



Change Management for Service Based Business Processes

Yi Wang

A thesis submitted in fulfillment
of the requirements of the degree of

Doctor of Philosophy

Department of Computing

Faculty of Science

Macquarie University

Supervisor: Prof. Jian Yang

April 2011

ORIGINALITY STATEMENT

I certify that the work in this thesis entitled “Change Management for Service Based Business Processes” has not previously been submitted for a degree nor has it been submitted as part of requirements for a degree to any other university or institution other than Macquarie University.

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

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

Yi Wang

April 2011

To my family.

Acknowledgements

This thesis would not have been possible without the support of many people: It is a pleasure to thank them all. First of all, I express my sincere appreciation to my supervisor, Professor Jian Yang, for her continuous support and guidance that have made my research possible. I would like to show my deep gratitude to my co-supervisor, Dr. Weiliang Zhao, for giving insightful and valuable comments about my research and exceptional support during these three and half years of studies. I sincerely thank all the people in the Department of Computing, Faculty of Science, Macquarie University, for their warm support and help. Finally, I am indebted to my parents and my husband for their love, support, and encouragement throughout the whole program of my study. Without them this work would have never been accomplished.

Abstract

In the service oriented computing paradigm, business processes and services are subject to change and variation arising from both the external and internal requirements of organizations from time to time. A service change can affect its internal supporting business process and a change occurred in a business process often has various levels of impact on its supported services. This thesis provides research results on the challenging issue of change management for service-based business processes.

Different from existing works in the fields of business process change management, this research focuses on the dependencies between services and business processes. In the real world, there are cases when multiple services are supported by a single business process. The changes of a business process and multiple services can affect each other. The dependencies between services and business process make change management complex and challenging. To manage such changes, it is crucial to identify different types of changes associated with services and business processes, analyse change impact patterns, and then decide the effective mechanisms to deal with them.

In this thesis, a service-oriented business process model is developed for capturing the major characteristics of the required change management in the context

described above. Based on the proposed model, the taxonomy is identified for the changes associated with services and business processes. A set of change impact patterns are specified. Each change impact pattern describes a specific type of change effect. With the help of the change taxonomy, the change impact patterns, and the mechanisms for dealing with individual changes, the cascading effect of changes within the service-based business processes can be analysed. As a proof of concept, a prototype has been developed to realize the change management mechanisms presented in this thesis.

Publications based on this Thesis

- [1] Y. Wang, J. Yang, and W. Zhao. Managing Changes for Service Based Business Processes. In Proceedings of the 5th IEEE Asia-Pacific Services Computing Conference, APSCC 2010, 6-10 December 2010, pages 75-82.
- [2] Y. Wang, J. Yang, and W. Zhao. Change Impact Analysis for Service Based Business Processes. In Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications, SOCA 2010, 13-15 December 2010, pages 1-8.
- [3] Y. Wang, J. Yang, and W. Zhao. Service Change Analyzer: An Enabling Tool for Change Management in Service-Based Business Processes, IEEE International Conference on e-Business Engineering, ICEBE 2011, 19-21 October 2011. (accepted)
- [4] Y. Wang, J. Yang, and W. Zhao. A Change Analysis Tool for Service-Based Business Processes, the 12th International Conference on Web Information System Engineering, WISE 2011, 13-14 October 2011. (accepted)
- [5] Y. Wang, J. Yang, and W. Zhao. Change Impact Analysis in Service-Based Business Processes, Service Oriented Computing and Applications. (under revision)

Contents

Acknowledgements	v
Abstract	vii
Publications based on this Thesis	ix
1 Introduction	1
1.1 Research Overview	1
1.2 Research Background	4
1.2.1 Service-Oriented Computing	5
1.2.2 Change Management	6
1.3 Research Requirements and Issues	9
1.3.1 Research Requirements	9
1.3.2 Research Issues	12
1.4 Contributions	15
1.5 Thesis Organization	18
2 Related Work	21
2.1 Change Management for Business Processes	26
2.1.1 Evolution of Workflow Processes	27

