EINDHOVEN
UNIVERSITY OF
TECHNOLOGY

A TxQoS-aware business transaction framework

Citation for published version (APA):
Wang, T. (2011). *A TxQoS-aware business transaction framework*. [Phd Thesis 1 (Research TU/e / Graduation TU/e), Industrial Engineering and Innovation Sciences]. Technische Universiteit Eindhoven. https://doi.org/10.6100/IR717753

DOI:
10.6100/IR717753

Document status and date:
Published: 01/01/2011

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 04. Oct. 2023

A TxQoS-aware Business Transaction Framework

Copyright © 2011 by Ting Wang. All rights reserved.

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN
Wang, Ting
A TxQoS-aware Business Transaction Framework / by Ting Wang. - Eindhoven : Technische Universiteit Eindhoven, 2011. - Proefschrift -
ISBN 978-90-386-2819-6
NUR 953
Keywords: Transaction Framework / Service-Oriented Architecture / Transactional Quality of Service / Business Process / Service Level Agreement

The work in this thesis has been carried out under the auspices of Beta Research School for Operations Management and Logistics.
Beta Dissertation Series D142
Printed by University Press Facilities, Eindhoven
Cover design: Paul Verspaget

A TxQoS-aware Business Transaction Framework

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische
Universiteit Eindhoven, op gezag van de rector magnificus,
prof.dr.ir. C.J. van Duijn, voor een commissie aangewezen
door het College voor Promoties in het openbaar te verdedigen
op maandag 14 november 2011 om 14.00 uur

door

Ting Wang

geboren te Jiangxi, China

Dit proefschrift is goedgekeurd door de promotor:

prof.dr.ir. P.W.P.J. Grefen

Co-promotor:
dr. S. Angelov

To my parents.

谁言寸草心，
报得三春晖。

Contents

Acknowledgements

It has been a long journey to reach to this point. I cannot wait to express my humble gratitude to those who contributed in all kinds of forms to the completion of my PhD thesis.

First I would like to thank prof.dr.ir. Paul Grefen, my promoter and supervisor, for the opportunity of bringing me on board at the XTC project. He patiently guided me through the PhD process , and shaped my work with his strict quality standard. Beyond his guidance and challenges during my research, Paul has impressed me in many ways – fancy restaurant treatments, his innovation on visual poems, and the funny teasers – all nice things for which he deserves the title of a 'super-wiser '. I learned a lot of things from him, among which the most important may be 'the best chocolate in the world', or 'everyone has a child inside'.

My deep gratitude goes to prof.dr.ir. Mike Papazoglou , the co-founder of the XTC project, for his helpful insights, which benefited the quality of the thesis. I am indebted to prof.dr.ir. Uzay Kaymak and prof.dr. Babara Pernici, the other core committee members, for their guidance and valuable comments. Prof.dr. YaoHua Tan and prof.dr.ir. Jos Van Hillegersberg are thanked for accepting the invitation as the committee members.

Special thanks go to dr. Samuil Angelov, my co-promoter, who, over time, shifted between many roles -my office mate, another 'poor PhD student' who is teased by Paul, an intellectually funny friend, and the supervisor of this thesis . Then I have to mention 'yet another poor PhD student', Sven, who together with Samuil, helped me on lots of things from day one, such as speaking English and moving furniture from Tilburg to Eindhoven. Their witty sparks brought me lots of hilarious moments both at work and during our spare time - we made a beautiful kite and often sneaked out of the office to fly it; we played board games regularly during weekends; we shared with each other the knowledge, experience, and of course, the food from our very different cultural background; we would engage on all conversations with persistent scientific spirit on all odd topics like, the origin of sour cabbage dish.

I would extend my sincere gratitude to the colleagues in the IS group, Ana Karla, Irene, and Mariska, who often organized girl's activities , and with which I had tours to some of the best spots of this country, including the park, the museum, and the beach. Needless to say that we had fun in the city when, for instance, going for dinners, movies, and out dancing. Sometimes I even forgot that we were pursuing a serious career named 'women in science'. Off course we also had nice guys in the group. Jochem saved me from time to time in the battles against software or hardware. Furthermore, he shared with me his insights on my research during the stimulative XTC discussions. Boudewijn and Remco are thanked for their tour guide during the conferences abroad. There was a very positive and friendly atmosphere in the group, where all colleagues were there ready for inspiring discussions and warm help. During the years working there, I was able to explore and grow through all kinds of 'first time in my life' experiences besides the research activities, and I have truly enjoyed this process.

The thesis could have not been written without the support from the following organizations. The XTC project was funded by NWO, the Netherlands Organization for Scientific Research. The research school of Beta provides an excellent academia environment and administration support. The partner from BearingPoint, Mary Ann, immediately agreed to my proposal when I suggested using the company's largest project as a case study. My manager Martin allowed me to work from home at times, so that I was able to meet with my supervisors on campus regularly. Towards the end of the thesis, André Vorage from Edit-Your-Work company proof-read the whole thesis and I would like to thank his professional service as well.

One of the best things I did in my university years here, is to make a lot of Chinese friends, most of which are from the student and scholar community, who gave color to my life in the Netherlands. Gu Bing, Li Ping, Zhang Shaoxian, Guo Wei, Liu Yan, Yang Jia, Liu Danqing, Zhang Yi, Sun Chunxia, Li Jing, Tsoi Shufen, Tang Dongling, Cheng Jieyin, and Zhang Ying have always been around me offering all what I could wish for from friends. Also I would like to express my deepest gratitude to the friends who nowadays have settled down in other cities or countries for their warm friendship ever-lasting: Li Zhonggui, Guo Lan, Xiao Yan, Jia Yuping, Zhao Jing, Wong Yuki, Wang Yue, Qiao Liang, Wang Ping, Jiang Zhouting, Han Wei, Yuan Ming, Li Zhili, Huang Rubin, and Huang Tiezhu. I will never forget the wonderful vacations and trips, the joyful games, the exhausted search for good restaurants, the inspiring conversations, and many more nice times we had together.

I would like to thank Jiang Ying and Xu Yu in China, who have been my best friends for over 23 years. It was them who ignited my interest in computer science and forwarded all the English test materials for studying abroad. It was them who sent a big FedEx box and soon thereafter a big EMS box, full of presents to help relieve me of homesickness in my first days after arriving in the Netherlands. It is them who have been closely watching my every move remotely and supporting me through all ups and downs.

At the end, I would like to thank my parents with great respect and appreciation. With their constant love and support in all possible forms, I was able to chase my dreams with lots of encouragement and freedom.

最后，谨以此书向亦师亦友的婷爸婷妈致以万分的敬意和感激。他们的爱护和支持始终伴我前行，给予我追逐梦想的勇气和自由。亲恩如海，未有一日敢忘。谢谢！

Chapter 1

Introduction

This chapter provides an overview of the research leading up to this thesis. Section 1.1 presents the research context of the XTC (eXecution of Transactional Contracted electronic services) project as background knowledge. Section 1.2 introduces the motivation to carry out the research in the given context. Section 1.3 presents the research goal and decomposes it into a number of objectives. Section 1.4 summarizes the intended contribution of our work. In order to generate a methodology for solving the problem set, Section 1.5 explains the research method and illustrates the research process. Finally, Section 1.6 outlines the structure of the thesis, corresponding to the process.

1.1 Research Context

E-business, a concept first promoted by IBM at a large marketing campaign in 1997 to raise awareness and knowledge of the new business paradigm [36], has revolutionized the business world. According to [47], e-business is 'the conduct of automated business transaction, by means of electronic communications networks (e.g. via the Internet and/or possibly private networks) end-to-end'. Backed by information technologies, such as e-mail, World Wide Web, workflow management systems, Web services, and many more, e-business has been widely accepted as an effective way to conduct business. Many commercial applications, such as ERP (Enterprise Resource Planning) and CRM (Customer Relationship Management) systems, have been developed to offer businesses new opportunities and empower business processes with the advantage of computerized information systems.

The latest trend in information systems shows the current shift from carefully planned designs that build information systems from scratch, to

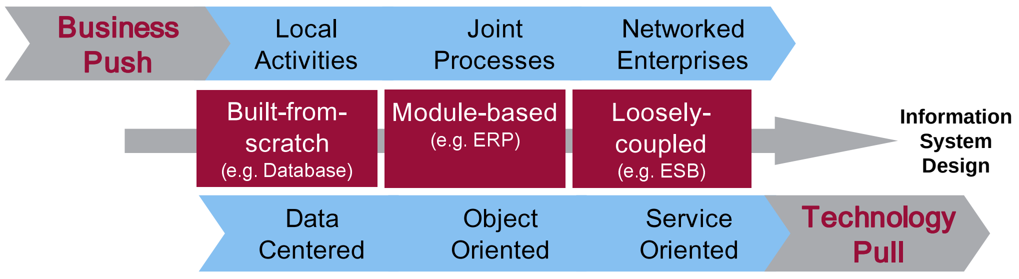


Figure 1.1: Trend of Information System Design

redesigns that build information systems based on existing application and components [58]. The legacy information systems centered around database applications are still in use. However, in many cases they are wrapped up as modules or services to integrate with other systems.

The emerging research efforts from both academia and industry on service composition using Service-Oriented Architecture (SOA), demonstrate the trend depicted in Figure 1.1. The basic idea of SOA is that service users discover needed services advertised by service providers through service intermediaries. This requires standardized approaches for service invocation and delivery. As the major implementing technology of SOA, Web services have evolved from standards and protocols, such as messaging (i.e. SOAP), description (i.e. WSDL), and discovery (i.e. UDDI), to many proposals addressing advanced issues, such as composition, QoS management, security, and transaction management. SOA, together with the business process technologies of information systems, have become an intriguing topic to address and have resulted in many research papers and commercial applications [17]. For example, among these technologies, the approved OASIS standard WS-BPEL(Web Service-Business Process Execution Language) is a language for composing distributed Web services into processes.

E-business has moved from the simple automation of business applications, to the complex integration and coordination of business processes [24]. Business processes have grown to be very complex, involving miscellaneous activities and resources. At the process level, the distributed applications are often treated as 'services', so that they can invoke, as well as be invoked, by each other, without considering the technical heterogeneity. In this dissertation, we call the processes executed in SOA paradigm *service-oriented business processes*. These service-oriented processes allow e-business to cross organizational boundaries, unify applications over proprietary networks, and offer customers comprehensive, yet easy-to-use services. In such an environment, dynamically composed activities usually have complex inter-related dependencies which make exceptions and errors prone to occur during process execution.

Transaction management, which has been widely used in information systems for exception handling and fault tolerance, guarantees a reliable and robust execution. However, the traditional approach of transaction management, by locking and afterwards releasing the shared resources per access, is not applicable in loosely-coupled, long-lasting business processes spanning organizational boundaries. In addition, each component activity, as well as the whole process, may differ in the need of transactional support. All these add to the complexity of today's transaction management and the demand for flexibility on top of robustness. For example, a booking process in a travel agency invokes three parallel Web services: hotel booking, car rental, and flight booking, where each Web service demands 'atomic' Web service transaction support. Furthermore, there are sequential activities like billing, payment check, etc., which are executed one after another, and as a result require a 'chained' transaction support. In addition, the whole process might need a 'rollback' mechanism, in case a customer cancels the booking before payment. In case of a cancelation after payment, the process cannot be returned to the exact same state prior to payment (e.g. the customer is charged with a fine due to late-cancel), thus the transaction mechanism of 'forward recovery' is needed.

According to Merriam-Webster, an English dictionary, the word *reliability* means 'the quality or state of being reliable' or 'the extent to which an experiment, test, or measuring procedure yields the same results on repeated trials' [33]. Relating it to transaction management, it means the extent to which a (transaction management) system handles exceptions and errors so that the execution produces expected results of consistent quality. We name this kind of reliability as 'technical reliability'. We noticed that, while transaction management provides technical reliability in a distributed and heterogeneous environment, contracts provides business trustworthiness between parties in complex processes. This kind of trustworthiness is seen by us as the business form of reliability, where one party expects consistent, measurable results delivered by another party. Technical reliability focuses on the execution and the mechanisms to guarantee the execution. Business reliability focuses on the results and the unambiguous interpretation of the expected results.

As a legally binding agreement between parties, contracts play an important role in regulating business relationships, especially in mission-critical business processes. If a process execution is triggered by some form of contract, such as an agreement between the parties or an form submitted by a client, then we name this process as a 'contract-driven' process. The traditional way of establishing and managing contracts manually is usually resource-consuming (in terms of time, cost, human effort, etc.). To enhance efficiency and lower costs, e-contracting has been introduced to the business world thanks to the advancements of relevant technologies. If a large amount of dynamic business relationships are involved, e-contracting can be a powerful means to guarantee business trustworthiness while meeting speed and flexibility requirements. Working together, transaction management and e-contracting ensure a reliable execution of complex and long-lasting collaborative processes from both technical and business angles. Here and throughout the thesis, we use the term 'reliability' in the context of complex business processes indicating a smooth execution with the results unambiguously interpreted by the involved parties.

This dissertation describes the research result of the XTC project funded by NWO (Dutch Scientific Organization), and carried out in the above research background. [1] More specifically, the aim of the XTC project is to address the process reliability issues within the crossing fields of SOA, business processes, transaction management, and e-contracting.

1.2 Research Motivation

Within the context of service-oriented business processes, there have been many research efforts in service composition (e.g. BPEL) and Business Process Management (BPM). However, we notice that some necessary properties such as reliability, security, and robustness of such processes are less well addressed, with few consensuses reached, and standards met. Therefore, we are motivated to conduct research, addressing the reliability issue within the whole promising picture of contract-driven service-oriented business processes.

After an extensive literature study on the effectiveness of the IT systems to address the reliability of process execution, we discover that current transaction management solutions are insufficient to address the reliability problem of complex processes. Taking the travel booking example, current technologies can already solve the reliability problem of a process purely composed of Web services in an accepted manner. However, it is a simplified illustration of complicated service-oriented processes, which may be composed of many more services and ad-hoc activities. Flexibility and comprehensiveness when using transaction mechanisms for such processes are required. The existing transaction models/frameworks are not able to address these requirements, as they are developed basically for one particular

[1]XTC is an abbreviation of XTraConServe, eXecution of Transactional Contracted Electronic Services, NWO Project, No. 612.063.305.

application domain (e.g. ACID transactions for database applications and Web service transactions for WS applications).

We notice that on one hand, research efforts addressing the reliability issue of automated business processes usually assess purely from a technical perspective, which results in advancements in many different disciplines such as transaction mechanisms and software reliability techniques. On the other hand, the business community views reliability as a broader issue than merely an executable process, which can include business, management, and legal perspectives. For example, the concept of transaction is perceived differently. In the IT world, transaction management is originally a database mechanism, while in the business world, a transaction is a trading term for expressing the exchange of values. These different interpretations of the same concept or subject very often result in technology solutions which are inadequate for business requirements. Designed to deal with the reliability issue, existing transaction management mechanisms are widely adopted in computerized information systems to handle exceptions and errors. However they do not guarantee reliable business process execution to be robust, despite of the exceptions and errors. It is especially true if the process is complex, long-lasting and full of ad-hoc activities, and is composed of services not implemented by means of automated computer systems.

Considering the below scenario taking place in a hospital which we discover in an elaborated case study [59]: Patients suffering from heart problems want to be treated and thus register for a particular cardiac surgery. From the patients point of view, the hospital is a provider that offers them a curing service by means of a treatment process lasting days or months. Within the hospital, there are different units, which are actually independent organizations, each specialized to offer some function such as intensive care, surgery, pre-operation screening and administration. We view each unit as a service provider offering services either to the patients directly or to other collaborating units. Obviously, this treating process is a complex and long-lasting service-oriented business process, and is full of exceptions (i.e. emergencies). Therefore, reliability (of both the execution and the result) is of highest concern. However, most of the services are implemented by human beings (e.g. surgeons, nurse practitioners, physicians, anesthetists, and staff) with no standardized serving result. Consequently, many of mature transaction mechanisms suitable for computers(e.g. two-phase commit) providing technical reliability are not applicable in this case.

From what we observed in cases and what we studied in the literature, we found the research on the reliability of complex business processes, that are typically service-oriented and contract-driven, a promising direction to pursue. A business transaction framework to address the problem is therefore our primary interest.

1.3 Research Goal and Objectives

Motivated by the lack of a comprehensive and flexible transaction support for service-oriented processes and the discovery of the gap between the IT and business world, we identify the overall research goal as:

The design and development of a comprehensive yet flexible Business Transaction Framework (BTF), that is not restricted to a specific application domain or application environment, and that supports explicitly specified reliability aspects of complex, contract-driven, and service-oriented business processes.

There are a number of objectives we need to achieve in order to achieve the research goal of the BTF:

- A problem statement and a requirements specification inspired from a case study, which together raise the main questions the BTF design needs to fulfill.
- The business-oriented design of the BTF components that allows for transactional qualities to be specified in the e-contracts.
- The technology-oriented design of the BTF components that allows for flexible composition of transactional constructs.
- The integration design of the BTF that combines the business and technology design results from the above.
- The validation of the usability and effectiveness of the BTF through the examination of a case study .

1.4 Contribution

What we intend to achieve from the XTC project is a ***Business Transaction Framework***, which is rooted in transaction management techniques, yet addresses the concern on process reliability by a business-aware approach. We develop such a BTF at conceptual level, with the concepts, scenarios, phases throughout life cycle, reference architectures, and mechanisms presented in this thesis. The thesis focuses on design; thus the implementation of the BTF is out of its scope.

The BTF provides comprehensive and flexible transaction support for contract-driven service-oriented business processes. Transaction support is indispensable for a reliable process execution and our intended BTF enhances the reliability by means of transaction management mechanism rooting in the IT world as well as a contractual approach reflecting the requirements from the business world.

At the end of the thesis, we will summarize what we have achieved along with the research on the BTF. The outcome from the XTC research, besides the BTF, is expected to include the method and other insights in this area.

1.5 Research Approach

Throughout our research, we applied a design science method that is widely used in research on technology fields. 'Research in IT must address the design tasks faced by practitioners. Real problems must be properly conceptualized and represented, appropriate techniques for their solution must be constructed, and solutions must be implemented and evaluated using appropriate criteria' [38]. In the XTC project, we have carried out a number of real-life case studies in order to identify problems and research solutions that address these problems. Furthermore, we use case studies to validate our research by verifying the utility and effectiveness of our solution. The actual research process in the XTC project is shown in Figure 1.2 below.

We started by planning, which was followed up by literature review (theory) and a case study (practice) to help identify the research problems. Afterwards, we carried out business-oriented research and technical-oriented research in parallel, and integrated them to develop a solution to address the problems. Finally we validated our solution through a large scale real life case study. Each phase of the research resulted in one chapter (including the appendix) of the thesis as shown in the diagram. The following research questions are answered:

1. What are the research context and relevant literature for the BTF design? - Chapter 2
2. What do we observe from a real-life case and what are the problems the BTF needs to address? - Chapter 3
3. What is the approach to address the business understanding of the transaction agreement? - Chapter 4
4. How does the approach (conceptualized in the last chapter) work? - Chapter 5
5. What is the technical approach to address the flexibility and how does it work? - Chapter 6
6. How do the business and technology oriented approaches work together? - Chapter 7

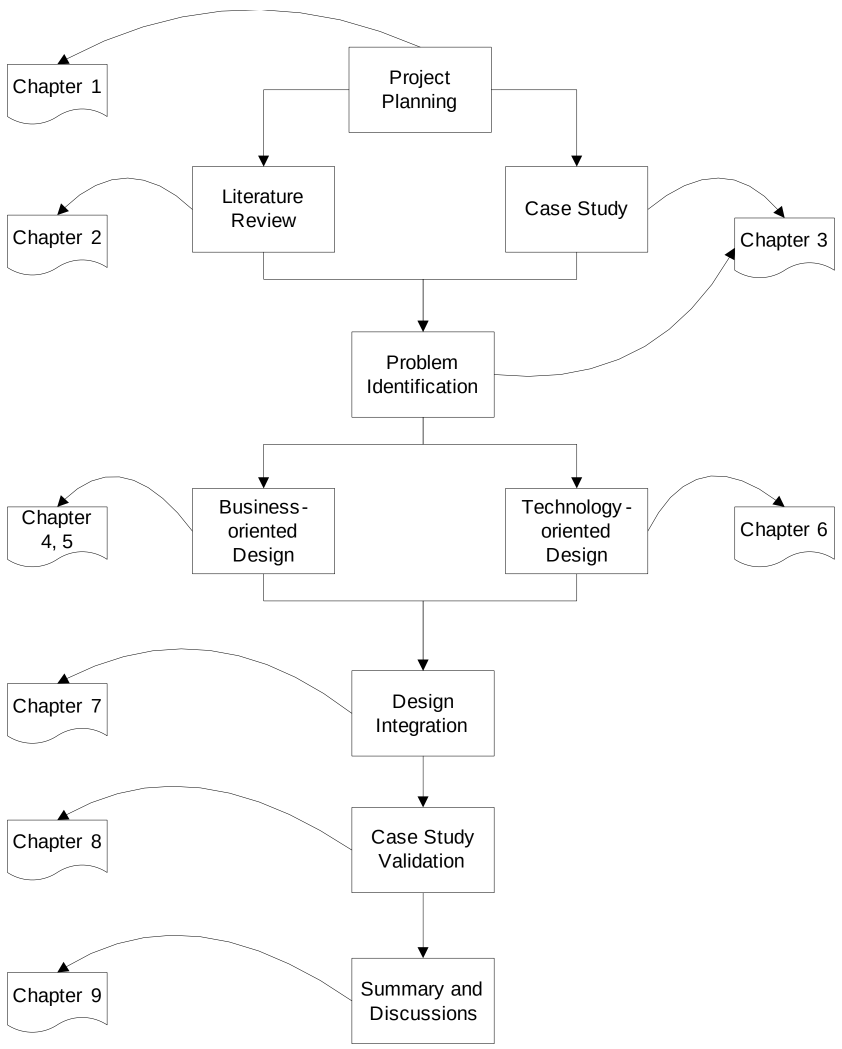


Figure 1.2: Research Process

7. How to verify the utility of our design (i.e. feasibility study)? - Chapter 8
8. What are the conclusions and limitations of our research on BTF? - Chapter 9

1.6 Thesis Outline

As stated in the last section, this thesis addresses the research questions identified above. Each chapter (apart from this chapter) answers one research question. Altogether, this thesis consists of 9 chapters, organized as follows.

Chapter 1, the current chapter, gives an overview of the research carried out in the XTC project with the aim to develop a Business Transaction Framework.

Chapter 2 gives the background information of the thesis and reviews the related research effort up to date.

Chapter 3 describes a case study which we model, observe and analyze for drawing out the statement of the research problems.

Chapter 4 presents a contractual approach to address the first problem of unambiguous reliability agreement raised in the previous chapter.

Chapter 5 continues the design of the solution proposed in the last chapter and presents the detailed specification method to enable the business-driven approach.

Chapter 6 proposes approach to address the second research problem focusing on transactional flexibility following technology-oriented design. Please note that in spite of the organization of the chapters, the research presented in this chapter was actually conducted in parallel with the business-oriented design carried out in the last two chapters.

Chapter 7 presents the intended business transaction framework based on both business and technology oriented design, with an integration of the two solutions proposed in the previous chapters.

Chapter 8 introduces a large scale real-life project, where the main properties fit exactly into our research context and interest: service-oriented, multiple parties, automated workflows, jointly executed by heterogenous applications across boundaries, and the e-contract in the form of web specification of products/services agreed by the users to initiate the business process.

Chapter 9 concludes the XTC project and the thesis. Also we discuss the future work that may further extend the research output.

In addition, there are appendices after the final chapter, containing supporting materials for interested readers. Appendix A presents the statistical model and relevant computation for supporting Chapter 4. Appendix B and C show the full set of diagrams from the case study discussed in Chapter 9.

Chapter 2

Related Work and Research Background

Transactions have been around since the Seventies to provide reliable information processing in automated information systems. Originally developed for simple database operations in centralized systems, they have moved into much more complex application domains, including aspects like distribution, process-orientation, and loose coupling. This chapter presents a historic overview of transaction models organized in several transaction management eras, thereby investigating numerous transaction models ranging from the classical flat transactions, via advanced and workflow transactions, to the Web Services and Grid transaction models. The key concepts and techniques with respect to transaction management are investigated. Placing well-known research efforts in historical perspective reveals specific trends and developments in the area of transaction management. Afterwards, the research background (i.e. contract-driven service-oriented business processes) is reviewed to clarify the basic concepts and outline the context where we carry out the research.

2.1 Introduction

This thesis proposes a TxQoS-aware transaction framework for contract-driven service-oriented business processes, which relates to several areas such as transaction management, SOA, QoS management and e-contracting. In this chapter, we review the research efforts and progress in these areas. First, we present a historic overview of transaction models to provide a comprehensive and structured overview of developments in the area. Next, we clarify the basic concepts and review the evolution of service-oriented business processes to outline the background of our research, which includes service-oriented computing and architecture, service level agreement, and QoS (Quality of Service).

The chapter is organized as follows. Section 2.2 provides a historical overview of transaction management, covering transaction models, frameworks, and standards developed since the 1970's until the present day. As the area of our research, the research and development in transaction management are presented in a temporal perspective, so that we reveal the streams of thought in the past as well as point out the current trend. Section 2.3 reviews the emerging research areas of service-oriented computing (SOC) and service-oriented architecture(SOA), and introduces the research efforts related to e-contracting in the service-oriented context (e.g. SLA and QoS management). Section 2.4 summarizes the content of this chapter.

2.2 Concept and History of Transaction Management

This section presents an extensive survey of the research efforts on transaction management from a historic perspective. The content of this section was published first in the report [67], which later became the main part of the paper [70]. The amount of research efforts in the area of transaction management is huge. Thus, we do not focus on technical details of each transaction model, protocol, or framework. Instead, we aim at bringing an up-to-date picture by reviewing the influential work that formed the basis for later development in this area. Furthermore, we summarize the picture with a clear overview, to reflect on the trends by analyzing and comparing the work in different transaction eras.

We first discuss the transaction concept. What exactly is a transaction? The concept was invented as early as 6000 years ago, when Sumerians noted down and scribed on clay tablets in order to keep records of the changes of royal possessions and trades [23]. A transaction is a transformation from one state to another. Over several thousand years, the concept has found its way into a broad range of disciplines. For example, in the business world, a transaction is defined as an agreement between a buyer and a seller to exchange an asset for payment. While in the database world, the real state of the outside world is abstracted from and modeled by a database where the transformation of the state is modeled by an update of the database. From this perspective, a transaction can be defined as a group of operations executed to perform some specific functions by accessing and/or updating a database. These operations are in fact a kind of program designed to consistently interact with a database system.

Later, with the wider use of transactional support in the IT domain, the original definition of a database transaction was extended and generalized by using complex structures to support a wide range of applications. In this thesis, the term 'transaction' refers to a reliable and coherent process unit interacting with one or more systems, independently of other transactions, that provides a certain service or function for a running application. This definition reflects the requirements for transactions that are able to capture more complex semantics arising from a broader range of application areas such as workflow management, Web services, and Grid computing.

2.2.1 Transactions for Databases

Originally a database concept, transactions are key mechanisms to guarantee consistent database updates. Along with the increasing complexity of databases, database transactions evolve from ACID transactions with no internal structure, to transactions designed for advanced databases.

ACID Transactions

In practice, a database is an abstract representation that models part of a real organization and keeps its state consistent with the state of that organization. Therefore, the programs that interact with the database need to reflect the requirements of real world. These requirements impose additional restrictions on transaction design. From the mid 1970s, a number of papers were published with early attempts to introduce these restrictions. These were the groundwork for the later defined properties generally known by the famous acronym 'ACID'. The ACID properties proposed in [30] are the basis for the classic form of transactions known as 'traditional transactions' or 'flat transactions':

- **Atomicity**: A transaction either runs completely or has no effect at all, which means from the outside, that a transaction completes and appears to have no observable intermediate states or appears have never left the initial state.
- **Consistency**: A transaction is a correct program and preserves all the integrity constraints. After the execution, the new state of the database complies with all the consistency constraints.
- **Isolation**: A transaction is executed as if there are no other concurrent transactions. The effect of the concurrent transactions is the same as

the effect when the transactions are executed serially.

- **Durability**: A transaction completes successfully and thus makes a permanent change to the state of the database. Consequently, the results from a transaction must be able to be reestablished after any possible failures.

The VCRP (Visibility; Consistency; Recovery; Permanence) can be used to describe the transactional properties in a more general way. In [71], the authors use these four notions to analyze and compare some transaction models such as nested transactions, sagas etc. The VCRP properties are actually four measurements of transactions: **Visibility** represents the ability of an executing transaction to see the results of other transactions. **Consistency** refers to the correctness of the state of the database after a transaction is committed. **Recovery** means the ability to recover the database to the previous correct state when failures occur. **Permanence** is the ability of a successfully committed transaction to change the state of the database without the loss of the results when encountering failures. By capturing the VCPR properties of the transactions, the author provides a standard framework to evaluate them.

We can apply VCRP to measure the traditional database transactions, also known as flat transactions, which are characterized as having no internal structures. The visibility is measured as ‘isolated’. The consistency is measured as ‘consistent’. The recovery is measured as ‘atomic’. The performance is measured as ‘durable’. The database transactions are ACID transactions measured by the VCPR framework..

The underlying transaction processing (TP) system is responsible for ensuring the ACID properties. A TP system generally consists of a TP Monitor, which is an application used to manage transactions. It controls access to one or more DBMSs (Database Management System), as well as a set of application programs containing transactions [34]. Atomicity and durability are guaranteed by the mechanism of recovery that is usually implemented by maintaining a log of update operations so that ’redo’ and ’undo’ actions can be performed when required. Isolation is guaranteed by the mechanism of concurrency control, which is usually implemented by using locks during the transaction process. A detailed overview of concurrency control and recovery techniques is available in [50]. Consistency is guaranteed by the integrity control mechanism usually provided by the TP system, though it is not complete in a strict sense. There are two approaches to guarantee consistency: One implementation is to incorporate integrity control into DBMSs presented in [26], while another is to comply with the integrity constraints through the effort of application designers instead of TP systems described in [23].

Advanced Transactions

As stated previously, ACID transactions, though very simple and secure, lack the ability to support long-living and/or complex transactions. Therefore, a lot of advanced transaction models have appeared to address such needs. The fundamental logic of advanced transaction models is to divide a transaction into sub-transactions according to the semantics of the applications. These sub-transactions, also referred to as component transactions, can also be divided if necessary. The advanced transactions can perform more complex and longer-lasting tasks. For instance, when a failure occurs during a long-living transactional process, the system is able to restart from the middle of the transaction instead of the very beginning.

Save Points The history of advanced transaction models can trace back to the mechanism of save point, a mechanism first introduced in [4], which enables a transaction to roll back to an intermediate state. The authors suggest that during the execution of a transaction, a save point can be marked which in turn returns a save point number for subsequent reference. At each save point, special entries are stored containing the state of the database context in use by the transaction and the identity of the lock acquired most recently. When a transaction fails, it rolls back to the recorded save point, where it restores the corresponding context and releases locks acquired after this save point. This way, rollback action can return the system to a previous state in case of failure.

When applying the rollback mechanism using save points, we should be careful about its constraints and limitations. For example, despite of the rollback of the database to the previously recorded state, the transaction's local variables are not rolled back, which means the transaction actually follows another alternative execution path after the rollback. Furthermore, after a rollback to one save point, the subsequently created save points are lost. Although the idea of a persistent save point had been proposed to overcome the deficiency, it is hard to implement this idea in reality. For example, the database content can be rolled back to the previous state, but the local programming language variables will be lost. Another notice is that rollback is different from abortion. When aborted, the transaction is rolled back to the state in which it started and the execution doesn't continue anymore. In contrast, a transaction rolled back to a save point still continues execution until it commits.

The save point mechanism led to the later development of advanced transaction models that had emerged since the mid 1980s, i.e. distributed transactions, nested transactions, chained transactions, etc. We will see later that a chained transaction is a variation of save points, while the nested transaction is a generalization of save points, as they both save middle status in the child-nodes. These models are more or less application-specific and each of them addresses the need of some given situation. If an organization needs to integrate several database systems residing in different servers to perform more comprehensive tasks in a multi-database system (MDBS), a distributed transaction (or sometimes referred to as a multi-database transaction) is needed. When considering complex-structured applications, a nested transaction properly addresses the need. For a time-consuming application with long-lasting transaction processes, a chained transaction is suitable to handle the problem. The above mentioned models are examples of applying the idea of save point in different cases. Next, we introduce these transaction models one by one.

Distributed and Nested Transactions Distributed transactions consist of sub-transactions that may access multiple local database systems. Consequently, in addition to meeting integrity constraints in local systems, a Multi-database System (MDBS) also imposes global integrity constraints on a transaction. Furthermore, a MDBS addresses other concerns like global atomicity and isolation. For instance, the whole transaction is aborted if any sub-transaction fails. In [7], the most popular model at that time, the 'base transaction model' is introduced. The model defines two types of transactions: local transactions and global ones. Local transactions are executed under the control of the local DBMS, while the MDBS is in charge of global transactions. Several approaches to realize transaction atomicity and database consistency are discussed. This paper also proposes the possible extensions to this basic model and provides an overview of the most recent work in the MDBS area. In addition, it raises some open problems for future research, such as a need for the standardization of the operating system, communication interfaces, and database systems.

In contrast to distributed transactions, which split a transaction into sub-transactions using a bottom-up technique from a geographical point of view, nested transactions adopt a top-down method to decompose a complex transaction according to their functionalities into sub-transactions or child transactions. In [39], this concept was first proposed by programming transactions in a structured way. As it claims, nested transactions overcome the shortcomings of single-level transactions by, for example, permitting parts of a transaction to fail without necessarily aborting the entire transaction. The idea is that a transaction is composed of sub-transactions in a hierarchical manner, which means a sub-transaction can be divided into further sub-transactions if necessary. Importantly, only the leaf-level sub-transactions really perform database operations, while others function as coordinators. A child transaction can only start after its parent starts and a parent can only commit after all its children have been terminated. The commitment of a child transaction is conditional on the commitment of its parents. Each child is atomic, and can abort independently regardless of its parent and siblings. After an aborted action, the intermediate state of the database is inconsistent, which is only a temporary state. The parent will take an action, such as triggering another sub-transaction, as an alternative. Thus, the whole nested transaction still meets the consistency requirement as if the aborted sub-transaction had never been executed. The mechanism of the model is very powerful and has a strong relationship with the concept of modularization in software engineering. Nested transactions gained a lot of attention – later on, some models appeared based on the idea.

Multilevel transactions (also called layered transactions) introduced in [72] and their generalization, open nested transactions, are based on the idea of nested transactions. The authors presented the concept of a multilevel transaction as a variation of a nested transaction, where a transaction tree has its levels corresponding to the layers of the underlying system architecture. Note that the leaf nodes are all at the bottom level, i.e. the depths of these leaves are the same. The authors introduce the concept of pre-commit, which allows for the early commitment of a sub-transaction before the root transaction actually commits, thereby making it impossible to roll back in a traditional way. When a parent transaction needs to roll back a sub-transaction, it uses a compensating sub-transaction to semantically undo the committed one instead of using a state-based undo. Note that there are three differences from the nested transactions. First, children are executed only sequentially, not concurrently. Second, all the leaf-level sub-transactions are at the same bottom level in the transaction tree. Third, the commitment of a sub-transaction is unconditional, thereby making the result visible to other concurrently executing sub-transactions at the same level.

Based on the multilevel transaction model, if the structure of the transaction tree is no longer restricted to layering, which means leaves at different levels are allowed, it evolves to open nested transactions. The authors investigated how open nested transactions relax the ACID properties to achieve the ideal orthogonality so that each of ACID properties can be omitted without affecting the others, to some extent. Compared to the nested transactions that guarantee global level isolation, which means the intermediate results of committed sub-transactions in nested transactions are invisible to other concurrently executing ones, open nested transactions relax the isolation property in the global level to achieve a higher level of concurrency.

Chained Transactions and Sagas Although the nested transaction and its extensions are more powerful than the classical flat transaction, they are only fit for some specific environments like federated databases, and are not suitable for environments requiring long-lived transactions. In such cases, the idea of chained transactions by decomposing a long running transaction into small, sequentially-executing sub-transactions was adopted. According to [23], the idea originates from IBM's Information Management System (IMS) and HP's Allbase database products. This idea is a variation of the save point mechanism that a sub-transaction in the chain roughly corresponds to a save point interval. However, the essential difference is that each sub-transaction itself is atomic, while each interval between every two save points is only part of an atomic transaction. In the chain, a committed sub-transaction triggers the next upon commitment, one by one, until the whole chained sub-transactions commit. When encountering a failure, the previously committed sub-transactions would have already made durable changes to the database so that only the results of the currently executing sub-transaction are lost. This way the rollback only returns the system to the beginning of the most recently-executing sub-transaction. Notably, from the application perspective, the atomicity and isolation properties are no longer guaranteed by the whole chain. For example, in the middle of execution, all the committed sub-transactions cannot be undone, which leads to a problem in aborting the whole chain. Another case is that other concurrent transactions can see the intermediate results generated during the execution of the chain.

Based on the idea of chained transactions, Sagas were proposed in combination with a compensation mechanism to roll back. The saga model described in [20] is a classic transaction model used as a foundation of many later transaction frameworks. Sagas divide a long lasting transaction into sequentially executed sub-transactions and each sub-transaction, except the last one, has a corresponding compensating sub-transaction. All these sub-transactions are atomic with ACID properties. When any failure arises, the committed sub-transactions are undone by those compensating sub-transactions. Unlike the non-atomic chained transactions that cannot undo the committed sub-transactions in the case of an abort, sagas can use compensating sub-transactions to return the whole transaction back to the very beginning. Note that the recovered start state is not exactly the same as the original start state but only equivalent to it from an application point of view. In this sense, sagas as a whole still preserve application-dependent atomicity. Similar to chained transactions, a saga transaction may be interleaved with other concurrent transactions, thus isolation is not guaranteed. Consequently, consistency in sagas is not realized by serializability, a common technique to keep the database consistent when accessed by multiple transactions. The saga model is an important transaction model which attracted a lot of attention. For example, some extensions of sagas are introduced in [15] with more recovery options.

Summary of Database Transactions

ACID transactions have proven to be very useful in traditional database applications, where the execution time is relatively short, the number of concurrent transactions is relatively small, and the database system only resides on one server. However, they lack the flexibility to meet the requirements of the complex applications. For example, multi-database operations need a certain level of transparency for the interactions with each local database; a workflow system needs to support long-living transactions etc.

Advanced transaction models can be viewed as various extensions to flat transactions that release one or more ACID constraints to meet with specific requirements. Two strategies have been adopted for extension purpose to achieve different structures inside a transaction. One is to modularize a complex transaction with hierarchies. Using these means, a large transaction is divided into smaller components, which can in turn be decomposed. This strategy has been applied in various transactions, including distributed transactions, nested transactions, multilevel transactions, and open nested transactions. With the modularization of a complex transaction, the structure is clearer from a semantic perspective. Another strategy is applied in chained transactions, sagas etc, where a long-lasting transaction is decomposed into shorter sub-transactions. By means of splitting the long processing time, each transaction can be divided into a sequential series of smaller components that are operated in a shorter time, thus minimizing the work lost during a crash.

Note that there are other works under this category. We skip the description of them because they do not exhibit an internal structure as typical as the above mentioned advanced transaction models. For example, the Split-transaction in [48] has been proposed for open ended applications where the finish time is unknown in advance, so it has a dynamic structure (split and join). The Flex transaction model proposed in [19] for MDBS applications can also be viewed as an advanced transaction model. It consists of an inexplicit hierarchy of local autonomous transactions, though it has some characteristics of workflow transactions.

However, the abundance of the advanced models does not mean that flat transactions have been replaced by these more powerful models. On the contrary, because of their simple structures and easily implemented ACID properties, flat transactions still dominate the database world.

2.2.2 Transactions for Workflow

Based on the advanced transaction models discussed in the previous section, specific transaction models have been designed for the support of business processes, usually identified as workflow transaction models. The concept of transactional workflow was first introduced in [53] to clearly state the relevance of transactions to workflows. Since the mid 1990s, two developments have taken place in the area of workflow technologies. One was the development of the transaction model supporting workflows and the other was the development of languages for workflow specification. From a transactional point of view, workflows are generalized extended transactions which focus on the automation of the complex, long-lasting business processes in distributed and heterogeneous systems. A workflow process may involve database transactions or human activities, so the ACID properties would not be the major concern anymore. Similar to the decomposition mechanism of advanced transaction models, a workflow process can be modeled by decomposition into some sub-processes in a hierarchical or sequential way. From this perspective, a workflow process can be viewed as a complex transaction hierarchically or sequentially consisting of sub-transactions and/or non-transactional tasks.

ConTract Model First proposed in [51] and finalized in [64], the ConTract model addresses the transactional challenge for long-lasting and complex computations in a formal way. It was not classified as a workflow transaction model. Instead, it was invented for large distributed applications. However, we position the ConTract model here for the consideration of 1) it uses a lot of workflow concepts like 'step', 'flow' and 'script'; 2) its basic idea is to model control flow by programming short ACID transactions into large applications, and guarantee reliability and correctness along the execution, which corresponds to the points in the previous section.

Unlike the advanced transaction models, the ConTract model does not extend the ACID transactions in structure but embeds them in the application environment and provides reliable execution control over them. It defines a unit of work as a step which has ACID properties, but preserves only local consistency. These steps are executed according to a script, which is an explicit control flow description. A reliable and correct execution of the steps is called a ConTract. The ConTract model offers control mechanisms like semantic synchronization, context management and compensation at the script level to provide transaction support to a long-lived and complex application.

WIDE Model In [29], a two-layer transaction model, known as the WIDE transaction model, was presented. The model introduces the concept of a safe point, which is similar to the save points in Sagas. The bottom layer consists of local transactions with a nested structure that conform to the ACID properties [6]. The upper layer is based on Sagas that roll back the completed sub-transactions using the compensation mechanism, thus relaxing the requirement of atomicity. The semantics of the upper layer have been formalized using simple set and graph theory [25]. The local transaction layer was designed to model low-level, relatively short-living business processes, whilst the global transaction was designed to model high-level, long-living business processes. The WIDE model was adopted later in [60] to develop a more comprehensive X-transaction model. Note that these two models address the needs in different contexts. The WIDE transactional model caters for intra-organizational workflow while the X-transaction model can deal with specific inter-organizational workflow.

X-transaction Model The X-transaction model [60] is a three-level, compensation based transaction model for inter-organizational workflow management. It is developed in the CrossFlow project, where a contracted service outsourcing paradigm was presented. The three levels in this model are the outsourcing level, the contract level, and the internal level, each with a different visibility to the consumer or the provider organization. The model views an entire workflow process as a transaction. For intra-organizational processes, they can be divided into smaller I-steps that adhere to ACID properties. Each I-step has a compensating step in case of failure. Similar to this idea, a contract-level cross-organizational process is divided into X-steps, each of which corresponds to one or more I-steps. With the components of I-steps, X-steps and compensating steps, the X-transactional model realizes a flexible intra- or cross- organizational rollback effect, so as to support all the scenarios with all the combinations of rollback scopes and rollback modes.

The authors also proposed an architecture to support this model. There are three layers in the architecture as in the transaction model, where a dynamically created upper layer is built on the top of the static layer, which involves local Workflow Management Systems. Between them, an isolation layer exists to provide portability with respect to specific WFMSs.

Summary of Workflow Transactions

The focus of workflow applications is the control-flow, which is different from the data-centric database applications. From the above discussion, we can see that the transaction support for workflows is not restricted to ensure ACID properties anymore. Workflow transactions usually leverage the traditional transaction mechanisms for recovery and concurrency control but meanwhile address more coordination requirement. We notice another feature of workflow transactions is that they are more or less platform/software dependent and often process-specific. With the development of Internet applications, transaction support for today's workflow processes has also evolved, placing more attention on the communication, distribution and coordination aspects. Along with this trend, we discuss transaction management in the Internet environment in the next section.

2.2.3 Transactions for Internet

Along with the growing popularity of Internet, there is a demand for a reliable execution of applications distributed using this medium. After the workflow era, the transaction management research community starts to look into transaction solutions that work with maturing Internet protocols and standards. Next, we provide a brief overview of two types of transactions: One for Web services and the other for Grid computing environment.

Web Services Transactions

Web services have become the mainstream implementation technology of Service Oriented Architecture (SOA). In this section we focus on transactions; SOA and related topics are reviewed in thereafter. From the late 1990s, more and more attention has been placed on the area of transactions in the loosely-coupled Web services (WS) world. Web services enable a standard communication between service parties (i.e. provider, user, intermediary), so that the implementation of a service is hidden and irrelevant when invoking this service.

In addition to other accepted standards such as SOAP, WSDL, UDDI etc., a technique to guarantee the consistency and reliability of WS applications is needed. We introduce briefly the standardization effort in this area.

Business Transaction Protocol (BTP) BTP [12] was developed by OASIS, which, as the name shows, is not exclusively designed for Web services but also for non-Web services applications. The latest version is Business Transaction Protocol Version 1.1, which was approved in 2004 as a Committee Draft. This is a revision of the Version 1.0 specification approved in 2002, based on feedback and implementation experience. BTP is an eXtensible Markup Language (XML) based protocol for representing and seamlessly managing complex, multi-step business-to-business (B2B) transactions over the Internet. It allows coordination of application work between multiple participants and uses a two-phase outcome coordination protocol to ensure a consistent result. It claims 'ideally suited for use in a Web Services environment', as it can coordinate between services provided by different organizations. It defines communications protocol bindings so that it can work both with Web Services as well as other communication protocols. BTP is a light standard, focusing on interoperability while avoiding dependencies on other standards. It defines roles and messages but does not define interfaces to be used by programmers, which is also said to help lower the hurdle to implementation.

Web Services Transactions (WS-Tx) WS-Tx consists of three components – WS-Coordination (WS-C) specification in [40], WS-AtomicTransaction (WS-AT) specification in [42], and WS-BusinessActivity (WS-BA) specification in [41]. The latest version is v1.2, which was approved in 2009. The standardization effort was initiated by Microsoft, IBM and BEA and the resulting specifications have been approved by OASIS as the standards for Web Service transactions.

The WS-C is a specification of an extensible framework that coordinates the actions of distributed applications towards consistent outcome via coordination protocols. The coordination framework enables existing transaction processing, workflow, and other systems for coordination to hide their proprietary protocols and to operate in a heterogeneous environment. Additionally this specification describes a definition of the structure of context and the requirements for propagating context between cooperating services.

The WS-AT is a specification that defines the Atomic Transaction coordination type to be used with the coordination framework from the WS-Coordination. There are three specific coordination protocols for the Atomic Transaction coordination type: completion, volatile two-phase commit, and durable two-phase commit. These three coordination protocols can be used alone or together to build applications from short-lived distributed activities (i.e. atomic transactions).

The WS-BA is a specification that defines two Business Activity coordination types, AtomicOutcome and MixedOutcome, which are used with the extensible coordination framework described in the WS-C. This specification also defines two specific Business Activity agreement coordination protocols: BusinessAgreementWithParticipantCompletion, and BusinessAgreementWithCoordinatorCompletion, which are used to build applications that require consistent agreement on the outcome of long-running distributed activities (i.e. business activities).

WS Composite Application Framework (WS-CAF) WS-CAF in [10] was also under the umbrella of OASIS, initiated by a consortium consisting of SUN, Oracle, Arijuna, and some other companies, with the purpose of developing an interoperable, an easy to use and implement, framework for composite WS applications. Similar with WS-Tx, it is also a series of specifications consisting of WS Context [8], WS Coordination Framework [9], and WS Transaction Management [11].

In [35], a comparison between BTP and WS-Tx was made to show that these two specifications both attempt to address the problems of running transactions with Web services. With a clear list of pros and cons, the authors conducted a comparative analysis of the two competitors, using a table. At the end, they conclude that the two specifications differ in some critical areas such as transaction interoperability. It also concludes that BTP lacks ‘the ability to use existing enterprise infrastructures and applications and for Web services transactions to operate as the glue between different corporate domains’. Considering the fact that large, strongly-coupled corporate infrastructures exist behind those loosely-coupled Web services, the authors call for the attention of leveraging ACID transactions, which underlying the internal corporate infrastructures, instead of replacing them with new models to design WS transactions.

Grid Transactions

Like an electricity power grid that pools together distributed electric energy, Grid computing is a form of distributed computing that involves coordinating and sharing computing resources across the web globally. Because of its vision to create a worldwide network of computers that act as if they are one, Grid computing makes the exclusive immense computing power previously only available to a few organizations now available to everyone. This new emerging technology has been gaining a lot of attention from its conception. With more and more projects launched, a lot of research is available within the areas of infrastructure and middleware. Much less effort is spent in the area of Grid transactions however.

One of the three efforts in this area is the Grid transactions TM-RG (GGF Transaction Management Research Group) initiated in Europe, working on Grid transactions with the goal of investigating how to apply transaction management (TM) techniques to Grid systems. It is stated in the charter [55] that 'a common grid transaction service would contribute a useful building block for professional grid systems'. Another effort at Shanghai Jiao Tong University proposed a new service-oriented Grid Transaction Processing architecture called GridTP based on the Open Grid Services Architecture (OGSA) platform and the X/Open DTP model [49]. The authors claimed that GridTP provides a consistent and effective way of making existing autonomously managed databases available within Grid environments. In [56], a protocol ensuring correct executions of concurrent applications on the global level, with the absence of a global coordinator, is proposed to realize the concept of distributed, peer-to-peer grid transactions. The approach is based on some known concepts and techniques, such as the recoverability criterion, serialization graph testing, and partial rollback. The main idea of the approach is that dependencies between transactions are managed by the transactions, so globally correct executions can be achieved even without the complete knowledge gained from communications between dependent transactions and the peers they have accessed. The idea is innovative in the sense that it combines old concepts and techniques for a new purpose.

Summary of Internet Transactions

We have been witnessing a rapidly increasing e-business, which often involves multiple organizations all across the world dynamically establishing business relationships over the Internet. We notice that, with the development of IT toward a broader geographical scope and larger scale, transaction management is correspondingly following a trend to address the need for more functionalities and better performance in the Internet environment (e.g. distributed, heterogeneous, and cross-organizational). To address this need, the proposed Web Services transaction standards and Grid transactions incorporated ideas from the transaction concept, mechanisms, and models in database and workflow environments.

2.3 Service-oriented Contract-driven Business Processes

In the last section, we reviewed research efforts in the transaction management area following a time thread, starting from the database era and finishing with Internet era. Bearing the research goal (i.e. a transaction framework for service-oriented, contract-driven business process) in mind, we look into the other related areas in this section. Next, we review the research efforts in the service and contracting areas.

2.3.1 SOA and SOC

The optimal integration of distributed IT resources to meet agile business requirements has long been a challenge. Many efforts have been made towards a cost-effective and easy-to-implement solution to address this challenge for increasingly complex business processes. One popular solution is SOA (Service Oriented Architecture), which emerged to address IT asset reuse and standardize communications between parties. The basic idea of SOA is shown in Figure 2.1 to illustrates inter-operability among parties which interact with each other to enable a service life cycle from publish until invocation. There are three parties involved in the SOA: a service provider, service user, and service intermediary. A provider designs a service to offer some kind of business functions for users to invoke through an interface. To enable the service discovery by the potential users, a service intermediary is often used as an advertiser. A user searches a desired service usually from a repository of an intermediary and then invoke through the interface of the provider.

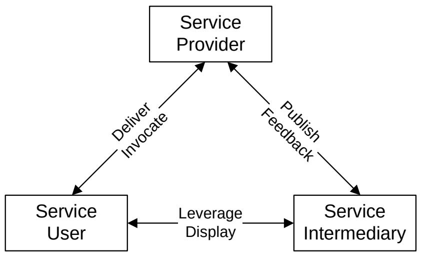


Figure 2.1: Basic service oriented architecture

According to [45], Service-Oriented Computing (SOC) is the computing paradigm that utilizes services as the foundation for developing applications. SOC covers the scope from basic service publishing to advanced service management. We see from the above Figure 2.1 that the basic SOA is limited for complex business scenarios, which often demand extensive properties when composing and managing services (e.g. security, management, etc.). Therefore, an extended SOA was proposed, which is shown as a SOA pyramid with three layers in Figure 2.2. It illustrates a broader scenario that involves various service roles and their interactions, and the authors point out in the paper that transaction management is a challenging topic in the composition layer.

