts, characteristics and directions. In *4th International Conference on Web Information Systems Engineering, WISE 2003, Rome, Italy, December 10-12, 2003*, pages 3–12, 2003.

[150] M. C. Paulk, B. Curtis, M. B. Chrissis, and C. V. Weber. *Capability Maturity Model for Software, Version 1.1 (CMU/SEI-93-TR-24, ADA 263 403). Pittsburgh, Pa.: Software Engineering Institute*. Carnegie Mellon University, 1993.

[151] S. Pearson, V. Tountopoulos, D. Catteddu, M. Südholt, R. Molva, C. Reich, S. Fischer-Hübner, C. Millard, V. Lotz, M. G. Jaatun, R. Leenes, C. Rong, and J. Lopez. Accountability for cloud and other future internet services. In

4th IEEE International Conference on Cloud Computing Technology and Science Proceedings, CloudCom 2012, Taipei, Taiwan, December 3-6, 2012, pages 629–632, 2012.

[152] F. Peter. *Drucker. The Practice of Management*. New York Harper Brothers, 1954.

[153] T. Phan, J. Han, J. Schneider, T. Ebringer, and T. Rogers. A survey of policy-based management approaches for service oriented systems. In *19th Australian Software Engineering Conference (ASWEC 2008), March 25-28, 2008, Perth, Australia*, pages 392–401, 2008.

[154] Protege. Swrltemporalbuiltins. In *http://protege.cim3.net/cgi-bin/wiki.pl?SWRLTemporalBuiltIns*, 2011.

[155] C. R. Rao. Large sample tests of statistical hypotheses concerning several parameters with applications to problems of estimation. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 44, pages 50–57. Cambridge Univ Press, 1948.

[156] I. Ray, I. Ray, and N. Narasimhamurthi. A fair-exchange e-commerce protocol with automated dispute resolution. In *Data and Application Security*, pages 27–38. Springer, 2002.

[157] C. Ré and D. Suciu. Approximate lineage for probabilistic databases. *Proceedings of the VLDB Endowment*, 1(1):797–808, 2008.

[158] W. Reilly. Public service accountability: A comparative perspective. by Joseph G. Jabbra and O. P. Dwivedi (eds.). *Journal of International Development*, 2(2):278–279, 1990.

[159] R. Reiter. *Knowledge in action: logical foundations for specifying and implementing dynamical systems*. MIT press, 2001.

[160] A. Rensink and H. Wehrheim. Process algebra with action dependencies. *Acta Inf.*, 38(3):155–234, 2001.

[161] P. Robinson, N. Cook, and S. Shrivastava. Implementing fair non-repudiable interactions with web services. In *IEEE International Edoc Enterprise Computing Conference*, pages 195–206, 2005.

[162] B. S. Romzek and M. J. Dubnick. Accountability in the public sector: Lessons from the challenger tragedy. *Public Administration Review*, pages 227–238, 1987.

[163] W. Royce. CMM vs. CMMI: From conventional to modern software management. *The Rational Edge*, pages 2–9, 2002.

[164] N. Sadashiv and S. D. Kumar. Cluster, grid and cloud computing: A detailed comparison. In *International Conference on Computer Science & Education*, pages 477–482. IEEE, 2011.

[165] N. Satoshi. Bitcoin: A peer-to-peer electronic cash system. In *http://bitcoin.org/bitcoin.pdf*, 2012.

[166] A. Schedler. *The selfrestraining state: Power and accountability in new democracies*. Lynne Rienner Publishers, 1999.

[167] B. Schneier. Secrets & lies: Digital security in a networked world. *International Hydrographic Review*, 2(1):103–104, 2001.

[168] J. Seijts and P. Bigus. Case study Toyota: Accelerator pedal recall. *Harvard Business Review*, 2011.

[169] Sitepoint. Preparing for semantic web services. In *https://www.sitepoint.com/semantic-web-services/*, 2016.

[170] J. Skene, F. Raimondi, and W. Emmerich. Service-level agreements for electronic services. *IEEE Trans. Software Eng.*, 36(2):288–304, 2010.

[171] M. Srivatsa, L. Xiong, and L. Liu. Exchangeguard: A distributed protocol for electronic fair-exchange. In *19th International Parallel and Distributed Processing Symposium (IPDPS 2005), CD-ROM / Abstracts Proceedings, 4-8 April 2005, Denver, CO, USA*, 2005.

[172] Z. Stojanović and A. Dahanayake. *Service-oriented software system engineering: challenges and practices*. IGI Global, 2005.

[173] R. Sturm, W. Morris, and M. Jander. Foundations of service level management. *Sams United States of America*, pages 25–47, 2000.

[174] S. Sundareswaran, A. C. Squicciarini, and D. Lin. Ensuring distributed accountability for data sharing in the cloud. *IEEE Trans. Dependable Sec. Comput.*, 9(4):556–568, 2012.

[175] N. Szabo. Formalizing and securing relationships on public networks. *First Monday*, 2(9), 1997.

[176] B. G. Tabachnick, L. S. Fidell, and S. J. Osterlind. Using multivariate statistics. 2001.