2.1.2	Flexibility of Business Processes	30
2.2	Change Management in Service-Oriented Environment	34
2.2.1	Compatibility of Services	37
2.2.2	Change Management for Web Services	41
2.2.3	Service Evolution	45
2.3	Service Adaptation	50
2.3.1	Adaptation of Service Interfaces	52
2.3.2	Adaptation of Service Protocols	54
2.3.3	Flexibility of Service-Based Business Processes	58
2.4	Discussion	61
3	Service-Oriented Business Process Model	63
3.1	A Motivating Example	65
3.2	Service-Oriented Business Process Model	67
3.2.1	Process Layer	68
3.2.1.1	Control Flow Schema	68
3.2.1.2	Information Flow Schema	70
3.2.2	Service Layer	73
3.2.3	Relations Between Process Layer and Service Layer	74
3.3	Discussion	78
4	Change Taxonomy	83
4.1	Service Changes	84

4.1.1	Operation Existence Changes	86
4.1.2	Operation Granularity Changes	87
4.1.2.1	Asynchronous Operation Granularity Change . . .	89
4.1.2.2	Synchronous Operation Granularity Change . . .	96
4.1.2.3	Complex Operation Granularity Change	98
4.1.3	Transition Changes	101
4.2	Process Changes	105
4.3	Discussion	116
5	Change Impact Analysis	119
5.1	Overview of Change Impact Patterns	121
5.2	Direct Impact Scope	124
5.2.1	Direct Impact Scope of a Service Change	124
5.2.2	Direct Impact Scope of a Process Change	129
5.3	Change Impact Patterns	133
5.3.1	Change Impact Patterns for Service Change	133
5.3.2	Change Impact Patterns for Process Change	141
5.4	Discussion	151
6	Change Handling	155
6.1	Handling Individual Changes	157
6.1.1	Dealing with Service Change: Add an Operation	157
6.1.2	Dealing with Service Change: Delete an Operation	159

6.1.3	Dealing with Service Change: Modify Operation Granularity	160
6.1.4	Dealing with Service Change: Reordering Transition Sequences	162
6.1.5	Dealing with Service Change: Modify Conditional and Looping Transition Sequences	165
6.1.6	Dealing with Process Changes	166
6.2	Handling Change Propagation	168
6.3	Change Isolation	173
6.4	Discussion	175
7	Service Change Analyser—A Prototype	177
7.1	Architecture	178
7.2	Data Structure	180
7.3	Components of Service Change Analyser	184
7.3.1	Operation Based Analysis	184
7.3.2	Transition Based Analysis	186
7.4	Running Examples	187
7.4.1	Example for Operation Based Analysis	187
7.4.2	Example for Transition Based Analysis	193
7.5	Discussion	196
8	Conclusions and Future Work	199
8.1	Concluding Remarks	199

8.2 Future Directions	203
Bibliography	205

List of Figures

3.1	A motivating example.	66
3.2	Control flow schema of the sales process.	69
3.3	Information flow schema of the sales process.	71
3.4	Services: (a) the service for buyer s_b ; (b) the service for financial institute s_f	74
3.5	Examples of internal processes and service.	75
3.6	(a) Abstract precedence relation; (b)-(e) internal processes. . . .	78
3.7	(a) Abstract parallel relation; (b) internal process.	79
3.8	(a) Abstract conditional relation; (b) internal process.	79
4.1	Taxonomy of service change.	85
4.2	Operation existence change.	87
4.3	Operation granularity changes.	89
4.4	AOGC type 1 one-to-one change.	90
4.5	An example for AOGC type 1 one-to-one change.	91
4.6	AOGC type 2 one-to-many/many-to-one change.	92
4.7	Examples for AOGC type 2 one-to-many change.	94
4.8	AOGC type 3 many-to-many change.	94
4.9	SOGC type 1 one-to-one change.	97