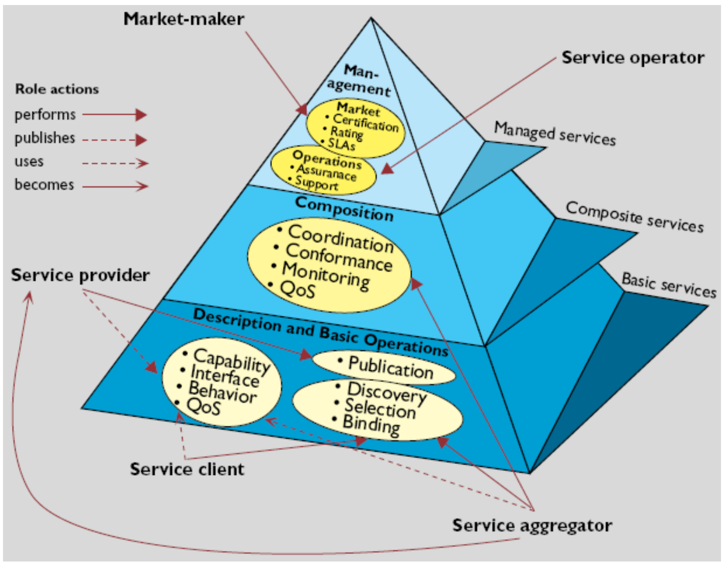


Figure 2.2: An extended service oriented architecture [45]

If implemented in Web service technology, the distributed applications are published as Web services for users to invoke using Web services standards such as SOAP[63], WSDL[13], and UDDI[57]. Web services technology has been backed by the industry with numerous products and research efforts that resulted in a package of maturing standards. However it is not the only way to realize a SOA implementation. For example, Grid services is a paradigm that utilizes Grid computing infrastructure for SOA implementation. In the context of the XTC project, we consider service a broader concept than Web services. Therefore, service-oriented in this thesis represents the way of how business processes are composed, which can be implemented by any type of services including manual services.

2.3.2 QoS and SLA

Quality of Service (QoS) and Service Level Agreement (SLA) are originally proposed to enhance network-level reliability with regard to traffic management, message routing, etc. Managing QoS has become increasingly critical in SOC paradigms, as shown in the upper layers of the SOA pyramid (e.g. service composition and management) in Figure 2.2. In this thesis we also address QoS in the SOA context, with the focus on transaction issues. The concept of TxQoS (Transactional Quality of Service) first appeared in [37], in which the authors discuss the overall QoS support for Web services. These QoS properties needed for the implementation of Web services applications include availability, accessibility, integrity, performance, reliability, regulatory and security. However they only mentioned the TxQoS term very briefly without a further explanation of how it should be defined and used. Another research effort in a different application domain is the QoS-aware transaction processing framework for mobile transactions proposed in [74]. The authors propose to assign the transaction service in multiple modes, depending on the situation of network traffic, and computing and power resources. Our approach of using TxQoS specification is not limited to any particular type of transactions.

As our work concerns SLAs, it has some similarities with the WSLA (Web Service Level Agreement) framework [31], which addresses SLA issues in a Web Services environment on SLA specification, creation and monitoring. The WSLA project is aimed at unambiguous and clear specification of SLAs that can be monitored via a distributed monitoring framework, and an XML schema to represent WSLAs. The monitoring framework allows provider components and third party services to perform the measurement and supervision activities. Another standardization effort that may be used and extended for our purpose is WS-Policy [32], a general framework to specify and communicate policies for Web services, which allows for expressing the capabilities, requirements, and characteristics of a web service as policies through a set of extensible constructs. WS-Policy provides a model and syntax to define a policy as a set of policy alternatives and each alternative contains a set of policy assertions. Each policy assertion describes an individual requirement, capability, or QoS characteristics.

Within the XTC project, the other researchers also addresses transactional reliability using business semantics citepapazoglou072. The motivation of their approach is the current inadequacy of Web service transactions on expressing application logic of common business functions. Thus, the authors proposed a business-aware Web services transaction model with supporting mechanisms, that allows expressing business functions in SLAs guaranteed by the QoS criteria. Our work shares the same motivation, that transaction solutions have limitations in terms of its guarantee to execute reliably for more and more complex processes. However, we take a complementary view of theirs, and develop the approach for general service-oriented processes. First, we do not limit our research to Web services transactions. We look into the widely used transaction management mechanisms instead (e.g. ACIDity, structured composition, save point, compensation, etc.), and design a more accommodated transaction framework. Second, we bring the transactional semantics with QoS management to find out QoS that measures transactional reliability.

2.3.3 Contract-driven Processes

In this thesis, we refine the research context of business processes as service-oriented and contract-driven. This means our transaction framework works for complex processes, utilizing the emerging paradigms of SOC and e-contracting. Besides the service and contracting perspectives, we also apply layered structure to analyze complex processes. Our analysis conforms to the three-level framework developed in [27], where business processes are specified on three levels:

1. Internal level, where business processes are specified, including all necessary (technical) details, so that they can actually be executed.
2. Conceptual level, where business processes specifications are abstractions of the internal level processes, leaving out (technical) details. This allows conceptual reasoning about the business process for design and analysis purposes. Multiple conceptual levels can exist, each higher level abstracting from certain details at the lower levels.
3. External level, which represents a business process as it can be viewed by the outside world, i.e., it does not include any technical details, but it also does not include those elements from the conceptual levels that the business does not want to make public to the outside world. Cooperation between organizations takes place on the external level processes.

Cooperation between organizations takes place on the external level processes. Cooperation involves the connection of the external level processes (called local processes) of the involved organizations, which leads to one large overall process, called business network process. Besides the three-level view of a business process, a dual view from both processes and services perspectives presented in [62] serves as the basis for our research. A business process in SOC paradigms can be viewed as a set of communicating services. Relating the process view with the service view, we are able to specify the activities performed by organizations with a clear order, and meanwhile reveal the interactions between organizations.

As a legally binding agreement between partners, contracts play an important role in regulating business relationships, especially in mission-critical business processes. The traditional way of establishing and managing contracts manually is usually resource-consuming (e.g. time, cost, human). To enhance the efficiency and lower the cost, e-contracting is introduced to deal with business and technical collaborations with delicacy and swiftness [1]. When a large amount of dynamic business relationships are involved, e-contracting can be a powerful means to guarantee business trustworthiness while meeting the speed and flexibility requirements.

It is pointed out in [54] that formal XML description in the Web services world is not directly useful to business. The notion of a service metadata document is invented, which contains both business aspects and technical artifacts so that the two communities work collaboratively on a common framework. The authors proposed a service contract template for their purpose of bridging the gap of the two communities. Following this thought, we define the service contract concept as a type of contracts which combine business clauses and SLAs and provide a contractual approach for transaction management.

With regard to the specification language for contracts specifying transactional qualities, we consider e-contracting languages and the WSLA language as candidates. Currently there are some proposals about e-contracting languages such as [5, 1], but none is widely adopted to our best knowledge. The WSLA language may well define normal Web services QoS, but it seems too limited for the expression of transactional semantics. In our research, we develop an approach to specify transactional qualities in contracts. However, our goal is to show a possible representation while a fully elaborated contracting language is not the goal of our research.

2.4 Summary of Chapter

In Section 2.2, we reviewed the area of transaction management from a temporal perspective. According to the detailed survey in [70], we distinguish the following 'ages' (i.e. from the 'early dark days' to modern time):

Stone Age In the stone age, no explicit transaction management models and mechanisms were available. Reliability of business processes running on (database) systems was often not yet considered an issue at all. If it was, its support was entirely the responsibility of application logic.

As this age is not too interesting from a transaction management point of view, we do not pay attention to it.

Classic History During the classic history, people realized that reliability of processes in multi-user, concurrent environments is an issue that deserves explicit attention - or rather requires explicit attention, in order to keep things running correctly. In this age, the basic transaction model and mechanisms saw the light.

Middle Ages In the middle ages, business application grew more complex and hence the requirements to transaction management rapidly increased. Consequently, the simple models and mechanisms developed in the classic history were not sufficient anymore. Consequently, a large variation of advanced transaction models and mechanisms supporting these requirements were developed for various application domains.

Modern Times In modern times, we see the emergence of new application domains, in which the Internet usually plays a prominent role. To allow the proper operation of business processes in this new environment, transaction management has to be 'ported' to the Internet as well. This means, that the results of the previous ages of transaction management history are made fit for application in the Internet environment.

In Section 2.3, we reviewed the work related to service-oriented contract-driven business processes. The basic SOA and an extended SOA pyramid, together with the research efforts in QoS and SLA have been introduced. A three-level framework for specifying service-oriented contract-driven processes is adopted in our research for process analysis. Meanwhile, a dual view on process is adopted as the basis for case analysis. While transaction management provides technical reliability for business processes over distributed and heterogeneous applications, e-contracting provides business trustworthiness between partners. When working together, transaction management and e-contracting ensure reliable execution of complex and long-lasting collaborative processes.

The research efforts reviewed in this chapter are comprehensive, but by no means complete, since the emerging areas like SOA have attracted a lot of attention from both academia and industry, and are undergoing rapid advancements. This chapter provides the basic ingredients used as the background for the development of a new approach to guarantee transactional reliability for service-oriented, contract-driven processes.

Chapter 3

Case Study on Process Reliability

In this chapter, we introduce an online advertising case. We perform an analysis on the technical elements, and examine them in the process context. In parallel, we analyze the business elements in a service context. Based on the process and service views on our case, we discover problems on transactional reliability. One problem is the lack of an unambiguous agreement between providers and users regarding service execution reliability (which is the transaction support to the process in technical terms). Another problem is the ability to accommodate changes in case of exceptions and/or errors. These problems are common for cases with similar properties (i.e. contract driven, service-oriented, jointly executed, and long-lasting processes). Using this case as an inspiration, we define two research problems: 'How to properly specify transactional reliability of a service' and 'How to ensure the transaction specification works in a changing environment'.

3.1 Introduction

Long-lasting processes tend to have more errors and exceptions throughout their execution. In addition, the growing number of parties and business relationships makes execution more likely to be interrupted. An interesting observation is that current approaches, which address reliability of business processes, come from separate domains and are usually not synergetic. Transaction management has long been applied as the effective means to handle exceptions and errors for running applications. Nowadays, in the emerging service-oriented computing paradigm, transaction management has become an important topic to address because processes tend to be long-lasting and are built over distributed services that cross organizational boundaries. Business people are usually not aware and do not understand these transaction mechanisms at the internal level of processes. They perceive reliability in terms of agreements or contracts between business partners to guarantee trustworthiness of processes. Therefore, the transaction support for applications is not their concern although it is the root for reliable process execution.

The demand for reliability from the business world has pushed the IT world for advancements in transaction management, from the original simple ACID transactions to the recent OASIS approved Web service transaction standard [70]. Progress in IT technology has provided opportunities to the business world for enhancing the process execution reliability. For example, the e-contracting paradigm emerged to provide a vision to automate contract establishment, monitoring, and management [1]. We notice that there is an awareness of process execution reliability gap between the IT and business world. Consequently, we are motivated to develop a business transaction framework to address the reliability concern from both transaction management and e-contracting perspectives.

In this chapter, we perform a case study to identify this gap and further refine our research goal. A real-life online advertising case is selected. Next, in Section 3.2, we introduce the case and model it by means of process and service models. Afterwards, in Section 3.3.3 we analyze the elements observed in this case and discuss the findings. We conclude the chapter and elicit the research questions in Section 3.4.

3.2 Case Description and Modeling

In [1], a case is presented on the e-advertising business of one of the largest newspapers in the Netherlands. This department is responsible for publishing advertisements on its website according to clients's requirements, through a feature-limited e-contracting system. We selected this case [68][69][65] because the interesting ingredients (e.g. contract, service, complexity) relevant to our research context are present here: 1) It is an e-contracting case where contracts are used to specify the activities, functions, and guarantees; 2) Several parties collaborate over the process, each performing a business role with specific functions, so that we can view it as service-oriented; 3) The whole process lasts long and exceptions/errors are likely to happen, so that we can analyze it from a transaction perspective.

Throughout the chapter, we use the term ‘medium’ to anonymously name the newspaper. In this case, there are two types of clients who publish advertisements via the medium’s website: individual clients and advertising agencies. For individual clients, the medium asks for full payment prior to the time the advertisements are published. For large advertising agencies, the medium allows them to pay after the advertising period. We only present the agency scenario because it involves more participants and therefore is more interesting and complex.

The dual view [62] is proposed for analyzing service-oriented business processes, and we use it as the basis for our modeling and analysis during the case study. The basic idea of dual view is that process and service can be used to describe the same case, with the former focusing on activity flow and the latter focusing on interactions. Relating the process view with the service view, we are able to specify the activities performed by organizations with a clear order, and meanwhile, reveal the relationships between organizations for contract-driven service-oriented processes. Our case study conforms to the three-level framework developed in [27], where business processes are specified on three levels: the external level for contracting purpose, the conceptual level for reasoning purpose, and the internal level for design purpose. In this case study we focus on the external level of business processes where collaborations take place.

3.2.1 Process View

We first take a process-oriented view, by which the case can be modeled as three processes: one payment process from the client to the agency, another payment process from the agency to the medium, and the main advertising process jointly executed by the three parties. As the payment processes are simple and not suitable for our purpose, we discuss only the advertising process in detail. The e-advertising process in Figure 3.1 presents our process view on the case.

The process starts when a client decides to outsource the advertising campaign to an agency. The agency negotiates with the client and analyzes its requirements. Afterwards, a contract named CA (Client-Agency) is established between the client and the agency, consisting of clauses about the campaign design, time, cost, etc. Then, the agency negotiates with the medium, and a contract named AM (Agency-Medium) is established. There are general provisions stated on the website of the medium as an umbrella of all contracts the medium offers. These general provisions include the payment method, force majeure conditions, etc., and are maintained by the medium without the need to inform its client if updated. As a trusted and frequent business partner, the contract AM is rather simple compared to the contract established directly between an individual client and the medium.

In this process, the two contracts are established and managed under the

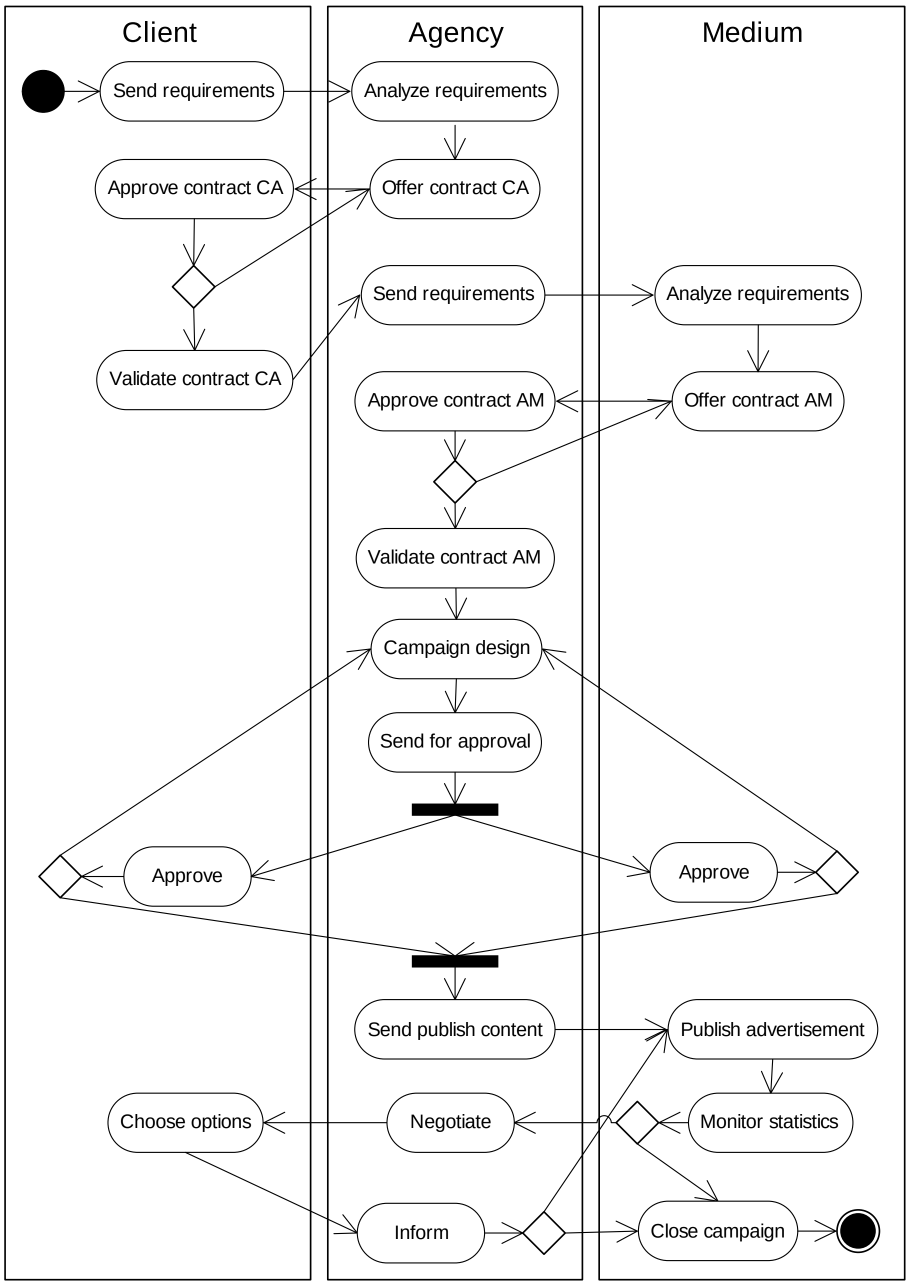


Figure 3.1: E-advertising Case: process view

support of simple e-contracting systems. Next, the agency starts to make a campaign design, including an online version of the advertisement (e.g. pictures with logos and text), according to the client's requirements from the contract CA and the medium's regulations from the contract AM. The campaign design has to be approved by both the client and the medium. The client checks the overall design while the medium checks its image compliance (e.g. format, size, etc.). Afterwards, the version approved by both parties is sent to the medium for publishing online. The campaign can end in two ways. One is impression due, which means enough impressions have been made within the agreed time. The second situation is time due, which means the agreed time is up, but not enough impressions are made in this time frame. In the latter case, the client can choose from three options: extend the campaign time, advertise on other websites owned by the medium, or get back a proportional amount of money. When the campaign ends, the advertising process ends. As the agency is delegated by various clients to work on a lot of campaigns, the medium asks for the payment of all the finished campaigns periodically in a sum. Therefore, a payment process with a different life cycle from the advertising process is needed to settle the finance matters between the agency and the medium. This way, the contract AM is fulfilled. Likewise, a process is needed for handling the payment from the client to the agency within the agreed time frame. In completion of this payment process, the contract CA is fulfilled.

3.2.2 Service View

We present the service view of the case in this subsection. Here we assume that a set of activities with a clear business function can be wrapped as a service with a standardized interface for invocation. These services are invoked by users by means of the requesting messages flowing into the provider side. Service providers perform certain functions according to user requests and the results are delivered back to the user side. Following the service view, we assume the process can be viewed as a series of services and their interactions. As shown in Figure 3.2, we identify six services along the e-advertising process: 'Contracting', 'Design', 'Decision', 'Approval', 'Intermediary', and 'Campaign' services.

In the dual view, we see service as another side of process, which in this case means that a service is seen as a process wrapped inside and communicates with other services through interfaces. Figure 3.3 depicts the details of the implementation activities within each service, which we call 'Service Realization'. The solid-end arrows reflect the message interactions, e.g. invoking, configuring, or replying. When invoked, the standardized interface of a ser-

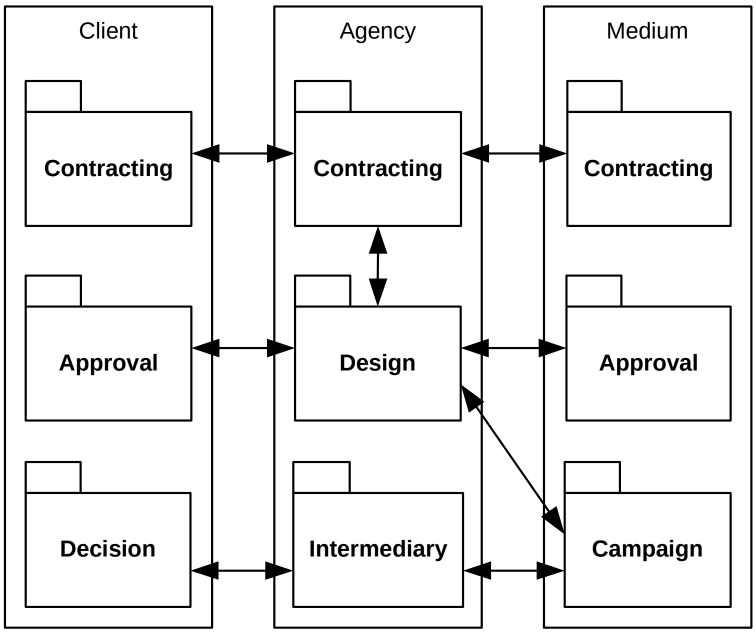


Figure 3.2: E-advertising Case: service view

vice triggers various internal implementations, which can be performed by a sales representative, a Web-service application, a database application, etc. This means that each party can host services, providing the same business function but through different internal implementation details. For instance, in our case, all ‘Contracting’ services provide the function of contract establishment, but their internal implementations are different. The ‘Contracting’ service residing at the medium side is provided through an e-contracting system that allows potential consumers to submit online, while the agency and the client implement the ‘Contracting’ service manually. The arrows going through the interface represent messages used for interaction with other services. Each service has at least one in-coming arrow and one out-going arrow. Please note that the activities within a service box may appear slightly different from the activities in the process view, as they are modeled for a different concern of business semantics.

The service view and the process view are ‘two sides of the same coin’. The terms ‘process’ and ‘service’ may be used interchangeably in the rest of the chapter, depending on the context. The term ‘Process’ is habitually used when we focus on activities or analyze transactions, and ‘service’ is used usually when we aim to emphasize interactions and contracting relationships.

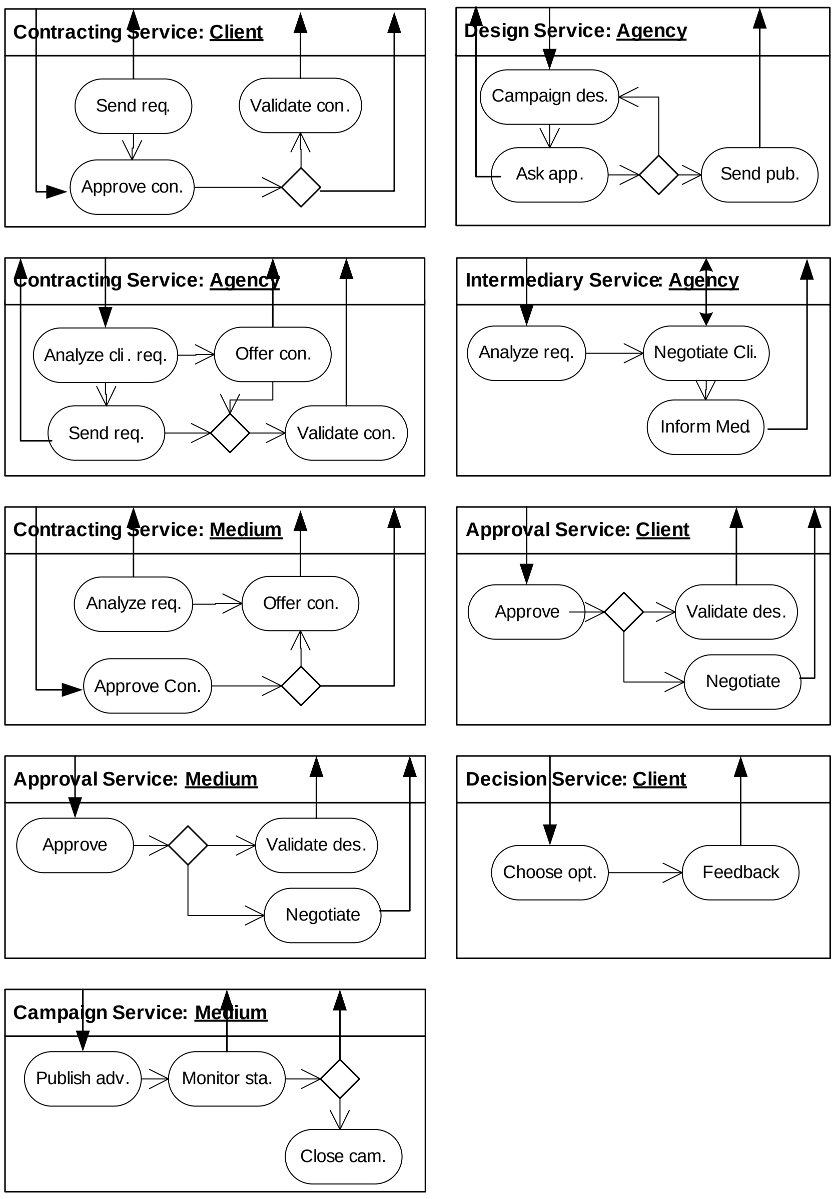


Figure 3.3: Service Realizations

3.3 Case Study: Observation and Analysis

In this section, we present our observations and analysis on the entities involved in the case. Here we define an entity as a generalization of a group of observed elements that can be modeled in a class diagram. We make a list of observed entities below, based on the process and service interaction models and the text description in the previous section. Every element that has appeared in the models can be mapped to one of the entities in our list (e.g. activity, process, contract, etc.). In addition, we add to our list entities that are implicitly stated in the textual description (e.g. role, reliability agreement, process specification, etc.). These implicit entities are not addressed in the process and service diagrams directly. For instance, we use a swimming line to distinguish roles in Figure 3.1, and the actualization of the 'Role' entity, Client, Agency and Medium, are explicitly stated. So the entity 'Role' is a generalization of the elements. Furthermore, we implicitly state that there are few words in the contracts which mention reliability. So we abstract the element into the 'Reliability Agreement' entity.

3.3.1 Process View

In this section, we make observations on the process view of our case study. Some of the explicitly stated or directly-observable elements can be generalized into entities. In other words, these are 'back-end' elements, that are too operational to specify in a business contract. Here we perform an analysis at a conceptual level and do not concern the detailed transactional techniques, mechanisms and systems.

Process: A process consists of activities and flows, where activities are connected and executed following certain logic. The e-advertising case is depicted as a process in Figure 3.1. This process can also be seen as a service if seen from the service view (where we identify service as an entity in the next group of business entities). The entity 'Process' is very generic, in a sense that it can be observed from most of the cases as long as there are some activities and flows of these activities. Processes can be self-containing, which means that processes can be decomposed into sub-processes.

Process Specification: A process specification states the process execution logic (i.e. which step, and in what order), and can be used for transaction design. It is a documented representation of a process abstracting its attributes for deployment. For instance, in our case study, 'Client', 'Medium' and 'Agency' each hosts a business process respectively. A process specification exists at each host, governing a part of the jointly-executed global process plus (not-revealed) internal processes. Upon triggering, a process is executed according to the process specification. The entity 'Process Specification' is not explicitly seen from the model, but it is implied by the case description where we see that some agreement on the execution path is established.

Activity: An activity is a basic unit of a process in our modeling. The activities in the process are the actual steps in a sequence-like chain, with intermediate forks for parallel performing with a starting point and an end-point. As mentioned, the activities appearing in service boxes are slightly different in names and orders as they are used for business descriptions to demonstrate service details. Some activities are performed by humans with the aid of tools (e.g. 'Send requirements' and 'Campaign design'). Other activities are fully automated, such as 'Monitor statistics'. All activities result in the changes of the states of the underlying systems either directly or indirectly.

Flow: Flows are the logic connection of the activities. In our case, the arrows in the process model which point from one activity to another are flows. The simple type of flow is a timely order to indicate one activity ends and the next starts. For example, the order from the client 'Send requirements' is followed by the Medium's order 'Analyze requirements'. The complex type of flow has a choice, which leads to splits and 'OR' joints. For instance, depending on the outcome of the activity 'Approve contract CA' at the client side, either the 'Medium' offer a revised contract or the contract is accepted by the client. It is important to know the flows of the activities in order to determine what transaction support is needed for the process. For instance, usually a long-lasting chain-like process (i.e. our case) requires 'Atomicity' for activities, and 'Compensation' at the process level.

Host: The host is an entity abstracted from the ownership of the processes, including activities depicted in Figure 3.1. A host in this case can be the client or the agency where least activities (usually not long-lasting) take place, and the medium where most of the activities (long-lasting or once-off) take place.

3.3.2 Service View

In this section, we examine the elements observed from the service view. They are generalized and abstracted into entities for analysis of contractual relationships. These entities are usually found in business contracts. The business entities listed below do not address technical issues, as those are typically avoided in business contracts.

Service: The case can be viewed as a series of interacting services as shown in Figure 3.2. Different services usually demand different transaction support. For instance, 'Design' service barely requires any specific transaction mechanism, as it is a human activity with few transaction semantics (e.g. ACID). On the contrast, 'Campaign' service requires comprehensive transaction support to make sure it is either committed or consistently executing after the start. Also note that the services addressing the same function, but are implemented in different ways, may require different transaction support. For example, there are three 'Contracting' services, each owned by each party. The one at the medium side is Web-based and supported by its e-contracting system, while the ones hosted by the agency and the client are manual services.

Party: There are parties involved in this case: the client, the agency, and the medium. From Figure 3.1 and Figure 3.2, we can see which party performs what activities and owns what services. Note that service can be executed by humans and/or machines. For example, the 'Intermediary' service is a manual service, and the 'Campaign' service is mainly implemented by the software at the medium side. So the entity 'Party' is a generic term to take roles and perform activities (i.e. provide/use services).

Contract: Two contracts are established in this case: contract CA (Client-Agency) and AM (Agency-Medium). The contract is a central theme in this e-advertising case, as it triggers the process and governs the execution path when facing choices.

Role: We distinguish two roles a party may play: a service provider and a service user. For each service, one party plays one role at a time. In our case, the agency acts the role of the service provider in the contract CA, and at the same time the service user in the contract AM. For instance, the agency, which owns a 'Contracting' service and acts as a provider, invokes the medium's 'Contracting' service acting as a user.

Reliability Agreement: An reliability agreement is the terms stated in a contract regarding execution reliability of the service one party provides to another party. The whole process that implementing the e-advertising service is a chained transaction that requires save points, given the process nature is long-lasting. In this way, the exceptions and errors will not cause the process to restart from the very beginning. The handling mechanisms of exceptions and errors are usually not present in a business contract. However, we do see there are some implicit statements about the reliability agreed on contract in this case. For instance, there has been some terms in the contract regarding the choice of the execution paths in case of not enough impressions made within the agreed time. These agreed terms can be seen as a reliability agreement that addresses the concern of both transactional reliability and business trustworthiness.

As listed and explained above, the entities are the abstraction of the actual elements observed from the case via dual view. To summarize and organize our findings, we depict their relationships by means of a class diagram, shown in Figure 3.4. In the next section, we discuss and further elaborate on the findings of the case.

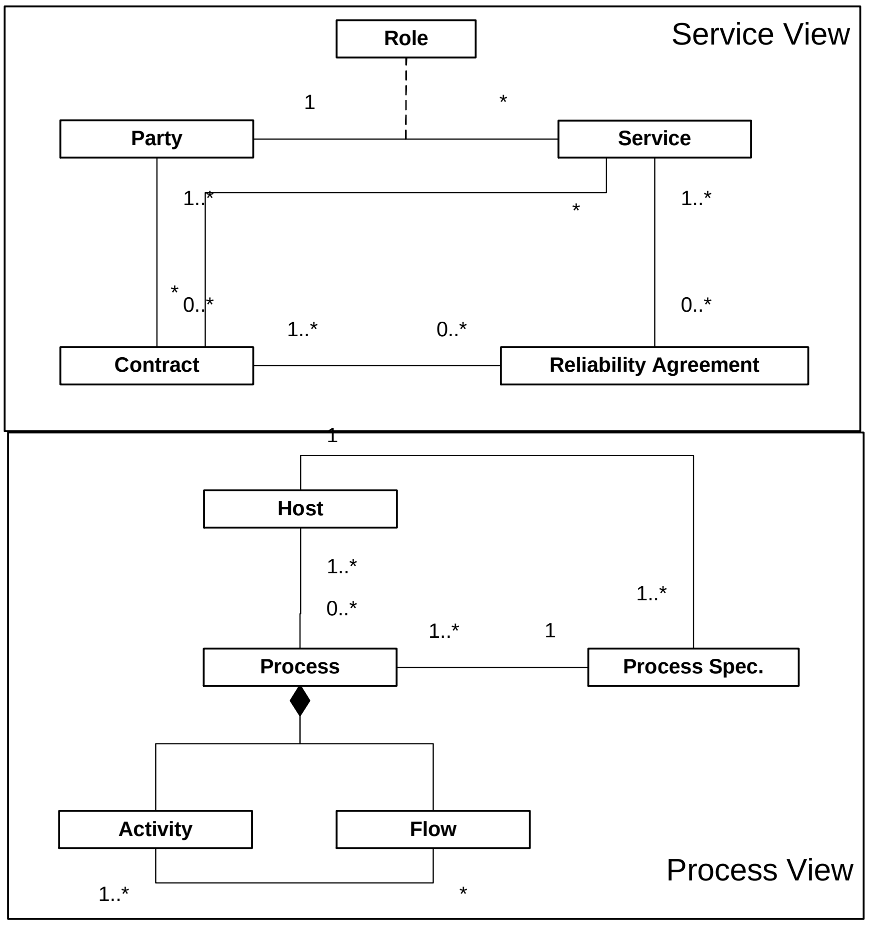


Figure 3.4: Dual View on E-advertising Case

3.3.3 Discussions and Further Elaborations

In Section 3.2, the global process of e-advertising is triggered by a ‘Client’ who submits a filled-in advertising campaign agreement, and is jointly executed by the three parties (i.e. Client, Medium, and Agency). The systems underlying the global process execution include workflow engines, CRM applications, e-contracting applications, and online-publishing applications. The auxiliary applications such as specialized graphical design software and e-mail tools are not our concern. From the analysis, we can see that the main process in this case is a typical long-lasting and cross-organizational process driven by e-contracts. The findings from the case study should not be special for this e-advertising case only, but meaningful in general in our research context. In this section we analyze and identify relevant research issues for the generic case.

3.3.4 Discussions On Case

The three parties collaborate effectively on this case, considering the fact that the medium's e-advertising business runs smoothly on a daily basis. On the one hand, we notice that some of the exceptions are well handled and the solutions are clearly stated. For example, usually the campaign finishes within the expected impressions, and in the agreed campaign duration. In case there are not enough impressions made within the agreed time, the 'Intermediary' service is invoked, which allows the client to choose from three options. One option results in a reduced cost, and directs the process to the activity of 'Close campaign' in the 'Campaign' service. Another option results in time extension, which directs the process back to the previous activity of 'Publish advertisement' in the 'Campaign' service. The third option results in a continuing online campaign on other websites owned by the medium, which is out of the scope of this investigation.

On the other hand, we also find out that some exceptions and faults are caused by opportunistic behavior, because the corresponding transaction mechanisms are missing. For example, negotiations may break down or the client may decline the contract offered by the agency, thus suspending the process. Another extreme example might be that the medium does not fulfill its obligations because of a unexpected hack into its website. As illustrated in Figure 3.5, all these possible exceptions and errors may lead the process to undesired states and break the execution. From the above discussion, we see that the presence of a reliability agreement is not adequate in this case. This finding requires further research on the issue of transactional reliability addressed by business agreements.

By identifying and revealing the generic entities, we have displayed their interrelationships in the process and service views in Figure 3.4. As we explained, the dual view means an entity in one view should have a corresponding entity in the other view. In other words, the dual view presentation is roughly a reflection, where business entities from the service view should see their more technical counterparts in the process view.

We reveal the correspondence between the two views in Figure 3.6, which is based on Figure 3.4 with entities 'Role', 'Activity', and 'Flow' removed, as

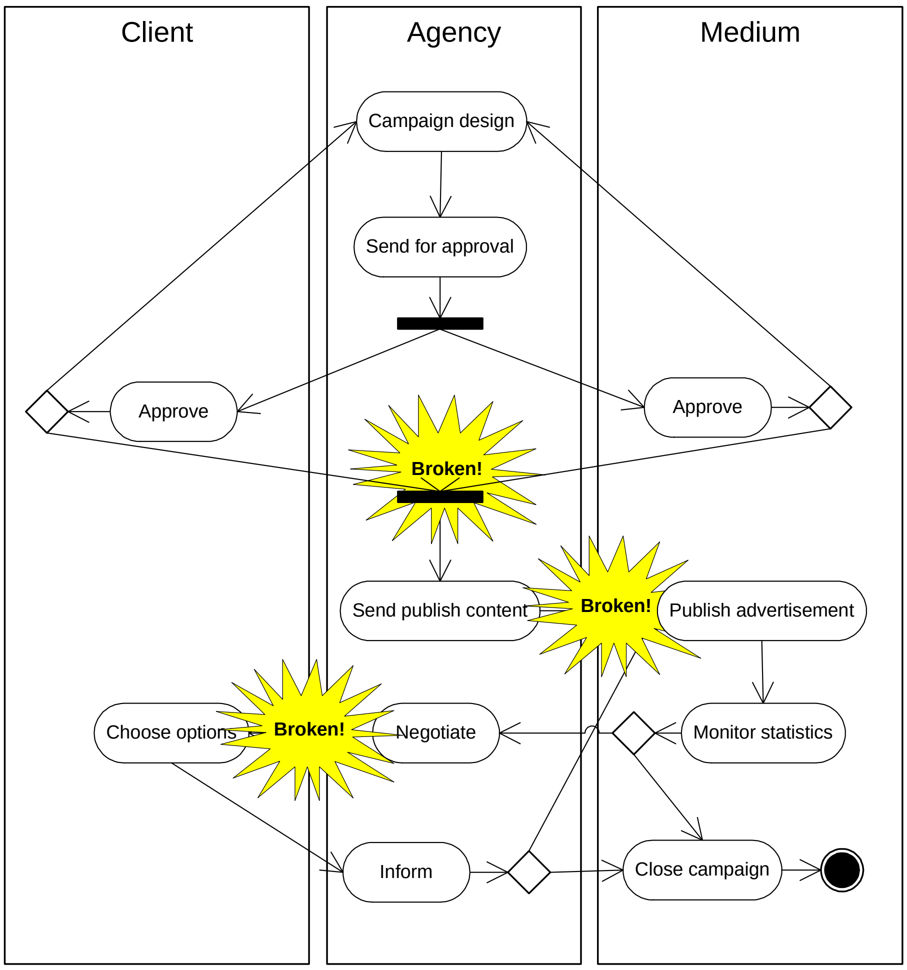


Figure 3.5: Unexpected Exceptions and Errors in Process Execution

they are not main classes in the model. 'Role is an association class, whereas 'Activity' and 'Flow' are children classes under the parent 'Process'. We can see that the entity 'Party' in the service view corresponds to 'Host' in the process view. The entity 'Service', if viewed from the process perspective, is the entity 'Process'. However, the '***Reliability Agreement***' entity, does not have a counterpart in the process view. In other words, we see an asymmetric picture where one entity can not find its counterpart in the other view. This asymmetry serves as an inspiration for us to deepen the analysis of the two views.

From the above discussions, we have found out 1) the entity 'Reliability Agreement' in not addressed adequately and explicitly in the case; 2) the service view and the process view on the case are not symmetric because of the

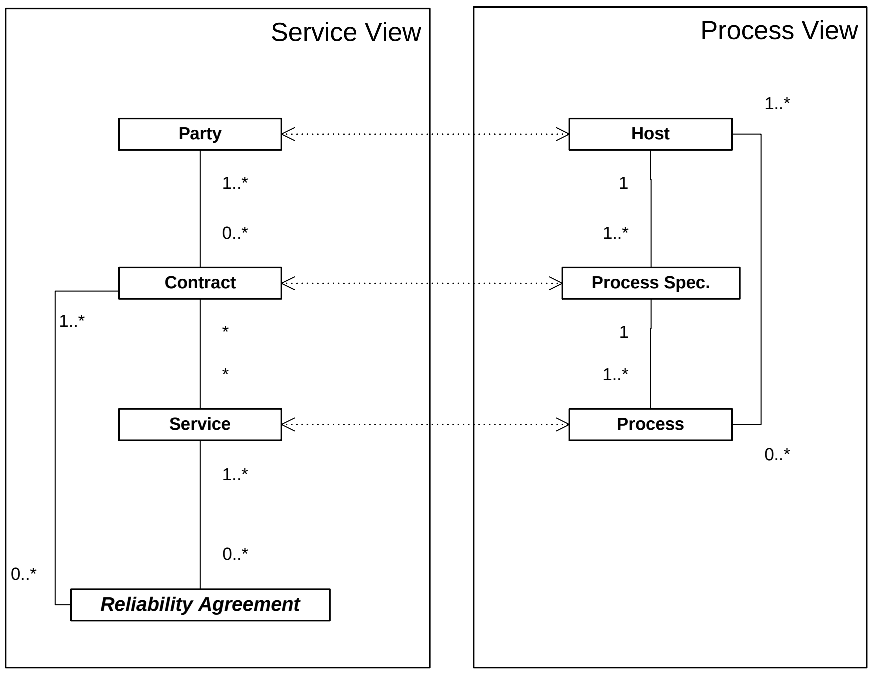


Figure 3.6: Mapping of Entities in Dual View

lack of a mirrored entity to ‘Reliability Agreement’. There is a need to further elaborate on these findings with the focus on the ‘Reliability Agreement’ entity to address the reliability concern. We are encouraged to address this problem in the inter-related areas of transaction management (i.e. reliability) and e-contracting (agreement).

3.3.5 Further Elaborations

We have found out that ‘Reliability Agreement’ is neither adequately addressed in the case nor balanced by a mirrored entity in the dual view analysis. This points into two directions: 1) Study ‘Reliability Agreement’ in further detail to make it explicit and adequate; 2) Balance ‘Reliability Agreement’ by identifying the missing entity in the process view. Therefore, we propose to use proactive agreements on transaction support shown by the grey-box entities in the to-be scenario depicted in Figure 3.7:

Following direction 1, we propose a new entity in the service view: ‘Business Tx Spec’. As discussed, an explicitly and adequately specified reliability agreement is needed, and an unambiguous specification of process execution reliability is the key. We model the ‘specification of execution reliability’ as the entity ‘Business Tx Spec’, which specifies execution reliability in business

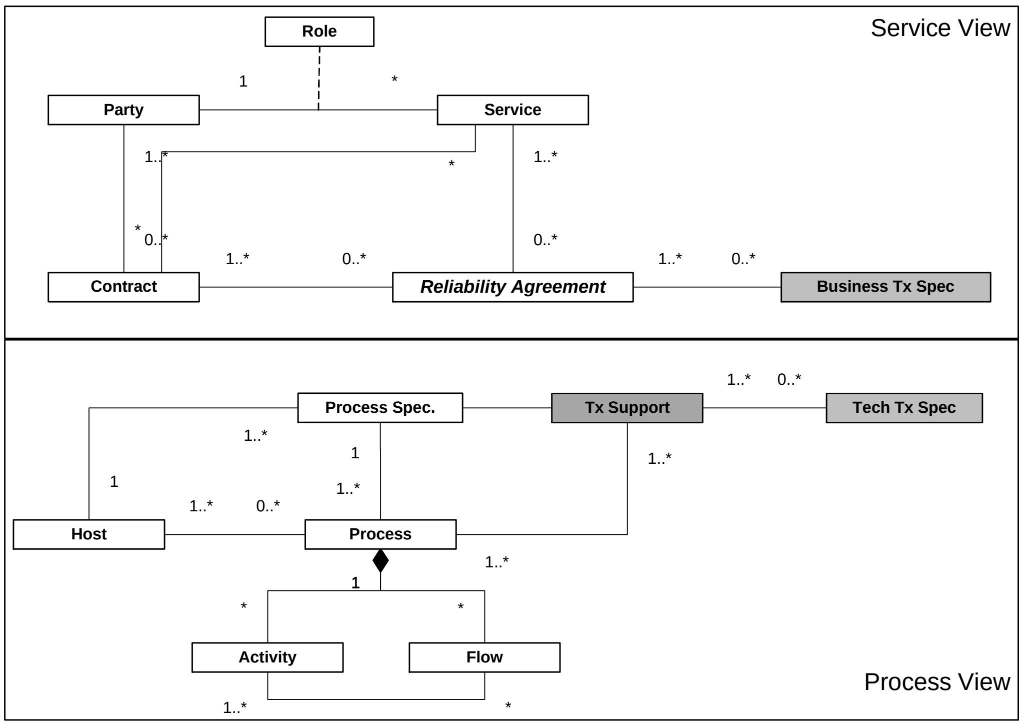


Figure 3.7: Balanced Entities and Relationships

terms for contractual purposes, and thus makes 'Reliability Agreement' explicit and adequate. To balance the two views, we propose an entity named 'Tech Tx Spec' in the process view in order to represent the technical specification of execution reliability in correspondence to business specification.

Following direction 2, the '***Reliability Agreement***' entity is balanced by an entity named 'Tx Support' (Transaction Support) in the process view, which represent the technical realization of the reliability agreement (i.e. transaction mechanisms).

The proposed entities promote the common understanding of the reliability (tx support) between the service providers and users. In similar cases (i.e. cases that are contract-driven, service-oriented, and include multiple parties), the handling of exceptions and errors can be agreed upon beforehand, so that the process execution is designed to be more robust and flexible. As we see in Figure 3.7, the entities 'Business Tx Spec' and 'Tech Tx Spec' are the root elements to achieve an unambiguous reliability agreement. The problem is such an agreement is lacking in the real-life case. As the result, we are inspired to look more carefully at these entities.

Lacking a transaction specification which is understandable by business hinders the feasibility of establishing such transactional agreement. Therefore, we are motivated to research a contractual approach, with a focus on the root entities ‘Business Tx Spec’ and ‘Tech Tx Spec’. We are motivated to research ‘Business Tx Spec’, as it facilitates establishing a ‘Reliability Agreement’ by parties in advance (i.e. before the process is executed). As a ‘mirrored’ entity of the ‘Business Tx Spec’, ‘Tech Tx Spec’ motivates us to design an approach to provide a desired transaction support which is understood by IT applications.

Furthermore, if looking at the ‘as-is’ scenario in Figure 3.4, the transaction support in this case is not change-friendly. In case of slight changes (e.g. when adding activities), the execution path (i.e. the ‘Flow’ entity) needs to be adjusted. As a linked entity, the transaction support is also subject to change. The business and technical transaction specification, reliability agreement, and contracts need to be changed accordingly. To summarize, we are inspired to do further research on ‘Business Tx Spec’ to find out a business-aware approach for contractual reliability between parties, and ‘Tech Tx Spec’ in search of a flexible mechanism that allows for comprehensive transaction support of processes in changing environment.

3.4 Conclusions

Transaction management has long been the way to guarantee execution for running applications. With the increasing complexity of business processes, transaction management is vital to address the reliability concern. However, in the business world, contracts are often related to the reliability concern. To investigate into the interdisciplinary area of process reliability, a case study has been developed.

In this chapter, we first introduced the case study and applied the dual view to describe and model it. Next, we identified the main elements of the case and listed the entities abstracted from these elements under the service and process view, respectively. Under the service view, the list shows the entities that help to analyze the business relationships. Under the process view, the list shows the entities that help transaction analysis. These entities can be observed from other cases, as long as the business processes showcase the characteristics like service-oriented, contract-driven and cross-organization.

Based on the findings from the asymmetric as-is diagram, we found out that the key to have complete and balanced views lies in a ‘Reliability Agreement’. One direction to pursue is to find out a realization approach to achieve the agreement, and another direction is to balance the views by entities from the process side. In addition, we also found out the as-is situation is not flexible to accommodate changes . Therefore, we proposed a more detailed and change-friendly to-be situation which is symmetric in two views. We see the gap between to-be and as-is as the research problem to address in the next phases of our research.

From the above discussions, we are inspired to address the problem of the gap by a business-aware approach that infuses transactional semantics into contracts or agreements. We also need to address the problem of a flexible transaction support that allows technical specification to change easily.

This raises two problems which need to be solved:

1. *How to specify the proper transaction support required for a service that is implemented by a complex process?* (Problem 1 focuses on the entity 'Business Tx Spec')
2. *How to ensure that these transaction specifications work in a changing environment?*(Problem 2 focuses on the entity 'Tech Tx Spec')

The first problem relates to research questions 3 and 4, as identified in Section 1.5, in the previous chapter. It concerns the approach for business transaction agreements, and is answered next in Chapter 4 and Chapter 5. The second problem relates to question 5, as identified in Chapter 1.5. It is about the mechanism for flexible transaction support, and is discussed in Chapter 6.

Chapter 4

TxQoS Approach

As identified in the e-advertising case study, the first research problem, 'How to specify proper transaction support required for a service?' in a business entity in the service view is addressed in this chapter. A contractual approach is proposed to bridge the gap between providers and users in terms of transaction awareness. FIAT (Fluency; Interferability; Alternation; and Transparency) attributes are introduced and designed to infuse transaction management semantics in contracts to promote the common understanding process/service execution reliability.

4.1 Introduction

As a result of the case study presented in the previous chapter, we have proposed a balanced and complete to-be diagram in order to address the missing entities in the as-is diagram. From this diagram, two problems have been identified, i.e. 1) How to specify proper transaction support required for a service, and 2)How to ensure these transaction specification work in a changing environment.

We focus on the entity 'Business Tx Spec' in this chapter. A TxQoS (Transactional Quality of Service) approach has been developed that addresses the first problem, by means of a specification of the transactional reliability. This is a contractual approach to enclose the business transaction specification into e-contracts. The TxQoS approach enables the interpretation of transactional reliability from a business perspective so that both parties can understand transactional reliability and agree on a reliability agreement.

The proposed approach conforms to the three-level framework developed in [27]. At the external level, transactional reliability should be expressed in a way that can be understood by all parties, no matter the business or technical perspectives they hold. This way, matching between the transaction requirements of users and offers of service providers takes place. Our proposed approach works on the external level with its root in the internal level where transaction management mechanisms are provided by the underlying systems, such as recovery, concurrency control, and compensation.

This chapter delivers an approach, designed to address the the first problem at a conceptual level, including the concept, scenario, life cycle, and specification method. The rest of the Chapter is organized as follows. Section 4.2 gives an overview of the proposed approach, and is illustrated using a scenario where all parties interact with each other by producing, exchanging, accessing, and evaluating data objects along the three-phased life cycle. We show the specification method in Section 4.3, where four attributes are developed to support the specification of transactional reliability. The chapter ends with conclusions in Section 4.4.

4.2 TxQoS Overview

We name the contractual approach as 'TxQoS' (Transactional Quality of Service), and define the term of TxQoS as the agreed reliability between a service provider and a service user with regard to the transaction performance of service execution. In other words, TxQoS is what a service provider promises to its users, based on its transactional capabilities, or a user requests from potential providers according to its transactional requirements of the regarding service execution reliability.

4.2.1 TxQoS Concept

The concept of TxQoS first appeared in [37], where the authors discuss the overall QoS support for Web services. These QoS properties which are required for the implementation of Web service applications include availability, accessibility, integrity, performance, reliability, regulatory, and security. The authors only mentioned the TxQoS term very briefly, without providing any further explanation of how it should be defined and used. We adopt the term with our own definition and furthermore, elaborate it into a contractual approach for transaction management.