[177] Y.-H. Tan and W. Thoen. A logical model of directed obligations and permissions to support electronic contracting. *International Journal of Electronic Commerce*, 3(2):87–104, 1998.

[178] C. P. Team. Capability maturity model® integration (CMMI SM), version 1.1. *CMMI for Systems Engineering, Software Engineering, Integrated Product and Process Development, and Supplier Sourcing (CMMI-SE/SW/IPPD/SS, V1. 1)*, 2002.

[179] C. Techapanupreeda, R. Chokngamwong, C. Thammarat, and S. Kungpisdan. Accountability in internet transactions revisited. In *14th International Symposium on Communications and Information Technologies, ISCIT 2014, Incheon, South Korea, September 24-26, 2014*, pages 378–382, 2014.

[180] C. Techapanupreeda, R. Chokngamwong, C. Thammarat, and S. Kungpisdan. An accountability model for internet transactions. In *2015 International Conference on Information Networking, ICOIN 2015, Siem Reap, Cambodia, January 12-14, 2015*, pages 127–132, 2015.

[181] P. E. Tetlock. Accountability and complexity of thought. *Journal of personality and social psychology*, 45(1):74, 1983.

[182] M. M. Tseng, C. J. Su, and Q. Ma. Accountability centered approach to business process reengineering. In *Hawaii International Conference on System Sciences*, pages 345–354, 1998.

[183] R. J. van Glabbeek. *The Meaning of Negative Premises in Transition System Specifications II*. 1996.

[184] W. Van Grembergen and S. Dehaes. *Implementing Information Technology Governance: Models, Practices and Cases*. IGI Global, 2007.

[185] W3C. OWL web ontology language overview. In *http://www.w3.org/TR/owl-features/*, 2004.

[186] W3C. OWL 2 web ontology language new features and rationale. In *http://www.w3.org/TR/2009/WD-owl2-new-features-20090611/*, 2009.

[187] C. Wang, S. Chen, and J. Zic. A contract-based accountability service model. In *IEEE International Conference on Web Services, ICWS 2009, Los Angeles, CA, USA, 6-10 July 2009*, pages 639–646, 2009.

[188] C.-H. Wang, C.-H. Yin, and C.-H. Juan. How to protect exchanged secrets in the fair exchange protocol with off-line TTP. *Computers & Electrical Engineering*, 32(5):364–375, 2006.

[189] P. Weill. Don't just lead, govern: How top-performing firms govern IT. *MIS Quarterly Executive*, 3(1), 2004.

[190] D. J. Weitzner, H. Abelson, T. Berners-Lee, J. Feigenbaum, J. Hendler, and G. J. Sussman. Information accountability. *Communications of the ACM*, 51(6):82–87, 2008.

[191] J. Widom. Trio: A system for data, uncertainty, and lineage. *Managing and Mining Uncertain Data*, 35, 2008.

[192] R. J. Wieringa and J.-J. C. Meyer. Actors, actions, and initiative in normative system specification. *Annals of mathematics and artificial intelligence*, 7(1-4):289–346, 1993.

[193] G. V. Wright. Deontic logic, mind new series. *Mind New Series*, 60(237):1–15, 1951.

[194] L. Xu. Monitoring multi-party contracts for e-business. *Open Access Publications from Tilburg University*, (127), 2004.

[195] E. Yakel. The social construction of accountability: radiologists and their record-keeping practices. *The Information Society*, 17(4):233–245, 2001.

[196] J. Yao, S. Chen, C. Wang, D. Levy, and J. Zic. Accountability as a service for the cloud. In *Services Computing (SCC), 2010 IEEE International Conference on*, pages 81–88. IEEE, 2010.

[197] J. Yao, S. Chen, C. Wang, D. Levy, and J. Zic. Modelling collaborative services for business and qos compliance. In *Web Services (ICWS), 2011 IEEE International Conference on*, pages 299–306. IEEE, 2011.

[198] A. R. Yumerefendi and J. S. Chase. Trust but verify: Accountability for network services. In *ACM Sigops European Workshop, Leuven, Belgium, September*, pages 50–51, 2004.

[199] S. Zadek. Wikileaks enter an era of disruptive accountability. In *http://www.zadek.net/wikileaks-enter-an-era-of-disruptive-accountability/*,

2010.

[200] Y. Zhang, K. J. Lin, and T. Yu. Accountability in service-oriented architecture: Computing with reasoning and reputation. In *IEEE International Conference on E-Business Engineering*, pages 123–131, 2006.

[201] J. Zhou and D. Gollmann. A fair non-repudiation protocol. In *1996 IEEE Symposium on Security and Privacy, May 6-8, 1996, Oakland, CA, USA*, pages 55–61, 1996.

[202] J. Zou and C. J. Pavlovski. Towards accountable enterprise mashup services. In *Proceedings of ICEBE 2007, IEEE International Conference on e-Business Engineering and the Workshops SOAIC 2007, SOSE 2007, SOKM 2007, 24-26 October, 2007, Hong Kong, China*, pages 205–212, 2007.

[203] J. Zou, Y. Wang, and M. Orgun. Modeling accountable cloud services based on dynamic logic for accountability. *International Journal of Web Services Research (IJWSR)*, 12(3):48–77, 2015.