4.10 SOGC type 2 one-to-many/ many-to-one change.	98
4.11 SOGC type 3 many-to-many change.	99
4.12 COGC type 1 asynchronous-to-synchronous change.	100
4.13 An example for COGC type 1 synchronous-to-asynchronous change. 101	
4.14 Transition sequence order change (TSOC).	102
4.15 Sequential (parallel) to parallel (sequential) transition sequence change (SPTSC, PSTSC).	103
4.16 Adding (removing) conditional transition sequence change (ACTSC (RCTSC)).	104
4.17 Adding (removing) looping transition sequence change (ALTSC (RLTSC)).	104
4.18 Taxonomy of process changes.	106
4.19 Serially insert a process fragment.	106
4.20 Examples of serially inserting an activity.	109
4.21 Parallel insert an activity.	110
4.22 Conditionally insert an activity.	110
4.23 Serially move an activity.	111
4.24 Parallel move an activity.	112
4.25 Conditionally move an activity.	113
4.26 One-to-one replacement.	113
4.27 One-to-many activities replacement.	114

4.28	Parallelize (sequence) activities.	114
4.29	Embed an activity in conditional branch.	115
4.30	Embed an activity in conditional branch.	115
5.1	Overview of change impact patterns.	121
5.2	Change impact patterns.	122
5.3	Structure of the change impact pattern	125
5.4	(a) Service change: TSOC in service s_b ; (b) direct impact scope of the service change.	129
5.5	(a) Process change: replace activities; (b) direct impact scope of the process change.	132
5.6	Change impact pattern 1 Insert a c-Activity.	135
5.7	Change impact pattern 2 Remove a c-Activity.	136
5.8	Change impact pattern 3 Replace c-Activities.	138
5.9	Change impact pattern 4 Move c-Activities.	140
5.10	Change impact pattern 5 Add, Remove or Modify Conditional Branches.	142
5.11	Change impact pattern 6 Add an Operation.	145
5.12	Change impact pattern 7 Remove Operations.	146
5.13	Change impact pattern 8 Change Operation Granularity.	148
5.14	Change impact pattern 9 Change Transition Sequence.	150

5.15	Change impact pattern 10 Add Conditional or Looping Transition Sequence.	152
6.1	Adapter for solving <i>type A</i> case at operation level.	158
6.2	Adapter for solving <i>type B</i> case at operation level.	159
6.3	Reordering activities.	163
6.4	Using reordering template.	164
6.5	An example of deadlock when using reordering template.	165
6.6	Propagation of a service change.	168
6.7	Propagation of a process change.	169
6.8	An example for propagation of a service change.	170
7.1	High level architecture for SCA.	179
7.2	Entity relationship diagram.	181
7.3	Hierarchy diagram.	185
7.4	Browse service operations.	188
7.5	Choose change types of service operations.	189
7.6	Add an operation in parallel to an existing operation.	190
7.7	Impact analysis for the change of adding operation in parallel to an existing operation.	192
7.8	Browse service transitions.	193
7.9	Choose change types of service transitions.	194
7.10	Change transition sequence order.	195

7.11 Impact analysis for change transition sequence order.	197
--	-----

List of Tables

6.1	Structure of reordering template.	164
-----	---	-----

Chapter 1

Introduction

1.1 Research Overview

Service-Oriented Computing (SOC) facilitates the low-cost and rapid composition of loosely coupled software applications. Service-oriented models are introduced to replace or extend traditional process models in order to develop flexible business processes and realise inter-organization integration [72]. Heterogeneous services can be integrated into distributed applications across organization boundaries. The SOC paradigm and service technologies provide a solution for making business processes of organizations accessible for their business partners in order to realise inter-organizational cooperation and collaboration. Organizations possess their own business processes, called private business processes, which are normally invisible to their business partners. In service-oriented environments, the functionalities of business processes are represented as services and exposed to business partners. Each service represents parts of the functions of the associated business process and is an external view of the business process from the view point of a specific business partner. Business partners interact with each other by