When a contract is established for regulating service-oriented business processes, we call it a service contract. In our TxQoS approach, a service contract typically consists of a set of clauses plus enclosed SLAs (Service Level Agreements), which are agreed and signed by the two parties. These

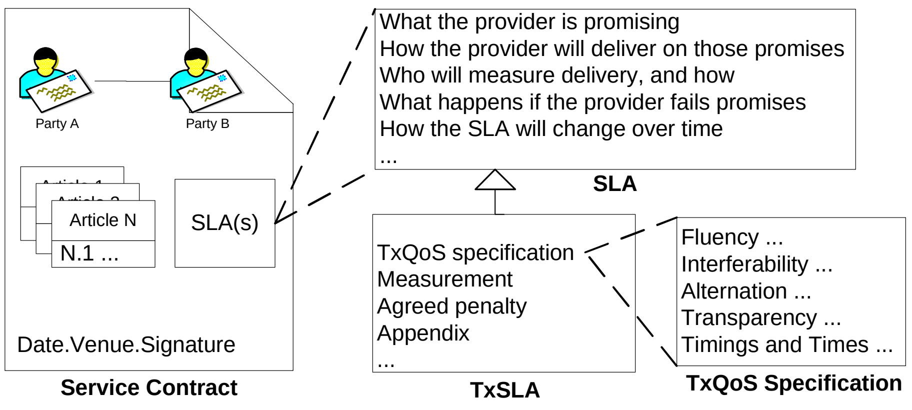


Figure 4.1: Contract structure

clauses are obligations and conditional rights, and are agreed for the purpose of exchanging value between the two parties. The enclosed SLAs in our setting state the non-functional agreements on the service qualities. An SLA specifies the promised qualities from the service providers and formalizes an agreement between the users regarding service availability, performance, measurement, and other aspects that enforce the operational qualities during service execution. When an SLA contains some specific components regarding TxQoS (e.g., TxQoS specification, measurement, penalty, and other appendices), we name it a TxSLA. Here a TxQoS specification can specify the TxQoS with differentiated parameters for the service users to choose from. Only when the TxQoS specifications are agreed by both the provider and user, TxSLAs are enclosed in a service contract and come into force.

Figure 4.1 depicts our proposed contract-SLA-TxSLA-TxQoS structure. In this figure, the example SLA contains the content from the SLA concept defined in [73]. The concepts of 'Service Contract', 'SLA', 'TxSLA', and 'TxQoS Specification' are the key elements of our proposed approach, which will be introduced in detail later.

From the contract structure shown in figure 4.1, we can see that the proposed service contracts can bridge the gap between the business and technical worlds by enclosing TxSLAs. The TxSLA contain a TxQoS specification to specify the agreed transactional qualities. When enforcing the proposed structure, contracts become a business tool that enforces execution reliability of a service and can meanwhile be mapped for technical transaction management mechanisms. In this way, the participating parties are aware of, and ensured of the transactional qualities of a service.

4.2.2 TxQoS Scenario

Following the TxQoS contract structure proposed above, in this section we provide an example that illustrates our approach. Figure 4.2 shows a basic scenario of the TxQoS. A full-fledged scenario based on this figure will be presented in Figure 4.5 at the end of this section. There are three parties involved, **provider**, **user**, and **intermediary(s)**, where the intermediary(s) is/are shown in a dashed box to indicate it is optional. There can be different intermediaries leveraged in our approach, namely 'Monitor', 'Advertiser', 'Reputation Registry', and 'Arbitrator'. Please note that this figure adopts the SOA scenario shown in Figure 2.1. The TxSLAs are established between the service providers and users. Then, services are invoked and delivered (i.e. executed) with the TxSLAs enforcing the agreed TxQoS specifications. We differentiate the communication arrows in the figure. The solid-end arrows are present in SOA, and the remaining arrows depict the communications specific for the TxQoS scenario.

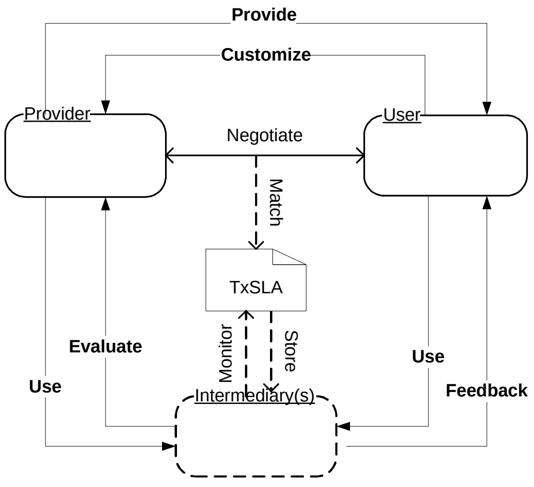


Figure 4.2: TxQoS Scenario

TxQoS provider A TxQoS provider offers differentiated TxQoS specifications for a particular service it publishes. Take a travel booking service for example, where the service can be associated with different levels of transactional reliability ensured by corresponding TxQoS specifications. One TxQoS specification offers a highly fluent service execution (e.g. always provide compensations for every step and no interruptions), which is specifically provided to its registered users. Another TxQoS specification offers standard service execution (e.g. time-out settings that may entail a new invocation), which is provided to random visitors.

TxQoS user A TxQoS user is a service invoker who intends to discover a suitable service with an appropriate level of transactional reliability according to its transactional requirements. Using the same travel booking example, a service user with a specific travel plan that has to be fixed within a certain time would have a number of TxQoS requirements.

TxQoS intermediary A TxQoS intermediary is a third party which monitors the ongoing service to collect and store TxQoS statistics, an advertising service provider which publishes TxQoS advertisement for potential users, a trusted reputation broker which records the past transactional performance of a list of services and may predict the future performance for users to check, or an arbitrator which settles a dispute when the agreed TxQoS is breached. An intermediary does not provide direct business functions in contrast to service providers. It has an auxiliary role to enable and facilitate TxQoS agreements. The enabling and facilitating tasks can be advertising, reputation repository, or post-service functions such as sending statistics and feedbacks of the actual TxQoS performance. In case a provider or a user does not trust each other, an intermediary (i.e. the monitor) is delegated to monitor the runtime TxQoS statistics. Otherwise, the monitoring task can be performed by the authorized party, so that a monitoring intermediary is skipped. Similar with the monitor, the other intermediaries (i.e. advertiser, reputation registry, or arbitrator) can be omitted from the scenario in case a provider and a user are tightly coupled and sufficient trust has been built up between them. These intermediaries can be used before, during, or after a TxSLA is established.

There are a number of data objects (e.g. messages, documents, or agreements) flowing between the parties along the TxQoS life cycle (which will be introduced in the next sub-section). Figure 4.3 shows (by party) the types of data objects, with a list of short descriptions below:

TxQoS Template is a base document used for the configuration, with the values of the TxQoS attributes to be assigned. A range of the values is given as a reference to provide differentiated offers.

TxQoS Offer is a configured TxQoS template with part or all the values of TxQoS attributes assigned by a service provider.

Requirement Template is a base document maintained by a service user to configure a TxQoS requirement document.

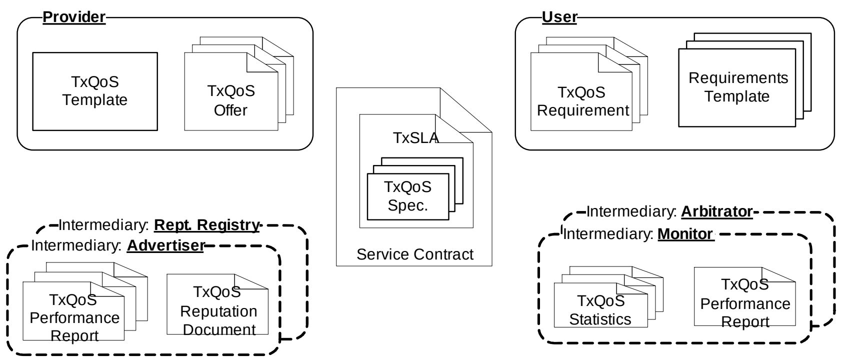


Figure 4.3: TxQoS Objects

TxQoS Requirement is a configured template with the required TxQoS values, and is used to search for a suitable offer.

TxSLA is an agreement for the transactional performance of a service established between a user and a provider. Inspired from the entity class 'Reliability Agreement' in Figure 3.7, a TxSLA is an object specially developed for the TxQoS approach.

TxQoS Specification is the main part of a TxSLA that specifies the TxQoS attributes and their values. This object is specially developed for the TxQoS approach, coming from the entity class 'Business Tx Specification' in Figure 3.7.

TxQoS Statistics is the runtime data for the transactional performance of a service. It is collected and recorded by the 'Monitor' intermediary to monitor the execution status, and can be used by the 'Arbitrator' intermediary for arbitrating purpose.

TxQoS Performance Report is the analysis of the transactional reliability of a service for the evaluation purpose based on the aggregation of the TxQoS statistics. It can be used by all types of intermediaries.

TxQoS Reputation Report is the aggregated analysis of the transactional reliability for similar services. It is used by 'Advertiser' and 'Reputation Registry' intermediaries for the evaluation purpose of multiple service providers based on the aggregation of the TxQoS performance reports.

Service Contract is an e-contract between a service provider and a user with regard to all the service related agreements. A service contract contains a TxSLA.

To reveal the relationships of these data objects, we model them by a class diagram in Figure 4.4. As we can see from the figure, a number of 'Transactional Specifications' are contained in a 'TxSLA', which in turn is part a 'Service Contract'. This structure conforms to the proposed contract structure shown in Figure 4.1. Note that there is no class named as 'Article' here, since articles specified in contracts are not relevant to our approach. In addition, we indicate the names of entities in the to-be situation in brackets to show the relationship of the TxQoS approach with the entities identified as the focus of our research. A TxSLA in our approach corresponds to the 'Reliability Agreement' entity, and a TxQoS specification corresponds to the 'Business Tx Specification' entity.

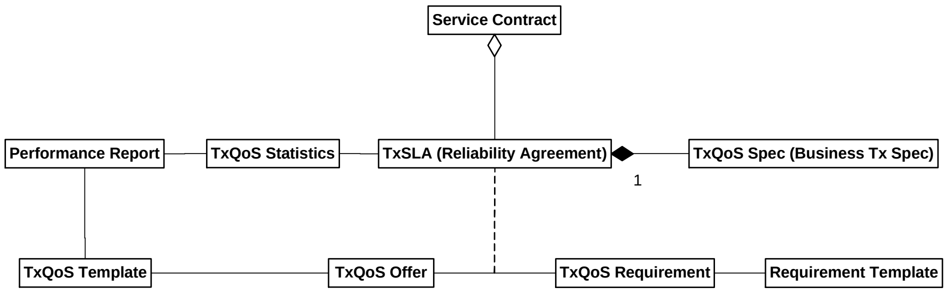


Figure 4.4: Class Diagram: TxQoS data object

4.2.3 TxQoS Life Cycle

Our approach is designed for the service oriented computing paradigm, where we assume that the functional aspects of services (e.g. publish, deliver, and leverage) are fulfilled with the support of SOA. In this section, we illustrate the life cycle of the TxQoS approach, which is designed in correlation to the SOA life cycle. A complete TxQoS life cycle comprises of 3 phases:

1. The **Design phase**, during which a service provider designs a TxQoS template of its service and offers one or more TxQoS specifications based on a template. The design of the template is realized by mapping from the internal-level transaction mechanisms to the external-level TxQoS specification. The differentiated offers are provided based on its past TxQoS performance and the analysis of users' requirements, so that various user requirements can be supported. A TxQoS offer can be pre-fixed, with every attribute configured by the provider, or user-configurable, with some attributes customized by a potential user. Also included in this phase is the option of a service user to design a TxQoS requirement template for a service according to its own reliability needs. Usually one requirement template is instantiated into one requirement document. This is in contrast to the provider side, where one TxQoS

template is instantiated into multiple offers. The reason for the difference is that a user usually has the same requirement every time it invokes a service, while one service is usually designed by a provider for multiple users to invoke.

2. The **Contract phase**, is the phase during which a TxQoS offer matches a TxQoS requirement document, so that a TxQoS agreement (named TxSLA) is established and enclosed in a service contract. A service user can use an intermediary to look up the transactional performance of a particular service. Afterwards, the user decides which service to invoke, and which TxQoS offer to take, if the service has differentiated TxQoS offers. Note that a TxQoS requirement document is usually not exactly met by any of the available offers. In this case, some negotiations are necessary. For pre-fixed TxQoS offers, a user may need to adjust its requirement and take one that fits best. The decision making for this adjustment is usually based on the weight of different TxQoS attributes (i.e. the least important requirement can be relaxed first). In cases where offers are user-configurable, an ideal offer is tailored to the user's requirement.
3. The **Evaluate phase** is where a TxSLA is monitored and managed by both parties or a monitor intermediary. This phase encompasses both the runtime (i.e. service execution period), during which real-time statistics is collected and recorded, and the post-execution period, during which the transactional performance of a service is evaluated. The statistics of multiple running instances of a service are aggregated by a monitor into a performance report, which describes the transactional reliability of that service for the purpose of evaluation. Performance reports of services are stored in a repository and updated from time to time. These reports are available for the other parties to access, so that potential users can search, monitor, and evaluate a service (via intermediaries) according to their transactional requirements. In addition, providers can adjust their TxQoS templates and differentiated offers based on the reports.

4.2.4 Summary

We have introduced the TxQoS approach, where a TxQoS specification is enclosed in a service contract to guarantee mutual understanding of transactional reliability between parties. We have proposed a scenario, where three parties interact with each other and a list of data objects at each side is described. Figure 4.5 gives an overview of our approach containing the parties,data objects, and their interactions.

A provider offers multiple TxQoS of a service to potential users. From available offers, a user can look up in the 'Performance Repository', to decide which one to take. Negotiations between a provider and a user can take place when no ready-to-agree TxQoS specification is suitable. After an offer and a requirement are matched, a TxSLA is agreed. Monitoring of the running transaction performance takes place by checking the compliance of the runtime statistics with the TxSLA enclosed in a service contract. The monitoring module is indispensable, and thus should be provided by at least one party or alternatively by an intermediary (shown in a dashed module at each party). The monitoring function is realized by accessing the 'Performance Repository' for 'TxQoS Performance Reports', generated on basis of runtime 'TxQoS Statistics'. The other intermediaries are also shown in dashed modules as they can be omitted from the scenario if the trust level between a provider and a user is high enough.

The data object 'TxSLA' is located in the central position, with extended relationships and objects surrounding it. This class refers to the 'Reliability Agreement' class, which we identified as the focus of our research in the case study in Chapter 3. In fact, the whole TxQoS approach we propose in this chapter is a result from research on the 'Reliability Agreement', as well as

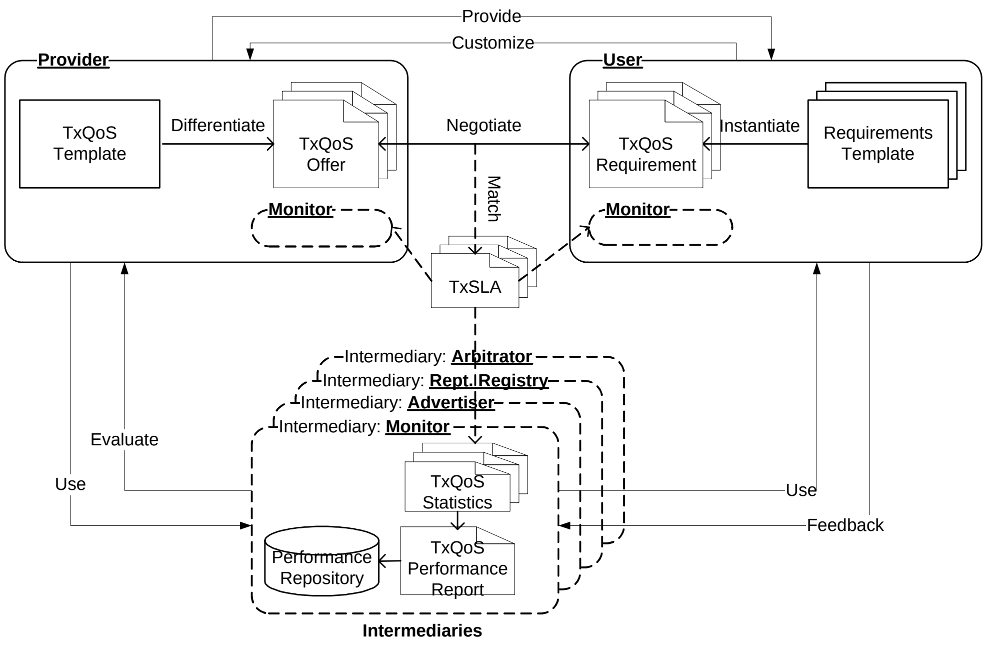


Figure 4.5: TxQoS Approach Overview

the proposed grey-box classes (e.g. 'Business Tx Spec'), which are used to balance the asymmetric dual view in Figure 3.7.

4.3 TxQoS Specification

A TxQoS terminology understood by both the technical and business world is essential to enable a common understanding of transactional reliability. In this section, we are going to develop a specification method, including a terminology for specifying transactional reliability in service contracts.

With a technical origin, the ACIDity can be extended into some transactional qualities that are more suitable for a business process or a service, e.g., payment atomicity in [46]. Taking the e-advertising case from Chapter 3 for example, 'Contracting' services hosted at all the parties should be atomic and consistent. The purpose of the TxQoS specification is to specify these qualities in a business-friendly language that can be enclosed in contracts.

At both provider and user sides, transactional reliability is usually guaranteed by the technical mechanisms listed below [23]:

Recovery, also called **forward recovery**, is a mechanism which fixes the exceptions and errors occurring halfway to ensure the activity or process continues to proceed until completion. The outcome of a recovery can be a restart or an alternative execution path. In the e-advertising case, for example, in case of insufficient impressions within the agreed time, the continuation of the advertising campaign can be seen as a recovery mechanism of the main process.

Concurrency Control is used to guarantee a consistent execution of applications when they operate simultaneously on the same data. For example, different instances of the same process may take place at the same time. Concurrency control is vital to keep the common objects safely updated when accessed by multiple process instances, if these instances take place one after another. If a design needs to be modified, the concurrency control ensures that people who access the design data do not interference with each other.

Compensation: Proposed in [20], compensation has been a very common transaction mechanism to guarantee forward execution by invoking a compensating activity when the original one fails. In the advertising case, imagine that the medium does not receive the requirements sent by the agency. To enable the process to continue, the failed 'Send requirements' activity performed by the agency needs to be compensated by an activity such as re-sending.

4.3.1 TxQoS Specification Attributes: FIAT

In the chapter discussing the case study, we have summarized the asymmetric as-is paradigm in Figure 3.4 regarding the transactional reliability, and found the problem of the missing business specification. We have proposed to add a few entities to transform the as-is into a complete and balanced to-be paradigm in Figure 3.7. The as-is entity 'Reliability Agreement' is what we aim to achieve by the TxQoS specification attributes. The to-be entity 'Business Tx Spec' is proposed to specify the transactional performance to be agreed in 'Reliability Agreement'. Interpreted in the TxQoS approach, the idea inspired from the case study is using TxQoS specification specifies the TxQoS in a TxSLA.

A TxQoS specification is necessary to promote common awareness and understanding of transactional reliability by both technical and business communities. When invoking a service, the users usually do not care about what techniques and mechanisms are applied by the service providers, and how these techniques and mechanisms work to enable transactional reliability. What the service users consider when they choose services are the qualities of the reliability the services can guarantee. In other words, service users care about the results instead of the technical details of transactional management. Our design of the TxQoS specification therefore comes from a user requirements analysis, i.e., what users expect for the reliably executed service and how they perceive transaction management. We express the expectations by attributes that are definable in contractual words and meanwhile measurable.

The requirement analysis of the specification attributes started with brainstorming sessions, with the process/service diagrams as a foundation and a number of 'user desires' as the output. These 'user desires' were then generalized, compared, and organized into attributes. At last, the attributes out of the brainstorming sessions were finalized into the TxQoS attributes that meet the following criteria: 1) A TxQoS attribute should reflect reliability at service level guaranteed by transaction management mechanisms; 2) A TxQoS attribute should be understandable by the business world; 3) A TxQoS attribute should be precisely specifiable and monitorable like other functional attributes (e.g. time, cost, capacity etc.); 4) The use of a TxQoS attribute should benefit both the service provider and the user.

The following transactional reliability attributes were defined: 'Fluency', 'Transparency', 'Interferability', and 'Alternation'. The attribute 'Fluency' represents the concern for customers on the smoothness of a service. The higher the fluency is, the less likely it is for the execution to go wrong, resulting in a higher user satisfaction as well as potentially raising cost. 'Transparency' specifies how much of the service execution details would be exposed to the outside. Many services, especially Web services, are black boxes from the user's perspective, from the viewpoint that they either execute successfully or not. However, in a case of close collaboration (e.g. the design service in the e-advertising case), service users concern themselves with the execution details of the services they invoke. Furthermore, the service users may wish to participate in the execution in case exceptions and errors occur. Therefore, we design the 'Interferability' attribute to express the limited user power in service execution that can be allowed and enabled by service providers. The attribute 'Alternation' is designed to provide an alternative execution path that is different from the execution path at the places where break-downs are likely to occur. The alternative paths are not routine, so they are not included into process/service design at the first point. This means these alternatives are like compensation mechanisms that specify the paths that can lead broken executions back into running till the end. We name the 'Fluency', 'Transparency', 'Interferability' and 'Alternation' the 'FIAT' attributes (an acronym with a slightly different order than in the presentation above).

As an element analyzed in the e-advertising case and presented in the previous Chapter, 'Time' is not directly related to transactional reliability but indispensable for a TxQoS specification. Depending on the execution time, the transaction requirement of the same service may differ. Generally speaking, the shorter the execution time is, the less chance the execution goes wrong. According to the users' reliability requirement of their local processes, the execution time can be a key factor to consider if a service executes reliably. Here we consider time as an important factor for a TxQoS specification but not a TxQoS attribute directly. Next, we explain the FIAT attributes one by one.

4.3.2 Fluency

The Fluency attribute gives an indication on the smoothness of service execution. To quantify the smoothness, we use the probability of 'breakdown' to define Fluency, so that the value of this attribute can be measured against the agreement.

Definition 1: *Fluency* is an attribute indicated in the form of a function or a numerical value that specifies the smoothness of the service execution by computing the probability of the *breakdowns* (see Definition 1.1) happening in the future based on the statistics (either from testing results or from past running data).

Definition 1.1: A *breakdown* is the ceasing of a running service so that the execution comes to a sudden end without delivering the intended results, and therefore requiring a fix to enable the execution to continue. We assume the distribution of breakdowns along service execution can always fit for a specific statistical model, which is widely adopted in the area of software reliability.

During service execution, a monitor keeps detecting errors and failures that prevent the ongoing execution to turn to undesired situations (e.g. dead loops and suspended process). The monitor counts the number of breakdowns and records their time stamps. After the execution, the smoothness indicator (e.g. a function or a numerical computation) is recalculated to represent the up-to-date status. The method to compute 'Fluency' is enclosed in Appendix A, where the GO NHPP model, which is widely used in software reliability research, is adopted to compute the function $f(n)$, which is the probability of having n breakdowns.

Following the method, $f(n)$ can be calculated given the testing statistics based on the functions. Then, the provider can publish multiple TxQoS offers with a precise 'Fluency' attribute to indicate the smoothness of service execution. For instance, $f(0)$ means the probability of having no breakdowns during service execution and can be used to specify the maximum fluency. Similarly, $f(1)$ means the probability to have at most 1 breakdown during the service execution. Usually in TxQoS offers, the provider can provide various fluency guarantees by means of fluency function values (i.e. $f(n)$). For example, assume the computation result shows that $f(0) = 0.8146, f(1) = 0.9235, f(2) = 0.9992$ during the tests, then the provider can confidently publish an offer where 'Fluency' is specified as '*we guarantee no more than 2 breakdowns during the execution*'. While in another offer, 'Fluency' is specified as '*we guarantee no breakdowns during the execution at the cost x*'.

An agreed TxSLA is monitored by a 'Monitor'. If, for example, in the case of $f(2) > 2$ (which means there are more than 2 breakdowns detected), the TxSLA is considered breached. As a consequence, the 'Penalty' clause in the TxSLA is enforced or an 'Arbitrator' intermediary is appointed to settle the dispute. After each execution, the newly-obtained data is collected, therefore recording the statistics, and updating a performance report. Such a real-time update enables the provider to offer the most appropriate TxQoS offers and equips the users with the up-to-date information to make a choice from available offers.

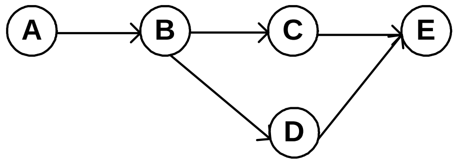
At design phase, 'Fluency' is specified by a service provider, and is based on its testing result of an unpublished service or past statistics of a published service. At runtime, 'Fluency' can be monitored by counting the number of breakdowns. If a service is canceled by a user, it is counted as an expected exception instead of a breakdown, and therefore belongs to the scope of 'Interferability' (see Section 4.3.5). While the monitoring of 'Fluency' is rather easy (by counting), the specification requires complicated calculation and prediction.

4.3.3 Alternation

The alternation attribute describes the predefined alternative execution paths in case of breakdowns happening along the service/process execution. Here the term 'breakdowns' means the same type of unexpected but fixable exceptions defined in Definition 1.1. During the design time, the service provider defines a set of execution graphs, including the main execution graph as well as alternative ones. At runtime, if a breakdown occurs, which deviates from the original execution path to an alternative path, a monitor detects the current path and compares it with the predefined set of graphs.

Definition 2: *Alternation* is an attribute indicating the allowed alternative execution paths when *breakdowns* occur along the ongoing path.

Every pre-defined alternative path can be specified using a graph. For example, imagine there is a service s consisting of activities A, B, C, D, E with the order of execution as



If we use $N = \{nodes\} \subseteq X$ to describe the domain of activities where X is the domain of nodes, and $E = \{edges\} = \{\langle node_m, node_n\rangle \in X \times X\}$ as the domain of edges, then $G = \langle N, E\rangle$ is the execution graph of the activities in the domain X. Correspondingly, in the above execution graph G_s is specified as

$$G_s = \langle\{A, B, C, D, E\}, \{\langle A, B\rangle, \langle B, C\rangle, \langle B, D\rangle, \langle C, E\rangle, \langle D, E\rangle\}\rangle$$

Here, '$\{A, B, C, D, E\}$' defines five nodes in the graph that represent the activities in this service that need to be executed, and '$\langle node_1, node_2\rangle(nodex \in \{A, B, C, D, E\})$' defines an edge of the graph that represents the execution path from '$node_1$' to '$node_2$'. In case a breakdown happens during the execution of the activity C or D, it is expected that an alternative activity P or Q is executed to compensate. Therefore, the alternative paths G_a can be specified as

$$G_{a1} = \langle\{A, B, P, D, E\}, \{\langle A, B\rangle, \langle B, P\rangle, \langle B, D\rangle, \langle P, E\rangle, \langle D, E\rangle\}\rangle$$

$$G_{a2} = \langle\{A, B, C, Q, E\}, \{\langle A, B\rangle, \langle B, C\rangle, \langle B, Q\rangle, \langle C, E\rangle, \langle Q, E\rangle\}\rangle$$
$$G_{a3} = \langle\{A, B, P, Q, E\}, \{\langle A, B\rangle, \langle B, P\rangle, \langle B, Q\rangle, \langle P, E\rangle, \langle Q, E\rangle\}\rangle$$

The above example indicates that three situations might occur: G_{a1} is chosen if C cannot be committed, so that P is adopted as an alternative node; G_{a2} is chosen if D cannot be committed, so that Q is adopted as an alternative node; G_{a3} is chosen if both C and D cannot be committed, so that P and Q are adopted respectively as the alternative nodes. This way, the predefined graphs representing the agreed execution path and alternative paths can be specified, which enables the later automatic monitoring on agreed Alternation specification. If, in this example, the monitor detects, during runtime, that the on-going path graph contains an unspecified edge '$\langle C, D\rangle$', a message is generated to notice such an error.

4.3.4 Transparency

This attribute describes the visibility of a service. There can be different ways to indicate the ability of users when viewing a service. This is the only attribute that needs to be specified at the design time but does not need runtime monitoring. At the design time, a service provider specifies a specific part of its service execution to expose to the outside, so that potential users can learn in advance what they are allowed to inspect. At runtime, there is no need for mechanisms to detect if a specified-as-transparent activity is actually transparent or not. When a possible dispute arises, we assume that the 'Monitor' (see Figure 4.5) keeps a log of the executing and executed service instances for this purpose. For example, only when a user notifies of a breach of the TxSLA; that an activity specified as transparent was not visible. In this case the log is needed to settle the dispute. Usually this is done by an 'Arbitrator' intermediary.

Definition 3: The *Transparency* attribute indicates the visible part of service execution by reflecting the proportion and/or the details of this visible part. This definition implies that when a (part of) the process is exposed, both the activities and the flows that direct these activities are visible by the service user.

We suggest two ways to specify transparency. In the first approach, it can be specified as the set of activities that are visible to the users at the external level of a process. We take the example from the previous section, where a process has five activities and two optional compensating activities. If the activities 'C, D, P, Q' are allowed to be visible, then these activities have to be included into the external-level process. We can specify the transparency of this example by the transparency set $N_{transparency}$ is specified as

$$N_{transparency} = \{C, D, P, Q\} \subseteq X \quad X = \{A, B, C, D, E, P, Q\}$$

Note that in this example, the flows are not included as there is no connection between the activities 'C, D, P, Q' in the execution path. In case that flows of visible activities exist, they can be specified as a subset of the execution graph shown in the 'Alternation' specification method. For instance, if the activities 'A, B' are allowed to be visible in the aforementioned example, then the transparency set $N_{transparency}$ can be specified as

$$N_{transparency} = \langle \{A, B\}, \{\langle A, B\rangle\}\rangle$$

This way of specification suits the scenario where users need to monitor and interfere with the service execution, like in most outsourcing paradigms.

Another way to specify the *Transparency* attribute is the proportion of visible activities among all activities. This way, this attribute can be defined as a percentage, or a number between the interval $[0, 1]$. Taking the above example, if activities 'C, D, P, Q' are allowed to be seen from the whole set of activities 'A, B, C, D, E, P, Q', the *Transparency* can be indicated as 57% or $\frac{4}{7}$. In this way, the flows between the visible activities are not specifically identified and lost. Therefore, this type of *Transparency* specification only gives users a general view of the visibility of the service. This approach suits the scenario where users are not interested in the details of service execution better.

4.3.5 Interferability

Interferability is an attribute that describes the control allowed from the user side upon a service invocation. During the design time, the service provider decides to expose part of its service to the users and lets them interfere with the execution. During runtime, users can issue some commands that are agreed beforehand to control (part) of the service execution [3]. Interferability is especially suitable in outsourcing scenarios where a different level of control is necessary [25]. Note that a user only has interferability to the activities that are specified as transparent, which means a user must be able to see things before being allowed to take any action.

Definition 4: *Interferability* is an attribute indicating the user control by means of issuing operational commands over the execution that are authorized by the provider.

This attribute can be interpreted as the set of commands from the users to intervene an activity (viewed as a node in execution path), plus the allowed timings to issue these commands. The below set contains the system functions that can be used to specify interferability:

$$I = \{operation(time, activity)\} \tag{4.1}$$

$$operation \in O_t \quad activity \in N_t \quad time \in \{Rules\} \tag{4.2}$$

Where O_t is the set of possible operations on certain activities such as cancel, revise, and rollback. N_t here is the transparency set where transparent activities are specified as a node in the set. *Rules* is the timing rule set of *time*, where *times* mean the timing an *activity* is allowed to perform. Timing rules can be specified in two ways. One adopts absolute machine times, such as '*before 10:00*', another adopts relative times, which invoke timing functions to indicate the semantics such as '*after the execution of activity N, but before the execution of activity M*'. Suppose a service s consists of four transparent nodes 'C, D, P, Q', where D allows user to cancel it after it has started but before the next node E starts. In this case, 'Interferability' can be specified as:

$$C = \{cancel(getProgress(D) = 0 \cap getProgress(E) = -1, D)\} \tag{4.3}$$

The above function $getProgress(N)$ invokes the system parameters to detect the execution status of the node $N \in X$. The parameters are set as '*-1=not started, 1=committed, 0=started but not committed*'. There can be other scales to indicate the execution status; here we just provide a numerical scale as an example. In a TxSLA, this example is stated as a rule '*the user has the right to cancel the activity D at any time during its execution*'. During runtime, when the monitor detects a cancel command from the user at the time when activity E has started, it judges the command as an invalid one and generates a warning. Thus, the monitor can filter every user command and pass it to the underlying system when it conforms with the TxSLA, and block the invalid commands.

4.3.6 Discussion of FIAT

As the key part of a TxSLA, we have developed four TxQoS attributes for specification and monitoring purposes: *Fluency*, *Interferability*, *Transparency*, and *Alternation*. A causal-effect diagram in Figure 4.6 shows the factors that affect TxQoS attributes and the dependencies between these attributes. These causes are identified under three categories: non-system, system, and TxQoS. The 'non-system' category contains the factors that belong

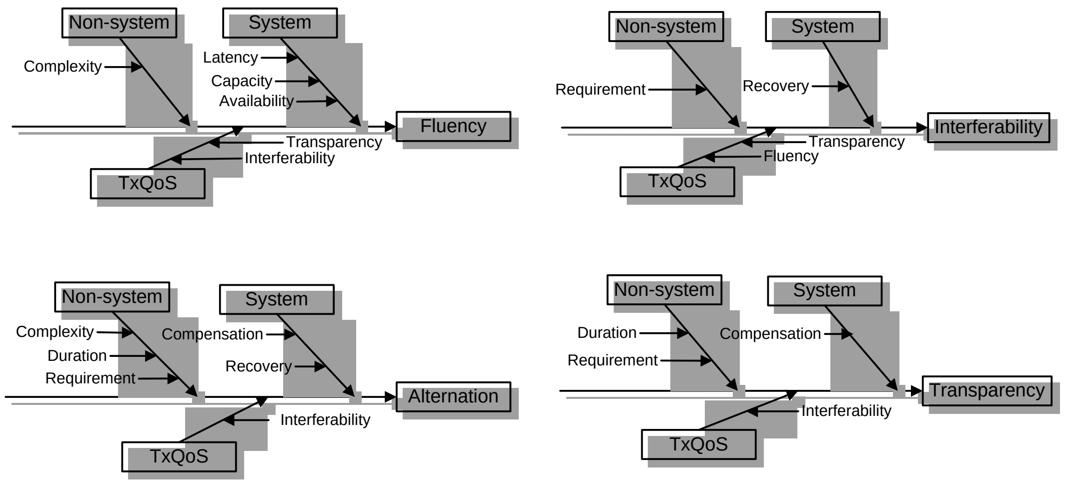


Figure 4.6: FIAT: Cause and Effect

to the business domain, while the ‘system’ category contains the factors from the technical domain. The ‘TxQoS’ category reveals the dependencies among the TxQoS attributes. For example, we can see that the *Fluency* attribute is dependent on the complexity of the service, i.e. the more complex a service, the lower its fluency can be. *Fluency* also relates to the system constraints, such as latency time due to network traffic, capacity due to the bandwidth, and availability due to the hardware reliability. Meanwhile, *Fluency* can be affected by *Transparency* and *Interferability*, and vice versa.

By means of the FIAT attributes, we aim to specify transactional reliability during design time (i.e. the ‘design’ phase in the TxQoS life cycle), so that they can be monitored at runtime and updated after the service execution (i.e. the ‘evaluate’ phase in the life cycle). A FIAT specification is the first and foremost concept in our contractual approach.

4.4 Conclusions

We have proposed the conceptual approach to address the first research problem identified in the previous chapter – How to specify proper transaction support required for a service? As inspired from the case study of on-line advertising, this chapters starts the business-oriented design with a focus on the entity ‘Business Tx Spec’, which in our approach corresponds to a method using FIAT attributes for contract specification. In this chapter, we design the conceptual TxQoS approach. An architecture, a contracting model, and a monitoring mechanism will be proposed in the next chapter as the supporting elements to implement the approach.

The FIAT attributes (*Fluency, Interferability, Alternation, and Transparency*) enable the specification of transactional reliability in a TxSLA. A TxSLA can be enclosed in a service contract and is understandable by both the technical and business worlds. Because of the FIAT specification, a TxSLA can be monitored and evaluated along the TxQoS life cycle.

These four attributes come from the transactional requirements of the user side and cover the need for exception and error handling. *Fluency* measures the robustness of service execution. *Interferability* indicates the extent of control from the outside during execution. *Alternation* represents the possible choices when encountering problems. *Transparency* reflects the interest of a service user on the internal process at the provider side. Applying FIAT through different stages of a service life cycle (i.e. design, publish, discovery, execution, evaluation) enhances transactional reliability by empowering users with more knowledge to choose reliable services. A framework to operationalize the TxQoS approach is needed to ensure things work as visioned. This part of the design will be described in the next chapter.

Chapter 5

TxQoS Framework

Following the answer to the question 'how to specify transactional reliability' by means of introducing a contractual approach at a conceptual level, we continue in this chapter to answer the second question: how to ensure such an approach at an operational level. A reference architecture, a contracting model, and a monitoring mechanism are proposed. These three components work together as the TxQoS framework, which deepens the understanding of the TxQoS approach and can be used as an implementation suggestion in practice.

5.1 Introduction

We have elicited the general existing problem of the lack of agreements on transactional reliability from a case study in Chapter 3. In Chapter 4, we have proposed the specification of transactional reliability by means of the FIAT attributes, which provides the conceptual solution to address the problem. In this Chapter, we provide the operational solution to address the problem. A TxQoS framework is developed to ensure a specified transactional reliability, which consists of three compartments: a reference architecture, a contracting model including matching and mapping, and a monitoring mechanism geared into the TxQoS phases.

According to [2], a software reference architecture is a generic architecture for a class of information systems that is used as a foundation for the design of concrete architectures from this class. For our purpose of providing an operational level solution to support the TxQoS approach, we designed a reference architecture based on a functional consideration. The architecture illustrates the modules required at each party to enable our approach. These functional modules are placed on top of business process engines and e-contracting systems. A contracting model that describes how the approach works across different process layers and organizational boundaries is developed to complement the understanding of the scenario. This model describes both the horizontal matching of the TxQoS documents between the involved organizations, and the vertical mapping from the bottom level transaction mechanisms to the upper level of TxQoS documents for external discovery and matching. A monitoring mechanism explains how the specified TxQoS agreements are monitored and evaluated. This monitoring mechanism, together with the reference architecture and the contracting model, address the operational level of the approach. We call this operationalized TxQoS approach as a TxQoS framework.

As mentioned in Chapter 4, a TxQoS life cycle comprises three phases: a 'design phase', a 'contract phase', and an 'evaluate phase'. The TxQoS framework supports the TxQoS approach along all three phases. With the in-depth investigation of various aspects, we aim to deliver a TxQoS approach that bridges the gap of transactional awareness between the technology and business communities, so as to enhance execution reliability for contract-driven, service-oriented business processes.

In the next sections, we first present the reference architecture in Section 5.2. Then in Section 5.3, we describe how the agreements on transactional reliability is realized across organizational boundaries and process layers by a contracting model. Next, the monitoring mechanisms of the TxQoS approach is explained in Section 5.4. We conclude the chapter in Section 5.5.

5.2 TxQoS Reference Architecture

As the size and complexity of software systems increased, designing and specifying the overall system structure emerged beyond the algorithms and data structures of the computation, which is referred as the architecture level of design [21]. Proposed in [52], an architecture of a specific system is a collection of computational components, connectors, constraints composition, containers, and configurations. The architecture design is a blueprint work to facilitate the understanding of a system and enable its later implementation. In this section, we design a reference architecture as a suggestion for the TxQoS approach, according to the overview scenario depicted in Figure 4.5.

5.2.1 Functional Requirements

An architecture should address all functions of a system. A reference architecture serves as a template for similar architectural designs. The design of our TxQoS reference architecture makes use of the typical architecture styles. According to [21], an architecture's style determines the vocabulary of components and connectors, together with constraints on how they can be combined. There are numerous styles for architectural design, such as 'Pipes and Filters', 'Layered Systems', and 'Repositories'. Each of them has its own structural pattern and specific advantages and disadvantages. These architecture styles and patterns can be combined, which is typically the case for most systems.

To design a TxQoS reference architecture, we first identify the functional requirements a TxQoS system must satisfy. Such functions include specifying, matching, agreeing, and managing the TxQoS-enclosed contracts, and are realized by the modules hosted at each party. Upon an analysis of the involved parties, we identify the functions necessary for the three phases of the TxQoS lifecycle:

- The **Design phase** is where both service providers and users design their TxQoS documents. In this phase, the templates are defined and configured into TxQoS offers or requirement documents. Therefore, functions of *publishing*, *definition*, and *configuration* are necessary for service providers and users. Meanwhile, intermediary functions (e.g. *Registry*) may be needed as well.
- The **Contract phase** is where a TxSLA is agreed and enclosed in an e-contract. In this phase, a *contracting* tool is necessary for both providers and users to establish such a contract. A contracting tool at the user side searches, matches, and negotiates with another contracting tool at the provider side to reach an agreement. Also the contracting tools are responsible to manage the established TxSLAs. Intermediary functions like *advertising* can help to facilitate the contracting process.
- The **Evaluate phase** is where which the execution, according to a TxSLA, is monitored and managed. This phase encompasses both the runtime (i.e. service execution period), during which real-time statistics are collected and recorded, and the post-execution period, during which the transactional performance of a service is evaluated. Therefore, a *monitoring* tool is needed. Such a tool can be hosted by either party or by an intermediary (in this case is called a 'Monitor'). Functions like *arbitrating* may be needed when disputes arise.

Besides the above functional requirements for the architecture design, *repository* tools are necessary throughout the whole lifecycle to gather TxQoS documents for service providers and users to access. Such tools are usually provided by an independent intermediary. In addition, a functional module *managing* all the above-mentioned tools and *coordinating* the related procedures serves as the foundation for the TxQoS architecture.

5.2.2 Architecture Design

In the previous subsection, we listed the functional requirements: publishing, definition, configuration, contracting, monitoring, managing, and coordinating. When designing a reference architecture, we also take some non-functional properties into account, such as security and reusability. For service providers and users, we suggest the equivalent modules, thus forming a symmetric architecture. Together with the modules from the intermediaries, we design the TxQoS architecture within the SOA paradigm, considering the context of our approach is service-oriented and contract-driven.

Internally, at the provider or user side, we adopt a 'Heterogeneous' design, which is a combination of 'Layered Systems' and 'Event-based, Implicit Invocation' architecture patterns [21]. The 'Layered Systems' pattern allows a partition of the complex TxQoS approach into incremental steps and supports enhancement which are reusable with the TxQoS layer on top of the process layer. The 'Event-based, Implicit Invocation' pattern is widely used for integrating tools and ensuring consistency constraints in applications where components and modules interact with each other through implicit invocations. With the similar benefit of the 'Layered Systems' pattern, the 'Event-based, Implicit Invocation' pattern also provides strong support for reuse.

An architecture of the TxQoS framework is shown in Figure 5.1, where a service provider or user integrates a business process engine for process control, and an e-contracting system for contract management. A 'TxQoS Manager' is placed on top of the basic layer, where a business process engine resides. The 'TxQoS Manager' lays the foundation for the tools (i.e. 'Definition', 'Publishing', 'Configuration', 'Contracting', and 'Monitoring' tools), which are built on top of it. It is the essential component; it is in charge of all

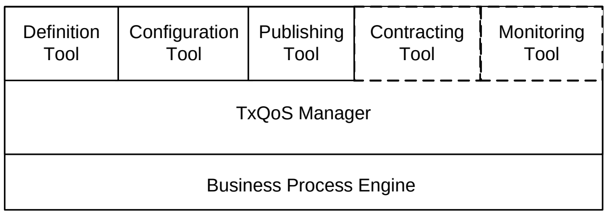


Figure 5.1: TxQoS Architecture: Provider/User

TxQoS management issues, such as coordinating the derivative tools and interacting with other parties. As e-contracting systems are not widely present in organizations, they do not appear in our architecture design. In case an e-contracting is not present, the 'Contracting Tool' is not needed anymore. In addition, the 'Monitoring Tool' is optional since the monitoring function can be performed by a 'Monitor' intermediary, instead of a provider/user itself, as mentioned in Section 4.2.4. Therefore, 'Contracting' and 'Monitoring' tools are presented in dashed boxes.

Besides service providers and users, there are (optional) intermediaries (i.e. 'Monitor', 'Advertiser', 'Reputation Registry', and 'Arbitrator') involved in the overview scenario of the TxQoS approach. Figure 5.2 shows that they are organized. These intermediaries provide auxiliary functions to facilitate the TxQoS activities along the life cycle and are considered trusted external parties. An 'Advertiser' advertises TxQoS offers in a indexed way so that users can easily find and choose services according their transactional requirements. An 'Arbitrator' is present when any dispute arises between the providers and users. A 'Reputation Registry' keeps the relevant ranking or indexing statistics on services from various providers, so that potential users can approve a provider. A 'Monitor' is a module for runtime compliance checking of the TxQoS specification according to the agreed TxSLA. The TxQoS performance repository is kept here for other parties to access.

Figure 5.1 and Figure 5.2 depict the architectures of service providers, users and intermediaries. When applied in the SOA paradigm, we get the

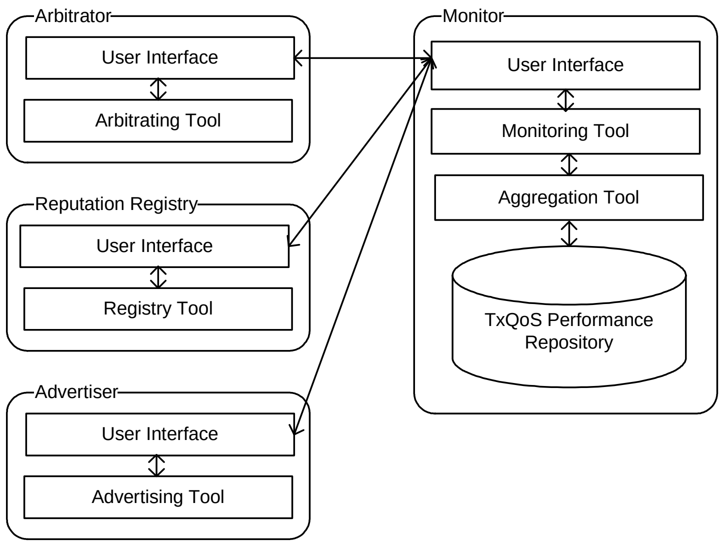


Figure 5.2: TxQoS Architecture: Intermediaries

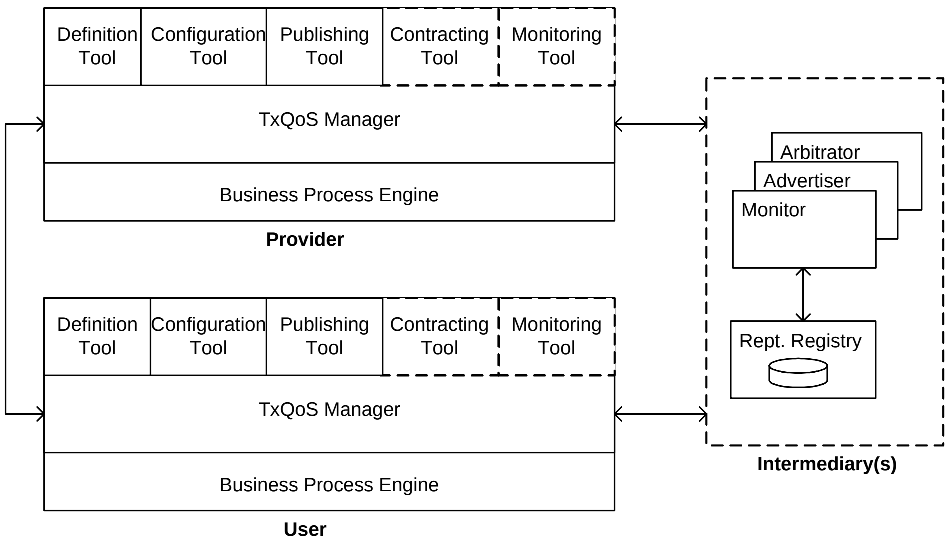


Figure 5.3: TxQoS Reference Architecture

TxQoS reference architecture in Figure 5.3. The dashed boxes indicate the optional components. This architecture can be used for understanding how the TxQoS approach works, as well as a suggestion for the implementation. Note that the TxSLA-enclosed contracts are preserved, either in the contracting tool or the e-contracting system at the provider/user side. As mentioned, due to the absence of an e-contracting system in most organizations, it is not depicted in our diagram. Interested readers are referred to [1] for a detailed e-contracting reference architecture design.

5.3 Contracting Model

Among the three phases in a TxQoS life cycle, the 'Design' and 'Contract' phases include a lot of interactions directly between service providers and users, which we see as relationships at the external level. Meanwhile, within each party, many cross-layer activities take place internally, which we see as the vertical relationships across the three levels (i.e. internal, conceptual and external levels conforming to [27]). The vertical relationships are developed during the 'Design phase' to produce TxQoS data (i.e. offers and requirement documents), and the horizontal relationships are developed during the 'Contract phase' to reach a TxSLA. We have depicted the relationship in Figure 4.2 in Section 4.2.2. In this section we investigate how a TxSLA is established from the perspective of two orthogonal dimensions.

To reach a mutually satisfactory agreement on transactional reliability, it takes place in three steps. For this we propose a contracting model shown in Figure 5.4 for mapping and matching. First, the provider maps the transactional mechanisms of its internal level to the conceptual-level transactional properties, and then interprets these properties into a TxQoS template, which can be configured into multiple offers, catering various needs at the external level. Second, the user figures out its TxQoS requirement document instantiated from its requirement template. This requirement template is based on the process reliability analysis at the conceptual level, which is rooted in the internal level process specification. Third, the user discovers a suitable service by matching its transactional requirement document to a provider's TxQoS offer, which is usually facilitated by an advertising intermediary. Note that the first step and the second step are independent and can take place in parallel.

From the provider's side, it offers, at the external level, multiple TxQoS offers of its service. This way a service user can choose a service provider with the most suitable TxQoS offer, based on its TxQoS requirement. There are mappings at each side between the internal level and the conceptual level, and between the conceptual level and the external level. With the mappings, the transaction properties (e.g. ACIDity) are translated into the external-level TxQoS terminology. At the external level, there is a match of the TxQoS documents, which results in an agreed TxSLA between the provider and the user. In the contract-driven service-oriented processes, the agreed TxSLA is included into a service contract and ensured by the contracting systems.

The TxQoS mapping (vertically) between process levels and matching (horizontally) between the service user and provider result in a successful fulfillment of the 'Design' and 'Contract' phase. For the mappings, the rela-

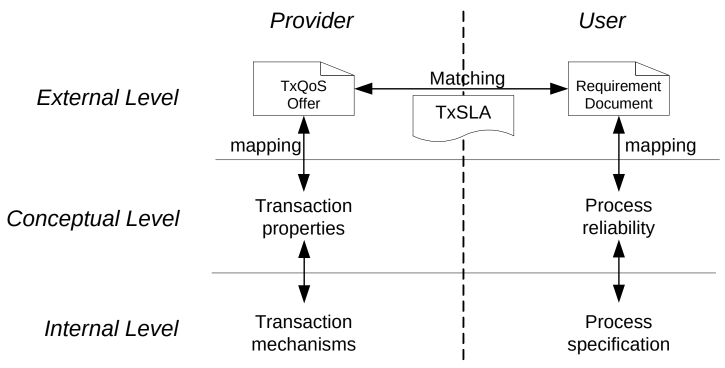


Figure 5.4: TxQoS Contracting Model

tionships between transactional properties in technical terms (i.e. ACIDity) and the TxQoS in business terms (i.e. FIAT attributes) are indicated in Table 5.1:

	Fluency	Interferability	Alternation	Transparency
Atomicity	↑	↓	X	↓
Consistency	X	↓	↓	X
Isolation	↑	↓	X	↑
Durability	X	X	↓	X

Table 5.1: Mapping Indication

Here, ↑ indicates a positive correlation between a technical property and a FIAT attribute and ↓ indicates a negative correlation. The mark 'X' indicates few or unknown correlations. For instance, if the execution time of a service is very short, this service is often supported by the transaction property of 'Atomicity'. This 'Atomicity' has a positive correlation with the 'Fluency' attribute (i.e., high atomicity $\longrightarrow$ high fluency) as an atomic transaction is short-lived and has less chance of getting wrong during the execution. The 'Atomicity' has a negative correlation with the 'Interferability' attribute (i.e., atomic service execution means no interferability from the user side). Note that Table 5.1 only gives a very rough indication of how to map the internal level transactional properties to the external level TxQoS documents.

After mapping, the essential step leading up to the establishment of a service contract is the match of a user's requirement with the most suitable offer from a provider's multiple TxQoS offers at the external level. The matching basically takes place among the FIAT specifications of the two sides. Afterwards, a TxSLA is reached and enclosed in a contract. How a contract is managed is out of the scope of the TxQoS approach as we assume that the e-contracting system supports the automation of contract fulfillment.

5.4 Monitoring

After the 'Contract' phase, a TxSLA is established. Consequently, the monitoring loop is initiated throughout the phases. As the architecture implies, a monitor, which monitors the agreed TxSLA at runtime, is indispensable for a complete TxQoS framework, although it may reside at any party. For transaction management properties (e.g. ACIDity), various transaction monitoring technologies and tools are already available (e.g. database benchmark tools). In this section, we explain how monitoring mechanism works in the TxQoS life cycle. Here, we assume that for any service, the FIAT attributes have been specified in the TxQoS offers. This means that the values agreed on the TxQoS attributes in the TxSLA come into force from the time a service provider publishes a service until the end of the service life cycle (i.e. the time a provider updates or closes the service).

Figure 5.5 shows the monitoring mechanism throughout the TxQoS life cycle. The objects to be monitored are the running instances of a particular service. A 'Monitoring Tool' or an intermediary 'Monitor' is delegated to detect if there are any inconformities with the agreed TxSLAs during runtime. Before publishing the service, the provider usually estimates the future possible performance according to its testing and probability computation. After publishing, the design template of the TxQoS offers at the provider side can be adjusted, based on the TxQoS performance report aggregated by the runtime statistics of all monitored service instances in the past (or between any certain time intervals). The accumulated runtime monitoring data of the TxQoS attributes for multiple service instances are aggregated and recorded into a TxQoS performance report, and all the TxQoS performance reports are stored in a repository, which can be accessed by the provider and the users, and/or other intermediary(s). A TxQoS performance report can be updated upon the execution of one or multiple service instances.

A monitoring loop starts from the 'Contract' phase, goes through the 'Evaluate' phase, and finishes in the 'Design' phase. Below we summarize the monitoring related steps (e.g. predict, specify, monitor, and evaluate) throughout the TxQoS life cycle:

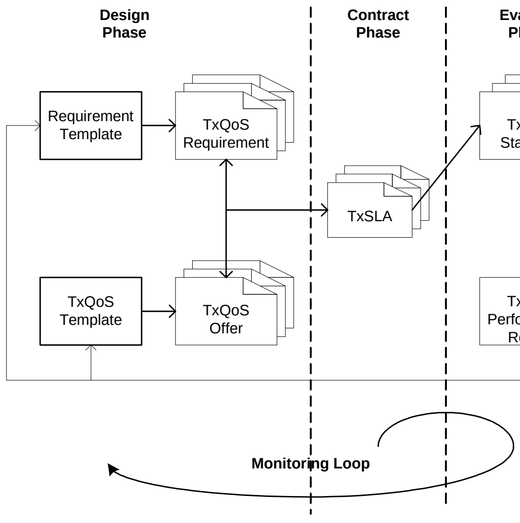


Figure 5.5: TxQoS monitoring

Design phase A provider designs a service and runs some tests before publishing it. Based on the testing data, a TxQoS offer template is designed. The information on fluency, such as the maximum, the minimum, the average fluency value, and the way to compute the function $f(p)$ etc., is used to predict the future performance. For the other attributes, the testing data can also be used to instantiate various offers (i.e. the provider is confident to guarantee the offers). Meanwhile, a user who looks for a suitable service also designs its requirement template and configures it into a requirement document for potential matching. The monitoring mechanism can results in feedbacks for both the provider and user to adjust their documents.

Contract phase A service user chooses one of the TxQoS offers based on its requirement document. A TxSLA is then agreed and enclosed in a service contract. Negotiation may take place if no exact match exists, for instance for the attributes of 'transparency' and 'interferability', which define the control a user has for a service. The desired values for these two attributes differ from one user to another, and are often open for negotiation. A monitoring loop starts from this phase after a TxSLA is agreed.

Evaluate phase The service starts to run, and a monitor who is authorized by both parties surveys the ongoing TxQoS statistics and compares them with the TxSLA. After the execution of this service instance, the runtime statistics are recorded and the performance report is updated as the latest evaluation. The updated report is then stored in a repository for future access. The service provider may accesses the report repository to adjust its TxQoS offer template and offers. The user may also wish to retrieve the up-to-date TxQoS performance report from the repository for decision-making. If there is any dispute, the runtime statistics in the report are used by the intermediary(s) as authorized evidence to check its conformance with the agreed TxSLA.

On the one hand, the monitoring mechanism enables the provider to offer TxQoS guarantees more confidently. On the other hand, users can compare different service offers with objective references from an authorized monitor for a best match, and protect their rights by means of a TxSLA agreed before service execution. If any dispute arises during or after the service execution, the monitor can provide the relevant statistics for a fair settlement.

5.5 Conclusions

In this Chapter, we have investigated the operational aspect of the TxQoS approach. A framework consisting of an architecture, a contracting model and a monitoring mechanism are developed to ensure the conceptual approach. Fist, a reference architecture is developed under functional consideration. Modules are needed to guarantee the necessary functions, such as publishing, definition, configuration, contracting, etc. In the architecture, these tools are placed on top of business process engines. Second, a contracting model is developed to describe the horizontal matching of the TxQoS documents that crosses the organizational boundaries, and the vertical mapping from the bottom level transaction mechanisms to the upper level of TxQoS documents for external discovery and matching. Third, the monitoring mechanism explains how TxQoS specifications are monitored throughout the TxQoS life cycle.

We have developed a TxQoS approach to bring together the reliability concern from both technical and business worlds that have long been separately addressed. The key of the TxQoS approach, a specification method via FIAT, completes the missing part of the picture (see Figure 3.4) and towards a balanced vision shown in Figure 3.7. So far, the business-oriented design, which resulted in the TxQoS approach, has been proposed and explained. In the next chapter, we perform technology-oriented design, focusing on the ‘Tech Tx Spec’ entity to address the problem of ‘how to ensure these transaction specifications work in a changing environment’.

Chapter 6

Abstract Transaction Construct

Business processes have become increasingly complex. However, no single transaction model is comprehensive enough to accommodate the various transactional properties demanded by these processes. In this chapter, we introduce and demonstrate the concept of the Abstract Transactional Construct (ATC) to address this problem. ATCs are abstractions of transaction models which can be selected, configured, and deployed in a service-oriented transaction framework according to process specifications. The reusable and extensible templates, with parameters to indicate structure, position, and ACIDity, enable the ATC composition to construct a flexible and comprehensive business transaction framework. A complex process can be decomposed into several subprocess (i.e. process chunks), which are each in need of a different transaction support in order to guarantee execution reliability. However, existing transaction models are not comprehensive enough to accommodate all these transactional needs. Therefore, with the unique features of generalization and abstraction of transactional qualities from the existing transaction models, ATCs allow for design time specification and runtime instantiation to support various transactional needs of the complex processes.

6.1 Introduction

In Chapter 3, we studied the entities and relationships of an on-line advertising case. A 'to-be' mapping of the dual view has been proposed in Figure 3.7, in contrast to the as-is dual view diagram. Based on the discussions on the entity of 'Reliability Agreement' and 'Business Tx Support' in the service view, we have proposed the TxQoS approach in Chapters 4 and 5. The essence of this approach is to specify transactional reliability in business-level terminology via FIAT attributes. In this chapter, we present

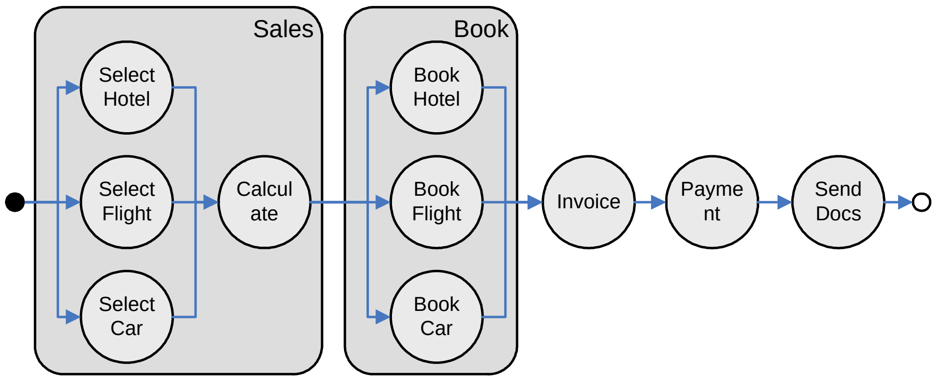


Figure 6.1: A Travel Booking Example

our research result on the entities mirrored to ‘Reliability Agreement’ and ‘Business Tx Support’ – ‘Tx Support’ and ‘Tech Tx Spec’ in the process view shown in Figure 3.7.

This chapter presents the result from the technology-oriented design, while the previous chapters (4 and 5) presented the result from the business-oriented design. As introduced in the research process in Chapter 1, these two branches of design activities took place in parallel. So the technology-oriented design is not the consequence or follow-up, but a complement of the business-oriented TxQoS approach.

First, we take a booking process of a travel agency (see Figure 6.1) which consists of a series of activities. The administrative activities take place in the agency, while three Web services (i.e. hotel booking, car rental, and flight booking) are executed elsewhere and are invoked by the agency. These services are composed in parallel and can be invoked in any order, according to the requirements of the clients. In this case, each Web service demands ‘atomic’ Web service transaction support, as the clients do not need to know how these services are implemented and where they are located. Besides the three parallel-invoked services, there are sequential activities in this process, such as billing, payment, etc., which are executed one by one. These sequential steps demand a ‘chained’ transaction support so that the execution does not need to roll back to the very beginning once an activity fails to execute. With regard to the transaction support at a global level, the entire process requires the ‘rollback’ mechanism, in case a client cancels the booking before the execution of the ‘payment’ activity. In an alternative case, if the process is canceled after the ‘payment’ activity, the process can not be rolled back to the exact same state to when it started, as the customer is fined for the late cancelation. Thus, a transaction mechanism of ‘compensation’ is needed to invoke compensating activities to 'undo' the entire process. The simple example gives an indication that transaction management in service-oriented processes can be a complex problem to tackle, as different mechanisms are required for different parts of a process. Furthermore, there can exist different levels of transaction scopes (e.g. activity level, service level, and process level), which can demand alternative transactional qualities.

As demonstrated by the example above, a multi-level complex business process requires robust execution in every process chunk. Here, a process chunk mean an arbitrary composition of activities that form a part of the process, which is sometimes referred to as a 'sub-process'. When these chunks join together, the execution needs to be smooth at the global level, in spite of heterogeneous underlying systems and application domains. As the example shows, the different parts of a process, which can be called process chunks or sub-processes, demand different transaction support and exhibit different transaction properties. For instance, the sequentially executed process chunk demands for transaction support suitable for long-living processes (i.e. chained transactions). As a result, this part of the process exhibits the transactional properties of non-atomicity and non-isolation (i.e. the intermediate results are saved for roll back purpose), while the process chunk consisting of parallel activities demands for transaction support suitable for distribute nodes (i.e. nested transactions) that releases the ACIDity of isolation.

As the building blocks of the Business Transaction Framework (BTF), Abstract Transaction Constructs (ATCs) play an essential role to achieve both comprehensiveness and flexibility [66]. The basic idea of an ATC is to analyze the existing transaction models and abstract the commonalities in terms of transactional properties, so that they can be utilized at design time by hiding their implementation details. Next, we demonstrate the concept of ATCs and explain their working mechanisms. ATCs provide the foundation for the BTF design and solve the problem generalized from the case study in Chapter 3, by introducing extra entities to move the 'as-is' towards the 'to-be' scenario. We are going to answer the following two questions one by one in this chapter – 1) What are ATCs? and 2) How do they work?

The rest of the chapter is organized as follows. First, we answer the 'What' question by introducing the concept of ATC, the ATC library, and explain the ATC features in Section 6.2. Furthermore, we define four types of ATCs, each with a parameterizable interface for later configuration and deployment. Second, we further explain how ATCs are designed, selected, configured, and deployed in Section 6.3, in order to answer the 'How' question. Section 6.4 concludes this chapter.

6.2 What are ATCs?

Business applications over distributed infrastructures are developed and deployed as reusable services that can be composed into a global process, as if they were local components. These function-as-local component services are composed in a plug-and-play manner using existing common standards (e.g. Web service protocols). The change of service implementation details does not affect the design. At the same time, the change of the process logic does not necessarily require the change of service implementation. The separation of the concern of the process design and service implementation render the services, each providing a specific function, to form multi-functional processes in a flexible and reusable way.

A complex business process demands execution reliability at each level of the composition (i.e. sub-process and/or component service). This requirement for a transaction management that offers various transaction qualities, is beyond what the existing transaction models can do. For example, in the service-oriented paradigm, Web service transactions have been proposed to meet the demands from service choreography/orchestration/composition. However, web services cannot stand for all types of services in a broader context. Thus, inspired by the benefit of reusability in a service-oriented paradigm, we propose reusable transactional building blocks, which are to be used for building complex transactions in order to address flexibility and comprehensiveness demanded by today's complex processes. These transactional building blocks are named Transaction Constructs (TCs). The general templates used for designing these constructs are named Abstract Transaction Constructs (ATCs).

6.2.1 ATC Concept

ATCs are design-time transaction constructs in the form of general templates, with parameters that allow for specification in run-time transaction constructs. They are organized and stored in a repository (ATC Library). The idea of the ATC approach is to develop reusable building blocks for the BTF that provides on-demand transaction qualities. Such on-demand transaction design requires the ATCs to be self-contained entities, that are ready to be selected and composed in a plug-and-play way. In other words, ATCs are used, and can be reused, as transaction schema design constructs. Proper ATCs can be selected to compose a transaction schema according to a specific business process at design time, and configured accordingly at configuration time, in order for the schema to be instantiated into an executable transactional process chunk at runtime. This way, a number of ATCs can be flexibly composed into a multi-level transaction scheme that guarantees reliability along the process execution in a best-effort manner.

To design the ATC concept in detail, we investigate the existing transaction models with the aim to generalize and abstract their commonalities. We classify and abstract various existing transaction models into three classes. A hierarchy presented in Figure 6.2 shows the classification [70]: The Flat Transaction Model (FTxM) with a flat structure, the Choreographed Transaction Model (CTxM) with a sequence and a complex structure, and the Nested Transaction Model (NTxM).

We identify these three classes according to their structure. Later we will introduce (in Section 6.2.2) the four ATC structures specified as: flat, sequence, complex, and tree-like. Here, the sequence and complex types are derived from the Choreographed group in the classification. For example, the X Transaction model proposed in [60] is a workflow transaction model, which leverages the idea of compensation originated from Sagas (and Saga ATC is a sequence type of ATC). In the classification according to Figure 6.2, the X model is under ESTxM (extended Saga transaction model) within the CTxM group. However, if it is defined in the template, the X model is specified as having a 'complex type' of structure.

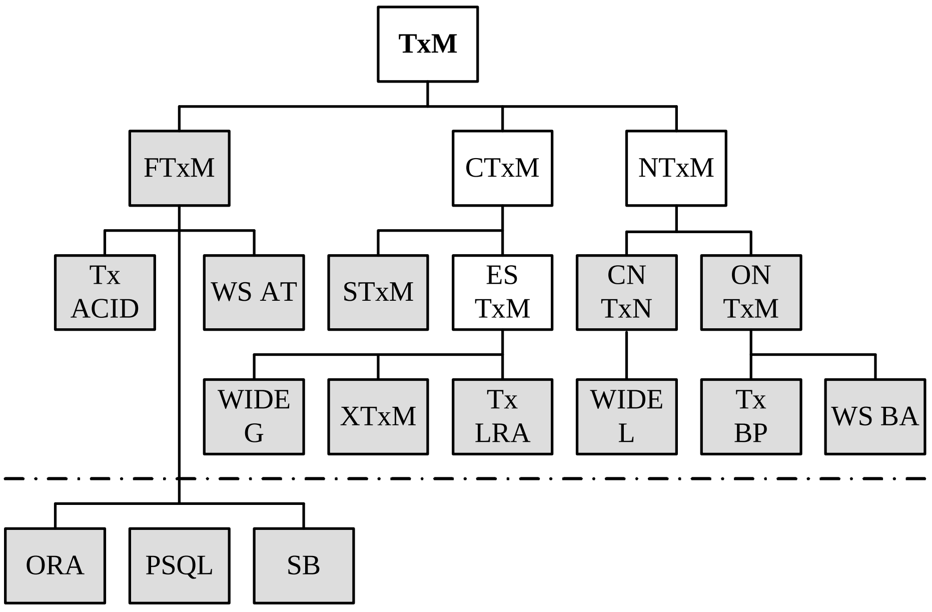


Figure 6.2: The ATC hierarchy

Figure 6.2 shows the transaction models, which are indicated by white and gray boxes. Models with gray boxes are specific enough to be executed (and which are thus usable at least in theory and which may or may not have an implementation) and white boxes represent transaction models that are only abstract and super-type like, i.e., *non-executable*. In other words, the gray boxes indicate the real existing transaction models and the white boxes are the generalization of transaction models, i.e. there is no existing transaction model called a 'Choreographed Transaction'.

The most abstract transaction model here is the empty transaction model called TxM (i.e., transaction model), which defines common properties of a transaction model, like the name, structural couplings, and transaction management events. For an example, please refer to [14, 18]. Below the most abstract model we can identify and classify three main groups of transaction models:

- The flat transaction model (*FTxM*)
- The choreographed transaction model (*CTxM*)
- The nested transaction model (*NTxM*)

The flat transaction model is the oldest, most simple and most widely used transaction model. It adheres to the ACID properties and has extensions, for instance the two phase commit protocol (2PC) in case of distributed flat transactions. In a service-oriented environment, the flat transaction model varies from, for example, *WS Atomic Transactions* (*WS AT*) to WS-CAF's *ACID Transactions* (*Tx ACID*).

The choreographed transaction models are used in environments that require long-lived transactions. In such cases, a choreographed transaction decomposes a long running transaction into small, (sequentially-executing) sub-transactions. This model can be further specialized, for example into the saga or extended saga transaction models. The extended saga transaction model can be decomposed further into workflow transaction models, like the *WIDE Global Transaction Support* (*WGTxM*) [61] and *X-Transaction* (*XTxM*) [60]. The *Long Running Action* (*Tx LRA*) model from WS-CAF also fits into this category.

The nested transaction model adopts a top-down method to decompose a complex transaction into child transactions according to the application semantics. Nested transactions overcome the shortcomings of flat transactions by permitting parts of a transaction to fail, without necessarily aborting the entire transaction. This model can be further classified into specializations, like the open nested and the closed nested transaction model, which can again be further specialized into, for example, the *WIDE Local Transactions model* (*WLTxM*) [6]. Examples of open-nested transaction models in a service-oriented environment are the *WS Business Activity* (*WS BA*) and the *Business Process Transaction Model* (*Tx BA*) from WS-CAF.

All existing transaction models can be classified into these three main categories of transaction models. Recently, we have seen new developments on transaction models in the realm of workflows, web services, and grid-computing, which have been identified as special types of the three main classes of transaction models.

Adhering to the principles of OMG's model driven architecture (MDA) approach [44], our hierarchy also provides support for platform specific transaction model implementations, for example Oracle (*ORA*) or Sybase (*SB*). Figure 6.2 shows a clear distinction between platform specific and platform independent models. Platform specific models are leafs below the dotted line that are ready to be used, and have a system which implements the higher level transaction model. Consider, as an example, a business transaction that intends to use a flat transaction model without specifying the concrete platform specific implementation which should execute this transaction. The BTF can decide then at run-time the concrete platform specific model.

ATCs are not a new type of transaction model. Instead, they are rooted in and abstracted from the existing transaction models. The ATCs and transaction models are both designed to meet specific transactional requirements. An ATC is designed based on the abstraction of a transaction model, which means a transaction model needs to be there before a corresponding ATC can be designed. In other words, ATCs come after the existing transaction models. It is configured and instantiated with process knowledge at configuration time.

To summarize, an ATC is a template designed after an existing transaction model for the construction of a business transaction schema. We propose this concept to address the flexibility requirement imposed by the second research problem.

6.2.2 ATC Features

In the above section, we have introduced the concept of an ATC. In this section, we explore the features of the ATC which are used for the specification of an ATC.

An ATC has a structure.

After an investigation of the existing transaction models as reviewed in Chapter 2, we have identified three classes of transactions according to their structures: Flat, Choreographed, and Nested. The choreographed transactions can be distinguished further as the chained transactions or complex workflow transactions. In total, the ATCs can specify four structure types, as shown in Figure 6.3: Flat, Sequence, Complex, and Tree.

Among the four types of ATCs, the basic type is the flat ATC, which

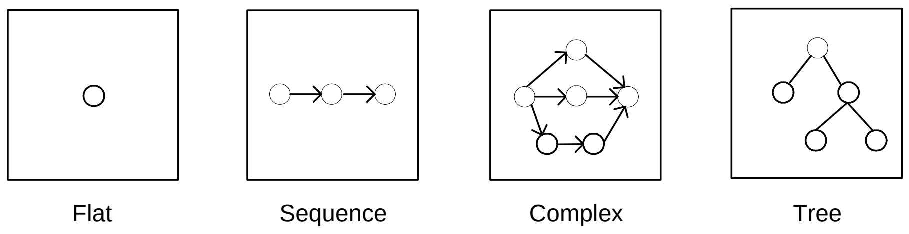


Figure 6.3: ATC Structure

is a template for specifying a flat transaction. We consider the traditional ACID transaction as a special case with a flat structure, which actually has no structure internally. Accordingly, we have the sequence type and complex type of ATCs, which specify transactions supporting chain-like sequential activities or complex-structured workflows. Here, a complex type of ATC has the internal structure of the mixed arbitrary sequences and parallels, which corresponds to some complex workflow transaction models. The nested transaction models can be specified by the type of ATC, with the internal structure like a tree. Please note that the nodes of a tree have parent-child relationships, and there is no control flow between the nodes. This means the siblings do not form a chain, which is different from the complex type where the children nodes connect one by one to form a chain.

When abstracting existing transaction models with various structures, we opted for a semantic abstraction. This means we only consider the logistics of the composition, e.g. splits, joints and flows, and the transaction properties exhibited, e.g. ACIDity. As a result, we do not consider their implementation details, such as the underlying transaction processing engines or any constraint from the application environment. With this semantic abstraction, ATCs can literally compose into any hierarchy of transaction schema exhibiting the needed transaction properties for supporting of similarly structured business processes.

ATCs can be composed in a recursive manner.

In a service-oriented business process, we can view the whole process as a complex business transaction. The sub-process (process chunks) can be viewed as sub-transactions. Every sub-transaction is a component in the business transaction and connects with each other horizontally or vertically. This way, the whole business transaction can be decomposed into several levels of processes. From the service view, this process hierarchy can be seen as a service composition.

We propose a recursive transaction composition in correspondence to the process composition. In our design, the ATC composition is very similar to

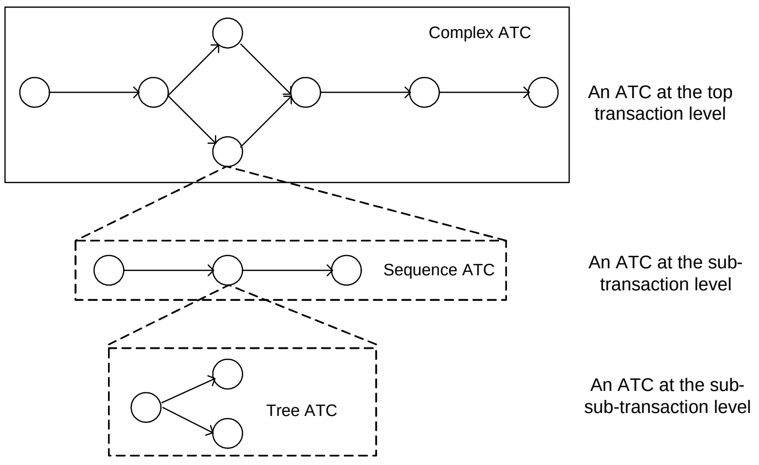

Complex ATC
An ATC at the top transaction level
Sequence ATC
An ATC at the sub-transaction level
Tree ATC
An ATC at the sub-sub-transaction level

Figure 6.4: Example of ATC Recursion

the process composition. One ATC can be decomposed into sub-ATCs, and a sub-ATC may be further decomposed into some ATCs as needed. When viewed by multiple levels, a top-level ATC can be decomposed into several lower-level ATCs, and the decomposition can go further into multi-levels if necessary.

Accordingly, a business transaction can be abstracted as an ATC, while each sub-transaction can be abstracted as a sub-ATC, which can be further decomposed if necessary. As shown in Figure 6.4, one sub-ATC (the circle dot with the expanded dashed box below) within the top-level ATC can be instantiated into transaction constructs that represent another process (in the center dashed box). This process has a chain-like structure, and thus can be supported by a chain transaction specified by a sequence type of ATC. If going further with this sequence ATC, one sub-ATC in the chain can be assigned as a tree ATC (the circle dot with the expanded dashed box at the bottom). This multi-level view of an ATC recursion allows a comprehensive transaction scheme to be specified at the design time that supports any particular business process, which may involve a lot of distributed sub-processes and activities.

An ATC specifies general transaction properties. Besides the features of having a structure and the ability to be composed recursively, an ATC has to specify the transaction properties abstracted from the transaction models, as these transaction properties form the foundation of the transaction support. In other words, transaction support for a business process means that certain transaction properties are needed to guarantee a smooth process execution.

From the investigation of the transaction management history in Chapter 2, ACIDity (i.e. Atomicity, Consistency, Isolation, and Durability) is what we know as the general transaction properties that a transaction either guarantees or not. For example, the original database transaction (flat transaction) guarantees ACIDity and hence is also called ACID transaction, while the chain transactions release the atomicity and isolation. So an ATC needs to specify the ACIDity to indicate what transaction properties are guaranteed by the transaction instantiated from it. This is done at the configuration phase upon the specific process knowledge, and by a composer who designs the transaction schema composed of ATCs.

In the previous chapters, we have proposed transactional qualities to be specified by the Transactional Quality of Service (TxQoS). In a service-oriented environment, TxQoS clauses are specified as contracts and enclosed in service agreements (i.e. TxSLAs). Table 5.1 in Section 5.3 gives an indication of the mapping between the ACIDity and the TxQoS. This mapping of the 'Tech Tx Spec' (ATC) and the 'Business Tx Spec' (TxQoS) can be traced back to Figure 3.7, where we propose a balanced service/process view to address the research problem. With the feature of transaction properties specification, recursively composed ATCs allow the instantiated transactions to execute with required transactional qualities.

6.2.3 ATC Representation

We have introduced the ATC concept as a configurable template. ATCs are templates abstracted from existing transaction models. A template has to be general and extensible (what we call as 'abstract'), where the configuration of the parameters are left for later, once the knowledge of the particular process and application environment is obtained. At the pre-design time, a template is written with most of the parameters empty for later configuration.

In this section, we are going to develop an ATC representation to specify an ATC in the form of a template. The main part of an ATC template is the specification of the three features explored in last section: Structure, Composition, and Transaction Properties. Next, we introduce the way to specify them one by one in a template.

The first part of an ATC template describes the internal structure of an ATC, which can only be one of the four values, either flat, sequence, tree, or the complex type. Note that a flat ATC (instantiated into an ACID transaction) has no value to put in within the structure section. In a general Saga example shown at the end of this section, the number of nodes in the sequence needs to be set. The second part indicates the position of an ATC in the ATC schema, which describes how it is connected with the other ATCs in an hierarchy, i.e. parents, children, and siblings (i.e. ATCs in front of or/and after it). The third part defines the transaction properties that an instantiated ATC conforms. These parameters are set 'true' or 'false'. For instance, a flat ATC will have all ACIDity set 'true' later on in the configuration. The last part is the transactional mechanism, which is important to know for the ATC that have applied transactional mechanisms. For instance, the savepoint and compensation parameters need to be configured in a saga ATC. Note that not all ATCs specify these transaction mechanisms, which means the existence of this part depends on the type of an ATC. In summary, the first three parts are the compulsory sections for specifying all ATCs, and the last part is optional.

We illustrate the basic idea of ATC specification by using a flat ATC template for example. The first part of the parameters specify the internal structure, which is flat. The second part of parameters specify the ATC recursion information, such as the inbound and outbound ATCs. This part shows the position of the ATC in the global ATC schema. The third part of the parameters specify the transactional properties to show the conformity of the ACIDity. The last (optional) part of the parameters specify if this flat ATC is the savepoint in the global ATC schema and/or if it is a compensation node. Below we use XML as the language to write a flat ATC template to illustrate the 'parameterizalble' feature. The values of the parameters such as 'ID', 'NumOfParent', 'Atomicity' and 'savepoint' will be determined later, when the ATC composer gets the process knowledge.

```
<ATC class="ACID"> <!-- This is an ACID ATC template>
   <parameter name="structure">
       <value>structure=flat</value>
   </parameter>
   <parameter name="position">
       <valueInput>
           name="ID" type=string:s
           name="NumOfParent" type=integer
           name="NumOfChild" type=integer
           name="NumOfFront" type=integer
           name="NumOfAfter" type=integer
           name="parent" type=atcIdType <!-- This is an ID for ATC>
           name="child" type=atcIdType
           name="front" type=atcIdType
           name="after" type=atcIdType
       </valueInput>
   </parameter>
   <parameter name="ACIDity">
       <valueInput>
           name="Atomicity" type=bool
           name="Consistency" type=bool
```

```
            name="Isolation" type=bool
            name="Durability" type=bool
        </valueInput>
    </parameter>
    <parameter name="Mechanism"> <!-- Optional part.>
        <valueInput>
            name="savepoint" type=bool
        </valueInput>
        <valueInput>
            name="compensation" type=bool
        </valueInput>
    </parameter>
</ATC>
```

The first part of structure parameters can be set during the selection phase, and the second part of parameters relevant to the position can be set during the configuration phase. The third part of ACIDity parameters can be set in either phase, as long as a TxQoS specification is ready. The fourth part of parameters relevant to the transaction mechanisms may be set during the deployment phase, where the integration with systems and other applications need to be considered. The structure of the templates is defined by the XSD (XML Schema Definition). Below we provide the main part of the XSD:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <xsd:element name="Parameter"
                     minOccurs="3"
                     maxOccurs="unbounded">
           <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="Structure" type="xsd:structureType">
                <xsd:element name="Position" type="xsd:complexType"/>
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="Parent"
                                         minOccurs="0"
                                         maxOccurs="unbounded"
                                         type=xsd:atcIdType>
                            <xsd:element name="Child" type=xsd:atcIdType>
                                         minOccurs="0"
                                         maxOccurs="unbounded"
                                         type=xsd:atcIdType>
                            <xsd:element name="Front" type=xsd:atcIdType>
                                         minOccurs="0"
                                         maxOccurs="unbounded"
                                         type=xsd:atcIdType>
                            <xsd:element name="After" type=xsd:atcIdType>
                                         minOccurs="0"
```

```
                                        maxOccurs="unbounded"
                                        type=xsd:atcIdType>
                         </xsd:sequence>
                     </xsd:complexType>
                     <xsd:simpleType name="atcId">
                        <xsd:restriction base="xsd:string">
                          <xsd:length value="6">
                       </xsd:restriction>
                 </xsd:simpleType>
                 </xsd:element>
                 <xsd:element name="Mechanism" type="xsd:complexType"/>
                     <xsd:complexType>
                         <xsd:sequence>
                             <xsd:element name="Savepoint" type=xsd:bool>
                             <xsd:element name="Compensation" type=xsd:bool>
                         </xsd:sequence>
                     </xsd:complexType>
                 </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
            <xsd:simpleType name="structureType">
               <xsd:restriction base="xsd:string">
                 <xsd:length value="20">
                 <xsd:enumeration value="flat"/>
                  <xsd:enumeration value="sequence"/>
                  <xsd:enumeration value="complex"/>
                  <xsd:enumeration value="tree"/>
               </xsd:restriction>
        </xsd:simpleType>
            <xsd:simpleType name="ACIDity">
            <xsd:restriction base="xsd:string">
                 <xsd:enumeration value="atomicity"/>
                 <xsd:enumeration value="consistency"/>
                  <xsd:enumeration value="isolation"/>
                  <xsd:enumeration value="durability"/>
            </xsd:restriction>
            </xsd:simpleType>
</xsd:schema>
```

If comparing ATCs with traditional transaction models, we see that they are both designed to meet specific transactional requirements. For instance, a Saga ATC is designed to be instantiated at runtime for the chain-like process, whereas a Saga transaction model was invented for long-living chained applications, such as updating a series of banking accounts in a row. They meet the same requirements, such as the compensation mechanism which enables a continuation from the execution breaking point if exceptions arise. ATCs and existing transaction models have mapping relations, e.g. a Saga ATC can be instantiated into, and only into, Saga transaction models. An ATC is designed based on the abstraction of a transaction model, which means a transaction model needs to be there before a corresponding ATC can be designed. However, an ATC is designed as a generic XML template, without considering a specific application context defining where it will be used in. Moreover, it can only be configured and instantiated with process knowledge, which means it needs process specifications at the configuration time. For example, a saga transaction model is designed for a chain-like application environment, while a Saga ATC is designed with no limit to the application environment. Only when the application environment has a process or process chunk (i.e. sub-process) demanding for compensation, a Saga ATC is selected.

According to what we have designed, an ATC is a general template where parameters still need to be specified. The XML examples are used to illustrate how ATCs look like according to our description. Please note that the topic of the ATC specification language is beyond the scope of this thesis. We develop the XML template to demonstrate the ATC concept can be realized and there can be other languages or formalism to realize our idea. Below we give an example of a simple Saga ATC template written according to the above schema. Through the parameters, a composer can assign the corresponding values based on the process specification. So next to the 'unparameterized' general Saga template, we also give a configured Saga specification, which has an additional section of the compensation mechanism configured as true.

```
<ATC class="SAGA"> <!-- This is a Saga ATC template>
   <parameter name="Structure">
       <value>structure=sequence</value>
       <valueInput>
           name="NumOfNode" type=integer:nn
       </valueInput>
   </parameter>
   <parameter name="Recursion">
       <valueInput>
           name="ID" type=string:s
           name="NumOfParent" type=integer
           name="NumOfChild" type=integer
           name="NumOfFront" type=integer
           name="NumOfAfter" type=integer
           name="parent" type=atcIdType
           name="child" type=atcIdType
           name="front" type=atcIdType
           name="after" type=atcIdType
       </valueInput>
   </parameter>
   <parameter name="ACIDity">
```

```
        <valueInput>
            name="Atomicity" type=bool
            name="Consistency" type=bool
            name="Isolation" type=bool
            name="Durability" type=bool
        </valueInput>
    </parameter>
    <parameter name="Mechanism"> <!-- Optional part.>
        <valueInput>
            name="savepoint" type=bool
        </valueInput>
        <valueInput>
            name="compensation" type=bool
        </valueInput>
    </parameter>
</ATC>
```

A configured Saga ATC:

```
<?xml version="1.0" ?>
<Saga xmlns="http://www.btf_atc.org" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.btf_atc.org/atc.xsd">
<ATC>
   <Structrue name=Sequence>
        <NumOfNode>10</NumofNode>
   </structure>
   <Position ID=sa003>
        <NumOfParent>0</NumOfParent>
        <NumOfChild>0</NumOfChild>
        <NumOfFont>1</NumOfFront>
        <NumOfAfter>1</NumOfAfter>
        <Parent></Parent>
        <Child></Child>
        <Front>sa002</Front>
        <After>sa004<After>
   </Position>
   <ACIDity>
        <Atomicity>F</Atomicity>
        <Consistency>T<Consistency>
        <Isolation>F<Isolation>
        <Durability>T<Durability>
  </TxQoS>
   <Mechanism>
        <Savepoint>{nodes|1,4,10}</Savepoint>
        <Compensation>T</Compensation>
  </Mechanism>
</ATC>
```

6.3 How Do ATCs Work?

In the previous section, we have introduced the concept of and described the features of the ATC to answer the question 'What are ATCs'. Next, we further explain the mechanisms to explain how ATCs are used in practice.

6.3.1 ATC Lifecycle

In Section 4.2.3, we have introduced the TxQoS lifecycle from the 'Design Phase', through the 'Contract Phase', to the 'Evaluate Phase'. If mapping this lifecycle, the design of the ATC templates actually takes place prior to the design phase of the TxQoS life cycle. An ATC starts its life cycle before any process knowledge, from which a transaction schema is composed. So herein we design a separate ATC lifecycle to illustrate.

In Figure 6.5, we show how ATCs transit from the general templates at the beginning until the ready-to-run transaction schema at the end. There are four phases in the cycle. The 1st phase is the pre-design time, during which a series of general ATC templates are designed and stored in a library. Pre-design here means this phase takes place before there is any requirement imposed by a specific process. In order to avoid confusion with the design time in a common sense, we name it as 'Pre-design Phase'. The outcome of this phase is a series of general ATC templates, which form a hierarchy stored in the 'ATC Library'. The 2nd phase starts from a process requiring some transaction support, during which proper ATC templates are selected from the ATC library. Selection is based on the information, regarding process structure and composition. The outcome of this phase is the properly selected ATCs that can be composed to roughly reflect the process transaction requirements. The 3rd phase is the period the selected ATC templates are configured, e.g. proper parameters are set. The last phase is the deployment time, during which the configured ATCs are deployed in the running environment. A number of factors need to be under consideration, such as the application domain, the integration with existing systems, etc. The outcome of the last phase is an exact and precise transaction schema supporting the process (in this diagram is the process appears since Phase 2). Note that the composition takes place from the 2nd phase. However, there is no need specify it in the first place.

The four phases do not always all take place. For instance, the pre-design phase can be omitted once the library is set up. Occasionally, the library may need an update, if there is a new type of ATC. Every time a new process arrives, or changes are required in the current process, an ATC life cycle starts again. There are several scenarios of restarting this life

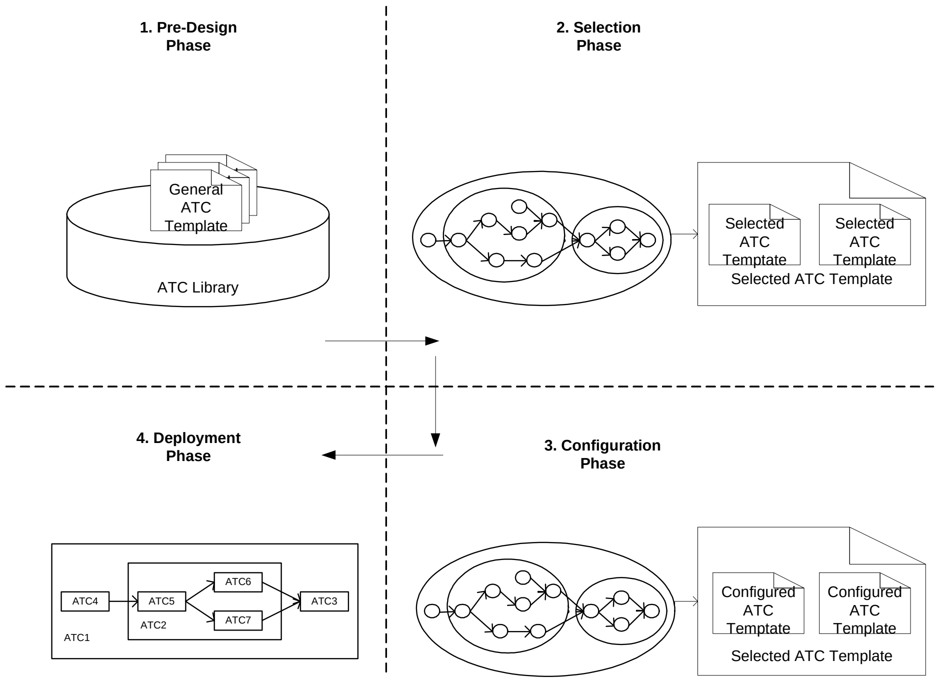


Figure 6.5: ATC Life Cycle

cycle. One scenario is when a new process arrives, it activates the ATC life cycle from the 2nd phase. Another scenario could be that some substantial changes have been made to the current process. In this case, parameters may need to be reset, and therefore the life cycle starts from the 3rd phase. If the change comes from the running environment, only the last phase of the ATC life cycle is activated for a slight adjustment. After the four phases, there is the process/service execution time. This is the time at which the deployed ATC schema is instantiated into a (multi-level) running transaction to handle exceptions and errors when they arise. It means that ATCs are transformed into executable transactional constructs in the end. However, the transformed ATCs are beyond the scope of this chapter. They will be discussed in the following chapter, where a TxQoS-aware business transaction framework built using ATCs is presented.

6.3.2 ATC Composition

The ATC library is constructed at the pre-design phase as described in the previous section. When designing transaction support for a given process, a composer selects one or more templates and configures the parameters using process knowledge, such as the structure and the TxQoS agreement etc. Afterwards, the configured templates are deployed according to the running environment of a process. This way, a transaction schema is composed with the on-demand manner. The top-level (global) ATC is, in fact, a transaction scheme for a process/service, which defines what general transaction templates will be selected and how they are composed. For each process/service, there is one corresponding top-level (global) ATC and the parts(chunks) from that process correspond to sub-ATCs in an ATC hierarchy.

We have defined in Chapter 4 how to specify the TxQoS and agree a TxSLA for a service/process between the parties. According to the the TxSLA, the composer selects proper ATCs, which guarantee some ACIDity to realize the TxQoS specified in the agreement. For instance, if the process demands a fluency valued 3, which means the breakdowns are allowed to happen three times maximally, then the parameter 'Atomicity' in the global ATC is set 'F', as this process cannot be an atomic transaction. Each (sub-)ATC in this case, when configured and deployed, is going to be instantiated in the process execution as a running (sub-)transaction. The other TxQoS specifications can be reflected in ATC templates in a similar way according to the table 5.1 in Section 5.3. Here we assume the composer has sufficient knowledge on the process and the process execution environment, such as process structure, running statistics, and TxQoS specifications.

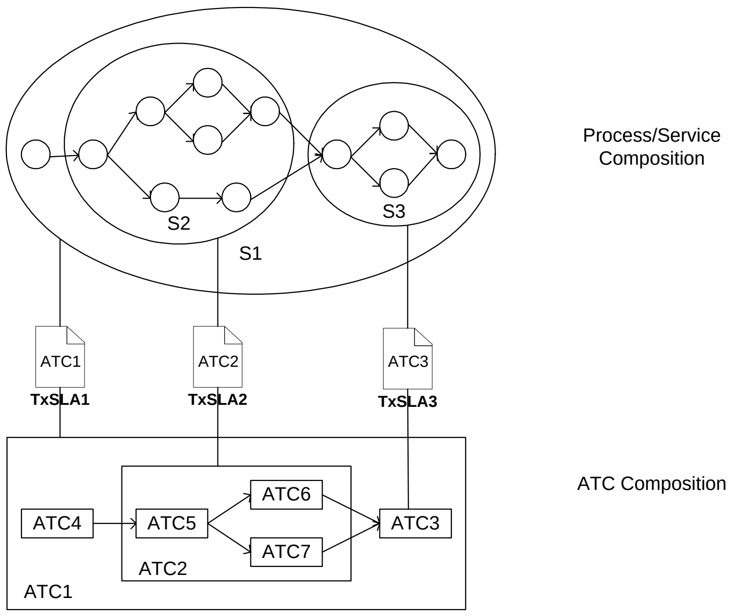


Figure 6.6: ATC-TxQoS Correlation

Figure 6.6 illustrates an example of ATC-TxQoS correlation. There is one global-level service/process named S1, which consists of two sub-service/process S2 and S3. In correspondence to the service/process composition schema is the ATC composition schema. In total, 7 ATCs are there, where ATC1, ATC2 and ATC3 correspond to S1, S2 and S3. Note that, the granularity of service/process composition is not equal to the granularity of ATC composition. In the below diagram, ATC2 can be further decomposed into ATC5, ATC6 and ATC7, while S2 is constructed by more than 3 steps.

For each service, it offers some TxQoS specifications for service consumer to choose. Upon a specific requirement (which is stated in a TxSLA), the composer needs to set the parameters in TxQoS1/2/3 accordingly. This means for different TxSLAs, the values of TxQoS parameters need to be set differently. After configuration of TxQoS parameters and running environment deployment, the ATC schema is instantiated into executable transactions at runtime. If anything goes wrong, for instance, the breakdowns exceed the agreed number in the TxSLA, the running transaction throws an error to the TxQoS monitor and a warning is sent to notice the violation between the values in the TxQoS specification and ATC schema. In this Chapter we focus on the ATCs, and the transaction framework built on ATCs to handle the TxQoS specification will be introduced in the next chapter.

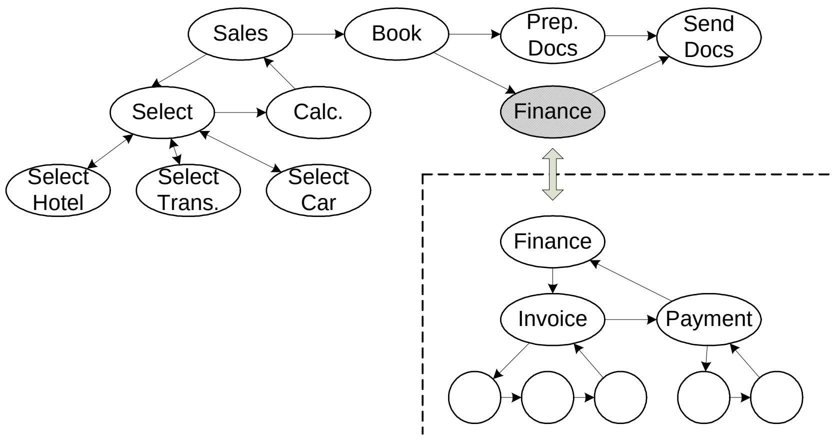


Figure 6.7: Example Travel Agency

To illustrate the ATC concept and especially ATC composition, we use a variation of the example of a travel agency, shown in Figure 6.7. Customers can create a trip by selecting a hotel, transportation, and an optional rental car (in parallel), after which the costs are calculated and the trip can be booked. In parallel , the required documents are prepared and the financial issues are dealt with (i.e., invoicing and payment checking), after which the documents are sent to the customer. Being a small travel bureau, the financial dealings are outsourced to a specialized organization, which offers this service as a Web Service. The interior of this service consists of invoicing and payment activities, which in turn consist of other activities not relevant to the example. The invocation of the Finance Web Service is represented in the travel agency process as a gray ellipse.

The resulting ATC schema for the travel agency example is shown in Figure 6.8. Eight ATCs are identified and named 'A' through 'H', which correspond to the activities/services shown in Figure 6.7. Note that the unnamed activities that belong to activities 'G' and 'H' are also ATCs but not relevant here. ATCs are represented by rectangles and the dashed lines represent encapsulation.

Assigning certain ATCs to different parts of this process will result in a specific behavior in case of exceptions, in which the transaction management system is involved. Assigning other ATCs, or by redividing the process over ATCs, the transactional behavior will be different in case exceptions occur, as explained before. For example, as the complete process (as seen by the travel agency) is a long-running process, the entire process might best be supported by a Saga like transaction model that comprises 'sales', 'book', 'prep. docs', 'finance' (the grayed-out one), and 'send docs'. The selection activities can be supported by an (variation of the) open nested transaction model, as these tasks can be done in parallel. The Web Service needs to be executed

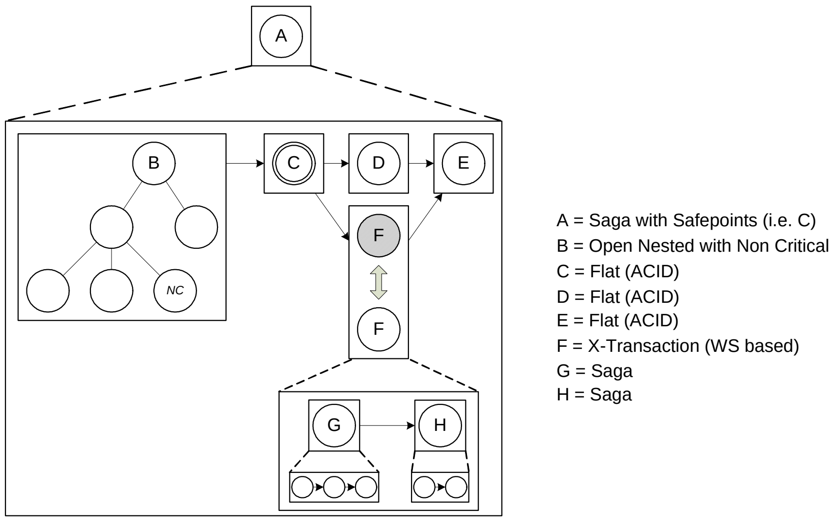

A
B
NC
C
D
E
F
F
G
H
A = Saga with Safepoints (i.e. C)
B = Open Nested with Non Critical
C = Flat (ACID)
D = Flat (ACID)
E = Flat (ACID)
F = X-Transaction (WS based)
G = Saga
H = Saga

Figure 6.8: Travel Agency ATC Schema

under some Web Services transaction model, while the internals of this Web Service, i.e., ‘invoice’ and ‘payment’ can be supported by a Saga again. As the Web Service cannot run in isolation, the travel agency might need to see intermediate results when its customers ask for status information, but needs to run in an atomic fashion so that the available web service transaction protocols (e.g., WS-BA) are not sufficient. In this case, we therefore choose a variation of the X-transaction model [60] that is suitable for the Web Services environment.

6.4 Conclusions

In this chapter, we proposed an ATC, which is not a newly-invented transaction model, but an abstraction and generalization of existing transaction models. ATCs abstract the existing models into four parts of the template: structure, position, ACIDity, and mechanism. A XML Definition Schema is designed, allowing the structure, position, ACIDity, and transaction mechanisms of an ATC to be specified. The ATCs transform through four phases, from the pre-design phase during which general templates are designed, to the selection phase where proper templates are selected on-demand, until the configuration phase during which the parameters are set, to the deployment phase when a specific ATC schema is fully composed and parameterized.

The proposal of an ATC concept is the outcome from the technology-oriented design to achieve our intended transaction framework. This chapter answers what is the outcome and explains how it ensures the design works for our purpose. Please note that the research on ATCs has been carried out in parallel with the research on TxQoS. Thus, the result presented in this chapter does not relate much to the research result from the previous chapters 4 and 5. A TxQoS-aware Business Transaction Framework integrating the TxQoS and ATC design will be presented in the next chapter to address this issue.

Chapter 7

BTF: Integrating TxQoS and ATC

In previous chapters, we have proposed the TxQoS approach as the contractual approach and the ATC approach as the technical support for reusable and flexible transaction management for service-oriented business processes. The TxQoS approach ensures transactional reliability via agreements between service providers and users. The ATC approach enables flexible and comprehensive transactional reliability via on-demand selection and composition. When working together, an integrated framework is needed to coordinate various life cycles and manage the functionalities through phases. In this chapter, we analyze the possible business patterns and the corresponding scenarios that can occur in our settings and present a Business Process Framework (BTF) that integrates the TxQoS and ATC approaches in our settings.

7.1 Introduction

With the increasing complexity of business processes, exceptions and errors are more and more prone to occur along execution. This poses new requirements on transaction management, which demands flexibility and comprehensiveness along with reliability. The existing transaction models have been designed and developed for certain application domains and lack the reusability which is a key benefit of the service-oriented processes. For example, a chain transaction supporting a long-lasting banking account update process requiring for the safepoint mechanism and visible intermediate results, does not fit an on-line payment web service.

Aiming for a comprehensible and flexible transaction management framework, we have proposed the TxQoS and ATCs approaches. The idea of

TxQoS is to complement the agreements/contracts with transactional clauses. As described in Chapters 3, 4 and 5, the TxQoS framework ensures process execution reliability via mutually agreed specification on transactional qualities. The idea of ATCs is to construct a proper transaction schema by selecting and configuring reusable transactional units (coming in the form of templates) according to process features. In Chapter 6, we introduced the concept and mechanism of ATCs which ensure process execution reliability via flexible composition of reusable constructs. The TxQoS approach creates business-level understanding of transactional reliability, while ATCs technically enable transactional reliability in a flexible and comprehensive manner.

To combine the TxQoS and ATC ideas and make them work together to address the reliability requirements that are increasingly challenging, we develop a Business Transaction Framework (BTF) which aims at laying a foundation to the transactional support for contract-driven, service-oriented processes. The basic idea of the BTF is to abstract the existing transaction models into Abstract Transaction Constructs (ATCs) and hide their implementation details that may reside in heterogeneous infrastructures. When a process is composed, according to the process specification, the proper ATC templates are selected from the ATC library and configured into a transaction schema, which is then deployed for execution. Meanwhile, for each service, the TxQoS template is configured into one or more TxQoS offers. These offers are designed based on the transactional capabilities of the BTF and are provided to users. A user selects a TxQoS offer according its transactional requirements imposed by the business process and then a TxSLA is established between the user and the provider. With the BTF-enabled transaction support, the TxSLA is flexible for changes if the user changes its requirements or the provider updates its service. One benefit of the integrated TxQoS-aware BTF solution is that the impact on transaction management from frequent changes of business processes is minimized because of the flexible ATC composition, therefore business agility is improved. Another benefit is that the user side is equipped with more knowledge and bargaining power because of the unambiguous TxQoS specification on service reliability, therefore improving customer satisfaction.

To demonstrate the integration of the TxQoS framework and the ATC mechanisms, herein the travel booking process from the previous chapters is used as an example. The process in Figure 6.1 in Chapter 6.1 starts from booking activities at the front office and ends with documentation. Beneath the process is the transaction support realized by ATCs as shown in Figure 6.7 and Figure 6.8. Suppose the travel agency provides the booking process as a service to its clients. Service reliability is guaranteed in the TxQoS offers based on the statistics from service (test) execution. Assume the booking service is published and registered for discovery and usage. In a B2B context, a client incorporates the booking service into its own process and therefore depends on the reliability of that service. The client does not care about the technical details such as the ATC composition as long as the service is guaranteed to be running robustly and smoothly at an acceptable level. This is stated by a TxSLA agreed between the travel agency and the client, and any violation to the agreement results in a repair and/or penalty from the provider to the client. While in a B2C context, a client directly makes use of the booking service and does not need to integrate it with another process. In this case a TxSLA is not necessarily established. However, the client still can choose another better service based on a reliability comparison of services that provide the same function. For instance, if the client would like to choose a booking service that allows more interaction with different sub-service providers (i.e. hotel, transportation etc.), then a service with a high T (Transparency in FIAT attributes) in TxQoS offers may be chosen. Or the client would like to configure the details of its trip, then a service with a high I(Interferability in FIAT) may be chosen. If the TxSLA needs to be established, usually it is the that client agrees to an offer by default, before usage.

The rest of the chapter is organized as follows. First, we present the scenarios of integrated TxQoS and ATC approaches in Section 7.2. In this section, the two scenarios of provider-dominant and provider-user equivalent are introduced and analyzed respectively. Then, we illustrate the life cycles of the BTF in Section 7.3.1, where a pattern matrix is proposed first, and life cycles are presented for each pattern. Next, in Section 7.4, we suggest a reference architecture. We conclude this chapter in Section 7.5.

7.2 BTF Scenario

As an integrated framework for the TxQoS and ATCs, the BTF needs to coordinate and manage the scenarios and mechanisms from the two approaches. Herein we still take the travel booking process as the example to demonstrate what a BTF is, and how it works. What we usually see in practice (without applying the BTF solution), a user invokes a travel (Web) service without making any impact on the provider-side, but just accept the service by default. It will become quite different if the BTF solution is introduced in this scenario. Let us first look back on the TxQoS scenario depicted in Figure 4.5 in Section 4.2.4. There are three parties interacting with each other. Intermediaries provide only auxiliary functions and can be dismissed in the scenario, so here we focus on the provider-user interaction. As we can see, the provider and the user have the symmetric mechanisms to enable the establishment of a TxSLA. It indicates that a user has the power to impose its requirements on the agreement. If introducing the TxQoS approach, the user side is empowered with better knowledge in selecting reliable services, and the user can negotiate with providers on transactional qualities, for instance asking for more interferability and transparency. As for ATCs, we have designed the templates in the role of a service provider. In contrast to the symmetric structure of TxQoS scenario and contracting model, a service user does not need ATCs to make better use of the service. Therefore the ATCs are from, and for the provider side. Again, taking the booking process in the introduction as the example, we will show the BTF Scenarios from two angles: Provider Dominant and Provider-User Equivalent. In the Provider-User Equivalent scenario, a service user can be a provider at the same time, invoking some services to form its own service, and afterwards provide the service to its users. Interested readers can refer to [28] for the picture of a global enterprise network, where services are networked and the owner of a service can be the consumer of another service, so on and so forth.

7.2.1 Provider-Dominant Scenario

In a provider-dominant scenario, service providers implement the full BTF framework, including the TxQoS mechanisms and ATCs. In contrast, service users do not need to implement all parts of the BTF framework. In the simplest form, the user is viewed just as a black box, which functions as a service invoker without any contracting activities. However, to fully leverage the transaction support from the provider side, some basic modules need to be presented, such as a communication agent. In this scenario, a user is a 'pure' user, so we do not look at its transaction framework, which is not relevant to its role as a user. A lot of Web services with default user agreements can be viewed as such a 'Provider-dominant' scenario. For example, suppose there is the travel booking process, as shown in the previous section, which is implemented by the TxQoS architecture and ATCs as described in the previous chapters. There is an individual user who makes use of the service to book a trip. In this example, this user does not care about what ATC schema has been designed or what techniques are applied at the provider side, as long as the standing TxQoS offer matches to his requirement, and the process is fluent and allows the intermediate results to be recorded. Figure 7.1 below gives an overview of such a scenario.

Like the TxQoS scenario in Figure 4.5 in Chapter 4.2.4, intermediaries are optional here. A TxSLA is not necessarily established in this scenario, as the

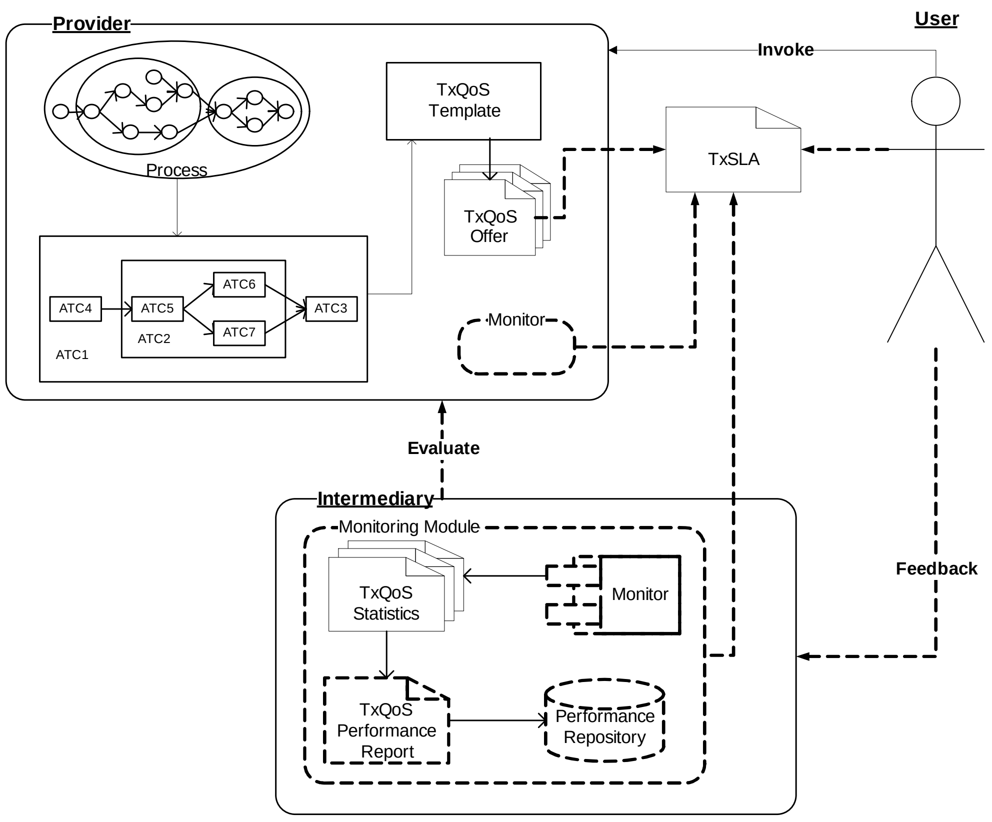


Figure 7.1: BTF Provider-Dominant Scenario

user agrees with the TxQoS specification the provider offers by default when invoking the service. Such a scenario usually presents in B2C context, where users invoke services in a simple manner. In this case, the user side does not incorporate the services into its own business processes and there is no need to know the details at the provider side. Since there is little integration between the provider and user, the user side is not able to participate in the monitoring life cycle directly. However, the user side can still leave feedback through intermediaries's interface via, for instance, SOAP/HTTP communication.

7.2.2 Provider-User Equivalent Scenario

In a B2B context, a party often invokes a service from another party to compose its own service and business process. Therefore, this party is a user when invoking a service provided by another party, and at the same time, it is a service provider when its own service is invoked by it customers. The provider-dominant scenario is then extended to a 'provider-User Equivalent' scenario. We suggested a symmetric reference architecture in Figure 5.3 in Section 5.2.2, where the user side adopts the same architecture as the provider side. This enables the negotiation and user interference on the transaction management at the provider side. Take the travel booking example, for instance: if a company uses the booking service to build up its own process and then provides a multi-functional web service to its clients, the transactional quality (i.e. reliability) of the booking service becomes an important factor to consider when composing its own service. This company may compare different offers on the market according to its transactional requirement and establish a TxSLA (besides other business contracts) with the booking service provider. When mapping the ATC recursion, this means the company's own transactional schema is a global level ATC, which is composed of some sub-ATCs, including the ATC abstracted from the booking process.

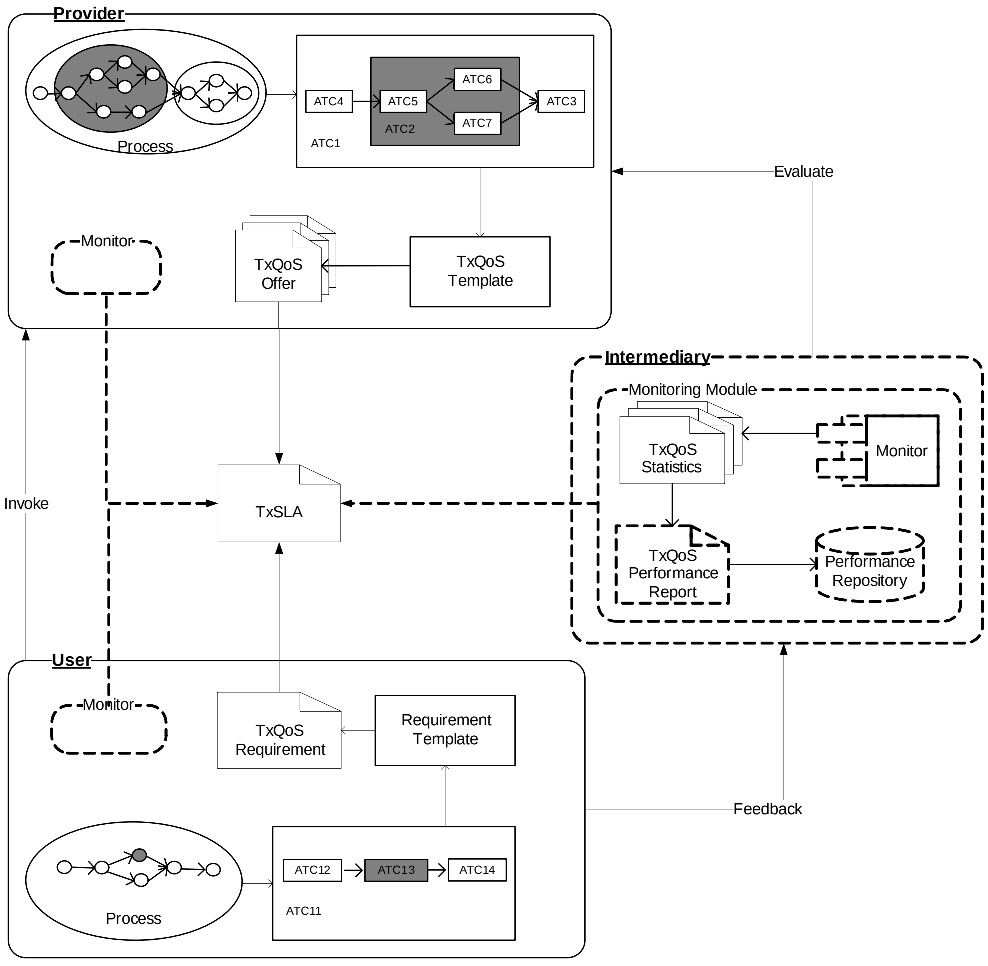


Figure 7.2: BTF Provider-User Equivalent Scenario

Figure 7.2 shows the Provider-User Equivalent scenario where the user side also implements the BTF framework. The offered service (implemented by the grey part of the provider process) and the corresponding ATC design (i.e. the grey part of the ATC recursion) are used by the user side to compose its own business process and design its ATC recursion (marked by grey parts at the user side to show the relevance). Upon any change on one party's process/service, only the corresponding ATC is re-configured, and its upper-level ATC is affected. For instance, if the booking process shown in Figure 6.7 in Chapter 6.3 has a structure change, and as a result the parallel booking activities become sequential, the ATC structure shown in Figure 6.8 in Chapter 6.3 is accordingly adjusted. Consequently, the company which invokes the service as part of its sub-process needs to update its ATC schema to accommodate the change (i.e. only the grey parts are affected).

Today's enterprises develop their information systems in a decentralized manner in which processes/services are intertwined, like a net. A service provider can meanwhile be a service user making use of another service provider's offering. Meanwhile a service user can take the role of a provider at the same time. All these services interact with each other in this net. We can conclude that the provider-dominant scenario is gradually developing towards a provider-user equivalent scenario with the trend moving towards networked enterprises.

7.3 BTF Life Cycle

Before discussing the life cycle of BTF, first we look at the TxQoS and ATC life cycles. The basic idea of TxQoS is to enhance reliability via an unambiguous agreement on transactional qualities between parties. Those agreements are specified based on the provider-side offers and the user-side requirements. The TxQoS mapping at the external level (as shown in Figure 5.4 in Section 5.3 is rooted in the internal level transaction mechanisms from the provider side. As the building blocks of the BTF, ATCs work at the internal level of the service provider, by building up an on-demand transaction schema. However, the user side can also adopt the BTF to provide transaction support as the role of provider, which happens in a provider-user equivalent scenario, as described in the above section. In this section, we first analyze the provider-dominant scenario. The provider-user equivalent scenario is an extension of the provider-dominant scenario. Thus, it has the same life cycle with a slightly different positioning. As described in Section 4.2.3, the TxQoS approach works from the first phase of 'Design', to the 'Contract', and ending at the 'Evaluate' phase. It is a cyclical structure where every phase is necessary and in order as the figure shows. As a contrast, the ATC life cycle described in Section 6.3.1 comprises of 4 phases, 'Pre-design', 'Selection', 'Configuration', and 'Deployment' respectively, which are are not necessarily looped. Depending on situation, the cycle can start from any of the phase and some of the phases are optional.

When integrated to work as a package of solutions, the phases of these two approaches run parallel, and sometimes overlap. The design of ATC templates should take place at the very beginning, before or in parallel with the activity of process design. After blank ATC templates are organized and stored in the ATC library, the configuration and composition can start only after a process has been designed. Meanwhile, the TxQoS specification is designed also at this point of time, as it needs to collect process execution data. ATC deployment takes place before process execution (i.e. runtime). Meanwhile TxQoS contracting also takes place before execution. Afterwards, the TxQoS evaluation is completed during or after the execution, by which time the BTF has worked through a full cycle. In the next section, we analyze the BTF life cycle of different business patterns according to the relation between service/process providers and users.

7.3.1 Business Patterns

In the past, suppliers were tightly bound to their customers, and vice versa. Thus, a business relation is hardly changing over time. However, in an e-business era, when information flows much more transparently and easily, providers can reach a much wider range of potential users from various time-zones, locations, and industries. Meanwhile, the users are able to choose from a lot more offers to get their desired services and products. It has led to an increasing business agility and flexibility during the business interactions, thanks to technology enablers such as Web services. We propose a matrix, as shown in Figure 7.3, in order to analyze the BTF life cycle (including ATC and TxQoS monitoring phases) in different scenarios. The business patterns in this matrix are identified by two dimensions: provider dominance and business dynamism.

During the past decade, we have seen the business pattern following a trend, moving from a provider-dominant process and a static relation to a user-involved process and a dynamic relation. The BTF can facilitate this shift because using two approaches. The TxQoS framework supports the trend from traditional business to e-business by differentiated TxQoS agreements (empowered user). The ATC approach enables the flexible design of a transaction schema to reduce the impact of changes on business processes (enhanced dynamism). In this section, we are going to integrate the TxQoS

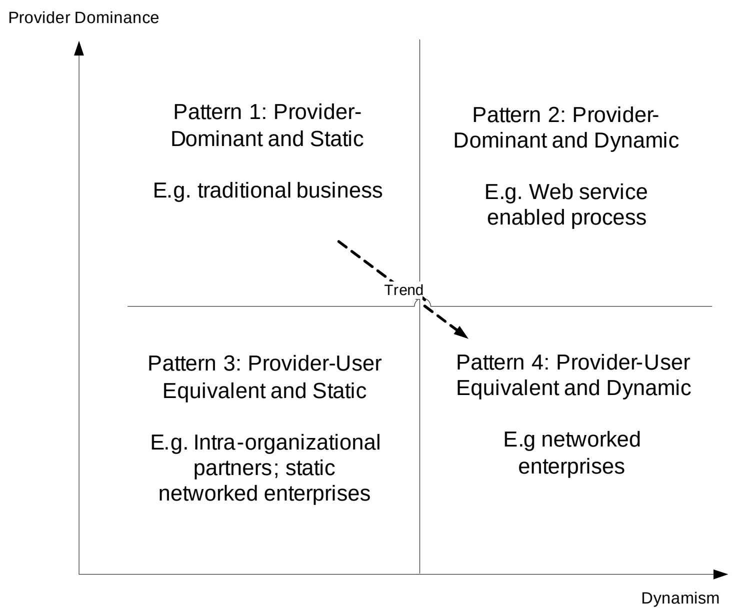

Provider Dominance
Pattern 1: Provider-Dominant and Static
E.g. traditional business
Pattern 2: Provider-Dominant and Dynamic
E.g. Web service enabled process
Trend
Pattern 3: Provider-User Equivalent and Static
E.g. Intra-organizational partners; static networked enterprises
Pattern 4: Provider-User Equivalent and Dynamic
E.g networked enterprises
Dynamism

Figure 7.3: Business Patterns

monitoring phases and the ATC phases, to identify phases in the BTF life cycle. There can be a slight difference, depending on scenarios presented by the above patterns. For instance, the starting times of the negotiation and contract-establishment vary in different scenarios. Therefore, the life cycle of a BTF does not match with fixed TxQoS monitoring phases. First, we analyze the business patterns 1-4. Next, we present the BTF life cycle for each pattern. Finally, we group the 4 slightly-different phased life cycles for an overview.

1. **Provider-Dominant and Static**
   The business pattern in this quadrant means providers have more bargaining power than users in their interactions, and their relations are bonded tightly with few changes over time. Traditional business usually fits in this pattern. For instance, suppose there is a branch of a big supermarket chain which gets their goods from the mother company, and serves its neighborhood. As long as the neighborhood is stable, the supply chain from the mother company (the supermarket chain) to the branch shop until the end customers is quite stable, without need for swift changes. In this type of business, the provider side has a dominant role, i.e. making the offers with fixed prices, while the user side takes offers without negotiation along the process. In addition, the roles in this case hardly change over time (e.g. a provider is always a provider

and processes rarely significantly change).

2. **Provider-Dominant and Dynamic**
The business pattern in this quadrant has the same provider-user balance with the above one, i.e. providers make offers with little user impact. However, it differs in the tightness of the provider-user bonding. Unlike the traditional static business relationship, in which providers and users are bonded with fixed business partners, this pattern exhibits flexibility and variety in the choices both providers and users have. This means a provider may make offers to a wide variety of users and contrastingly, a user also may choose from a wide variety of offers. One typical example in this pattern is Web services, e.g. a hotel-booking service invoked by users in Internet. In this example, differentiated room rates are set by hotels without user participation. During the business process from a user clicking on the service until a room is booked, there is little he/she can do on the rate offers. Thus, it is a typical provider-dominant scenario. Meanwhile, this booking service is not bonded to any specific user, and vice versa. The bonding is very loose in the manner that a user can cancel at any time to close the business relationship with the provider.

3. **Provider-User Equivalent and Static**
In this quadrant, a user is equipped with enough power to bargain on the offers a provider makes. Meanwhile, the bonding between the user and the provider is very tight. In other words, the supply chain in this pattern is very stable, with strong user participation. A typical example in this pattern is intra-organizational units, which are maybe financially independent, but usually share common technical infrastructures, e.g. a business department requests the IT department to deliver a project within a certain budget. Another typical example in this pattern can be an exclusive supply chain in which a company supplies a certain material to another company, with long-term contracts binding their relationship, which is in fact a static networked enterprise such as plane manufacturers. In this type of business, the user side may ask for customized products and services with a higher level of freedom. Sometimes providers and users even share common infrastructures and leverage the same technologies or standards. Regarding the BTF scenario, the user side in this pattern may implement the same framework as the provider side, as the symmetrical architecture we proposed for the TxQoS framework.

4. **Provider-User Equivalent and Dynamic**
The business pattern in this quadrant exhibits dynamic business rela-

tionships with strong user presence. Customer-driven is the trend of the business, and the relationship between business partners has become more and more dynamic. In the e-business era, flexible choices and ample choices are realized without being restricted by geography, time zones, and information accessibility. This empowers the user side, giving them a higher bargaining power than ever before. Customization has become a trend and providers make offers which are more and more influenced by the user. A typical example in this pattern is the instant virtual enterprise [28], where enterprises are organized like a dynamic network to make business deals. In such a network, providers and users are like nodes in a net, interacting with each other autonomously. Business deals are made on mutual agreements in which users may negotiate with providers for customized offers. A user is a provider at the same time, for instance in a B2B paradigm, who incorporate processes/services/products from other providers into its own so as to provide to its users. In this case, each node in this network implements common infrastructures (e.g. BTF) and therefore, a symmetric architecture is proposed in this type of business.

7.3.2 BTF Phases

For every business pattern, we apply the ATC phases, which are shown in Fig 6.5 and described in the previous chapter, and the TxQoS monitoring phases shown in Figure 5.5 in Section 5.4. The BTF is an integrated framework of TxQoS and ATC approaches. Therefore, the BTF life cycle comprises the phases from both approaches. Below we identify and analyze different business patterns:

1. **Provider-Dominant and Static**
   For this type of business pattern, coined 'Provider-Dominant and Static', only the provider side fully implements BTF. The life cycle starts from the design of ATC templates, which also applies to all other patterns. In parallel with the ATC composition phase, TxQoS offers can be designed and published. While the ATC deployment takes place, the TxQoS negotiation may happen at the same time, leading to a TxSLA. At runtime, the TxQoS performance reports may be collected for monitoring, along with the execution of Transaction Constructs (i.e. instantiated ATCs). Figure 7.4 below shows the integrated BTF phases, which are made up of ATC phases and TxQoS monitoring phases. Note that a TxSLA is not necessarily established in the provider-dominant scenario, as explained in the previous section. So, the optional TxQoS

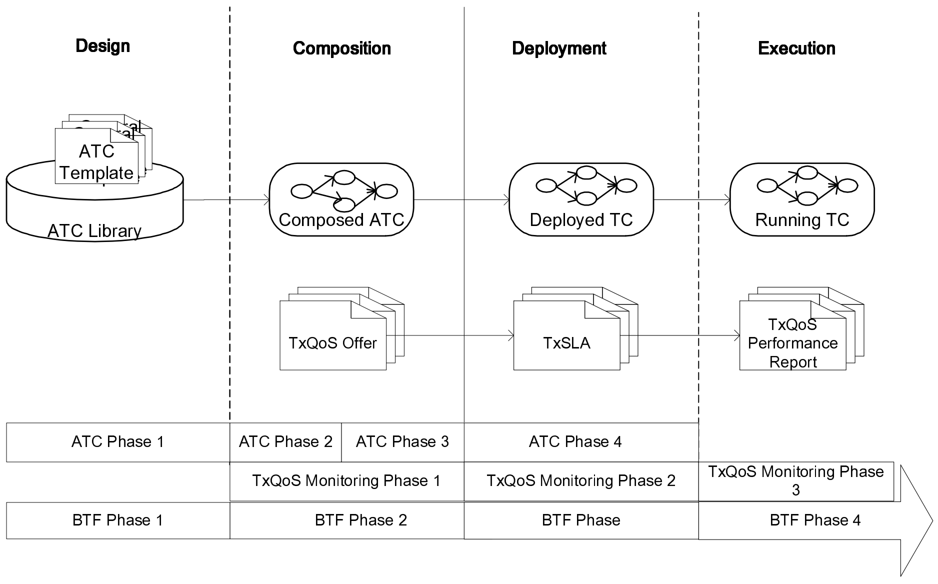


Figure 7.4: Provider-Dominant and Static Business Pattern

monitoring life cycle appears here to identify the BTF phases. Also please note that only part of the artifacts from the two approaches are depicted in all BTF life cycle diagrams. The remaining artifacts, such as the 'business specification' and the 'requirement template', appear in the relevant figures in previous chapters, and are not shown here because they do not make facilitate identifying BTF phases.

2. **Provider-Dominant and Dynamic**
   In the 'Provider-Dominant and Dynamic Pattern', the users do not necessarily need all TxQoS functions. For instance, as explained in the pattern analysis above, a user invoking a booking service can be a natural person using just a PC and Internet, and does not implement the BTF framework. This user agrees an offer, without negotiating with the provider. Sometimes the user even does not notice the establishment of an agreement. For this type of business, the BTF life cycle starts from the design of the ATC templates; the same as the business pattern above. Because of dynamism in business relationships, TxQoS offers can only be made after the composition phase, i.e. upon the completion of the process and transaction design. Figure 7.5, above, shows the integrated BTF phases. The difference compared to Figure 7.4 is the starting time the latter offers for making TxQoS.

3. **Provider-User Equivalent and Static**
   The BTF life cycle for the pattern of 'Provider-User Equivalent and

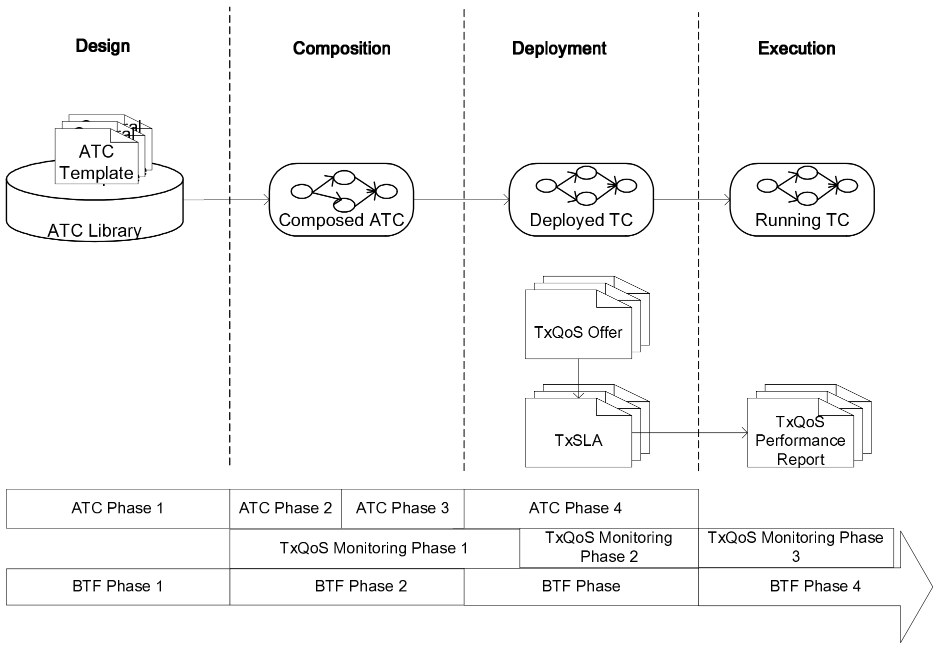


Figure 7.5: Provider-Dominant and Dynamic Business Pattern

Static', shown in Figure 7.6, stands for a highly-trusted type of business relationship. Noticeably, TxQoS offers are made in parallel with the transaction design at the composition phase. Users in this type of business are involved in, or their requirements are considered by providers during, the services/processes design. Negotiation may begin in either the composition or the deployment phase. As a result, TxQoS agreements (TxSLAs) are established in either phase which are depicted as positioning across the two phases.

4. **Provider-User Equivalent and Dynamic**

   The pattern of 'Provider-User Equivalent and Dynamic' is the trend in business, thanks to the development of service technologies allowing ad-hoc integration and collaboration. In a dynamic enterprise network, providers and users are loosely coupled and supply chains are formed with maximized efficiency (e.g. cost-efficient and reduced effort). The provider-User equivalent scenario requires a symmetric implementation of BTF, in which users are allowed to make requirements that impact the design of the provider side (i.e. it is customer-driven). As such, the phases of the BTF life cycle differs from the above three in the timing of TxQoS activities. As shown in Figure 7.7, TxQoS offers and the establishment of TxSLAs may take place at either phase, which are po-

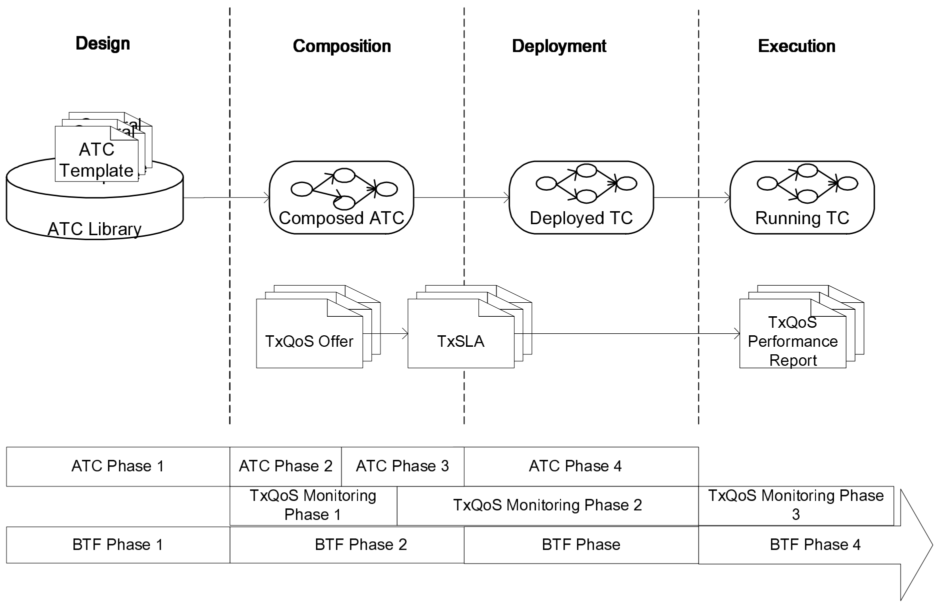


Figure 7.6: Provider-User Equivalent and Static Business Pattern

sitioned in the middle. Please note that TxQoS offers have to be made before TxSLA establishment, although in the figure, they are positioned both in the middle without an obvious time line. The figure implies three possibilities: both TxQoS offers and TxSLAs are established in the composition phase; TxQoS offers are made in the composition phase while TxSLAs are established in the deployment phase; both TxQoS offers and TxSLAs are established in the deployment phase. However, the negotiation can take place already in the earlier phase, so that user requirements do impact the provider design. This pattern exhibits the most flexible BTF life cycle.

As classified and depicted above, the phase composition of a BTF life cycle varies per business pattern. The BTF life cycle always starts from the ATC template design and ends at runtime (i.e. at the execution phase). In the first phase, the design of the ATC template is finalized, and is matched to the ATC phase of 'Pre-design'. During this period, an ATC library is set up as the preparation for all possible processes. The 'Composition' phase matches to the 'Design' phase of the TxQoS life cycle and the 'Selection' and 'Configuration' phase of the ATC life cycle. In this phase, proper ATCs are selected on demand, and are configured into a platform-independent transaction schema (named as 'Composed ATC') according to the process specification. Meanwhile, TxQoS offers are designed for user selection and

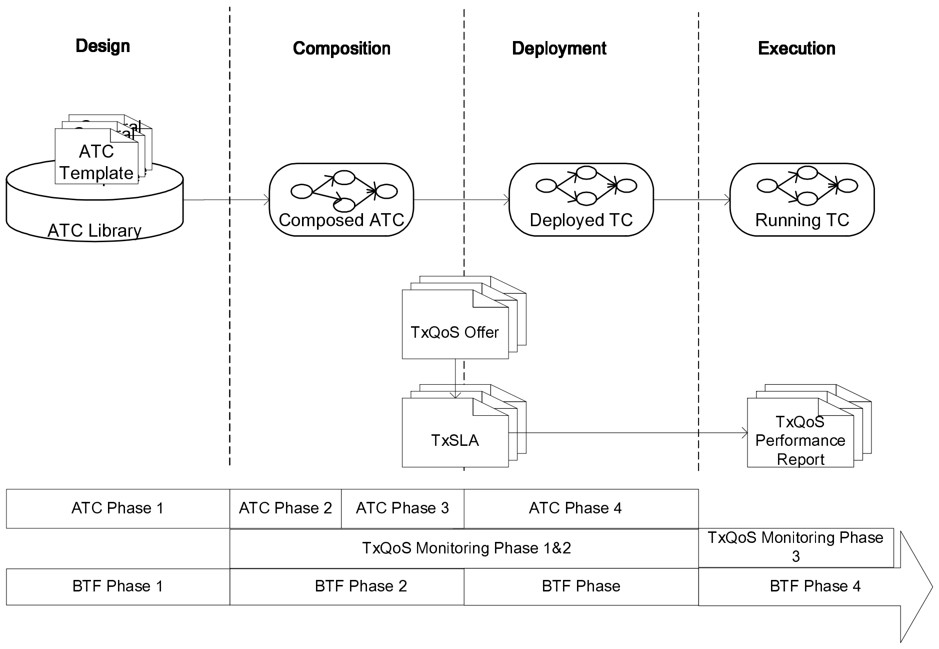


Figure 7.7: Provider-User Equivalent and Dynamic Business Pattern