invoking corresponding services. A business process may have multiple business partners and each business partner contributes part of the entire functionalities of the business process. For example, a sales process offers the functionalities as receiving purchase orders from customers, checking stock availabilities, sending order acknowledgement, arranging shipment, processing payment and issuing invoices. This sales process interacts with three business partners as a buyer, a shipping company, and a fanatical institute. A business process that involves multiple business partners must expose suitable services for the purpose of interacting with its partners.

In the SOC paradigm, business processes and services are coupled with each other when services expose functionalities of business processes [53, 69]. Services and business processes may change from time to time due to various reasons such as business regulations and application environments [70]. In particular, services may need to change their external specifications including signatures, business protocols, and behaviors in order to engage in business collaborations. A business process specifies a flow of activities and associated data connections in order to fulfil a business goal. Changes associated with business processes at the business logic level include modifications on its structures such as control flows and data connections of activities. For instance, a business process may introduce new activities to its control flow schema in order to provide more functionalities. A specific service change usually affects the associated business process and services and a change that occurs in a business process often has various levels of impact

on the associated services. Service-based applications and information systems will need to operate correctly despite of expected and unexpected changes related to services and business processes. Change management, which is a traditional problem in IT, is critical and challenging in the development and maintenance of service-based applications and information systems [70, 68].

Significant researches have been done in business process change management. These researches focus mostly on business processes without considering too much on their relationships with services. They are inherently inadequate to support change management in the service-based environment. Work also has been done in service evolution [9, 10, 70], service adaptation [12, 17, 28, 39, 46, 75, 113], change management for composite Web services [8, 7, 57, 58, 80, 108] and service protocols [83, 93]. These researches only consider the features of services without considering the internal supporting business processes.

There exist various types of dependence relations between services and business processes. The relations between services and business processes are crucial for the change management in the service-oriented environment for the reason that any change can introduce cascading effects on the associated services and business processes. In particular, a single business process may support multiple services. As mentioned above, a sales process can support three services as a buyer service, a shipping service, and a payment service to its partners when fulfilling a purchase order. The three services are invoked by its partners and will contribute to accomplish the entire functions of the sales process. The change

management becomes complicated due to the dependencies between the business process and related services.

In this thesis, we present our work in dealing with the change management for service-based business processes. The goal of this research is to manage the various types of changes associated with services and business processes by developing effective mechanisms for analysing their impact on associated services and business processes. We propose a service-oriented business process model for capturing the major characteristics of change management issues in the service-oriented environment. The proposed model provides the foundation for building up the taxonomy of changes and generic solutions for detection, analysis, and reaction to various types of changes of business processes and services. Our research targets techniques for understanding and identifying various types of changes, analysing the impact of changes, and facilitating the evolution of services and business processes in the service-oriented environment.

1.2 Research Background

As stated in the previous section, our research deals with the change management for service-based business processes. In this section, we present the background information of the research reported in this thesis. We first introduce the basic concepts of service-oriented computing and then discuss the existing researches about the change management in the field of workflow processes and

in the service-oriented environment.

1.2.1 Service-Oriented Computing

Service-Oriented Computing (SOC) is the computing paradigm that facilitates the development of loosely coupled and distributed applications by using services as fundamental elements [71]. Services are platform independent software components that export their functionalities and properties by external specifications. Services can vary in functions ranging from checking a credit number to executing complicated business processes [69]. Services, which may be provided by different organizations, can interact with each other based on well-defined service descriptions. Descriptions of services contain the information about signatures, business protocols, QoS properties, and behaviors of services that are necessary for service publication, discovery, selection, and integration. Service-based applications and information systems are developed by composing existing services to provide add-value functionalities. The SOC paradigm benefits enterprises in the system development and maintenance phases by reusing existing software components and resources to create highly flexible business processes with low cost.