negotiation. Depending on the business pattern, TxQoS offers can either be made within the composition phase or until the deployment. For instance, in a static business relationship, where parties are bonded tightly and with highly integrated infrastructures, the design of the TxQoS offers is completed usually after the platform information is obtained, i.e. the process is available for real-time testing. In this case, TxQoS offers are made throughout the 2nd and 3th phases. Then in the ‘Deployment’ phase, where the environment and the platform parameters are set, the transaction schema is deployed for execution and one or more TxSLAs are established. This phase matches the ’Deployment’ phase of the ATC life cycle. The last phase is the ‘Execution’ phase, during which the instantiated ATCs (named as TCs as they are not abstract anymore) are executed. Also included in this phase is the generation of performance reports through the collection of running statistics of TxQoS (e.g. breakdowns). This phase starts after the last ATC phase, which ends until runtime, and matches to the ‘Evaluate’ phase of the TxQoS monitoring life cycle.

Note that we only show part of the artifacts from the TxQoS and ATC approaches here. The big arrow at the bottom represents a full BTF life cycle. However, as in the ATC life cycle, changes on process specifications or platforms may result in a cycle directly starting from the 2nd or 3rd phase. For instance, shifting to a new platform asks for a new deployment therefore the Transaction Constructs (TCs) are redeployed while the transaction schema resulted from the previous phase still remains the same.

In short, we make a unified life cycle diagram for all business patterns in Figure 7.8. As we can see, the ATC phases are matched to the BTF phases. While the difference between the 4 patterns is reflected by different timings of TxQoS monitoring phases. The gray box after the ATC life cycle indicates that instantiated TCs are running in the execution phase.

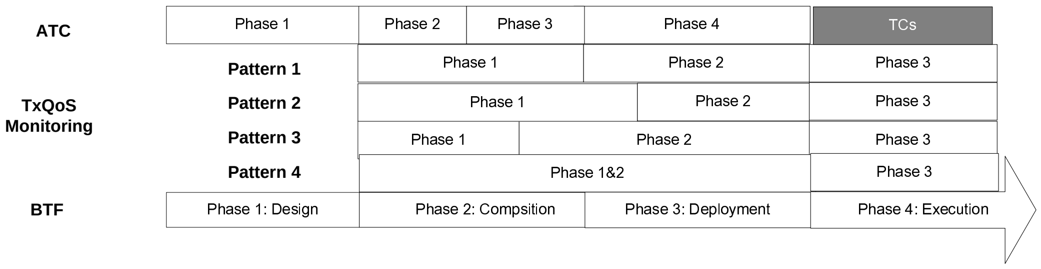


Figure 7.8: BTF Life Cycle

To support the transformation through the phases, coordinating and managing activities must be present. Here we depict the BTF life cycle only. The supporting functional components of the BTF framework and a reference architecture will be presented in the next subsection.

7.4 BTF Architecture

The basic idea to achieve a contract-driven service-oriented transaction framework BTF is to leverage the contractual approach TxQoS, to ensure process reliability via on-demand composition of reusable constructs (i.e.ATCs). So far in this chapter, we have proposed the integrated BTF scenarios and analyzed its life cycle pattern by pattern. In this section, we propose a BTF architecture, which delivers the necessary functions (i.e. integration, coordination and management) to support the BTF scenarios and phases. The design method is similar with the TxQoS reference architecture in Chapter 5: From various classic architecture styles (e.g. Pipes and Filters, Layered Systems, etc.), we adopt the heterogeneous style where the functional modules and artifacts are distributed across the layers with a hierarchy-like structure. Here we use 'artifact' to represent requirement templates, process specifications, transactional agreements, transaction schema, etc. It is a general term for all things created along the TxQoS, ATC, and BTF life cycles. Next, we identify and analyze the functions the architecture needs to deliver based on the requirement analysis. Then, we propose a three-layer reference architecture that satisfies the functional requirements.

7.4.1 Requirements Analysis

As described in previous sections, the BTF solution is an integration of the TxQoS and ATC approaches. For every business pattern, we have applied the ATC phases, as shown in Figure 6.5, and the TxQoS monitoring phases, as shown in Figure 5.5, from which we identified the BTF phases. To make a BTF architecture design, first we need to identify what functions are needed to integrate and coordinate the two approaches. Next, following the heterogeneous style for architecture design [52], we analyze, phase by phase, the components, connectors, and containers. Below we analyze the requirements along the BTF phases to get a list of components supporting the required functionality in architecture design.

1. **Design Phase**
   In this phase, ATC templates are designed, organized, and stored in a library for future configuration and composition. The main functionality from the architecture perspective is the management of the ATC library. Therefore, an ATC library manager performing the functions of designing, organization, and storage is the key component here. Meanwhile, auxiliary components such as a ATC repository, ATC templates and connectors with the other components are required as well.
2. **Composition Phase**
   After the templates are designed and stored in the ATC library in an organized manner, proper ATCs are selected, composed and configured into an initial transaction schema, according to specific process specifications. During this phase, a key component, a designer, must be present to perform multiple functions including selection, composition, configuration, etc. The designer enables the transformation from ATCs to a ‘composed ATC schema’.
3. **Deployment Phase**
   During this phase, the ATC schema and process specifications are deployed together with platform and environment parameters. Therefore, a component to enable the deployment is needed to transform a ’composed ATC schema’ into a ‘deployed transaction’. Note that ATC is not abstract anymore and we call the deployed ATC schema ‘deployed transaction’, which is ready for execution.
4. **Execution Phase**

The execution phase means runtime during which a deployed transaction consisting of TCs (Transaction Constructs, i.e. sub-transactions) is executed. During this phase, the main functionality is to collect performance reports and/or errors for monitoring purpose. As described in Figure 5.3, the TxQoS manager contains a component for runtime monitoring. Therefore, a component needs to be present in the architecture to interact with the TxQoS and ATC managers for monitoring and managing purposes.

Based on the above analysis, we identify the main components of the architecture. These components are distributed in the bottom layer, which directly interact with the DBMS and other underlying systems (e.g. the workflow engine), and in the middle layer, where coordination and management take place. On top of these functional components, a managing component taking control to complete the phases is required, and we name it the BTF manager.

So far we have performed a functional analysis, and have not touched the non-functional aspects. These non-functional aspects include security, the communication pattern (e.g. synchronous/asynchronous messaging), the network QoS (e.g. bandwidth), and the technical error handling (e.g. hardware recovery). They need to be considered as well for the architecture design, but are beyond the scope the thesis, as our aim is to suggest an architecture without considering the constraints from the hardware, platform, or implementation environment.

7.4.2 Architecture Design

We have introduced the idea, presented the scenarios, and identified the phases of the BTF. As the last step to design a comprehensive business transaction framework at the conceptual level, an architecture design, which is used for implementation referencing, takes place. We propose a three-layer, cross-phase architecture, as shown in Figure 7.9. The heterogeneous style of the architecture design separates concerns by the layered structure and simplifies the communication by wrapping up the relevant functionality into common modules. It is developed with the similar logic of the TxQoS architecture design, where the heterogeneous style is also adopted.

At the top is the management layer, where managing and coordination activities take place. At the middle layer is the function layer, where design, composition, configuration, and monitoring activities take place. At the bottom layer is the artifacts layer, where all created and generated artifacts (e.g. specifications, constructs, templates, and documents) come into play. Among

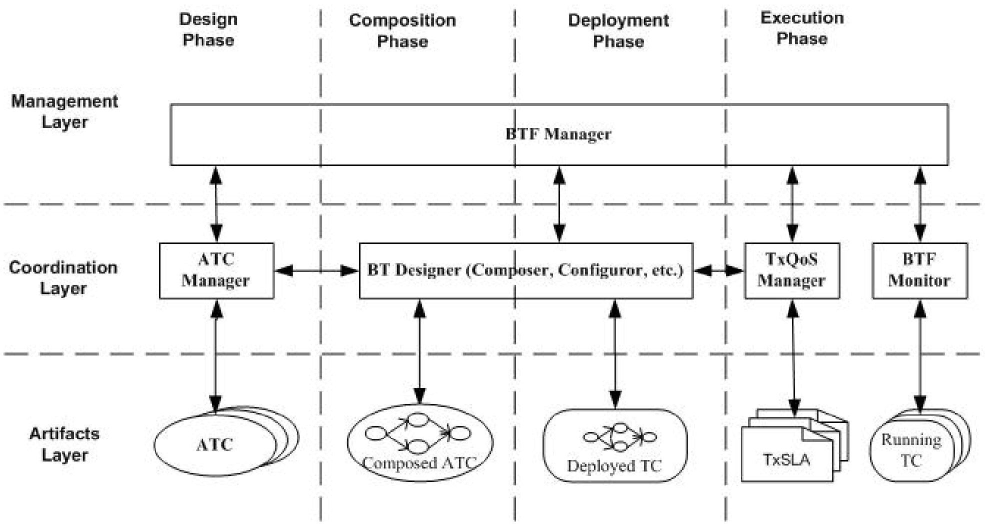


Figure 7.9: BTF Architecture

all the components, the 'BTF Manager' is the central hub, which coordinates and controls the lower-level modules. It serves as the hub of the framework and takes charge of the general management issues. The real functioning modules, which enable design, composition, configuration, database access, and monitoring, are positioned below the BTF manager. These functional modules create, manage, and control the artifacts along through phases, and also communicate with the underlying systems like DBMS and WfMS through the communication infrastructure, for instance a Enterprise Service Bus (ESB). As a whole, these modules form an architecture to support the idea of the TxQoS and ATC. A BTF architecture can work in a standing-alone way or with other BTF architectures implemented over heterogeneous IT infrastructure.

The components in the above figure can be further decomposed. We have outlined the top-level design in Figure 7.9. Below we present a list of the modules in the architectural components.

BTF Manager

The main component in the architecture, the BTF Manager, sits at the top layer for the overall control and management of transactional activities across the whole life cycle. It contains the modules for specific functionality. A 'Messenger' receives and replies messages from the lower-level component, such as the ATC manager. A 'Coordinator' serves as a hub for the data exchange between the components. A 'Controller' monitors runtime performance, detects errors, and solves conflicts. An 'Administrator' registers, accesses, and updates the artifacts across all phases.

ATC Manager

The 'ATC Manager' works during the design phase, where creates and organizes ATC templates, as well as manages the access to the ATC library. It can be further decomposed into an 'ATC Editor', used for writing and editing the ATC templates, and an 'Administrator', who manages communications with other modules which need to access the ATC library.

BT Designer

The 'BT Designer' is a component for designing business transaction schemas. As shown in Figure 7.9, it works across the composition and deployment phases, transforming original ATC templates into deployed and ready-to-execute transactions. It consists of three modules: a 'Composer', a 'Configurator', and a 'Deployer', allowing human designers/developers to work with ATC templates. The 'Composer' interacts with the 'ATC Manager' to select proper ATCs and compose them into transaction schemes according to process specifications. The 'Configurator' allows the designer/developer to configure the ATC schema, using environment parameters such as the ATC as well as alternate execution paths. The 'Deployer' is to deploy the configured transaction schema with platform parameters into a ready-to-execute transaction.

TxQoS Manager

The 'TxQoS Manager' has been designed and introduced in Figure 5.1, where we see it lays the foundation for a number of tools. It works during the execution phase to manage and coordinate between parties and internal components.

BTF Monitor

An optional monitor has been proposed in Figure 5.2 for runtime monitoring of TxSLA compliance. Here the monitor module of the BTF architecture goes beyond the TxQoS framework, since the ATC approach is integrated. Besides a 'TxQoS Monitor' monitoring the TxSLA compliance, a BTF Monitor also contains a 'TC Monitor' which signals runtime error arise from running TCs.

7.5 Conclusions

The aim of the XTC project is to design a transaction framework for service-oriented business processes, with flexibility and comprehensiveness as primary concern. In this chapter, we integrated the TxQoS and ATC design to deliver such a transaction framework. We started with proposing the two BTF scenarios, provider-dominant and provider-user equivalent respectively.

Based on this, we developed a matrix to distinguish four business patterns. We analyzed the features and life cycles of each pattern. The BTF life cycle consisting of four phases is identified and analyzed. Note that this life cycle integrates the ATC and TxQoS monitoring life cycles as well. The phases of the TxQoS monitoring life cycle vary in the segments of the BTF time line according to the different business patterns.

As a suggestion to implement a BTF that integrates the TxQoS and ATC approaches, as well as satisfy the functional requirements from each phase, we proposed an architecture. Consisting of the components distributed in layers, the architecture supports the integration of the TxQoS and ATC approaches. Thus, we complete the design of a TxQoS-aware business transaction framework. In the next chapter, we will introduce a large, real life project to show how the BTF can be applied in complex cross-organizational business processes that are contract-driven and service-oriented.

Chapter 8

Case Study for Validation

We have designed a TxQoS-aware business transaction framework (BTF) built on ATCs. The BTF integrates the scenarios, phases, and architecture components to meet the flexibility and comprehensiveness requirements in transactional reliability. In this Chapter, we present a large, real-life integration project as the context in which we perform a case study. A telecom B2B order-installation-billing process is selected because it is long-lasting (several weeks or months), complex (multiple organizations and processes/services are involved), and contract-driven (establishing contracts in forms of customer order is the starting point of the processes). We use this case as a context to carry out a feasibility study to demonstrate the BTF can be applied in contract-driven service-oriented processes, which validates our BTF design from the previous chapters.

8.1 Introduction

Today's business processes have become increasingly complex. On the one hand, the close collaboration of multiple parties on a business process is very common, which brings with it many concerns on issues like contractual bonding, application integration, and data security. On the other hand, the structure of a business process can grow into a very large hierarchy, which brings concerns about execution, management, and monitoring. In the previous chapters, we have proposed ATC and TxQoS approaches to build up a transaction framework BTF for complex business processes in the contract-driven and services-oriented context. The BTF is developed to support processes that may cross organizational boundaries and integrate applications over heterogeneous systems. These processes can be seen as a hierarchy, consisting of processes and sub-processes at all levels from the perspective of a process view. According to the service view, these processes meanwhile can be seen as composite services with (part of) the execution details not visible from outside.

During the design process, we carried out limited case studies to inspire and support the concepts and ideas of the BTF. Case studies are commonly used as a research methodology for explaining and describing a concept. During the initial phase of the XTC project, and guided by our research questions, we first studied a case study to further clarify the research scope and base the initial results on the case study. In Chapter 3, we introduced an e-advertising process (shown in Figure 3.1) where customers order commercial online-advertisement from one of the biggest Web-based media company in the Netherlands. As a starting point for our research, we identify the problems of this case regarding transaction management. After the analysis, we concluded that an approach for business-level understanding, to ensure technical reliability of the process, is needed. Following this conclusion, we continued to address the problem identified from the case study and develop the concepts, scenarios, life cycle, and abstract architecture. This is what we call the TxQoS framework presented in Chapter 3, 4, and 5. The approach of TxQoS is generalized from the e-advertising case and can be applied in similar cases (where contract-driven and service-oriented processes prevail). We go on to address the construction of a business transaction framework in Chapter 6. ATCs are proposed as reusable constructs to build such a transaction framework which needs to be flexible and comprehensive to support today and tomorrow's business processes. The ATC approach is demonstrated with a travel agency example, which is used very often used in the literature as a mature application of Web services composition/choreography.

In this Chapter, we apply the research result from the previous chapters (i.e. TxQoS and ATC approaches). We have conducted a case study on e-advertising in a relatively smaller scale than this one. The basic idea of the case study method is to analyze the observations to sort out various relationships between the case entities and our research. The analysis in turn can lead to findings about service/process reliability when applying our research in the case. In the case study in Chapter 3, the analysis has led to findings on unbalanced views, which then clarified our research problems. In this case study, we first follow the same case practice as in Chapter 3, by listing our observations on the case entities and analyze them from both process view and service view. Then based on the analysis, we are able to further study the feasibility of our research result.

Within the case study, we focus on feasibility study to examine how the BTF can be applied in real-life business environment in the context of contract-driven and service-oriented processes. As pointed out in the introduction of the thesis, the main contribution of our research lies in design and development of the concepts, mechanisms, life cycles, and architectures. So in this case study, we make reasoning at a conceptual level instead of real technical implementation. We take a large-scale telecom B2B project that integrates heterogeneous applications across organization boundaries. We only elaborate on the most relevant part of the project for feasibility study, i.e., we focus on the scenarios, the business processes, the participating parties, and the high-level architecture of the solution.

Next, we give an detailed overview of the telecom project in Section 8.2. The case is introduced from both the process and service views, and analyzed using a similar case study method used in Chapter 3. In Section 8.3, we review the results from the previous chapters, and explain how they can be applied in the project. The chapter ends with a summary of the case studies in Section 8.4.

8.2 Case Description and Analysis

The Epacity project was delivered for one of the largest telecom companies in Europe (hereafter, the name 'Tele' will be used to stand for this company). The company Tele applies advanced technologies and products to streamline and optimize their business processes, striving for a high operational efficiency. The main focus of the Epacity project was a system, application, and process integration to enable fully automated B2B order and billing functionality.

The business process starts from business customers ordering broadband Internet and/or telephone equipment and services via the Web. During the order execution, specified routers and/or other equipments are installed and configured by a third-party company. In the end, customer orders are fulfilled, and bills are automatically processed. With rapid development of telecommunication technologies, the telecom company has been constantly improving the variety of their products and services to serve the customer's need. Therefore, the processes and applications in this case need to be updated from time to time, which leads to constant new releases over the past years. This asks for a system integration approach that is both flexible, to allow constant changes, and extensive, to enable new products/service offers on top of current reliability offer.

NGOSS (New Generation Operations Systems and Softwares) is a widely used term in the Telecom industry, referring to the business support systems, which are enabled by IT. According to the TMForum (http://www.tmforum.org), an established thought-leader in Telecom industry, the key principles of

NGOSS are: 1) Separation of Business Process from Implementation; 2) Loose Coupling of the applications; 3) Shared Information/Data Model; 4) Common Communications Infrastructure; and 5) Contract defined interfaces as an extension of API (Application Programming Interface) specifications.

In the Epacity project, the desire to conform to these principles, and because it is imposed by the requirements of flexibility and extensibility, a SOA based integration framework CTF (Configure-To-Fit) was designed and implemented. This project is very appealing to our research, as it exhibits the essential properties we seek for in a case study: multiple organizations, customer orders in the form of agreements/contracts that trigger the process execution, hierarchical structure in process composition, service-oriented architecture design, long-lasting execution, and heterogeneous applications. Here we describe only part of the Epacity solution that is relevant to our research. The purpose of the case study is to validate the general feasibility of BTF in our research context. So here we mainly examine: 1) If the BTF is applicable in this real-life project (which is a typical case of our research context, i.e. contract-driven and service oriented); 2) If the application of BTF provides a basis to enhance process reliability as it supposed to deliver.

To achieve the goal of the BTF feasibility study, our areas of focus in this case are the processes, services, and the architecture. Parties, roles, applications, and design requirements will be briefly explained, as they are necessary to understand the case. The remaining elements of the project, such as implementation, deployment, testing, and system integration may be mentioned occasionally, but are out of the scope of this thesis.

8.2.1 Architecture

The integration project, Epacity, aims to provide an automated end-to-end solution without human intervention during the workflow execution. It needs to integrate applications (such as ordering, CRM, and billing) from Tele's internal departments (e.g. business market and customer support) as well as its external parties (e.g. company responsible for installation of routers). It is implemented in a best-of-breed manner, which means various software, hardware, and service vendors are chosen for their best-known products and reputation in some domain to build solutions. Therefore, integration becomes vital and more complicated than the single-brand approach, which has the internal consistence between applications. The architecture design in this case is service oriented with the focus on integration of the functional components. The basic idea is Configure-To-Fit (CTF) and the role of CTF expands over years.[1] As stated in the architecture design document, "The project evolves from pure ordering solution to a more complete integration solution that combines the currently-in-place process with data integration and composite application attributes".

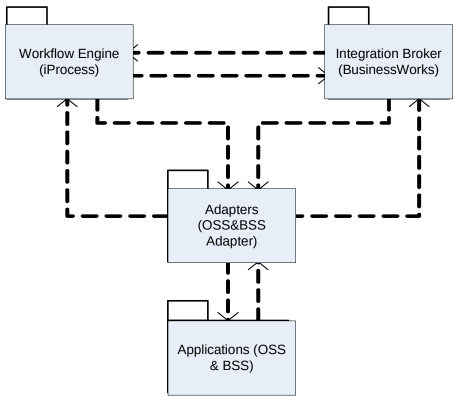

Workflow Engine (iProcess)
Integration Broker (BusinessWorks)
Adapters (OSS&BSS Adapter)
Applications (OSS & BSS)

Figure 8.1: Epacity CTF Architecture

The current integration approach allows for flow control, and it supports multiple transport protocols. A sketch of the Epacity architecture is shown in figure 8.1, where 4 main components are connected for integration. The workflow engine iProcess and the integration broker BusinessWorks (BW)[2] orchestrate the OSS/BSS applications via adapters. OSS (Operational Support Systems) refer to telecom applications. OSS of telecom service providers manage network services, such as maintaining inventory, provision access, and configure components. BSS (Business Support Systems) manage daily business operations such as ordering, billing, and customer support. These OSS/BSS are integrated by a service bus, in this case BW, through various adapters for messaging coordination. The BW is the core component, which acts as the service broker as well as the message broker. It can be further decomposed into several components, with most of them adapters as shown in Figure 8.2. These adapters (e.g. Siebel adapter connecting BW with CRM application Siebel, ECCO adapter connecting BW with ordering application ECCO, etc.) ensure the communication of the data from heterogeneous systems, and are updated once the data structure and parameters of an application are updated. For instance, if in the CRM application the

[1]CTF is a intelligent property belongs to the consulting firm that delivered the Epacity series project. Therefore, introduction to the project refers to proprietary design documentations is not available in the references list.

[2]iProcess and BusinessWorks are registered brands from TIBCO Software Inc.

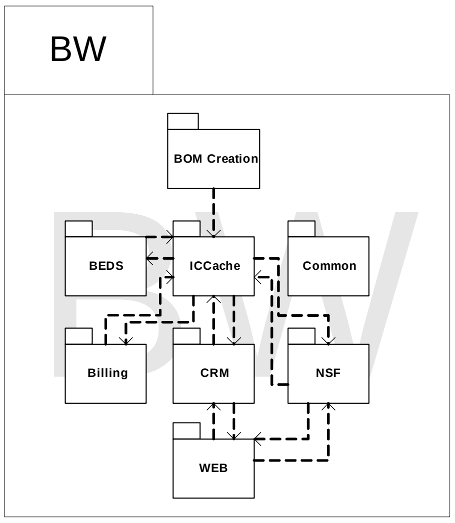


Figure 8.2: BW architecture

structure of a customer's account has been changed to include more fields, then the Siebel adapter needs to be updated accordingly.

8.2.2 Dual View on the Case

In Chapter 3, we have introduced the dual view and applied it to the e-advertising case. In this case, we continue to use the dual view to analyze the case. We first take a process view and introduce the processes involved in this project. Next, we take a service view and introduce the service composition. Then, based on the observations we make, we analyze it in preparation for the feasibility study, which is discussed in the next section.

The integrated processes allows business customers to configure their request for Internet services/equipments such as speed, timing, and capacity by themselves, and then submit to trigger the process via an order management application (which in this case called ECCO). Business customers here are those organizational users who have the access to Tele's portal. Via the portal interface, business customers can specify and submit orders. When receiving a customer order, the CRM sub-process is initiated to confirm the order and create an account if it does not exist yet within the CRM application domain. In this case, the order, using the XML format, arrives in Siebel, which is a CRM application. Upon receiving the CRM request, relevant actions are taken to ensure the customer order is fulfilled. These actions include IP address allocation, router installation and configuration, customer communication, as well as change management in case of exceptions. In this case, exceptions can include a change in the order specification (such as the date the customer wishes to shut down the old site, or desires a different speed, etc.), or any unexpected delay along the process. In the end, when an order is fulfilled, the billing application is set to work automatically to bill the customer.

After the initial delivery of the project years ago, the relevant processes and system applications have been updated constantly in order to accommodate the changes from the business side (e.g. new products and new functions). Therefore, this project requires a continuous effort in maintenance, and is updated or redesigned to have new releases. So there may be a slight difference between our description (based on the 2008 release) and its current situation. Below we introduce the case by looking at the process view:

Process View

In this project, the integration takes place in a TIBCO environment. Here the core is an information bus, Business Works (BW), which connects to all other applications. There are a set of business processes executed in the TIBCO workflow engine, iProcess, which sits next to the BW. We call the executing processes ‘current flows’, which means they have been deployed with platform parameters and can be triggered anytime. In comparison, there are future flows in this case, which are updated at process level upon a change requests, but not yet deployed and tested. The current flows include 7 main processes: ‘Create Site’, ‘Change Site’, ‘Move Site’, ‘Cancel Site’, ‘Terminate Site’, ‘Create Special Site’ and ‘Create VRF Site’. Each process describes a scenario corresponding to a specific type of customer request.

Our documentation effort on iProcess diagrams resulted in a set of three-level plain process models. We do not show the screen-shot of iProcess diagrams because they contain lots of dummy steps that have no business meanings (e.g. for logistic of the flows), and are redundant in semantics. Most of these dummy steps are placed in parallel with the other semantic steps as a modeling technique, so that each of the semantic step can be optional. In this chapter, we present a simplified version of the diagrams for understanding and analysis. All dummy activities are removed. Interested readers can take a look at the full set of current flows in this project in Appendix B and C, depicted by the three-level processes models.

The process models are reorganized in structure, because in the display window of iProcess, the original sub-processes are organized per domain instead of per process. In addition, the bottom level NSF (Network Service

Factory) activities are not shown directly in iProcess figures but in the scripts of the activities. These NSF steps modeled our process diagrams in the same way as we model the other levels of activities. A set of process diagrams describing all workflows in this case and including all sub-processes, is attached in Appendix A and B. Please note that those process diagrams have been modeled on the basis of actual flows deployed and executed in the workflow engine iProcess, instead of the diagrams in previous chapters modeled on basis of business processes management theory. Therefore, the processes represented by the diagrams are not necessarily optimal in theory but more a trade-off between the ideal business process, and environment constraints. For instance, sequential steps are predominantly used, although according to business semantics those steps can be modeled as parallel.

The trigger of these processes is a customer request submitted via a Web interface in the form of contract-like order. Then, a confirmation is sent upon an order, marking the establishment of a contract. Functional activities are depicted as the first-level steps with the names in their natural language. In the middle level of the model we see the decomposed application actions, indicating which applications are invoked and what actions are taken to realize the corresponding top level activities. There are four application domains here: CRM (Customer Relationship Management), BILL (i.e. Billing), NSF (Network Service Factory), and CORE (i.e. database applications). The bottom level is the specific 'NSF' (Networked Service Factory) level for NSF type of actions only. The middle-level steps are mandatory as they are the decomposition of the top-level steps. However the bottom level steps are not mandatory and exist if the application domain is NSF.

Let us take the 'Move Site' process, for example, to explain the process diagrams. Figure 8.3 depicts a main process, in case a customer moves its office/factory location and therefore, requests Tele to shut down the Internet/phone services of the old site and provide services to the new site. Two threads start upon the request. In one thread, a customer account is set up with the service status set to active in the CRM domain. Meanwhile, a billing account is set up with the service status set to active in the BILL domain. VPN (Virtual Private Network) resources are prepared afterwards, and include the access point and the router installation. Here a router is installed by a third-party, which is shown as the step 'Schedule CPE Install'. In the full diagram in the Appendix B, we can see this step is supported by a middle-level step, 'create a TPIP (Third Party Installation Platform) WO (Work Order)', in the CRM domain. In this scenario, time is an important factor to consider, because the operation of the customer's business needs non-breaking telecommunication support, which means no down-time interval is allowed. This is solved in reality by configuring overlapped service

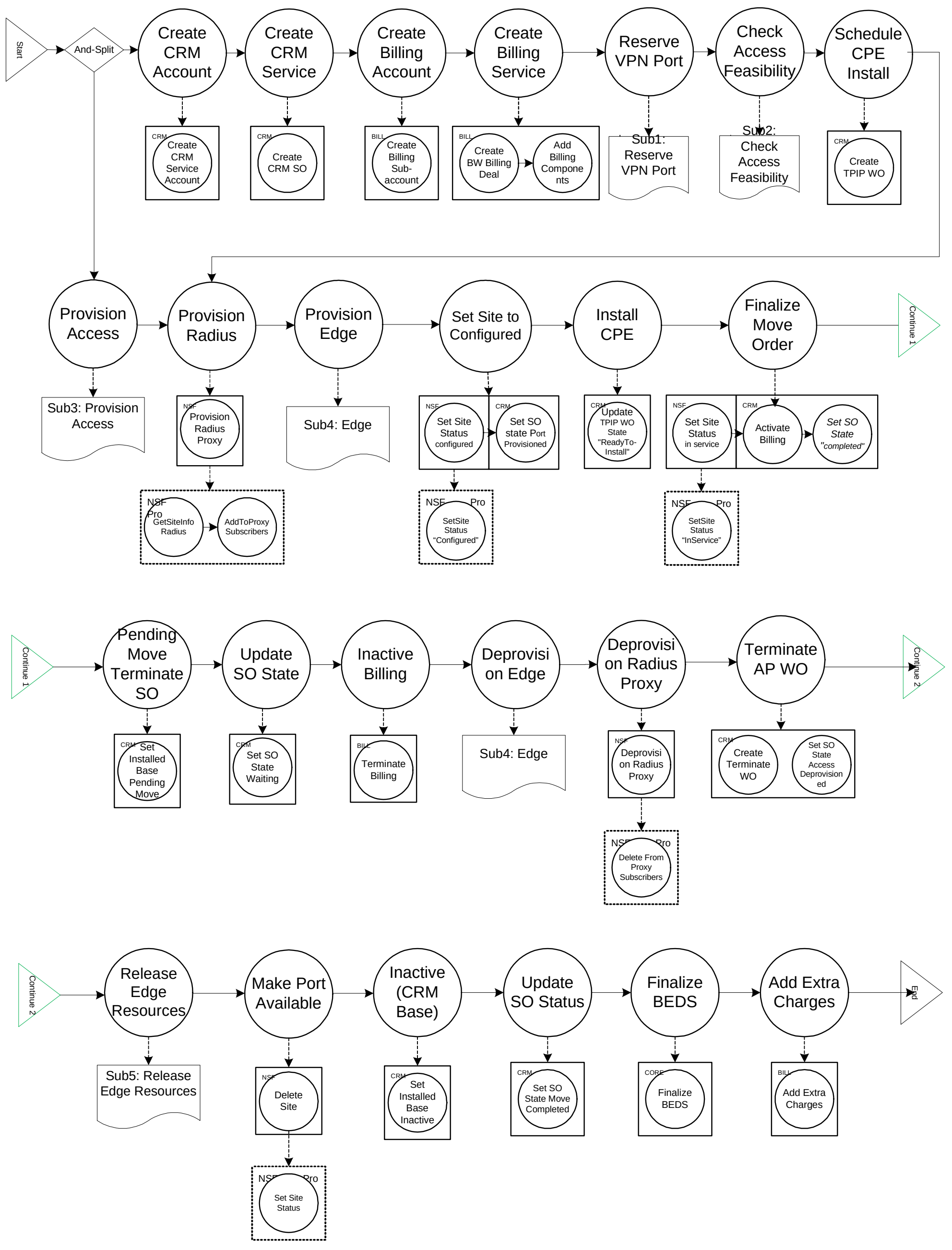


Figure 8.3: Epacity Process: Move Site

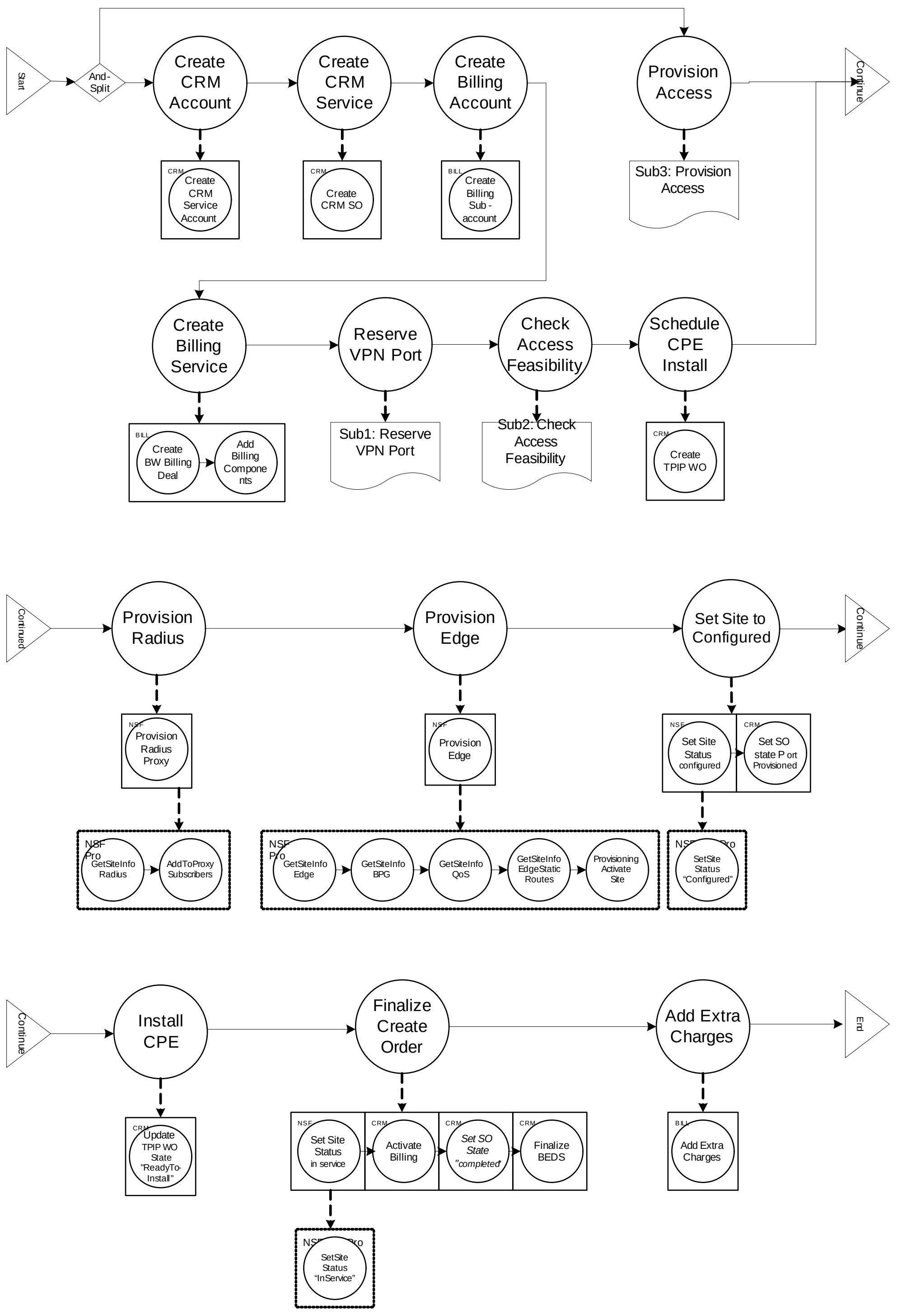


Figure 8.4: Epacity Process: Create Site

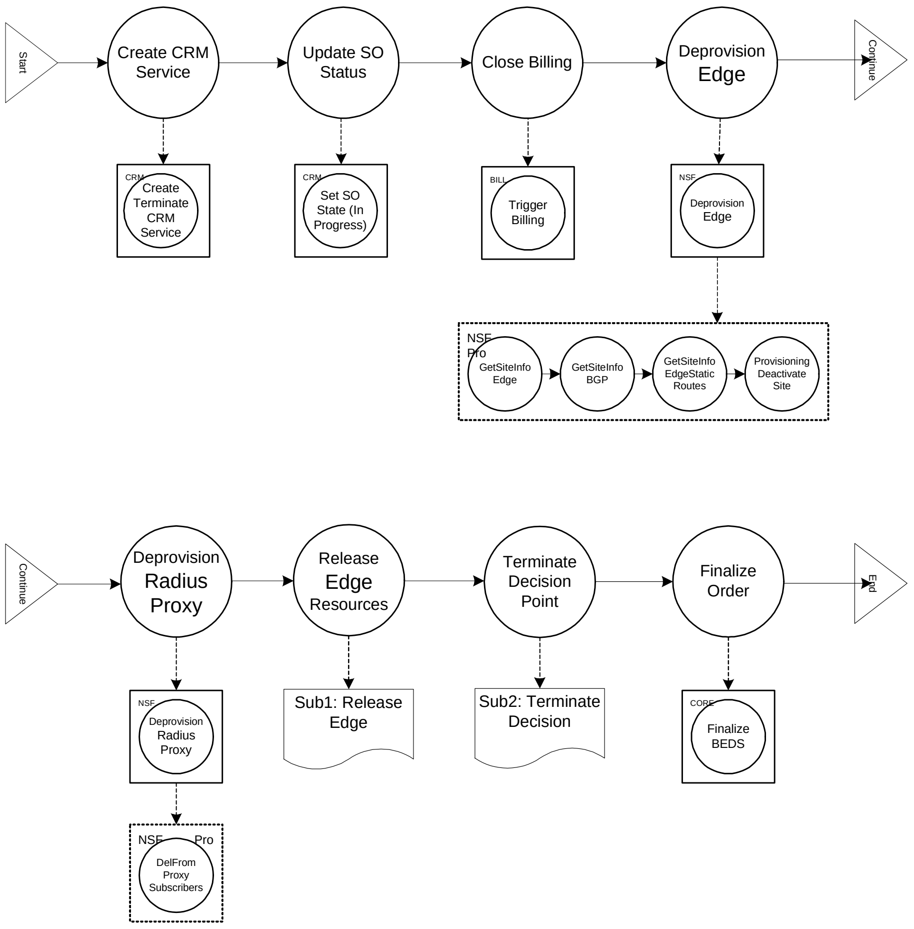


Figure 8.5: Epacity Process: Terminate Site

days between the end of the old site and the start of the new site. As mentioned above, there are 7 main processes currently deployed in the workflow engine. This 'Move Site' process actually combines the other two main processes – 'Create Site' in Figure 8.4 and 'Terminate Site' in Figure 8.5. The full-level and complete set of processes are attached in Appendix B and C for interested readers.

Service View

In the past years, the role of Configure-To-Fit has expanded beyond the original multi-step processes, and now includes data integration (for messaging through all heterogeneous applications) and an integration model (which consolidates application calls into higher-level services). The Epacity architecture covers all common integration models (i.e. process integration, data integration, and composite application) by employing SOA. Besides flexibility and vendor-independence, the SOA-based CTF also enables service abstraction, asynchronous and autonomous messaging, and service orchestration based on common messaging bus (i.e. implemented with Rendezvous[3]).

Figure 8.6 gives an overview of the service architecture. We can see that the 'Broker Services' box is the hub to integrate all other services. Each service is implemented by some specific vender application(s) as indicated after the service names. This is an overview of all services in the Epacity project, which include both functional services and system services. We use a dashed box to depict system services so as to distinguish from functional services, as they do not have business semantics, and are invoked by applications instead of humans.

Functional services are Fulfillment Services, Virtual Services, and User Facing Services, which provide functions to meet business requirements. However, these services serve different user groups. Fulfillment services are provided by various telecom BSS/OSS systems to support Tele's business operations. For instance, inventory service, implemented with a Cramer product, provides a user interface specialized for storing, retrieving, and displaying information about networks. User facing services are all implemented with the WebLogic product, providing an interface for Tele's business customers to enter and monitor their orders on-line. These services and their interactions in Epacity can be seen as the service view on the case.

System services here include broker service and workflow services (referred as CTF Core services in project documentations). These services prepare an environment for the functional services to execute. Broker service, imple-

[3]Rendezvous is a registered trademark of TIBCO Software Inc.

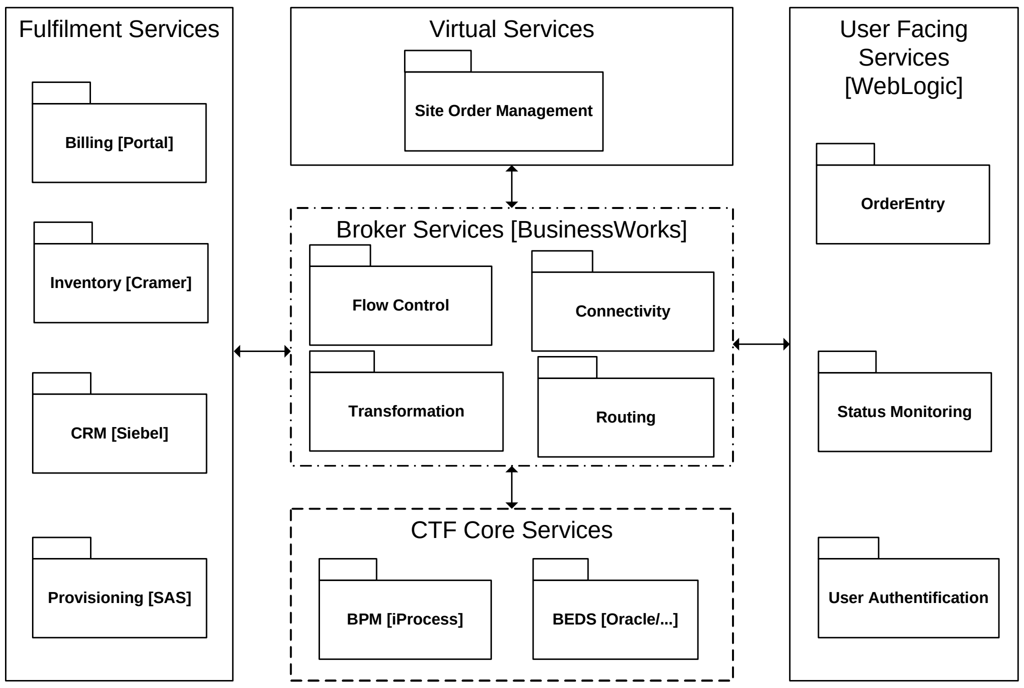


Figure 8.6: Service View on Epacity

mented using TIBCO's BusinessWorks product, enable a secure and robust integration. Workflow services, implemented with TIBCO's iProcess product and Oracle's product, provide logistic and database functions for the functional services. In other words, the CTF Core services enable smooth workflows to realize fully-automated business processes.

As introduced in the process view, if we look at the second level of the process diagrams (all diagrams available in Appendix B and C, we can see that each step is boxed and labeled with capital letters (which can be 'BILL', 'CORE', 'CRM', 'NSF', or 'BEDS'). These labels indicate the service domains in which these steps take place. For example, the 'CRM' box means a CRM service that is implemented by Siebel product in Epacity. Together, the processes and services form the dual view on this case.

8.2.3 Observation and Analysis

The Epacity project is a B2B case focusing on integrating heterogeneous applications by employing SOA, BPM, and Web technologies. All together, 7 processes(workflows) are deployed in the workflow engine iProcess, which are illustrated in three-level details in Appendix for interested readers. There are three parties involved in this case: Tele, Tele's business customers, and third parties which providing equipments and/or services to Tele. Contractual relationships exist between Tele and its customers, as well as Tele and its suppliers.

In this case study, we limit our study on the contracts between Tele and its business customers, and exclude third-party contracts. The contracts submitted by its customers trigger the business process. Such contracts are established and submitted on-line, which in this case are called 'customer orders'. These orders specify the Tele's products/services (e.g. broadband service for an office building) and the non-functional qualities (e.g. QoS). For example, a 'Move Site' web order (in Dutch) is shown in Figure 8.7 (placed under the business entity of 'Contract'), which we can see as a contract between Tele and a customer.

Technical Entities: Process View

First we examine the technical elements for transactional analysis.

Process: There are a number of processes investigated during the case study (interested readers can read Appendix B and C for the whole collection). They are complex processes, designed following SOA principles and triggered by customer order, a form of contract. So we see this case as typical service-oriented, contract driven processes that cross organizational boundaries.

Activity: Activities are unit actions which realize certain functions. For example, at the first level in the process model are activities with business semantics in Figure 8.3. Each activity has one or more actions, and each carried out in one application domain. In this case, most activities are automated, and consequently few activities are performed by humans with the help of applications, such as 'Create CRM/Billing Account'. An activity results in the change of the state of an application.

Flow: Most of the activities in the processes of this case execute in a sequence. In fact, some activities could have been modeled as parallels, reducing the execution time. In this case they built the workflows largely in chains for the following reasons. First, there is hardly any demand on a fast process execution. For instance, as long as the delivery of the new site is ahead of the agreed date, it does not matter if the billing account and service account are processed in parallel or in sequence. Second, the chained flows do not require much effort in terms of control and coordination, therefore lowering the risk of breakdowns during the execution phase.

Process Specification: A process specification specifies process execution logic (i.e. which step in what order), and can be used for transaction design. It is a documented representation of a process abstracting its attributes for deployment. In the Epacity case, a process is a workflow that is designed for the realization of a function.

Host: A host in this case can be the Tele company and its business customers (various companies). Tele hosts a major part of the processes, while its customers host processes to enable web-ordering.

Business Entities: Service View

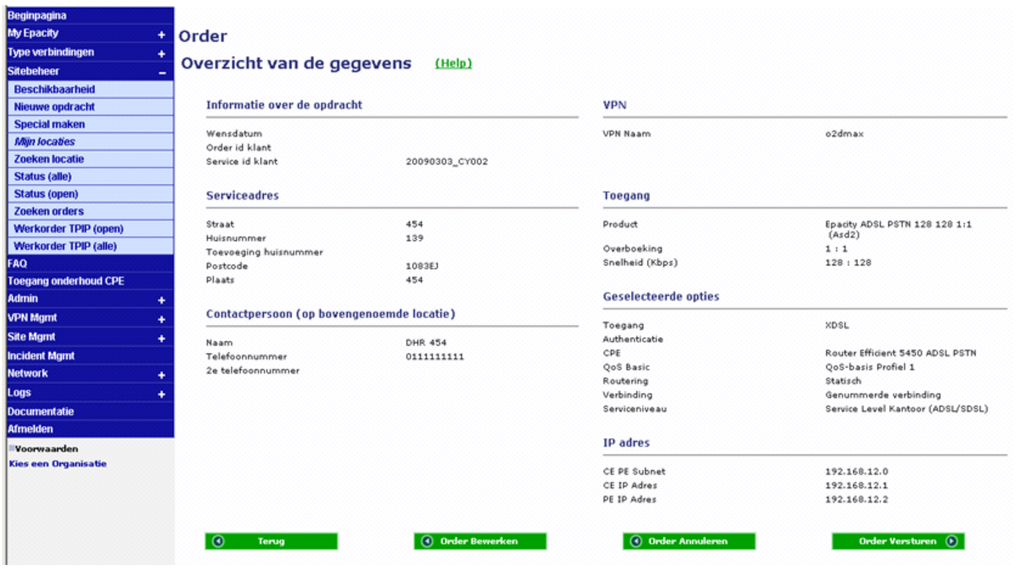
Next we examine the elements for business analysis in a service context. Here we use 'business' to describe elements that do not concern the technical details of underlying systems which appear as common elements in business collaborations:

Party: Three parties are involved in this case: Tele, the business customers of Tele who make orders on-line, and Tele's partners who perform some of the activities (i.e. router installation) as agreed beforehand.

Contract: There are two types of contracts relevant to the case. One type is the contracts established outside the scope of the our research. For example, most of the customers in this case are already contracted with Tele, and already own a business account allowing for internal allocation of network resources. In addition, Tele has a long-term relationship with third parties, such as a company who provides a router installation service There must be contracts for this outsourcing business. The other type of contracts are established in the form of online-order specifications, which trigger the processes in this case. This includes a customer-specified order submitted via Web requesting, a Move Site service shown in Figure 8.7. As we can see from the screen snapshot below, it is a summary of what is previously specified before the sending action. Upon clicking the button 'Order Versturen', the 'Move Site' process starts to execute.

Service: In Figure 8.6, we see the overview of the services. The case can be viewed as a series of interacting services integrated through a broker service. Different services usually demand for a different transaction support. For instance, the 'BPM' service handles the workflows, and it therefore requires long-lasting transaction management. Meanwhile 'BEDS' service handles the database update, and requires atomic transaction mechanisms. In a service-oriented context, the services are implemented with various vender-specific applications and provide only interfaces to the outside for invocation. Thus, it enables flexibility and agility by allowing changes to be made to the applications.

Role: There are two roles a party may play: service provider and service user. For each service, one party plays one role at a time. For instance, Tele provides the 'MoveCite' service acting as a provider, and invokes the third