The Web service technologies provide a technical foundation for building service-based applications and information systems based on the SOC paradigm. Web services are services that are defined by the Web Service Description Language (WSDL) [1], published and discovered in service registries by the Universal Description, Discovery, and Integration (UDDI) [3], and communicate with each

other over the Internet by using the Simple Object Access Protocol (SOAP) [4]. Existing Web services defined by WSDL can be composed into business processes by using the XML based specification: Business Process Execution Language for Web Services (BPEL4WS or simply BPEL) [2]. A BPEL process consists of a set of structured activities that are implemented by Web services offered and accessed over the Internet.

1.2.2 Change Management

Change management is an important issue in the development and maintenance of software systems because of the evolving nature of software. Change management has been studied in a wide range of areas such as software engineering [52, 45, 51, 64, 76], distributed systems [95, 50, 30], database management systems [92, 11, 112, 34], and information systems [90, 26]. In particular, the change management for workflow processes, which is an important area related to our research, has been studied extensively since mid 1990s [97, 19, 40, 41, 77, 79, 78, 85]. The workflow change management concentrates on workflow process evolution and process flexibility. The goal of workflow process evolution is to allow business processes to evolve in a disciplined, controlled, and dynamic manner. The researches on workflow process evolution focus on managing static changes of workflow processes (modifying process schemas) and dynamic changes (process instance migration and adaptation). Process flexibility refers to the capabilities of business processes to dynamically modify their process schemas and instances at

runtime in order to cope with both expected and unexpected circumstances. The works on process flexibility provide various mechanisms for dynamically changing process schemas and individual process instances at run time at certain context.

Service-based applications and information systems operate in a highly dynamic world and are subject to changes arising from the internal and external requirements of organizations from time to time. Change management is a challenging issue in the service-oriented environment due to the distributed and dynamic characteristics of services. The research on the change management in service-based environments is still at its early stages and existing work provides only partial solutions to the issues of service changes and business process changes. The current research in relation to change management in the context of services can be categorized as following:

- **Service compatibility.** In the SOC paradigm, applications and systems are built upon distributed services offered by different organizations. The problem of compatibility between interacting services in software systems arises from many reasons and in many aspects. Proposals have been published in the literature for solving the compatibility of service interfaces, business protocols, and service behaviors [12, 16, 17, 28, 27, 31, 47, 60, 62, 67, 75, 74, 84, 83].
- **Change management for Web service compositions.** A Web service composition comprises a number of Web services that may belong to different ser-

vice providers. Participant services in a service composition are frequently subject to changes due to various reasons such as business rules and regulations. In the current studies on Web service change management, proposals for dealing with changes related to service protocols [83, 93], managing existence changes of services and service operations [8, 7, 57, 58], and aligning choreographies and orchestrations of BPEL processes [80, 108] are reported.

- **Service evolution.** The central problem of service evolution is about service versioning control and the compatibility issue between different versions of a service. Researches on this topic aims to provide effective mechanisms to manage different versions of services and achieve compatibilities between different versions. In his paper [70], Papazoglou has proposed methodologies and theoretical approaches for supporting service evolution. Service changes are categorized into shallow changes and deep changes according to their impact on the entire system. The methodology of change-oriented service life cycle is presented to support handling deep changes of services. There are also a few researches about Web service evolution and mechanisms for controlling evolution of WSDL defined Web service specifications are proposed [18, 44, 43].
- **Service adaptation.** The problem of service adaptation is closely related to the service change management. The research on service adaptation mainly concentrates on providing mechanisms to overcome various types of mis-

matches that may occur among services developed by different parties with the purpose of enabling service interoperability and service replacement. In the literature, the adaptation of service interfaces [28, 39, 75, 113], business protocols [12, 17, 46, 63, 66, 89, 91], and service behaviors are studied [6, 21, 20, 22, 37, 38, 48, 65, 111, 110, 114].