Order

Overzicht van de gegevens

Informatie over de opdracht

Wensdatum	
Order id klant	
Service id klant	20090303_CY002

Serviceadres

Straat	454
Huisnummer	139
Toevoeging huisnummer	
Postcode	1083EJ
Plaats	454

Contactpersoon (op bovengenoemde locatie)

Naam	DHR 454
Telefoonnummer	0111111111
2e telefoonnummer	

VPN

VPN Naam	o2dmax

Toegang

Product	Epacity ADSL PSTN 128 128 1:1 (Asd2)
Overboeking	1 : 1
Snelheid (Kbps)	128 : 128

Geselecteerde opties

Toegang	XDSL
Authenticatie	
CPE	Router Efficient 5450 ADSL PSTN
QoS Basic	QoS-basis Profiel 1
Routering	Statisch
Verbinding	Genummerde verbinding
Serviceniveau	Service Level Kantoor (ADSL/SDSL)

IP adres

CE PE Subnet	192.168.12.0
CE IP Adres	192.168.12.1
PE IP Adres	192.168.12.2

Figure 8.7: Move Order Contract

party's 'Installation' service acting as a user. Note that Figure 8.6 only shows the services within the boundary of Tele and does not reflect any third party services.

Reliability Agreement: We hardly observe specific agreements regarding transactional reliability. What Tele's customers know are mainly functional guarantees, such as bandwidth and the available date, which are specified in the contract.

Analysis

There are 5 types of applications: CRM, Billing, Core, BEDS, and NSF (Network Inventory Management) involved in the workflows as shown in the business process diagrams. These applications are supplied by various vendors, or developed in-house, based on a best-of-breed strategy. They implement the services illustrated in Figure 8.6. The SOA-based integration model focuses on information exchange across participating applications and software components. With coordination from the BPM component (i.e. iProcess workflow engine) and the integration component (i.e. BusinessWorks), these heterogeneous applications work together to enable the long-lasting cross-organizational processes triggered by web ordering, and then flow to the site configuration until billing.

In the above subsections we have revealed the process and service entities in the Epacity case. They are very similar with the entities and relationships we observed in the previous e-advertising case study shown in Figure 3.4 in Chapter 3. We can make an assumption that the missing of unambiguously specified reliability agreement is a common situation in long-lasting, service-oriented, and contract-driven processes. One reason is that such processes all have some form of agreements specified in the contracts to guarantee some confidence in process/service reliability, although ambiguously. Another reason is that the customers, or in other words, the service users, are usually in a weak and passive position compared to the providers.

As a result of the vague reliability guarantee and the dominant position of the provider side, the users are not able to fully integrate providers' services/processes well into their own business processes. Therefore, we continue the feasibility study in the next section to investigate how the situation can be improved by means of a Business Transaction Framework that helps today's companies move towards becoming networked enterprizes.

8.3 BTF Feasibility Study

We have introduced the architecture, provided a dual view of process service, and elaborated on the project entities of the case in the sections above. With a clear overview of the flows, parties, scenarios, and architecture design, we apply the research findings from the previous chapters to the complex real-life case. There are many other important issues in the case, such as implementation, deployment, testing, and management. However, they are not relevant to our research objective and will not be discussed here. What we focus on here is the operational side of the contractual relationship between the provider and users, and how a BTF can be applied in this context.

The processes in this case span across the boundary of Tele and reach the interfaces of applications resided in third parties. For instance, the router installation is not handled by Tele. A partnered company, which is delegated by Tele, send engineers to the sites to install and configure routers. Inside Tele, the applications involved in Epacity belong to different domains, such as CRM, billing, etc. We can view business customers as service users of Tele whose orders trigger the execution of Epacity processes. The company Tele in this case acts the role as the service provider at a global level. Contrastingly, at the lower level, outsourcing activities may take place. For example, Tele acts the role of a service user, while the router installation company acts the role as the service provider. From the perspective of Tele's business customers, the only concern is the reliability of the global level process. However, from Tele's perspective, the reliability of the third party process is important to specify the service agreement with its customers.

As a result of this case study, we demonstrate that the TxQoS-aware business transaction framework, which is inspired by the e-advertising case, can be used in the generic context of service-oriented, contract-driven business processes across organizational boundaries. The research result, which includes the problems we identified as well as the framework to address the problems, should be applicable to the Epacity case. In other words, the problems we identified should map the problems we observed in the case. Furthermore, the solution framework BTF should be able to address the problems in a real-life setting. Given the properties of our research, which stays at the conceptual level, the case approach is to practically validate the utility of our results (i.e. feasibility study). We start by applying the TxQoS approach.

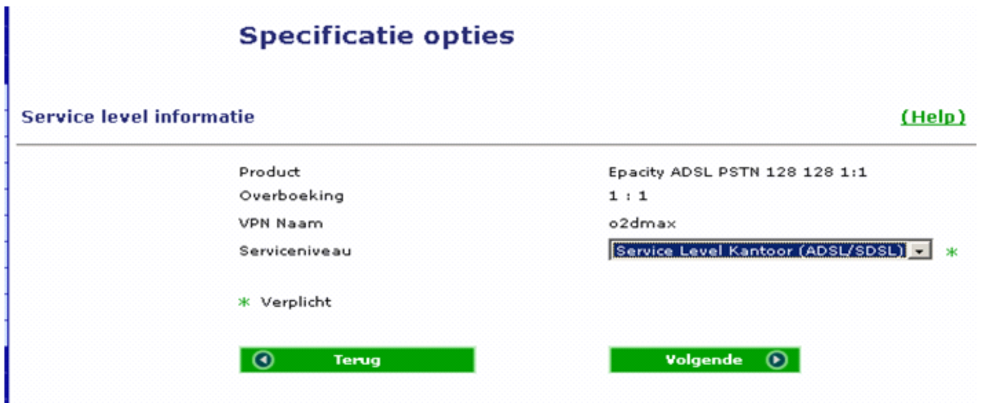
8.3.1 Apply TxQoS in Epacity

Offering TxQoS-enabled contracts based on its current products/services is a benefit for both Tele as well as its customers. An enhanced customer impact on process execution reliability is expected. The strengthened customer impact can lead to a move from the current provider-dominant pattern towards an equivalent one (illustrated in Figure 7.3). Tele will benefit from the realization of a more dynamic enterprize network. In this section, we discuss how, in this case study, the TxQoS approach can be applied in order to enhance process execution reliability.

In Tele's current ordering interface, as is shown by the sample order screenshot in Figure 8.7, the network QoS is specifiable, along with a few other service details. Through the ordering system ECCO, customers can specify the orders in detailed figures and specifications. The specification of an order takes place step by step, starting usually from the customer data input until all details are specified. For instance, a screen snapshot of one step during the order specification is shown in Figure 8.8. As the service provider, Tele can offer different levels of quality for its consumers to choose. Here, the differentiated offers are for its telecom products and services, for instance the bandwidth if ordering broadband Internet service. Once submitted, the agreement is established which we can see as one type of e-contract. Both Tele and its customers can view the status of the order for monitoring purpose during the order fulfillment (i.e. process execution).

The key finding from the e-contracting case in Chapter 3 is the lack of unambiguous specification on transactional process quality. To address this problem, we have proposed a TxQoS framework to ensure transactional quality by means of establishing TxSLAs. The core of the TxQoS framework consists of the FIAT attributes used to specify process reliability at the business-level, which supplement the technical terminology of transaction management at the back end. Using the process shown in Figure 8.3, for example, yields us with the TxQoS scenario shown in Figure 8.9.

From the above figure, we see that the TxSLA is the key when implementing the TxQoS approach in the case study. Adding a TxQoS specification page to an ECCO order can be one way to implement the idea of TxSLA. The customers can choose from the TxQoS offerings in the same fashion as they specify the other service requests (e.g. download speed, upload speed, overbook rate, etc.) from the menu. Following the above scenario, where Tele provides TxQoS offers, we assume that contracts(i.e. specified, agreed, and submitted orders) can be established with an enclosed TxQoS specifica-


Specificatie opties

Service level informatie (Help)

Product	Epacity ADSL PSTN 128 128 1:1
Overboeking	1 : 1
VPN Naam	o2dmax
Serviceniveau	Service Level Kantoor (ADSL/SDSL) *

* Verplicht

Figure 8.8: Epacity Case: Order Specification

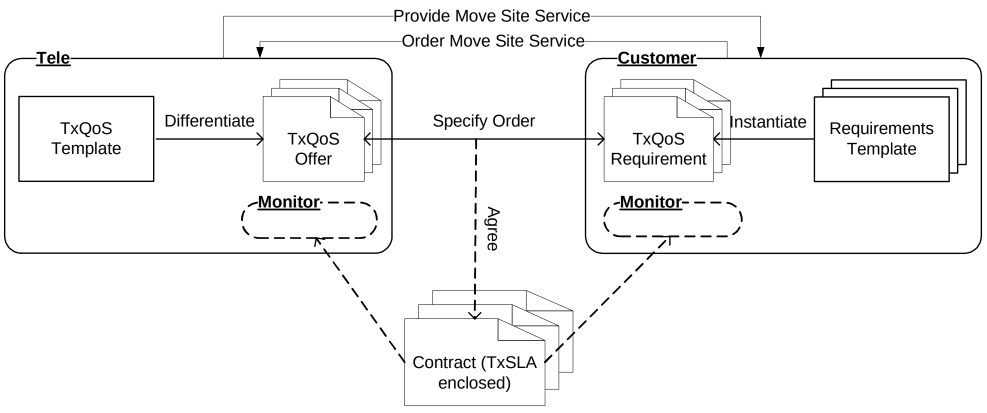


Figure 8.9: Epacity Case: TxQoS Scenario

tion. As stated, upon receiving a customer order, a corresponding process is triggered to instantiate a running instance. This applies to all processes in this case, including CreateSite, CancelSite, and the rest processes (attached to Appendix B). These contract-driven processes can well support our idea of specifying transactional reliability via business contracts.

Consequently, we demonstrate the TxQoS feasibility by an example of the 'MoveSite' process. The logic to apply TxQoS in the other processes in this project should be the same as this one. Besides the contract establishment, the next point of focus when applying the TxQoS approach is the specification of the transactional quality of the 'MoveSite' service via FIAT attributes. As defined in Chapter 4, Fluency, Interferability, Alternation, and Transparency are designed to indicate the process/service reliability at a business understanding level. We have pointed out that adding one or more pages through the ordering system ECCO is feasible. Below, we demonstrate what can be the options listed on the TxQoS ordering page, and how Tele support this idea by transforming its current systems.

Fluency

In Section 4.3.2, Fluency is defined as an attribute specifying a function or a numerical value, that measures the smoothness of service execution by computing the probability of the breakdowns happening in the future. The probability of future performance is based on statistics collected either from the past, or a test. Here a breakdown is defined as 'the ceasing of a running service so that the execution comes to a sudden end without delivering the intended results, and therefore requiring a fix to enable the execution to continue'. An assumption here is that the distribution of breakdowns along service execution can always fit for a specific statistical model. Another assumption is that the provider and the customer have agreed on the definition of breakdowns.

We take the 'Move Site' process as an example. This process can last for weeks, or even a month. Thus, it is likely to suspend or pause due to a number of reasons, such as a customer request for a cancelation, or due to exceptions/erros during the execution. From a customer's point of view, the top concern of the service/process is having on-time delivery with desired quality. More precisely, in our case, it means the new site with the VPN, network, or other services should be available before the desired date as specified in the order. In addition, the billing of the old site should stop before the billing of the new site starts. Therefore, the customer would like to keep the number of breakdowns well in the safe range (i.e. for the breakdown not to affect the delivery of the new site before the termination of the old one). According to the statistics of testing and/or past runs, Tele should be able to figure out the distribution of 'Fluency' of its 'Move Site' process.

Technically it is feasible to offer fluency as one of the TxQoS indications in contracting options. We have suggested an example of how to specify 'Fluency' in TxQoS offers in Section 4.3.1: assume the statistic of the number of breakdowns during process execution exhibits a Go NHPP distribution, where $f(n)$ is the function to calculate the probability of having n breakdowns using the model. For instance, $f(0)$ is the probability of smooth execution without any breakdowns. This way, the specification can come in the form of '*no more than 2 breakdowns during the execution*' or '*no breakdowns during the execution at the cost x*'. The same methods can be applied here on Tele's processes/services in our case. For example, assume the provider Tele finds out $f(0)$ in the long-lasting process 'Move Site' is very close to 1, which means it rarely breaks down. Tele can provide the maximum Fluency for this process specified as '*Fluent execution*'. While $f(0)$ in another long-lasting process 'Cancel' is around 0.8, then Tele may provide a drop-down style menu in the ordering page specifying various offers of 'Fluency' such as '*x breakdowns allowed*' (here n is an integer between 0 and 3).

The monitoring mechanism of the TxQoS approach can also be applied here. In the scenario in figure 8.9, no intermediaries exist, and therefore the monitor can be placed in the Tele and/or customer systems. After each execution, the newly-obtained data is collected and a performance report is updated. Such a real-time update enables the provider to offer the most appropriate TxQoS offers and equips the users with the up-to-date information to make a choice from available offers.

Transparency

Transparency has been defined in Section 4.3.4 as the visibility of service execution as the basis for user interference. This attribute can be specified in an absolute manner by a set of activities that are visible from outside by the service users, or in a relative manner by a numerical value, to reflect the proportion of visible activities against all activities.

In our case, the processes allow the customers to view some intermediate results from the process execution. For example, the status of the order fulfillment at every stage of the lifecycle can be checked online after login. Here we continue taking the 'Move Site' process as the example to illustrate the feasibility of the FIAT attribute 'Transparency'. From a customer's perspective, Tele provides a 'Move Site' service on request in the form of an online order submitted by its business customers. We specify the transparency of this service/process by a set of visible activities, where customers are allowed to view the intermediate result. Here, we assume that during execution, customers can access the Web interface to view the order details they submitted in the same way they check the order fulfillment status (i.e. account established, site ready, etc.).

$$T_{Move} = \langle\{A, B, C, D, M, O, \}\rangle$$

Here the nodes in the set T_{Move} are taken from the execution graph shown in Figure 8.10 (i.e. the first level of the process model in Figure 8.3): 'Create CRM Account'(A), 'Create CRM Service'(B), 'Create Billing Account'(C), 'Create Billing Service'(D), 'Finalize Move Order'(M), and 'Update SO (Service Order) State'(O). With this transparency specification, the intermediate outcome of the CRM and billing accounts, as well as the execution status of CRM and billing services, are allowed to be seen from outside Tele's boundary. As suggested in the FIAT design, there is another way to specify the transparency attribute, namely through the calculation of a numerical value which discloses the proportion of visible nodes. This vague method of specifying transparency is not suitable in our case, where customers need to know exactly which steps they are allowed to control in the long-lasting execution.

The application of 'Transparency' in the case allows Tele's customers know exactly which part of the process they are allow to 'see'. Thus, the service/process from user perspective are more trustworthy. Meanwhile, the possibilities to adjust the orders to better meet their current requirements increase. The reason may be that customers see the intermediate results, and decide to make a change.

Interferability

We have defined 'Interferability' in Section 4.3.3 as the indication of how much control a user has over the process/service execution. One way to specify this attribute is 'a set of commands from the users to intervene in an activity (viewed as a node in execution path), plus the allowed timings to issue these commands'. We have used the 'Transparency' attribute to specify how many nodes on an execution path can be seen from the outside. Following the method in Section 4.3.2, we define N_t as the transparency set containing nodes (i.e. the activities in the process diagrams) that are visible from the outside, O_t as the operations allowed for customers, and *time* as the timing of the interfering operations upon N_t.

$$I = \{operation(time, activity)\}, \tag{8.1}$$
$$operation \in O_t \quad activity \in N_t \quad time \in \{Rules\}, \tag{8.2}$$

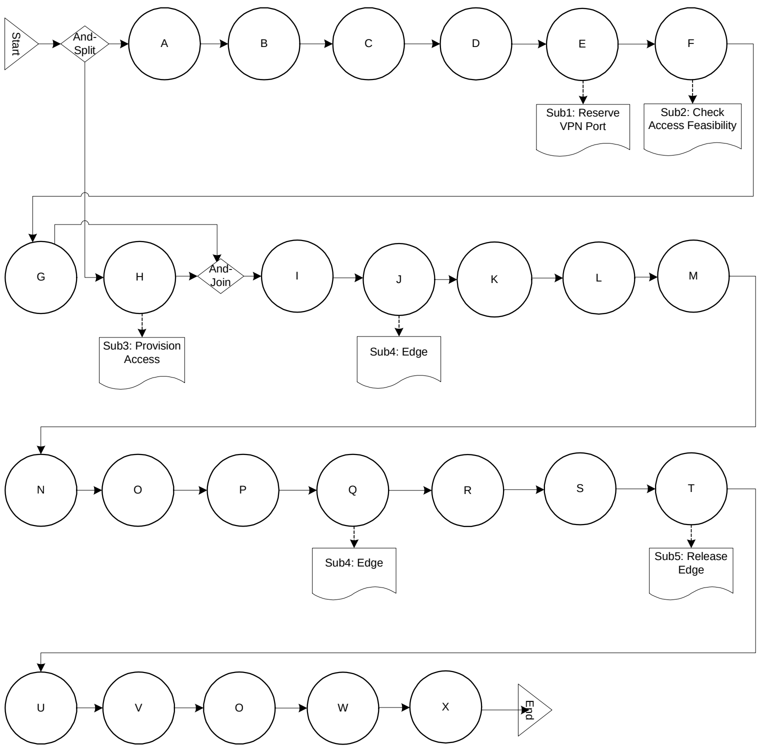


Figure 8.10: Execution Path of Move Site Process

The Interferability in our case can be specified as a set of operations on the activities that are visible to the customers with proper timings. The activities in the processes are executed to provide certain functional services/products to Tele's customers. Unlike some type of processes, where both parties own part of activities and jointly execute the whole, our case is a typical provider-dominant scenario where customers do not see most of the intermediate results and have few control power on the process execution. More specifically, interferability in this case means the power of Tele's customers to cancel and/or revise the orders. During the execution, which usually lasts weeks, the customers are allowed, under some conditions, to revise part of the order specifications.

For instance, we make an assumption that the download/upload speed is revisable before the billing in the 'Move Site' process. Meanwhile, let us assume that the cancelation of the process by a customer is only allowed before the order confirmation the in ordering process, as illustrated in Figure 8.7. This means once the order is specified, confirmed, and submitted (i.e. the contract is established), the 'Move Site' process starts to execute and is not cancelable. Referring to a 'cancelable workflow design', is not a paradox in this case. The 'cancelable workflow design', as we have explained in the case introduction, allows most activities in the processes to be optional by using a parallel dummy activity (not shown in the re-organized diagrams in Appendix B and C) for each semantic step. So in theory, a process is cancelable at each step, from which dummy steps are executed until the end. However, this design-enabled cancelation can only be performed by Tele in case something goes wrong and needs human intervention, whereas the customers have no power to leverage these 'dummy' steps. Below, we can specify the attribute 'I' (Interferability) for the 'Move Site' process:

$$I = \{CANCEL(time, activity)\}, \quad activity \in \{\phi\} \quad time \in \{\phi\} \tag{8.3}$$

$$I = \{REVISE(IFgetProgress(FinalizeMoveOrder) < 1, activity)\} \tag{8.4}$$
$$activity \in \{SetSitetoConfigured, FinalizeMoveOrder\} \tag{8.5}$$

As explained in the FIAT specification method, in this environment timing rules can be specified in two ways: Absolute machine times or relevant times indicating the semantics. The above specification adopts the semantic time rules. Specification 8.3 specifies that a cancelation is not allowed (as the time set is empty). However, specifications 8.4 and 8.5 specify that the command 'REVISE' from customers is only allowed on the activities 'Set Site to

Configured' and 'Finalize Move Order'. Note that the 'Finalize Move Order' is executed at after all previous activities have committed. Thus, the 'revise' command can only be allowed before the activity 'Finalize Move Order' commits. The function $getProgress(FinalizeMoveOrder)$ invokes the system parameters to detect the execution status of the activity 'Finalize Move Order'. The parameters are set as '*-1=not started, 1=committed, 0=started but not committed*'. As explained, the value -1 is a numerical scale for example.

At runtime, if the system detects a 'REVISE' command from a customer, the timing is judged to decide if the command is valid. Afterwards, actions take place accordingly (reject/accept the command) to ensure the 'Interferability' as specified in the order (here we mean a suggested TxSLA in addition to the order in this real-life case). The application of 'Interferability' can give customers more confidence over the long-lasting execution of the Move Site process, as they know what kind of control they have and on which part of the process. Not much effort is needed for Tele to provide the Interferability in a contract, as the actual process remains the same. The reliability of the process, from a customer's perspective, is strengthened.

Alternation

We have defined the 'alternation' attribute as the predefined alternative execution paths, which are used in case of breakdowns happening along the service/process execution. In our case, the workflow design makes each step(activity) literally optional, which is comparable to the 'cancelable workflow design' stated previously. This implies that, for each activity, there is always an alternative step (the dummy step). In theory we can define the 'Alternation' of the processes in this case as all possible combinations of meaningful activities plus dummies (i.e. the complete subsets of the full execution paths). Here, a full execution path contains all meaningful activities at the first level of the process models (where we have purposely removed all dummies).

The pre-defined alternative paths can be specified, according to the specification method of 'Alternation', by a set of graphs containing dummy nodes with some meaningful nodes. Take the 'Move Site' process for example, the execution path G_{Move} can be mapped from the process diagram (shown in Figure 8.3). The execution path of the 'Move Site' process is depicted in Figure 8.10, where each main-level activity in the process is depicted as a node in the graph.

As shown in the diagram, here we use letters A, B, ..., X to indicate the activities: 'Create CRM Account' (A), 'Create CRM Service' (B), ..., 'Finalize Move Order' (M), 'Update SO Status' (O), ..., 'Add Extra Charges'

(X). The order fulfillment actually consists of the fulfillment of two orders 'Create' and 'Terminate'. Thus, the status of the service order needs to be updated twice to release execution status. So we see in the diagram that the node 'O' (i.e. Update SO Status) is executed twice.

As explained in FIAT specification, $N = \{nodes\} \subseteq X$ is the domain of activities, and $E = \{edges\} = \{\langle node_m, node_n \rangle \in X \times X\}$ is the domain of edges. Thus, $G = \langle N, E \rangle$ is the domain of execution graph of the activities in the domain X. Following this method, the G_{Move} (i.e. the execution graph in normal times) in Figure 8.10 can be specified as:

$$\begin{aligned} G_{MOVE} = \langle \{A, B, ..., Y\}, \{ & \langle A, B\rangle, \langle B, C\rangle, \langle C, D\rangle, \langle D, E\rangle, \langle E, F\rangle, \\ & \langle F, G\rangle, \langle G, I\rangle, \langle H, I\rangle, \langle I, J\rangle, \langle J, K\rangle, \langle K, L\rangle, \langle L, M\rangle, \langle M, N\rangle, \\ & \langle N, O\rangle, \langle O, P\rangle, \langle P, Q\rangle, \langle Q, R\rangle, \langle R, S\rangle, \langle S, T\rangle, \langle T, U\rangle, \langle U, V\rangle, \\ & \langle V, O\rangle, \langle O, W\rangle \langle W, X\rangle, \}\rangle \end{aligned}$$

Suppose there is an alternative execution path G_{Alt} for the 'Move Site' process. The path G_{Alt} is selected if a customer requires a revision on the order as explained in the 'Interferability', which can happen at any time ahead of the 'Finalize Move Order'(M). The specified alternative path correspondingly needs to roll back from the node *Rollback* (*Rollback* indicates any interferable node) to the beginning of the process, and re-execute again. The node *Rollback* is to be determined at runtime. It becomes known to the system when the customer command 'REVISE' is issued.

The assumptions we make here are: 1) The interference is a real-time operation, which means it always results in suspension of the on-going execution; 2) Except the meaningful (shown in our process diagrams) and dummy activities (removed from the diagrams), no other intervention activities will be performed. Thus, an interference of 'REVISE' will ask the workflow engine to first roll back from the current node to the first node. We can use *preRollback* to indicate the previous node of *Rollback* in the normal executing graph. A value-set action should be performed to determine which nodes are *Rollback* and *preRollback*. When applying the 'Alternation' specification in our case study, taking the process 'Move Site' for example, we can specify the 'Alternation' in the TxSLA using a graph G_{Alt} as follows:

$$\begin{aligned} G_{Alt} = \langle \{A, B, ..., X\}, \{ & \langle A, B\rangle, ..., \langle preRollback, Rollback\rangle, \langle A, B\rangle, \langle B, C\rangle, \langle C, D\rangle, \\ & \langle D, E\rangle, \langle E, F\rangle, \langle F, G\rangle, \langle G, I\rangle, \langle H, I\rangle, \langle I, J\rangle, \langle J, K\rangle, \langle K, L\rangle, \\ & \langle L, M\rangle, \langle M, N\rangle, \langle N, O\rangle, \langle O, P\rangle, \langle P, Q\rangle, \langle Q, R\rangle, \langle R, S\rangle, \langle S, T\rangle, \\ & \langle T, U\rangle, \langle U, V\rangle, \langle V, O\rangle, \langle O, W\rangle, \langle W, X\rangle\}\rangle \end{aligned}$$

The application of 'Alternation', in this case, allows the customers to understand that the process is still taken care of by an agreed execution logic in case the normal process execution is changed. The change may be the result of a customer interference (e.g. a revision of the order) or have other preapproved causes. This way, customers understand their orders are still being processed. This includes unusual cases, which also enjoy a reliable execution because of the alternative execution graphs.

So far, we have demonstrated, through illustrations, that it is feasible to apply the FIAT specification in Tele's current ordering contract. We also discussed the benefit in each attribute application example when introducing such an extra TxQoS agreement, in addition to the current contract. The application of a TxQoS approach does not ask Tele to technically update the processes, architecture, or business logic. The TxQoS framework implementation, in this case, aims at extending the contract/agreement with TxQoS specifications, and make add-on tools as suggested in the TxQoS architecture. This way, from the customer's point view, the process/services are more reliable as they gain transparency and confidence because of the contractual assurance.

8.3.2 Apply ATCs in Epacity

In the previous section, we have demonstrated how to specify a TxQoS-enclosed 'Move Site' order. In this section we continue to apply the ATC concept to show what ATCs can be selected, and how they can be composed to support the order process reliability.

In Chapter 6, we have proposed the ATCs as the constructional pieces to flexibly compose a transaction schema in support of the process execution. An ATC is an abstracted transaction model taking the form of a XML template for a customized configuration, resulting in a configured and ready-to-enact template following the process specification. According to the ATC life cycle shown in Figure 6.5, the first phase is to design the templates (i.e. to create the 'empty' templates). In the appendix we have given a series of ATC templates; here we take them as the starting point for later steps. The second phase is to select proper types of ATCs for supporting specific process. Taking the 'Move Site' process as the example, according to the process diagram and the execution graph, we can see that the activities are largely executed in sequence, with only one parallel of 'AND-join' at the beginning. Therefore, we select a chain ATC as the global ATC, with the following detailed schema:

From the above ATC schema, we can see that the global 'Move Site' process/service 'S1' corresponds to the global ATC, named 'ATC1'. 'ATC1' is a

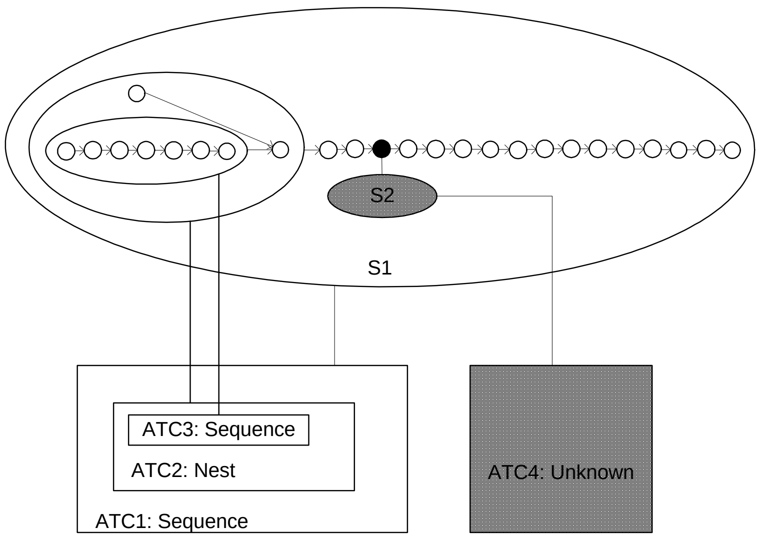


Figure 8.11: ATC Schema in Move Site Process

compound ATC, with the internal structure as a chain. The sub-level ATC, 'ATC2', is the child of 'ATC1'. Moreover, 'ATC3' is the child of 'ATC2'. According to the process structure, we specify them as a nest type of the ATC and a sequence type of the ATC, respectively. Note that they do not correspond to any sub service, and come into being for structuring purposes. The service $S1$ invokes an external service $S2$ (marked as gray), which is provided by a third party requesting a router installation. For this third party, $S2$ may correspond to transaction schema $A4$. However, the provider of $S1$, Tele, is not authorized to see the details of $S2$. In other words, the details inside $S2$ and $A4$ are not visible from the outside. After the schema design, the next step is to apply the ATC approach as a template specification and configuration, so that process chunks get the transactional support accordingly. In our example above, the ATC1, ATC2 and ATC3 can be designed from the general templates, with the knowledge from the 'Move Site' process. A general chained ATC template can be configured into ATC1 and ATC3 respectively. A general nested ATC template can be configured into ATC2. Here we only demonstrate the link between the ATCs and Tele's processes. A full-blown ATC library implementation, containing the complete set of ATCs (e.g. ATC1, ATC2, ATC3, ...) for all Tele's processes, is out of the scope of the thesis. We assume that all these ATCs are organized in a library. Because of the ATC library, the transaction management of Tele's complex processes can be designed in a plug-and-play manner to achieve flexibility and comprehensiveness.

8.3.3 Apply the BTF Patterns in Epacity

In the previous chapter, where we proposed a BTF to integrate the TxQoS and ATC approaches, we analyzed two scenarios: a ‘Provider-dominant’ and a ‘Provider-user equivalent’ one. Using these as one dimension, and crossing them with another dimension of business dynamism, we have created a matrix to distinguish four business patterns, and identify each of them with features and examples. Taking the pattern matrix as the context, in this subsection we analyze the feasibility of a BTF in the Epacity case.

The company Tele has long-term business relationships with its business customers. The profile of the customers are companies who rely on Tele for Internet, telephone, and other telecom services. How they establish contracts with Tele to begin the usage of Tele’s products and services are out of the scope of this case study. We assume that there are business contracts between Tele and its customers regarding delivery, price, customer support, dispute settlement, etc. The orders which trigger the business processes in the Epacity case become agreements once submitted by customers on-line. We see these agreements as the contracts which can be transaction-aware by adding TxQoS enhancements. The business processes are quite stable, with few periodical updates which are requested by the business department in Tele over a period of years. At the customer side, requests for products/services are put through the ordering system via the Web interference, which is provided by Tele. During the long-lasting execution of the processes, the customers have few interference power. They are only allowed to change the product specification, delivery date, etc. but do not directly have impact on Tele’s business processes or service implementations.

From the above observations on Tele’s external relationships (i.e. with its customers or third parties), we identify the Epacity case as an example in the ‘**provider dominant and static**’ quadrant of the business pattern matrix. In this business pattern, providers have more bargaining power than users, and their relations are bonded tightly, with few changes over time. The provider side has a dominant role, i.e. making offers with fixed prices, while the user side passively accepts offers. The users in the Epacity case, which are the business customers and collaborators of Tele, have limited bargaining power and may implement very different technical infrastructures from Tele. Tele’s customers and collaborators are required to use the standardized web interface offered by Tele. We have limited insights into the implementation details of the integration of the standardized web interface with their internal applications.

Compared to the global picture illustrated above, the pattern is different within the company Tele. The parties participating in these processes include customer support, IT, finance, etc. These business units and departments are usually financially independent and have internal business relationships, specifically highly trusted supplier-customer/provider-user relationships. They use the same or similar infrastructures and information resources. They collaborate in a consistent way. Although the collaboration internally still encounters changes over time, due to a number of factors, such as a reorganization, changes on budget, and staff movement, the partnership between stakeholders remains tight. Therefore, we can fit the scenario of Tele's internal collaboration into the 'provider-user equivalent and static' pattern. It is a clear example of this pattern, as the relationship between the independent parties is very close and hardly changes over time. In addition, the parties collaborate in a manner that is more open, transparent, and communicative, compared to the relationship between external parties.

As explained in the last chapter, the benefit of using BTF can facilitate the move of business patterns from a more rigid quadrant to a more dynamic one, as well as moving along with the trend of increasing customer power. Suppose the idea of a BTF is applied in this case. Tele's customers gain more awareness, and can directly make an impact on the processes by asking for more favorable TxQoS offers (e.g. they can increase transparency and interferability, or change the alternative execution path). The increased customer power results in the move from a provider-dominant to a provider-user equivalent. However, as the user side does not necessarily implement BTF, a shift from a static paradigm to a dynamic one may not happen. This implies that we can adopt the life cycle in the 'provider-user equivalent and static' pattern to analyze the phases in the scenario, provided that the BTF is applied in the case.

As we can see in Figure 7.6 in Chapter 7, which shows the BTF life cycle for this pattern of business, TxQoS offers are made parallel to the transaction schema design (i.e. composing ATCs to support the business processes) at the BTF composition phase. In this business pattern, the customers are involved or at least their requirements are considered by providers. This indicates that Tele specifies TxQoS terms according to its capability on the reliability of the processes and the customers' requirements and feedbacks. In the same phase or thereafter, the customers can accept suitable offers or negotiate special requests with Tele, until they reach an agreement by means of TxSLAs. Once the TxSLAs are established and the processes/services start to be executed, the monitoring activities over Tele's TxQoS performance and the TxSLA compliance start. At this phase, statistics regarding the TxQoS performance are collected for logging and feedback purposes. In this last phase of execution, the ATC life cycle ends as ATCs are instantiated into running transactions. Usually this phase last from days to weeks, depending on the complexity of customer orders. It starts from customers specifying their desired telecommunication equipment and services on-line, until the equipment is properly installed, the services are delivered, and the billing cycles start. In the exceptional case where processes may be canceled, the TxQoS performance may still be monitored along the execution.

8.3.4 Apply the BTF architecture in Epacity

In order to illustrate the main functional modules, we have discussed, from the design documents, the architectures of Epacity as shown in Figure 8.1, and Figure 8.2. These figures provide an overview at the highest level, and show how the applications are integrated. Figure 8.6 illustrates the services provided by the applications and how they are composed in this integration project. The basic idea is to leverage an integration broker (i.e. BusinessWorks) which connects to a workflow engine (i.e. iProcess) and all the other applications (CRM, Database, Inventory, etc.) through 'adaptors'. One 'adaptors' is a specific interface which communicates with an application via the predefined XML schema as a data structure. From the SOA perspective, we can view the integration broker as the provider of an integration service, and the other applications as the providers of services such as CRM, Inventory Management, and so on. All these services compose into a composite service called the 'OrderToDeliver'.

According to the duel view, which sees services and processes as two sides of the same solution, the OrderToDeliver service provides business customers of Tele telecommunication products and services through automated ordering-delivery-billing processes. In other words, this service is implemented by a number of business processes, and the external parties invoke it through its web interface. Due to the unknown infrastructures and application environment of the other parties in this case (with the exception of Tele), we do not consider a symmetric BTF architecture design in this case. This means we consider only the BTF architecture design at the service provider side (i.e. Tele), whilst the user side (i.e. Tele's customers and collaborators) is not considered. It is not necessary for the customers to apply the BTF design in their own services/processes. To establish TxSLAs with Tele, and when making use of the 'OrderToDeliver' service, the customers only need to update the contracting-related part of their current IT systems.

We have discussed, in Figure 7.9, a BTF architecture that sits on top of a business process engine. As a workflow engine, iProcess is already present in this case, and connects to various OSS and BSS applications. The key to applying the BTF architecture design principle in Epacity is a BTF adapter. Such an adaptor enables the communication between the BTF manager and

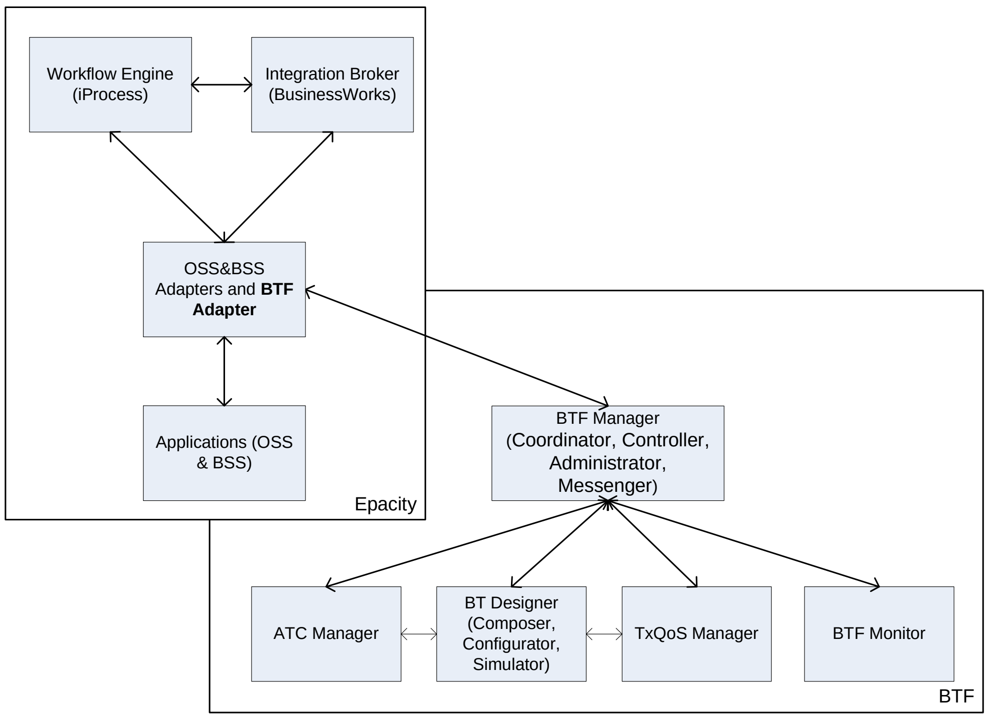

Workflow Engine (iProcess)
Integration Broker (BusinessWorks)
OSS&BSS Adapters and **BTF Adapter**
Applications (OSS & BSS)
Epacity
BTF Manager (Coordinator, Controller, Administrator, Messenger)
ATC Manager
BT Designer (Composer, Configurator, Simulator)
TxQoS Manager
BTF Monitor
BTF

Figure 8.12: BTF-Epacity Architecture

iProcess engine under a common XML schema via the integration broker (i.e. BusinessWorks). The common XML schema allows the data to be forwarded and understood by relevant applications.

As shown in Figure 8.12, the 'BTF Manager' is positioned at the same level as the OSS and BSS applications in Epacity. In fact, the components of the BTF manager (i.e. Coordinator, Controller, Administrator, and Messenger) are various BTF applications, which realize the ATC and TxQoS approaches.

When the BTF manager is taken as a whole, and interfaced by the BTF adapter, we can take a service view on all the services choreographed in Epacity. This includes the BTF service, which connects to the messaging bus in the same way as the other services e.g. billing, CRM, and so on. The service choreography is shown in Figure 8.13. The original figure (without BTF) is taken from the project design document, where the Epacity relevant services are connected to various transportation means. Herein we add the BTF as one more service which connects to the RV transport. RV is an information bus from the BusinessWorks (BW) solution pack, where adaptors are used for the integration of heterogeneous data schemas of various applications.

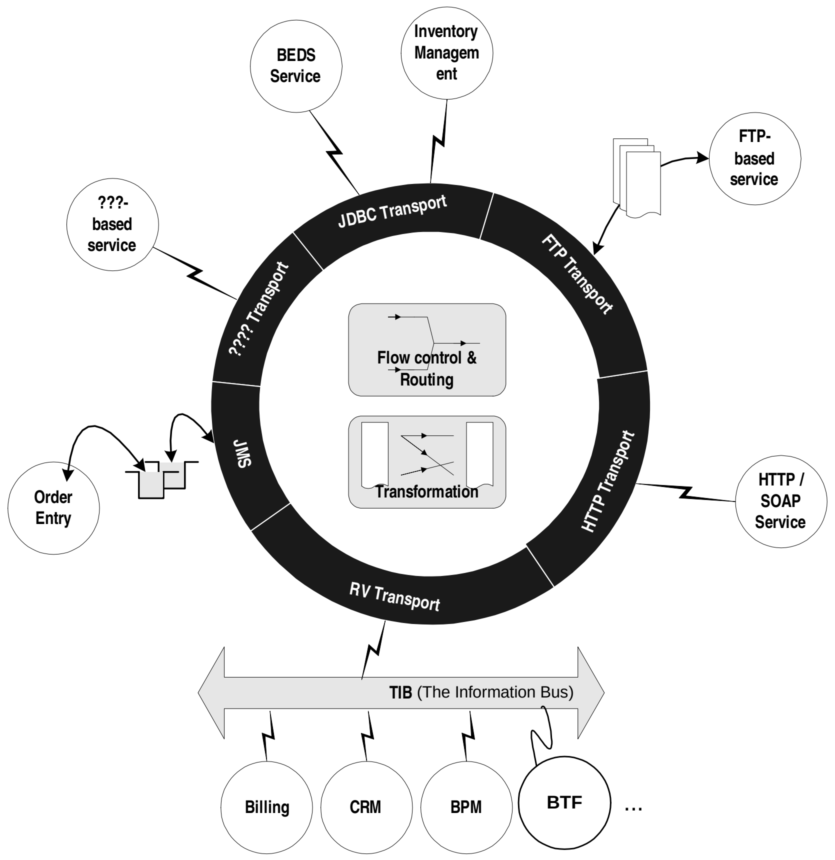


Figure 8.13: BTF-Epacity Service Choreography

To apply the BTF architecture in this case, we introduced the BTF adaptor that connects BTF with Epacity. In addition to the adaptor implementation, the updates on data schemas of the BSS and OSS applications are needed during the implementation. Thus, the Epacity systems are able to communicate and accommodate with the BTF systems, without changing its current design in most places when introducing the BTF into the case study.

8.4 Conclusions

In this Chapter, we presented a case study based on a real life integration project in one of the largest telecommunication companies in Europe, named here as 'Tele'. Internally it has several independent business units and departments which are involved in this case study. External participants are the business customers and the collaborators. Collaborators refer to particular third parties, including an engineering company who is in charge of equipment installation and configuration. The business processes are supported by zero-touch workflows, from the order stage, to processing and billing, and are long-lasting and cross-organizational. The workflows are implemented as part of the Epacity integration solution. Most of the steps in the processes are automated (i.e. without human intervention), except for the installation of equipments.