1.3 Research Requirements and Issues

In Section 1.2, we have outlined the context of our research reported in this thesis. In this section, we first discuss the research requirements for the change management in the service-oriented environment and then we identify the important issues on service change management that have not been addressed by the existing works.

1.3.1 Research Requirements

Service-based applications and information systems operate in a highly dynamic environment. Services and business processes are subject to frequent change and variation due to various reasons such as business regulations and application environments. The change management in the SOC context needs to provide mechanisms for:

- Detecting and understanding the varieties of changes that will occur in service-based applications and information systems. When a change hap-

pens, effective mechanisms must be provided for detecting the occurrence of this change and identifying its change type.

- Analysing the impact of various types of changes in service-based applications and information systems. Once a change occurs, its impact on the entire system needs to be clearly analysed before any treatment can be decided to cope with the change. The change impact analysis needs to provide the information about the cause of this change, the change region including the direct and potential impact scopes of this change, and how this change affects the entire system. In particular, the propagation of changes in service-based systems must be clearly analysed.
- Handling various types of changes in service-based applications and information systems. This requires mechanisms for dealing with various types of changes with the goal to control their cascading effect on the entire value-chain.
- Separating change impact analysis and change reaction. Change impact analysis refers to the process of understanding the effect including direct and cascading effect a specific change on the entire value chain. Change reaction refers to the process of deciding mechanisms and then handling a specific change based on its impact. This requirement for managing changes of service-based applications and information systems is based on the following reasons. First, the tasks of change management related to change

impact analysis and change reaction in the services environment described above are complex and difficult. Second, change reaction is context sensitive because a proper change reaction is based on the contextual information such as business regulations, customer requirements and preferences, and costs. In addition, multiple change handling mechanisms may exist for a specific change. Contextual information needs to be taken into consideration when deciding a suitable treatment for coping with a specific change based on its impact. Third, the change impact analysis and change reaction should be reusable for the reason that repetitive types of changes will occur in service-based applications and information systems at different locations. Therefore, to reduce the complexity of change management tasks, develop effective and efficient change handling mechanisms, and advocate reusability in the development and maintenance of service-based applications and information systems, change impact analysis and change reaction need to be separated.

- Evolving services and business processes. Services and business processes may need to evolve frequently to meet the changing requirements arising from different origins. This requires techniques for enabling consistent modifications of services and business processes, managing different versions of services and business processes, and checking compatibilities between different versions of services and business processes.

1.3.2 Research Issues

Current studies on change management in the service-based environment provide partial solutions which cannot fully satisfy the above requirements. We identify the following research issues which are critical for addressing those requirements.

- The complex relations between services and business processes in service-based applications and information systems need to be identified. In the SOC paradigm, business processes and services are coupled with each other when services expose business functionalities of business processes. There exist complicated dependencies between services and business processes. A specific service change usually affects the associated business processes and a change occurs in a business process often has various levels of impact on the associated services. The coupling relations between services and business processes will be crucial for the detection, analysis, and reaction of various types of service changes and process changes.

There are a number of researches on change management that focus on Web services [8, 7, 57, 58, 59] and BPEL processes [80, 108]. These researches concentrate on the features of services and business processes respectively. Unfortunately, the dependencies between services and business processes have not been fully investigated in the existing works about change management in the service-oriented environment.

- A taxonomy of changes related to service-based applications and infor-

mation systems needs to be identified. A change taxonomy provides the foundation for developing mechanisms for change detection, change impact analysis and change reaction.