The integration of heterogeneous applications (e.g. a CRM, the database, the workflow engine, etc.) is based on a service-oriented architecture. These applications, which all have a communication interface, exchange data and information through a common messaging broker with a compatible data structure. All these features make this case study an interesting context to apply the BTF in.

We analyze the processes and the underlying applications using a method developed during the case study introduced in Chapter 3. This analysis method provides the starting point to gain an understanding of the case. After an overview of the Epacity is presented, we start to apply the research insights from previous chapters. To validate the feasibility of the BTF in this real-life case, we show how our research can be applied, and what benefit the BTF renders. In other words, we aim to demonstrate the practicality and usefulness of the BTF design in addressing the problems we discovered in the previous case study. We achieve this by providing illustrations and applying the research results one by one. From the feasibility study in the Epacity case, we conclude that the BTF solution, including the TxQoS and ATC approaches, can be applied to a real life case, whose characteristics of the concerned processes are long-lasting, contract-driven, and service-oriented. The case analysis method applied to the two cases in this thesis can be used for this type of processes.

With the increasingly complex business collaborations, the requirements on the reliable execution of business processes asks for a more flexible and comprehensive solution. As the business-oriented design, the TxQoS approach enforces a reliability, contracting relations between providers and users. The key to enabling the TxQoS approach is an unambiguous specification at a business level understanding. From this case study, the FIAT attributes have proved to be able to specify the transaction qualities in real-life cases. The ATC approach allows the flexibility of transaction design at the provider side. In this case study, we also see that the ATC approach enables a flexible composition of transaction support for processes. Together with the application of the BTF pattern analysis and architecture design, this case study demonstrates the feasibility to implement BTF in long-lasting, contract-driven, and service-oriented processes.

Chapter 9

Conclusion

This chapter concludes the thesis. First a summary of the conducted research is presented, followed by a discussion explaining the contributions to our research field. At the end, we discuss the limitations of the thesis, based on which extensions are possible for future research activities.

9.1 Research Summary

As stated in the beginning of the thesis in Chapter 1, the research carried out in the XTC project aims to design a flexible and comprehensive transaction framework (BTF), in order to address the challenge of providing an increasingly demanding process/service execution reliability. The research started with a theoretical literature review, and was followed by a case study in order to identify and define the precise research questions. First, we performed a thorough investigation on the current state of affairs, followed by an assessment of the case study. Based on our findings, we were able to define the research problems: *How to specify a proper transaction support?* (focusing on the entity 'Business Tx Spec') and *How to ensure these transaction specifications work in a changing environment?* (focusing on the entity 'Tech Tx Spec'). The design of the BTF thus follows two threads: one business-oriented and the other technology-oriented. This solves the two problems mentioned above. Afterwards a framework is developed in order to integrate the two approaches.

A TxQoS framework was established from the results of the business-oriented design, allowing the transactional qualities to be specified and guaranteed through a contractual approach. The technology-oriented design resulted in the ATC approach, which enables a flexible composition of transactional process chunks. Together, the TxQoS and ATC approaches form the core concepts of the BTF that integrates these two approaches and describes their application. In the end, we validated the feasibility of the BTF by a feasibility study, centered around a case study of a real-life telecom project, which consists of service-oriented and contract-driven processes and complex business relationships.

9.2 Contribution

The main contribution of the thesis is a transaction framework, called BTF, which integrates the TxQoS and ATC approaches in order to address the process reliability concern. We notice that the research in transaction management often focuses on specific technical means, easily losing the overview of the business environment. The BTF brings together two types of process reliability (i.e. transactional reliability and business trustworthiness), which have long been separately addressed by the IT and business worlds.

Next we elaborate on the contributions of this thesis on the research field:

9.2.1 TxQoS

The TxQoS approach enables the specification of transaction qualities in terms of FIAT (Fluency, Interference, Alternation, and Transparency) attributes. This business-friendly approach allows providers and users to agree on transactional qualities before execution time. As a result, the customer side is able to understand processes better, and has more power in selecting better services. Meanwhile the guaranteed transactional reliability raises the confidence level as well as the level of satisfaction of the customers. In addition, the statistics on data and information collected through the TxQoS life cycle regarding its transactional capabilities help providers serve their customer needs better.

The concepts, scenarios, life cycle, and the specification method of transactional reliability have been developed as a framework to operationalize the TxQoS approach. Research on the proposed entity 'Business Tx Spec' has resulted in a TxQoS-TxSLA (Transactional Service Level Agreement)-contract structure, which is used to enclose the TxQoS specification into e-contracts. The unique feature of this approach is the interpretation of transactional reliability from a business perspective, which has not been addressed by existing research efforts.

The essential part of the TxQoS is the transaction specification, for which we developed FIAT attributes (*Fluency, Interferability, Alternation, and Transparency*). They are designed to enable a precise interpretation of transactional performance in a TxSLA. It is understandable by both the IT and business communities, and can be monitored and evaluated along the TxQoS life cycle. *Fluency* measures the robustness of the service execution. *Interferability* indicates the control power from the service user during service execution. *Alternation* represents the possible choices when encountering problems. *Transparency* reflects the interest of a service user on the internal process at the provider side. Applying FIAT through different stages of a service life cycle (i.e. design, publish, discovery, execution, and evaluation) contributes to enhanced transactional reliability, while empowering users with more knowledge to choose reliable services. Due to the contractual agreement, customers gain more confidence in being able to be provided with a reliable execution.

9.2.2 ATC

The ATC approach provides a method leveraging the existing transaction models by abstraction and configuration. ATCs contain four parts of information: structure, position, ACIDity, and mechanism. An XML Definition Schema has been designed, allowing the structure, position, ACIDity, and transaction mechanisms of an ATC to be specified. The ATCs transform through four phases, from the pre-design phase, during which general templates are designed, to the selection phase, where proper templates are selected on-demand, through the configuration phase, during which the parameters are set, and finally to the deployment phase, when a specific ATC schema is fully composed and parameterized.

The comprehensive transaction support for process execution is enabled by an ATC schema that allows for the combination of different types of ATCs (i.e. atomic, chains, nested, and complex). These ATCs are abstracted from the current transaction models predominantly according to the structural similarity. They are organized and stored in the ATC library as general templates that can be reused and configured to meet various transaction needs. Compared to the existing transaction models, ATCs can cover every type of process structure, by using flexibility and comprehensiveness as a base.

Existing research on technical transactions very often emphasize the application environment. For instance, the ACID transaction was designed for database and web service transactions for web applications. The unique feature of the ATC approach is the plug-and-play logic that overcomes application constraints. Thus, ATCs provide a more flexible way of reusing the current transaction logic to meet various reliability needs.

9.3 Limitations and Future Work

The XTC project asks for a variety of deliverables related to the areas of transaction management, service oriented computing, e-contracting, and enterprise architecture In this thesis, we have looked into the aspects relating e-contracting with transaction management in order to realize the BTF complex processes. Due to the time constraints, there are still aspects which have not been analyzed in detail, which provide an interesting area for future work. In this section, we go over these aspects and discuss what can be done to improve their understanding.

9.3.1 Integration of TxQoS and ATC

The TxQoS design includes a business element, in order to balance the picture (refer to Figure 3.4) and move towards the vision laid out in Figure 3.7. Similarly, the ATC design complements the missing technical element to balance the picture. We performed both designs in parallel to each other, and integrated the two threads of research at the end. In this approach, we integrate the two sets of design output, by positioning them into common scenarios and patterns and explaining their joint operation in these settings. The result presented in Chapter 7 has demonstrated a TxQoS-aware transaction framework, integrating the TxQoS and ATC approaches.

However, this integration is loose and we still see there is room to enrich and improve it. For instance, the data alignment has not been addressed in the BTF design. A common data schema for both the ATC template specification and the TxQoS contract specification is key to realize full integration. Thus, it can be a direction to follow up in the future. In addition, we see a possibility in specifying TxQoS information in an ATC specification. Following this thread of research, a more tightly-integrated BTF can be achieved.

9.3.2 Feasibility Study

Our design took place mainly at the conceptual level and is validated through a case study instead of by means of a prototype implementation. This validation approach has certain limitations. Having only one case study is not sufficient to draw general conclusions but rather to only obtain indications for the quality of the design. Clearly, a larger set of cases would provide a stronger demonstration of the feasibility of our approach and design results.

A next validation step would be performing a comparison of performance in process reliability before and after implementing the BTF in a real business environment. To achieve this, a collaboration between industry professionals and academia is key. Such a validation demands from the industry to provide a technical environment and allow changes to be made to its current systems that would deliver a quantitative demonstration for the advantages of our approach.

9.3.3 Full-blown ATC Library

For the ATC design we used several examples of XML specification snippets to illustrate the conceptual design. The XML-based language we used to write the ATC specifications is a demonstration that our idea of ATCs can be implemented. However, complex examples may prove the current ATC specification method inadequate and it therefore needs to be extended. Due to time constraints, we have not created a formal set of specification method that can abstract all important transaction models.

We have performed a survey [70] on transactions in order to organize the existing transaction models. Based on the hierarchy of transactions and ATC specification method presented in the thesis, a full-blown ATC library can be an interesting topic to address in future. Please note that a full-blown ATC library indicates that both the design and implementation are needed in order to produce a complete set of ATCs corresponding to existing transaction models. For instance, we can start from Figure 6.2, where an ATC organization chart is presented. By researching the features of the models, one can find a method to capture these features by abstraction and configuration. As an outcome, each type of transaction model can find an ATC correspondence in the library.

Appendix A

Fluency Specification Method

Theoretically, if we assume that breakdowns happen stochastically and meet the following two conditions: (1) No simultaneous breakdowns can happen at any time; (2) the causes of the past breakdowns are fixed and do not affect future execution, then we can view the service execution a NonHomogeneous Poisson Process (NHPP). A NHPP is a generalization of a Homogeneous Poisson Process where events occur randomly over time at an average rate of λ events per unit time. The rate λ varies with time as determined by the intensity function $\lambda(t)$, which is an integrable function of time interval $(0, t]$. The cumulative intensity function $\Lambda(t)$, which is interpreted as the expected number of events from starting time 0 by time t, is defined by [16]

$$\Lambda(t) = \int_0^t \lambda(\tau)d(\tau), t > 0 \tag{A.1}$$

Accordingly, the exact number of events occurring in the interval $(a, b]$ is given by

$$\Lambda(a, b] = \int_a^b \lambda(t)d(t), b > a > 0 \tag{A.2}$$

Therefore, according to the above formula 4.2, the probability of exact n events occurring in the interval $(a, b]$ is given by

$$P(n) = \frac{\left[\int_a^b \lambda(t)d(t)\right]^n e^{-\int_a^b \lambda(t)d(t)}}{n!}, \quad \text{for } n = 0, 1, \ldots \tag{A.3}$$

If we view each breakdown as an event, and use the available statistics of executed service data to determine the intensity function $\lambda(t)$, which in our case means breakdown happening rate, then in theory a prediction of the fluency in the next execution can be computed. Depending on the service characteristics, the breakdown happening rates can vary. For example, a few extensive NHHP models using different $\lambda(t)$ computation techniques have been proposed using different intensity function in the area of software reliability, such as GO NHPP [22], Delayed S-shaped NHPP [75], Inflection S-shaped NHPP [43]. It is up to the provider to choose the most appropriate one that interprets the real testing data and can be adjusted according the runtime statistics.

We explain below how an extensive NHPP model is used for specifying 'Fluency' by an example. Suppose a service that should be executed within time $T(T = max(t))$, any execution not committed after T is viewed as a failure and is excluded in the fluency statistics. Before publishing the service, the provider made a number of testing runs. From the testing results, it shows the breakdown rate $\lambda(t)$ is a constant. According to the applicability of above mentioned extensive NHPP models, the GO NHPP model is therefore adopted. The mean value function of $\lambda(t)$ is given by

$$\lambda(t) = m(1 - e^{-rt}), \tag{A.4}$$

where $\lambda(0) = 0$. Here, m is the number of breakdowns that will be eventually detected and r is the breakdown detection rate according to the semantics of the GO NHPP model. These two parameters can be obtained from the tests, thus the breakdown rate *lambda* is a computable number given the time (i.e. the variable t).

If we define the fluency function $f(n)$ as the probability of having no more than n breakdowns during execution (i.e. within the time interval $(0, T]$), according to (4.3), $f(n)$ can be calculated as

$$f(n) = \sum P(n) = \sum \frac{\left[\int_0^T m(1 - e^{-rt})d(t)\right]^n e^{-\int_0^T m(1-e^{-rt})d(t)}}{n!} \tag{A.5}$$

According to the above function(4.4), we replace the $\lambda(t)$ by $m(1 - e^{-rt})$ to calculate function f(n) where the variables T, m and r are given by the testing results:

$$f(n) = \sum \frac{\left[m(T + \frac{e^{-rT}-1}{r})\right]^n e^{-m(T+\frac{e^{-rT}-1}{r})}}{n!} \tag{A.6}$$

Appendix B

Legend and Main Processes in Epacity

In this appendix, the complete business processes in the Epacity project are shown in three-level abstract diagrams. These diagrams capture the workflows that are deployed in the TIBCO iProcess workflow engine and serve a business-level understanding. Note that every activity of these processes does not stand for a single workflow step in the engine. One activity shows a meaningful workflow step or a series of workflow steps and we deleted the dummy steps that are built up for workflow logistics. Meanwhile, the structure of workflow set has been reorganized, as the original structure is flat and levels are hidden in the scripts of activities and flows.

B.1 Legend Description

Throughout the case study, we draw the diagrams using the legend shown below. The triangled 'continue' is used to switch from one line to the next when the process is too long. The circle indicates a step in all three levels. The square with solid line has different labels representing different application domains: NSF (Network Service Factory), CRM (Customer Relationship Management), CORE (database applications), BILL (billing applications). Sub-processes appeared in the main process diagrams are shown by document-like symbols with names only. Full-fledged sub-processes are depicted in the next Appendix.

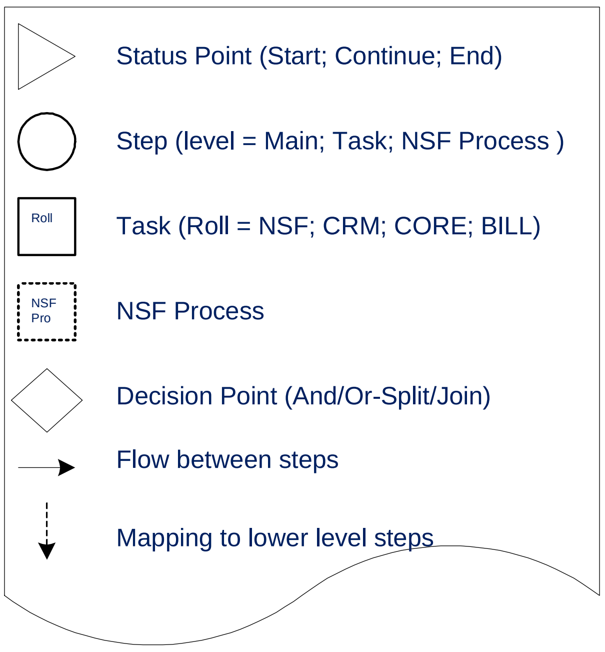

Status Point (Start; Continue; End)
Step (level = Main; Task; NSF Process)
Roll
Task (Roll = NSF; CRM; CORE; BILL)
NSF Pro
NSF Process
Decision Point (And/Or-Split/Join)
Flow between steps
Mapping to lower level steps

Figure B.1: Legend of Process Diagrams

B.2 Main Processes

In total, there are 7 workflows designed and deployed in the Epacity project. Over the years, they have been updated along with the changes of the requirements. In this thesis, we depict these process diagrams according to the current flows deployed in the workflow engine (i.e. iProcess) by June 2009.

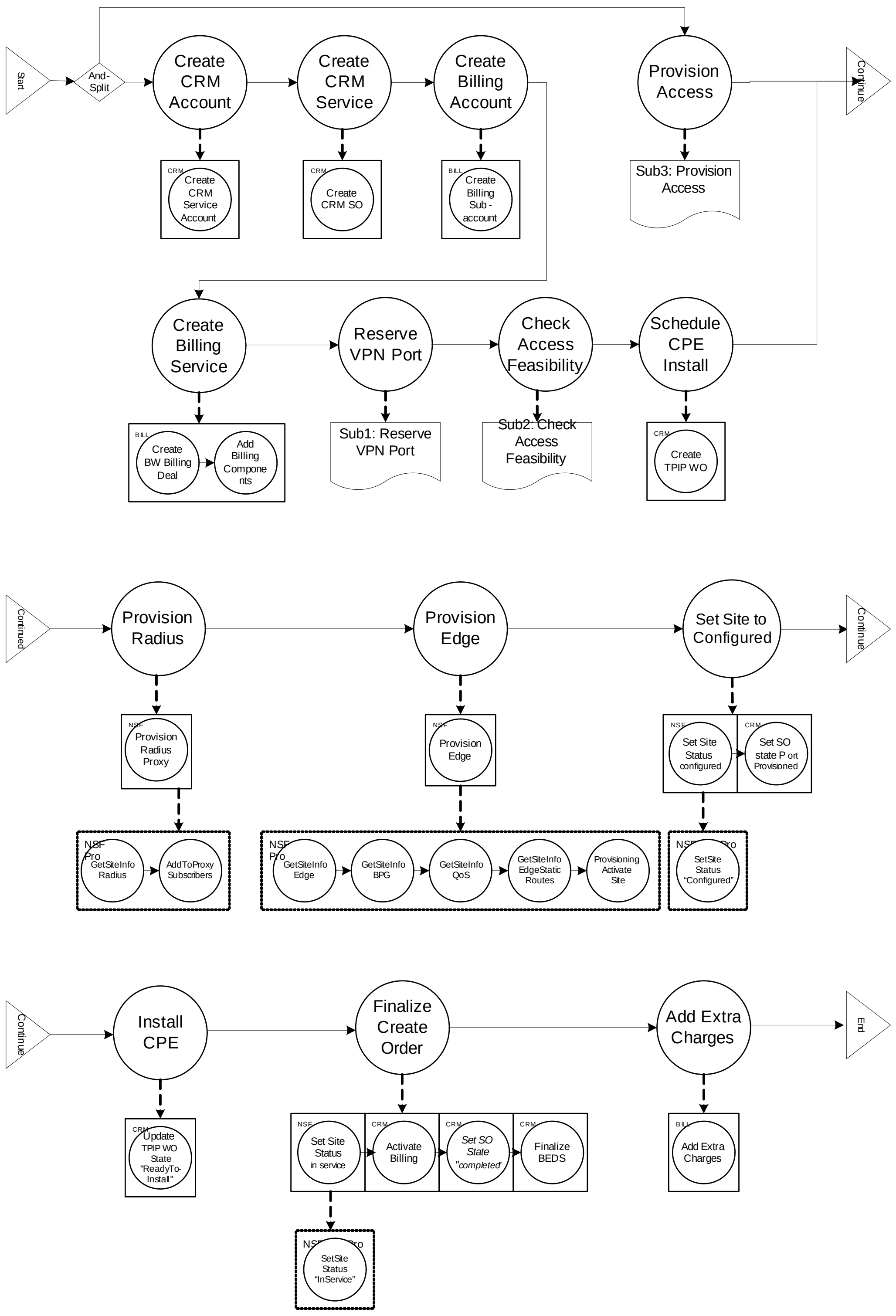


Figure B.2: Epacity Case: Create New Site

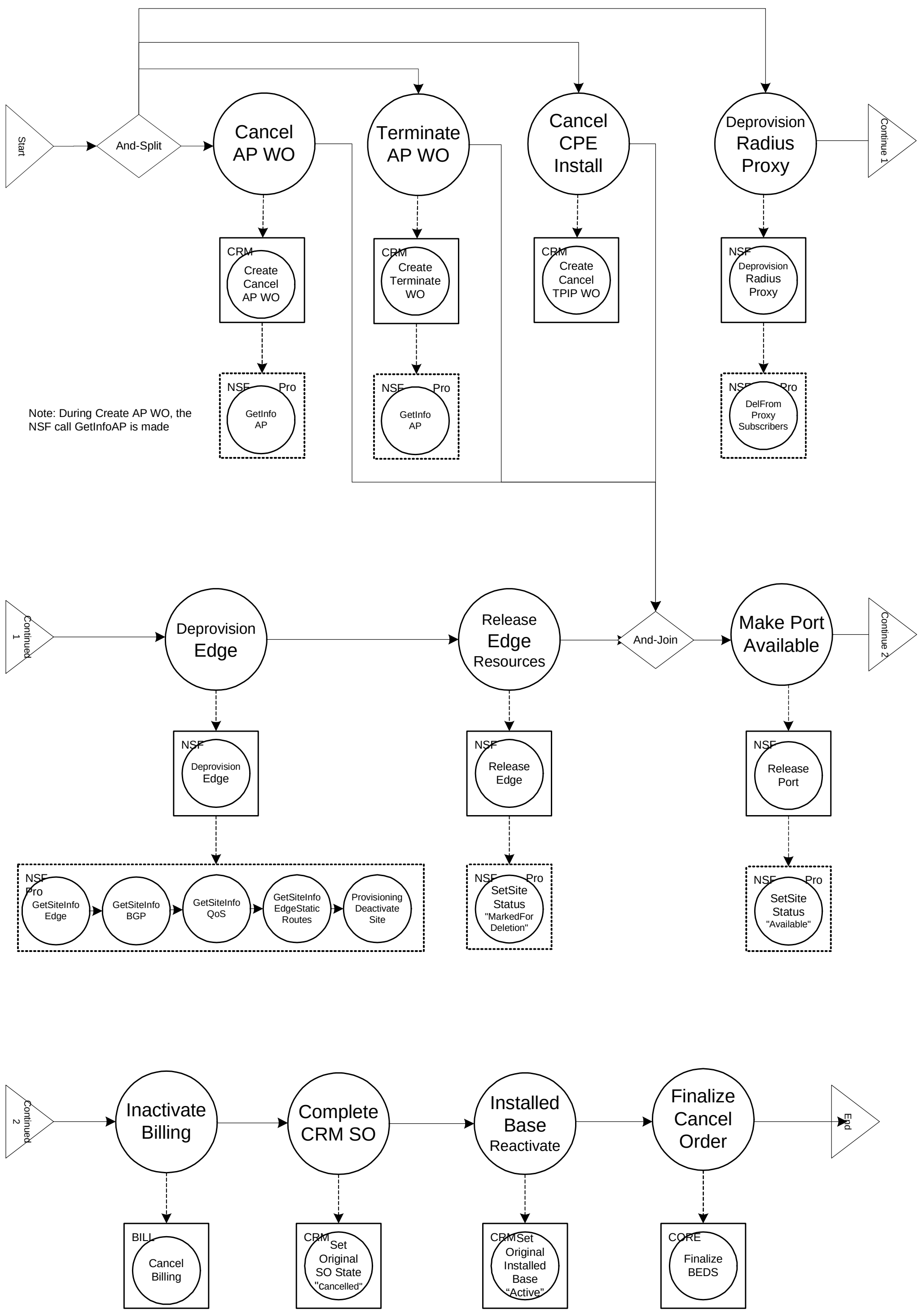


Figure B.3: Epacity Case: Cancel Site

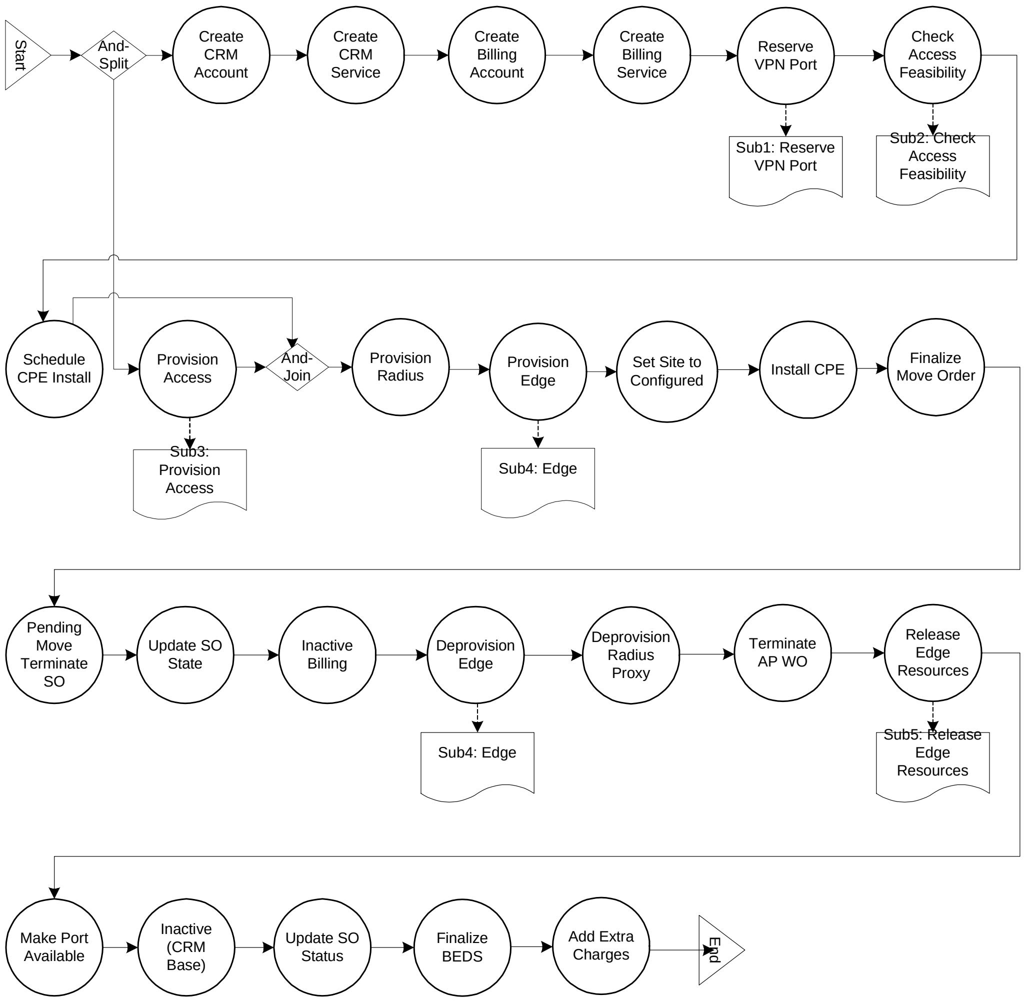


Figure B.4: Epacity Case: Move Site

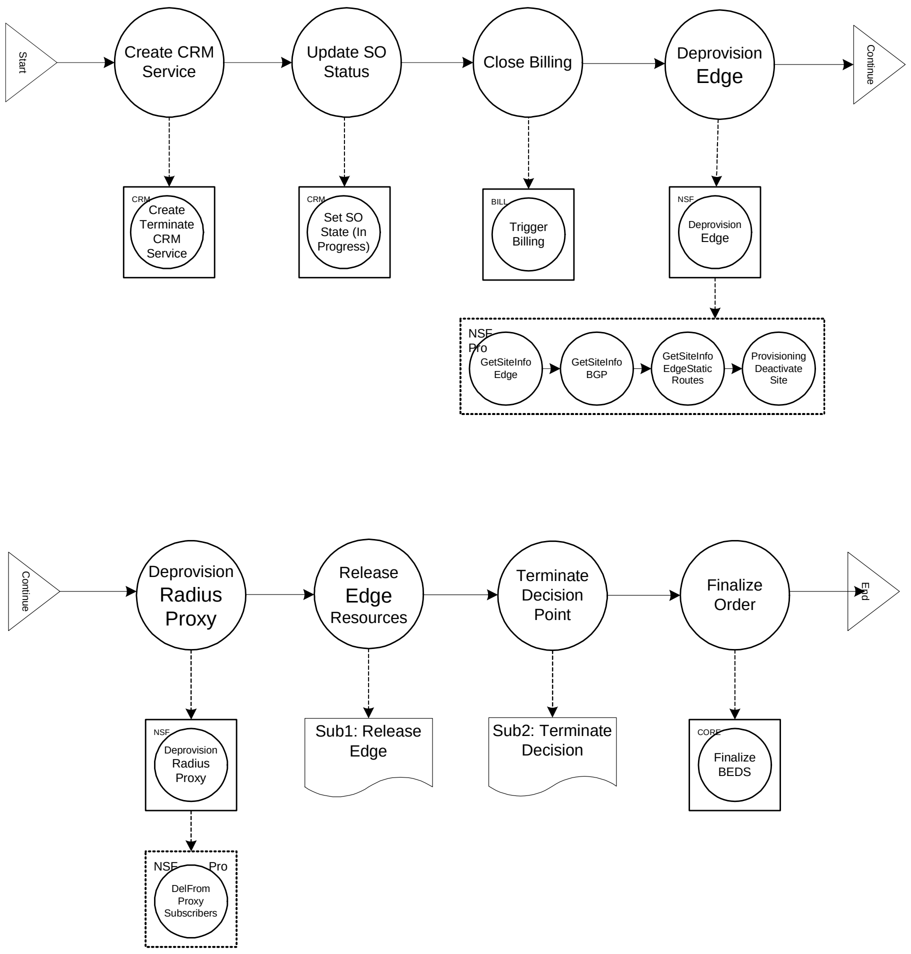


Figure B.5: Epacity Case: Terminate Site

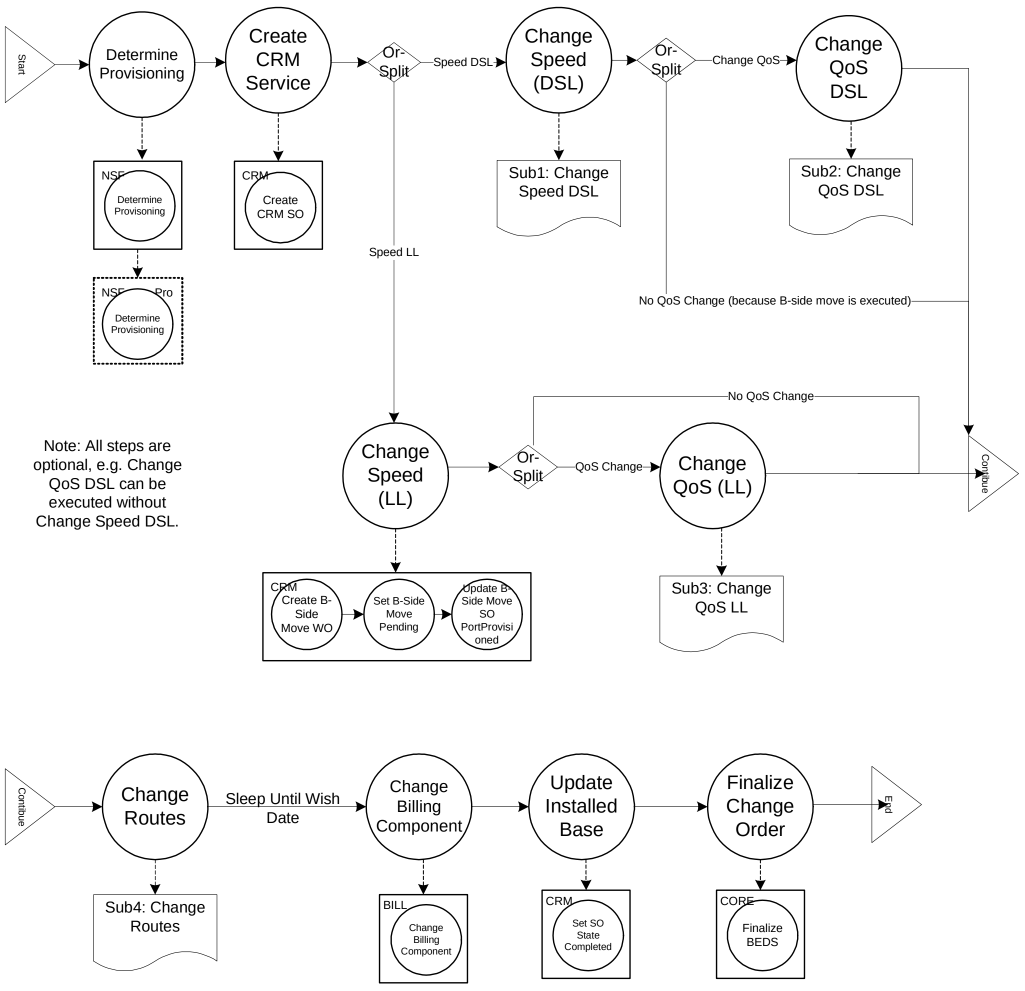


Figure B.6: Epacity Case: Change Site

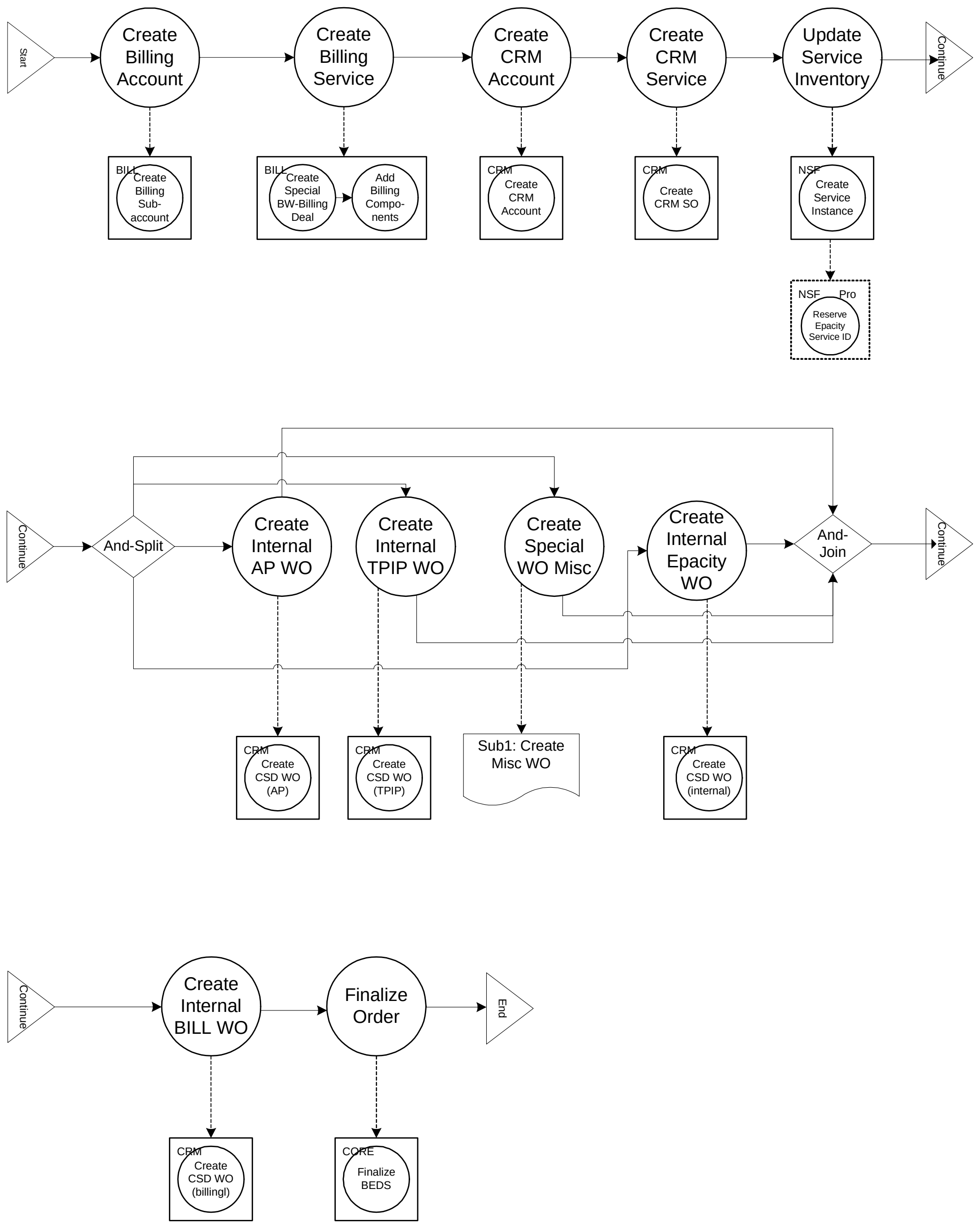


Figure B.7: Epacity Case: Create Special Site

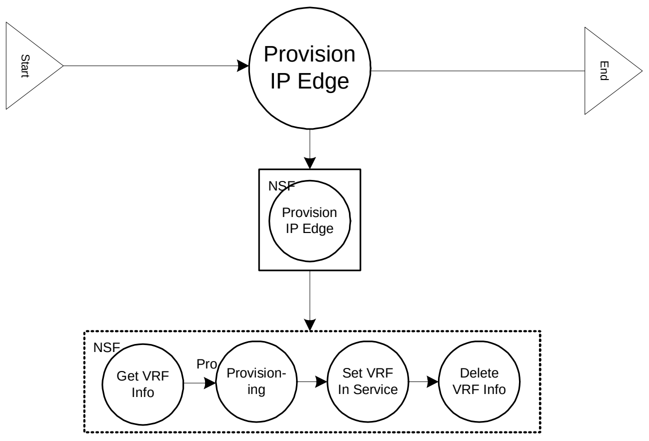


Figure B.8: Epacity Case: Create VRF Site

Appendix C

Sub-processes in Epacity

This appendix shows the sub-processes of the processes in the previous Appendix and applies the same legend.

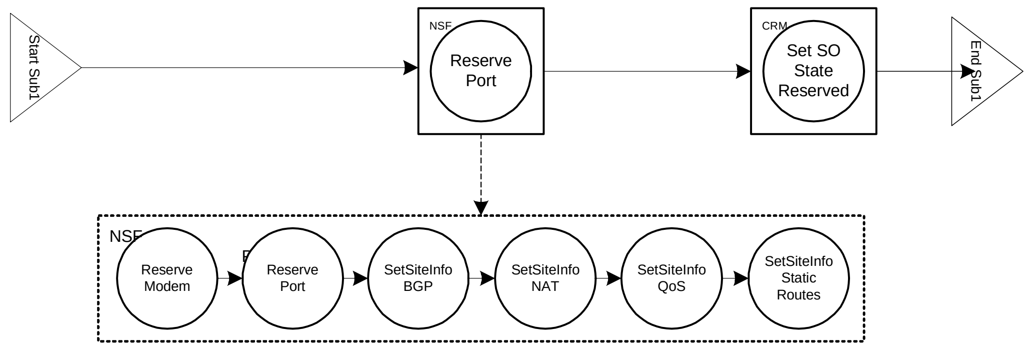


Figure C.1: Create New Site - Subprocess 1

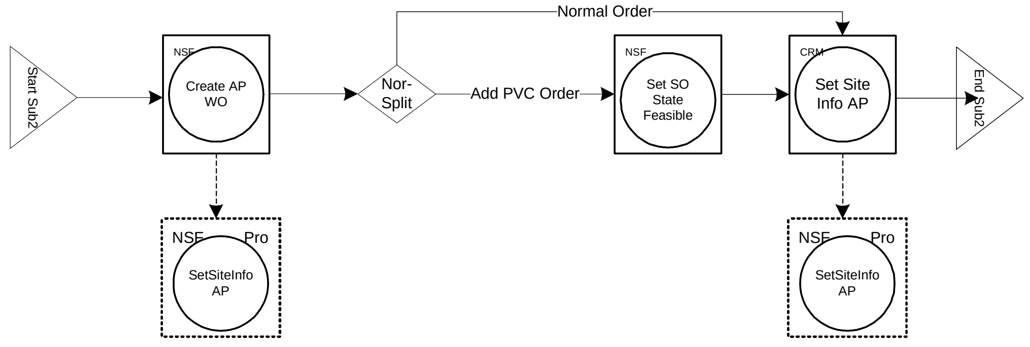


Figure C.2: Epacity Case: Create New Site - Subprocess 2

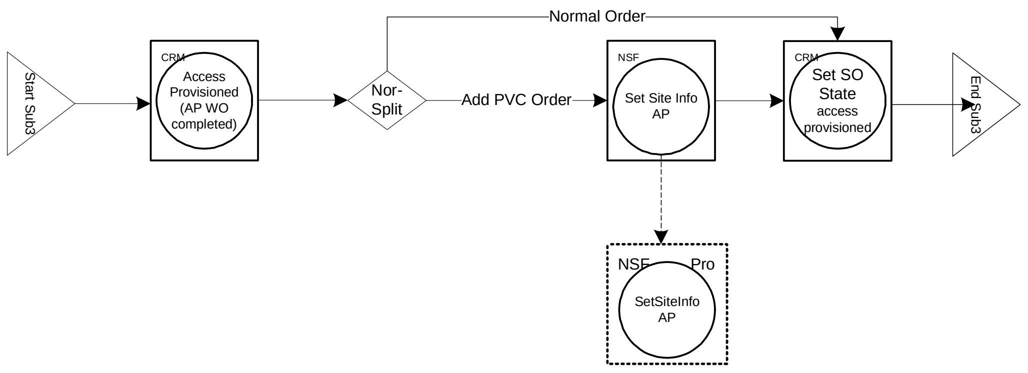


Figure C.3: Epacity Case: Create New Site - Subprocess 3

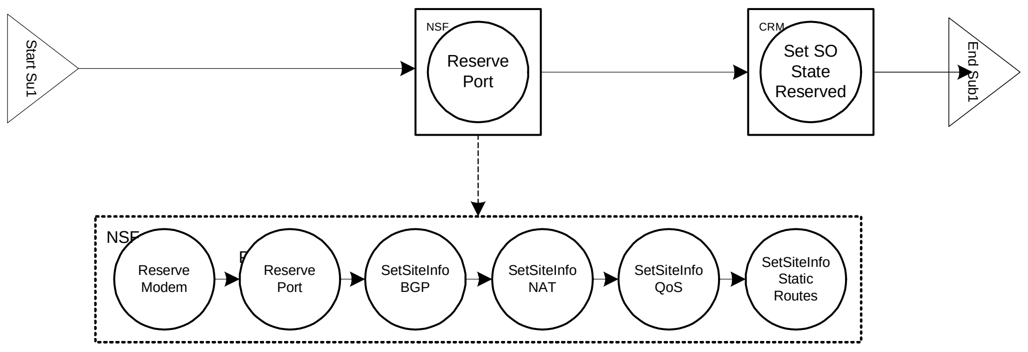


Figure C.4: Epacity Case: Move Site - Subprocess 1

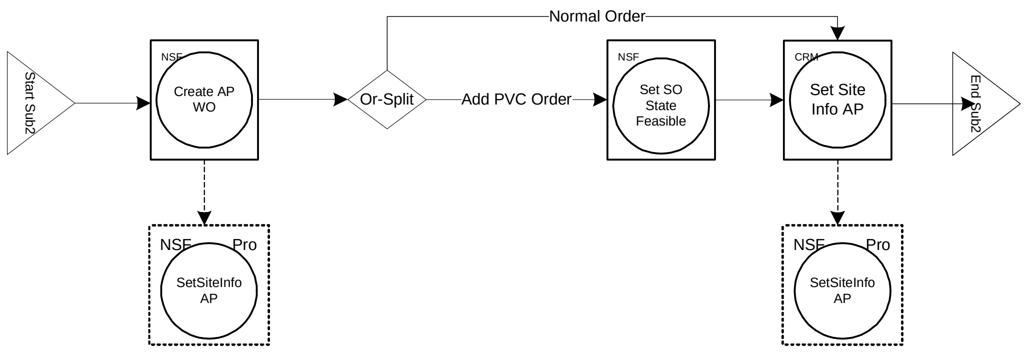


Figure C.5: Epacity Case: Move Site - Subprocess 2

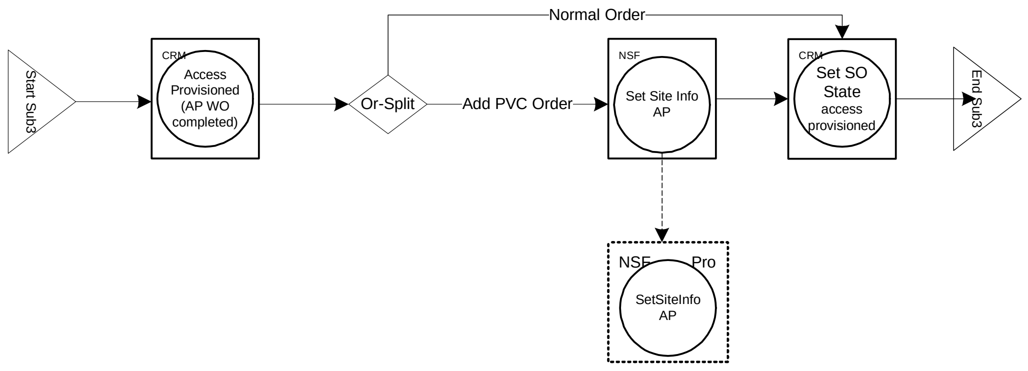


Figure C.6: Epacity Case: Move Site - Subprocess 3

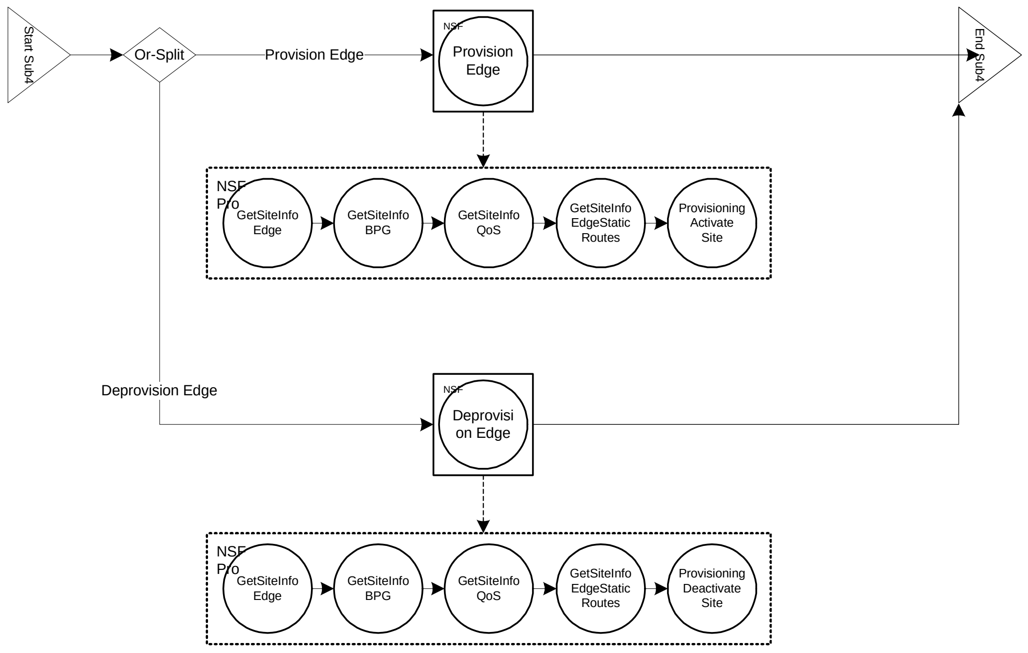


Figure C.7: Epacity Case: Move Site - Subprocess 4

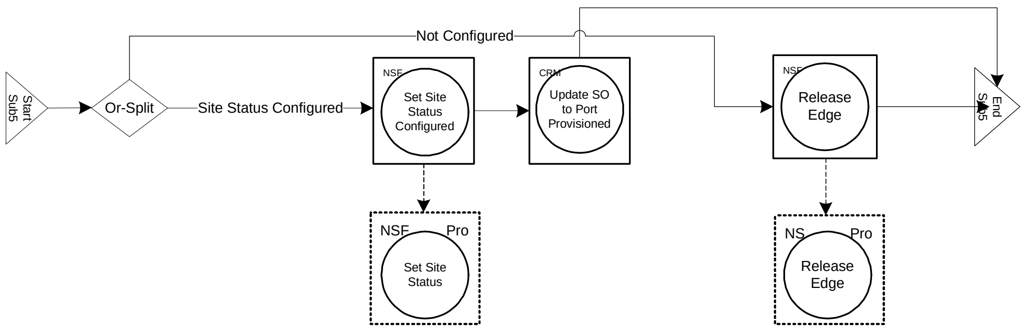


Figure C.8: Epacity Case: Move Site - Subprocess 5

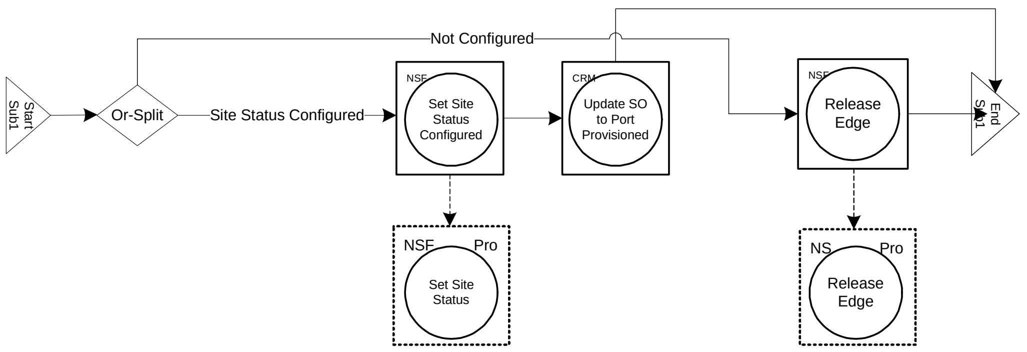


Figure C.9: Epacity Case: Terminate Site - Subprocess 1

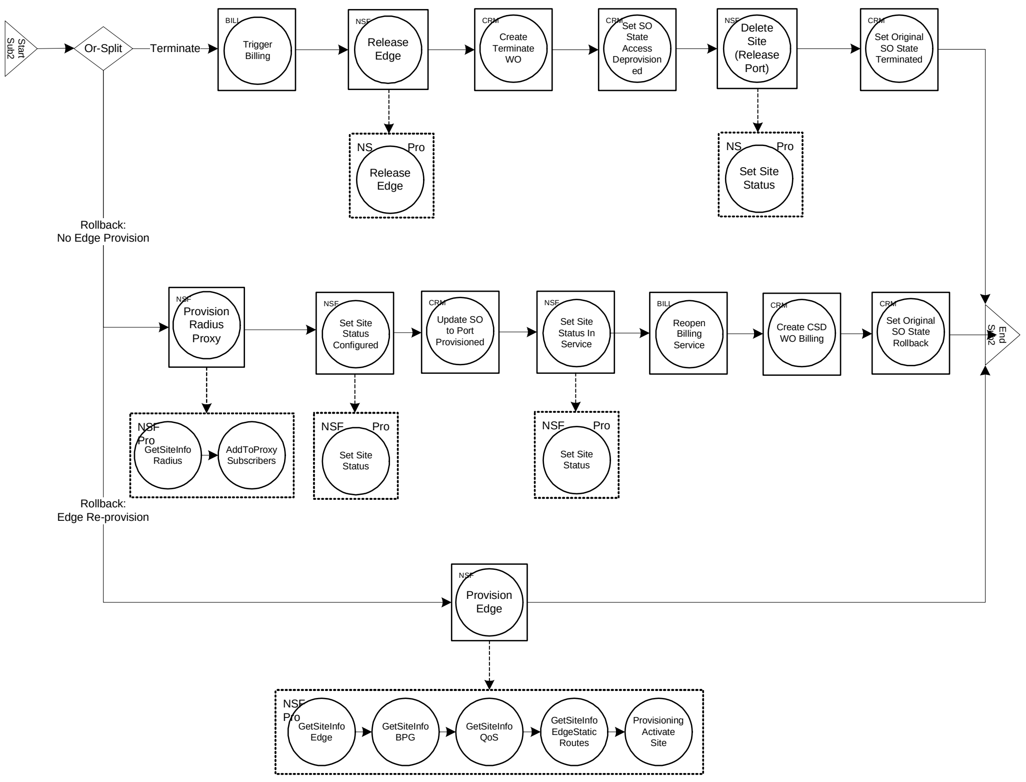


Figure C.10: Epacity Case: Terminate Site - Subprocess 2

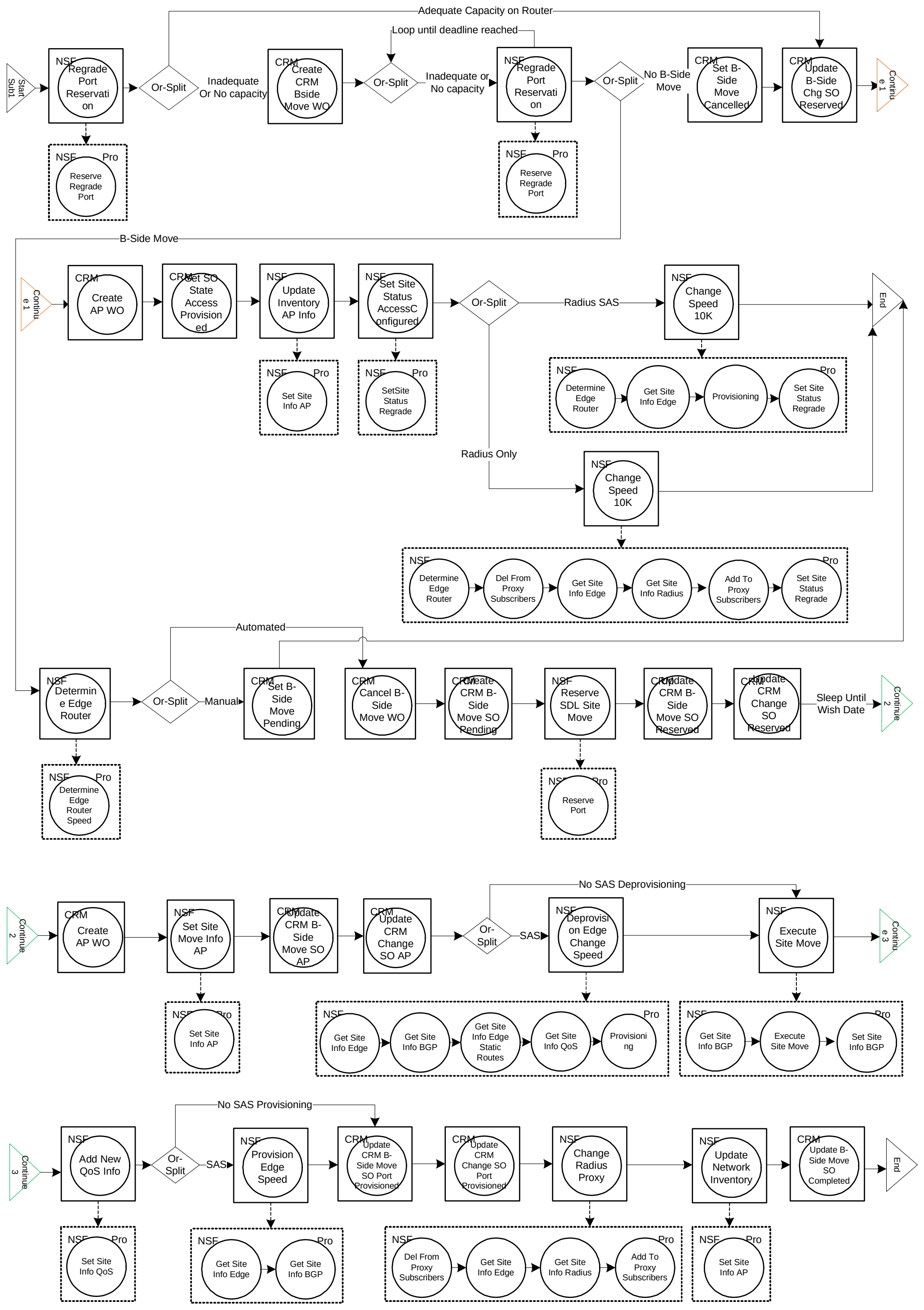

Start Sub1
NSF Regrade Port Reservation
NSF Pro Reserve Regrade Port
Or-Split
Adequate Capacity on Router
Inadequate Or No capacity
CRM Create CRM Bside Move WO
Or-Split
Loop until deadline reached
Inadequate or No capacity
NSF Regrade Port Reservation
NSF Pro Reserve Regrade Port
Or-Split
No B-Side Move
CRM Set B-Side Move Cancelled
CRM Update B-Side Chg SO Reserved
Continue 1
B-Side Move
Continue 1
CRM Create AP WO
CRM Set SO State Access Provisioned
NSF Update Inventory AP Info
NSF Pro Set Site Info AP
NSF Set Site Status AccessConfigured
NSF Pro SetSite Status Regrade
Or-Split
Radius SAS
NSF Change Speed 10K
NSF Pro Determine Edge Router, Get Site Info Edge, Provisioning, Set Site Status Regrade
End
Radius Only
NSF Change Speed 10K
NSF Pro Determine Edge Router, Del From Proxy Subscribers, Get Site Info Edge, Get Site Info Radius, Add To Proxy Subscribers, Set Site Status Regrade
Automated
NSF Determine Edge Router
NSF Pro Determine Edge Router Speed
Or-Split
Manual
CRM Set B-Side Move Pending
CRM Cancel B-Side Move WO
CRM Create CRM B-Side Move SO Pending
NSF Reserve SDL Site Move
NSF Pro Reserve Port
CRM Update CRM B-Side Move SO Reserved
CRM Update CRM Change SO Reserved
Sleep Until Wish Date
Continue 2
Continue 2
CRM Create AP WO
NSF Set Site Move Info AP
NSF Pro Set Site Info AP
CRM Update CRM B-Side Move SO AP
CRM Update CRM Change SO AP
Or-Split
No SAS Deprovisioning
SAS
NSF Deprovision Edge Change Speed
NSF Pro Get Site Info Edge, Get Site Info BGP, Get Site Info Edge Static Routes, Get Site Info QoS, Provisioning
NSF Execute Site Move
NSF Pro Get Site Info BGP, Execute Site Move, Set Site Info BGP
Continue 3
Continue 3
NSF Add New QoS Info
NSF Pro Set Site Info QoS
Or-Split
No SAS Provisioning
SAS
NSF Provision Edge Speed
NSF Pro Get Site Info Edge, Get Site Info BGP
CRM Update CRM B-Side Move SO Port Provisioned
CRM Update CRM Change SO Port Provisioned
NSF Change Radius Proxy
NSF Pro Del From Proxy Subscribers, Get Site Info Edge, Get Site Info Radius, Add To Proxy Subscribers
NSF Update Network Inventory
NSF Pro Set Site Info AP
CRM Update B-Side Move SO Completed
End

Figure C.11: Epacity Case: Change Site - Subprocess 1 - Change SpeedDSL

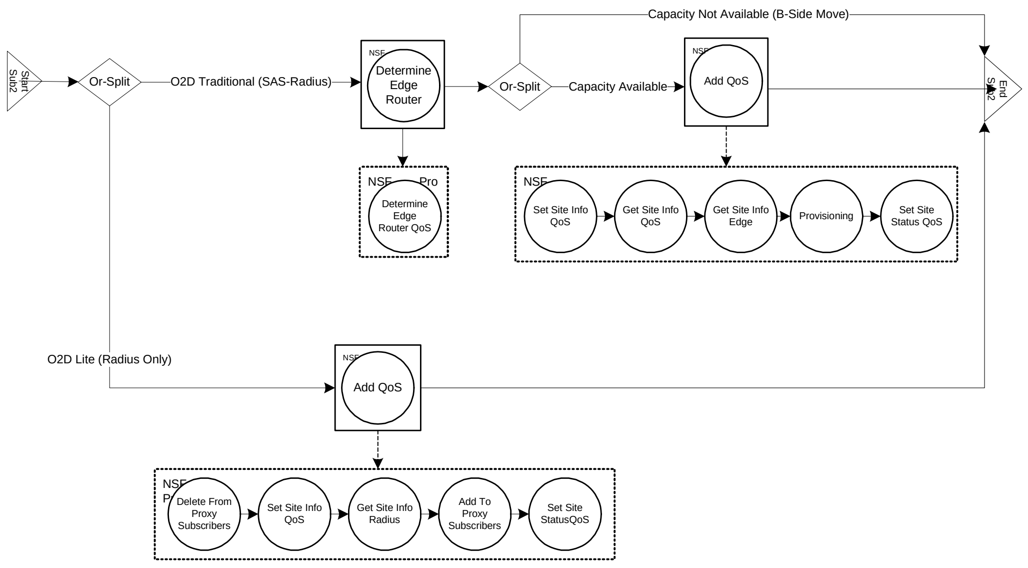


Figure C.12: Epacity Case: Change Site - Subprocess 2 - Change QoS DSL

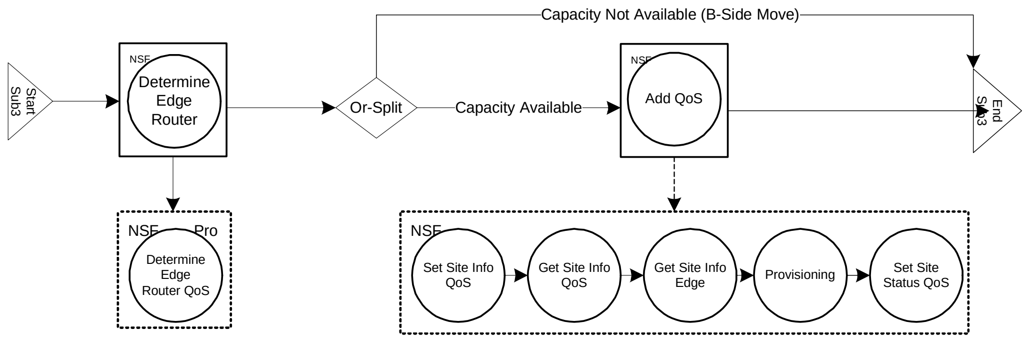


Figure C.13: Epacity Case: Change Site - Subprocess 3 - Change QoS LL

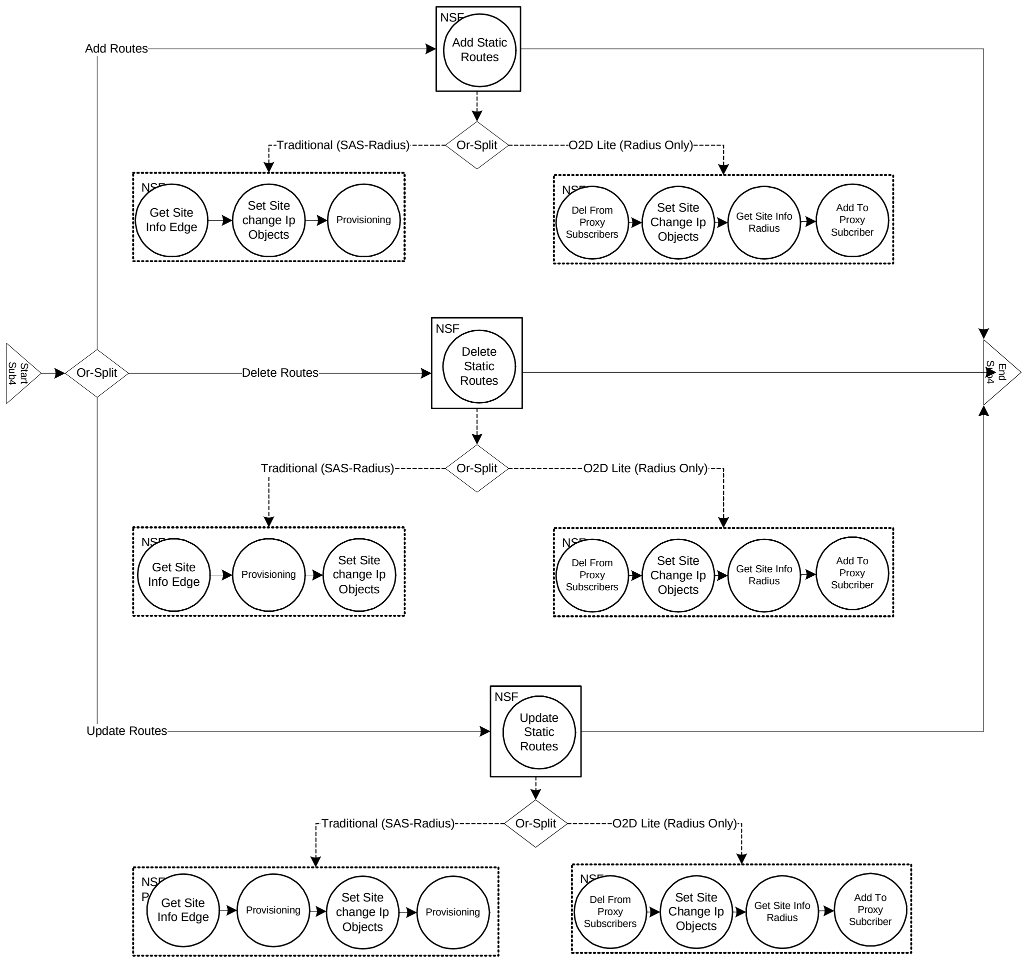


Figure C.14: Epacity Case: Change Site - Subprocess 4 - Change Routes

Bibliography

[1] S. Angelov. *Foundations of B2B Electronic Contracting.* PhD thesis, Eindhoven University of Technology, 2006.

[2] S. Angelov, P. W. P. J. Grefen, and D. Greefhorst. A classification of software reference architectures: Analyzing their success and effectiveness. In *WICSA/ECSA*, pages 141–150, 2009.

[3] S. Angelov, J. Vonk, K. Vidyasankar, and P. W. P. J. Grefen. Supporting cross-organizational process control. In *PRO-VE'09*, pages 415–422, 2009.

[4] M. M. Astrahan and et. al. System R: Relational approach to database management. *ACM Transactions on Database Systems (TODS)*, 1(2):97–137, 1976.

[5] A. Berry and Z. Milosevic. Extending choreography with business contract constraints. *International Journal of Cooperative Information Systems*, 14(2-3):131–179, 2005.

[6] E. Boertjes, P. W. P. J. Grefen, J. Vonk, and P. M. G. Apers. An architecture for nested transaction support on standard database systems. In *Proceedings of the 9th International Conference on Database and Expert Systems Applications*, DEXA '98, pages 448–459, London, UK, 1998. Springer-Verlag.

[7] Y. Breitbart, H. Garcia-Molina, and A. Silberschatz. Overview of multidatabase transaction management. *VLDB Journal*, 1(2):181–239, 1992.

[8] D. Bunting and et. al. *Web Service Context (WS-CTX)*, July 2003. `http://www.oasis-open.org/committees/ws-caf/`.

[9] D. Bunting and et. al. *Web Service Coordination Framework (WS-CF)*, July 2003. `http://www.oasis-open.org/committees/ws-caf/`.

[10] D. Bunting and et. al. *Web Services Composite Application Framework (WS-CAF)*, July 2003. `http://www.oasis-open.org/committees/ws-caf/`.

[11] D. Bunting and et. al. *Web Services Transaction Management (WS-TXM)*, July 2003. `http://www.oasis-open.org/committees/ws-caf/`.

[12] A. Ceponkus, S. Dalal, T. Fletcher, P. Furniss, A. Green, and B. Pope. Business transaction protocol version 1.0, June 2002. `http://www.oasis-open.org/committees/business-transactions/`.

[13] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. *Web Services Description Language (WSDL) 1.1.* W3C, March 2001. `http://www.w3.org/TR/wsdl`.

[14] P. K. Chrysanthis and K. Ramamritham. ACTA: a framework for specifying and reasoning about transaction structure and behavior. In *Proceedings of the 1990 International Conference on Management of data (ACM SIGMOD'90)*, pages 194–203, New York, NY, USA, 1990. ACM Press.

[15] P. K. Chrysanthis and K. Ramamritham. ACTA: The SAGA continues. In *Database Transaction Models for Advanced Applications*, pages 349–397. Morgan Kaufmann, 1992.

[16] E. Cinlar. *Introduction to Stochastic Processes.* Prentice Hall, 1975.

[17] W. M. P. Van der Aalst, F. Leymann, and W. Reisig. The role of business processes in service oriented architectures (editorial). *International Journal of Business Process Integration and Management*, 2(3):75–80, 2007.

[18] A. K. Elmagarmid, editor. *Database transaction models for advanced applications.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.

[19] A. K. Elmagarmid, Y. Leu, W. Litwin, and M. Rusinkiewicz. A multidatabase transaction model for interbase. In Dennis McLeod, Ron Sacks-Davis, and Hans-Jörg Schek, editors, *VLDB Conference*, pages 507–518. Morgan Kaufmann, 1990.

[20] H. Garcia-Molina and K. Salem. Sagas. In *Proceedings of the 1987 ACM International Conference on Management of data*, SIGMOD '87, pages 249–259, New York, NY, USA, 1987. ACM Press.

[21] D. Garlan and M. Shaw. An introduction to software architecture. In *Advances in Software Engineering and Knowledge Engineering*, pages 1–39. Publishing Company, 1993.

[22] A.L. Goel and K. Okumoto. Time-dependent error-detection rate model for software reliability and other performance measures. *IEEE Transactions on Reliability*, R-28(3):206–211, 1979.

[23] J. Gray and A. Reuter. *Transaction processing: concepts and techniques.* Morgan Kaufmann Publishers, 1993.

[24] P. W. P. J. Grefen. *Mastering e-Business.* Routledge, 2010.

[25] P. W. P. J. Grefen, K. Aberer, H. Ludwig, and Y. Hoffner. Crossflow: Cross-organizational workflow management in dynamic virtual enterprises. *International Journal of Computer Systems Science and Engineering*, 15:277–290, 2000.

[26] P. W. P. J. Grefen and P. M. G. Apers. Integrity control in relational database systems - an overview. *Data and Knowledge Engineering*, 10:187–223, 1993.

[27] P. W. P. J. Grefen, H. Ludwig, and S. Angelov. A three-level framework for process and data management of complex e-services. *International Journal of Cooperative Information Systems*, 12(4):487–531, 2003.

[28] P. W. P. J. Grefen, N. Mehandjiev, G. Kouvas, G. Weichhart, and R. Eshuis. Dynamic business network process management in instant virtual enterprises. *Computers in Industry*, 60:86–103, 2009.

[29] P. W. P. J. Grefen, J. Vonk, E. Boertjes, and P. M.G. Apers. Two-layer transaction management for workflow management applications. In *Database and Expert Systems Applications (DEXA)*, pages 430–439, 1997.

[30] T. Härder and A. Reuter. Principles of transaction-oriented database recovery. *ACM Computing Surveys*, 15(4):287–317, 1983.

[31] IBM Corp. Web Service Level Agreements (WSLA) project, 2004. http://www.research.ibm.com/wsla/.

[32] IBM Corp. Web Services Policy Framework. http://www.ibm.com/developerworks/library/specification/ws-polfram/, 2006.

[33] Encyclopædia Britannica Inc. Merriam-Webster Dictionary Online. http://www.merriam-webster.com/dictionary/reliability.

[34] P. M. Lewis, A. J. Bernstein, and M. Kifer. *Databases and Transaction Processing: An Application-Oriented Approach.* Addison-Wesley, 2001.

[35] Mark Little. Business transaction protocol: Transactions for a new age. *Web Services Journal*, 2(11):50–55, 2002.

[36] A. Mand. IBM best online and offline integrated campaign. *Brandweek Online*, 1999.

[37] A. Mani and A. Nagarajan. Understanding quality of service for web services. Technical report, IBM DeveloperWorks, 2002.

[38] Salvatore T. March and Gerald F. Smith. Design and natural science research on information technology. *Decision Support Systems*, 15(4):251 – 266, 1995.

[39] J. E. B. Moss. *Nested transactions: an approach to reliable distributed computing.* PhD thesis, EECS Department, M. I. T., 1981.

[40] E. Newcomer, I. Robinson, M. Feingold, and R. Jeyaraman. *Web Services Coordination (WS-Coordination) Version 1.2.* OASIS, February 2009. `http://docs.oasis-open.org/ws-tx/wscoor/2006/06`.

[41] E. Newcomer, I. Robinson, T. Freund, and M. Little. *Web Services Business Activity Framework (WS-BusinessActivity) Version 1.2.* OASIS, February 2009. `http://docs.oasis-open.org/ws-tx/wsba/2006/06`.

[42] E. Newcomer, I. Robinson, M. Little, and A. Wilkinson. *Web Services Atomic Transaction (WS-AtomicTransaction) Version 1.2.* OASIS, February 2009. `http://docs.oasis-open.org/ws-tx/wsat/2006/06`.

[43] M. Ohba. Software reliability analysis models. *IBM Journal of Research and Development*, 28(4):428–443, 1984.

[44] OMG. OMG Model Driven Architecture, 2011. `http://www.omg.org/mda/`.

[45] M. Papazoglou and D. Georgakopoulos. Service-oriented computing. *Comunications of the ACM*, 46(10):25–28, 2003.

[46] M. Papazoglou and W-J Heuvel. Service oriented architectures: Approaches, technologies and research issues. *VLDB Journal*, 16(3):389–415, 2007.

[47] M. Papazoglou and P. Ribbers. *E-business: Organizational and Technical Foundations.* John Wiley Sons Ltd., England, 2006.

[48] C. Pu, G. E. Kaiser, and N. C. Hutchinson. Split-transactions for open-ended activities. In F.Bancilhon and D. J. DeWitt, editors, *VLDB Conference*, pages 26–37. Morgan Kaufmann, 1988.

[49] Z. Qi, X. Xie, B. Zhang, and J. You. Integrating x/open dtp into grid services for grid transaction processing. In *International Workshop on Future Trends in Distributed Computing Systems (FTDCS)*, pages 128–134. IEEE Computer Society, 2004.

[50] K. Ramamritham and P.K. Chrysanthis. *Advances in Concurrency Control and Transaction Processing.* IEEE Computer Society, 1997.

[51] A. Reuter. ConTracts: A means for extending control beyond transaction boundaries. In *Third International Workshop on High Performance Transaction Systems*, 1989.

[52] M. Shaw and D. Garlan. *Software architecture: perspectives on an emerging discipline.* Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1996.

[53] A. P. Sheth and M. Rusinkiewicz. On transactional workflows. *IEEE Data Engineering Bulletin*, 16(2):37–40, 1993.

[54] A. Simon and T. Rischbeck. Service contract template. In *Proceedings of IEEE International Conference Service Computing (SCC'06)*, pages 574–581, Washington, DC, USA, 2006. IEEE Computer Society.

[55] T. Steinbach, J. Webber, and C. Türker. Proposed grid transaction RG - charter. `http://www.data-grid.org/tm-rg-charter.html`.

[56] C. Türker, K. Haller, C. Schuler, and H.-J. Schek. How can we support grid transactions? Towards peer-to-peer transaction processing. In *Proceedings of Conference on Innovative Data Systems Research (CIDR)*, pages 174–185, 2005.

[57] Universal description, discovery and integration (UDDI), 2006. `http://www.uddi.org/`.

[58] W. M. P. van der Aalst, A. H. M. ter Hofstede, and M. Weske. Business process management: A survey. In *Proceedings of International Conference on Business Process Management*, pages 1–12, 2003.

[59] J. Vonk and et al. An analysis of contractual and transactional aspects of a cardiothoracic surgery proces. Technical report, Eindhoven University of Technology, 2008.

[60] J. Vonk and P. W. P. J. Grefen. Cross-organizational transaction support for e-services in virtual enterprises. *Distributed and Parallel Databases*, 14(2):137–172, 2003.

[61] J. Vonk, P. W. P. J. Grefen, E. Boertjes, and P. M. G. Apers. Distributed global transaction support for workflow management applications. In *Proceedings of the 10th International Conference on Database and Expert Systems Applications*, DEXA '99, pages 942–951, London, UK, 1999. Springer-Verlag.

[62] J. Vonk, T. Wang, and P. Grefen. A dual view to facilitate transactional qos. In *Proceedings of 16th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE'07)*. IEEE Computer Society, 2007.

[63] W3C. *Web Services Description Language (WSDL) 1.1*, Marchh 2001. `http://www.w3.org/TR/wsdl`.

[64] H. Wächter and A. Reuter. The contract model. In *Database Transaction Models for Advanced Applications*, pages 219–263. Morgan Kaufmann, 1992.

[65] T. Wang and P. W. P. J. Grefen and. Ensuring transactional reliability by e-contracting. In *Proceedings of 20th International Conference on Advanced Information Systems Engineering (CAiSE'08)*, pages 262–265. Springer, 2008.

[66] T. Wang, P. Grefen, and J. Vonk. Abstract transaction construct: Building a transaction framework for contract-driven, service-oriented business processes. In *Proceedings of 4th International Conference on Service Oriented Computing (ICSOC'06)*, pages 434–439. Springer Verlag, 2006.

[67] T. Wang and P. W. P. J. Grefen. A historic survey of transaction management from flat to grid transactions. Technical report, Eindhoven University of Technology, 2005.

[68] T. Wang, J. Vonk, and P. W. P. J. Grefen. TxQoS: A contractual approach for transaction management. In *Proceedings of 11th IEEE International Conference on Enterprise Computing (EDOC'07)*, pages 327–338. IEEE Computer Society, 2007.

[69] T. Wang, J. Vonk, and P. W. P. J. Grefen. Towards a contractual approach for transaction management. *Enterprise Information Systems*, 2:443–458, 2008.

[70] T. Wang, J. Vonk, B. Kratz, and P. W. P. J. Grefen. A survey on the history of transaction management: from flat to grid transactions. *Distributed Parallel Databases*, 23.

[71] J. Warne. An extensible transaction framework: Technical overview. Technical report, ANSA Architecture for Open Distributed Systems Project, 1993.

[72] G. Weikum and H-J Schek. Concepts and applications of multilevel transactions and open nested transactions. In *Database Transaction Models for Advanced Applications*, pages 515–553. Morgan Kaufmann, 1992.

[73] E. Wustenhoff. Sun blueprints: Service level agreement in the data center, 2002. `http://www.sun.com/blueprints/0402/sla.pdf`.

[74] W. Xie, S. B. Navathe, and S.K. Prasad. Supporting qos-aware transactions in a system on mobile devices (syd). In *Proceedings of 23rd International Distributed Computing Systems Workshops*, pages 498– 502. IEEE Computer Society, 2003.

[75] S. Yamada, M. Ohba, and S. Osaki. S-shaped reliability growth modeling for software error detection. *IEEE Transaction on Reliability*, 32(5):475–478, 1983.

Summary

In this thesis, we propose a transaction framework to provide comprehensive and flexible transaction support for contract-driven, service-oriented business processes.

The research follows the research method outlined below. Initially, a thorough investigation on current state of affairs was made. Afterwards, we carried out a case study, which we utilized to identify the problems that are likely to occur during the execution of business processes. As the result of the solution design, the concepts, scenarios, life cycles, reference architectures, and mechanisms were proposed to address the problems. The design took place on the conceptual level, while the coding/programming and implementation is out of the scope of this thesis. The business-oriented solution design allows for transaction qualities to be specified and guaranteed by a contractual approach named as TxQoS (Transactional Quality of Service). The technology-oriented design enables flexible composition of ATCs (Abstract Transaction Constructs) as a transaction schema to support the execution of complex processes. As the last step of research, we validated the feasibility of our design by a utility study conducted in a large telecom project, which has complex processes that are service-oriented and contract-driven. Finally, we discussed the contributions and limitations of the research.

The main contribution of the thesis is the BTF (Business Transaction Framework) that addresses process execution reliability. The TxQoS approach enables the specification of transaction qualities in terms of FIAT (Fluency, Interference, Alternation, Transparency) properties. This business-friendly approach allows the providers and users to agree on transaction qualities before process execution time. The building blocks of the proposed framework, ATCs, are reusable and configurable templates, and are abstracted and generalized from existing transaction models. The various transaction requirements of sub-processes and process chunks can be represented by corresponding ATCs, which allow for a flexible composition. Integrated, the TxQoS and ATC approaches work together to form a TxQoS-aware business transaction framework.

Samenvatting

Samenvatting

In dit proefschrift wordt een raamwerk voor transacties voorgesteld teneinde volledige en flexibele transactie-ondersteuning te bieden aan contract-gedreven, diensten-georiënteerde bedrijfsprocessen.

Het onderzoek volgt de onderzoeksmethode zoals hierna geschetst. In eerste instantie is een uitgebreid onderzoek uitgevoerd naar de huidige stand van zaken. Daarna hebben we een casus geanalyseerd, die we gebruikt hebben om de problemen te identificeren welke met een grote waarschijnlijkheid optreden tijdens de uitvoering van processen. Als resultaten van een ontwerpproces worden concepten, scenario's, levenscycli, referentie-architecturen en mechanismen voorgesteld om deze problemen aan te pakken. Het ontwerp heeft plaatsgevonden op het conceptuele niveau, waardoor codering en implementatie buiten het aandachtsgebied van dit proefschrift vallen. Het bedrijfsgeoriënteerde ontwerp van het Business Transaction Framework (BTF) maakt het mogelijk dat transactie-karakteristieken worden gespecificeerd en gegarandeerd middels een contract-gebaseerde benadering met de naam TxQoS (Transactional Quality of Service). Het technologie-georiënteerde ontwerp ondersteunt flexibele compositie van ATCs (Abstract Transaction Constructs) tot een transactieschema voor de ondersteuning van de uitvoering van complexe processen. Als de laatste stap van het onderzoek hebben we de uitvoerbaarheid van ons ontwerp gevalideerd middels een bruikbaarheidsstudie die is uitgevoerd in een groot telecom-project, waarin complexe processen voorkomen die diensten-georiënteerd en contract-gedreven zijn. Tenslotte hebben we de bijdragen en beperkingen van het onderzoek besproken.

De belangrijkste bijdrage van dit proefschrift is een raamwerk voor transacties dat gericht is op de aspecten van de betrouwbaarheid van de uitvoering van processen. Een contractuele aanpak, genaamd TxQoS, maakt het mogelijk transactiekarakteristieken te specificeren in termen van de FIAT (Fluency, Interference, Alternation, Transparency) eigenschappen. Deze bedrijfsgerichte aanpak maakt het mogelijk dat aanbieders en gebruikers overeenstemming bereiken over transactiekarakteristieken voordat processen worden

uitgevoerd. De bouwblokken van het voorgestelde raamwerk, ATCs, zijn herbruikbare en configureerbare sjablonen die geabstraheerd en gegeneraliseerd zijn uit bestaande transactiemodellen. De verscheidene transactievereisten van deelprocessen en procesdelen kunnen worden gerepresenteerd door corresponderende ATCs, hetgeen een flexibele compositie mogelijk maakt. De TxQoS en ATC aanpakken vormen samen een geïntegreerd TxQoS-gebaseerd raamwerk voor bedrijfstransacties.

Curriculum Vitae

Ting Wang was born on September 11, 1977 in Nanchang, Jiangxi Province, People's Republic of China. After finishing a gifted youth program combining middle and high school education at Nanchang No.10 school in 1993, she started higher education at Nanchang University in China. In 1997, she graduated with a B.Sc. degree in Information Management and Science. From 1997 to 2002, she was employed by Jiangxi Police College as a police officer, where she lectured courses, and worked as an IT staff in the library. Additionally, from 1999 to 2002 she attended a part-time Master program for professionals in the department of Computer Science at Nanchang University. She obtained all course credits and passed national qualification exams, but then quit the thesis to study abroad. In 2002, she started a Research Master program in CentER graduate school at Tilburg University, the Netherlands, and graduated with a M.Sc. degree in Information Systems in 2003. From 2004 to 2008, she was employed by Eindhoven University of Technology, the Netherlands, as a PhD student working for a NWO (The Netherlands Organization for Scientific Research) project. In 2008, she joined the Amsterdam office of BearingPoint as a consultant in Business IT Alignment.