In the workflow systems, change patterns have been proposed for capturing the major types of changes associated with business processes [100, 105]. These change types consider the features of change management for business processes only and cannot meet the requirements of the service change management. In the area of service computing, changes of services and BPEL processes have been studied in some research works. In addressing the challenges of service evolution, Papazoglou [70] classifies service changes into two broad types as shallow changes and deep changes according to the change impact on the entire business processes. Theorems and methodologies are provided for dealing with the two types of service changes respectively. In the researches about Web services change management, types of Web service changes such as top-down change and bottom-up change in virtual enterprises [57, 58, 59] and internal and external changes in Web service environments [8] have been addressed. These change classifications focus only on the characteristics of services and do not consider the complicated dependencies between services and business processes. In [80, 108], two types of changes as: subtractive change and additive change related to BPEL processes are discussed. This change classification considers only the features of Web service based business processes and does not consider

the associated services.

Classification of mismatches related to service interfaces and business protocols in the area of service adaptation has been identified in [12]. The mismatch types at the level of service interfaces are classified into signatures mismatch and parameter constraints mismatch. The mismatch types at the level of business protocols include message order mismatch, extra message mismatch, missing message mismatch, message split mismatch, and message merge mismatch. These mismatch types are identified with the purpose for developing adapters in order to overcome service incompatibilities and facilitating service replacement.

To support the change management for service-based business processes where services and business processes are coupled with each other with various types of relations, a change taxonomy for services and business processes needs to be identified as the foundation for the development of change analysing and change handling mechanisms.

- The evolving, dynamic, and distributed nature of services requires effective and efficient change analysis mechanisms for service-based applications and information systems. These mechanisms must provide general techniques including calculating impact scopes of a specific change and understanding the direct and potential impact made by this change. In particular, change propagation within service-based business applications and systems caused

by the dependencies between services and business processes needs to be clearly analysed and controlled with the goal to minimize the impact of a specific change. Unfortunately, the existing researches about service change management provide little support for change impact analysis. In most of the research works about service change management, change reactions are provided for handling service changes. We believe that change impact analysis is crucial for developing effective and efficient mechanisms for handling various types of changes.

In conclusion, we have identified the research issues that hinder the development of effective change management solutions for service-based applications and information systems when services and business processes are closely related to each other. In the following section, we will present our approach for solving the above important issues.

1.4 Contributions

In this thesis, we propose an approach for dealing with the change management for service-based business processes. This research targets techniques for understanding and identifying various types of changes, analysing the change impact, and facilitating the evolution of services and business processes in the service-oriented environment. The proposed change management solution aims to manage the various types of changes associated with services and business processes

by developing effective mechanisms for analysing changes and their cascading effect. The proposed approach is based on the established service-oriented business process model that captures the dependencies between services and business processes where multiple services are supported by a single business process. A number of change impact patterns are specified on the basis of the identified types of service changes and process changes. The change propagation within service-based business processes can be analysed with the help of these change impact patterns.

We summarize our contributions as follows:

- A service oriented business process model is developed for capturing the major characteristics of required change management in the context described above. The proposed model provides the foundation to build up the taxonomy of changes and generic solution for identifying, analysing, and reacting to various types of changes of business processes and services.
- The taxonomy for the changes associated with services and business processes is presented. The operation changes and transition changes are identified as the two major types of service changes. The operation changes are further classified into operation existence changes and operation granularity changes. The process changes are classified based on the defined control flow schemas of business processes.
- A set of change impact patterns are specified and the functions for calculat-

ing the impact scopes of service changes and process changes are defined. Each change impact pattern captures a specific type of change effect. The change impact patterns provide intermediate results in the analysis process and they can be reused in the development and maintenance of service-based information systems. With the help of identified impact patterns and specified impact scope, the change impact within service-based business processes can be clearly analysed.

- Mechanisms for handling the various types of individual changes are presented. With the support of the identified change taxonomy, the specified change impact patterns, and the mechanisms for dealing with individual changes, the techniques for analysing the change propagation in service-based business processes are presented. The actual impact scope of a change is defined based on the analysis of the change propagation. The issue of change isolation is discussed and the principles for controlling and cutting off the cascading change effect within service-based business processes are provided.
- A prototype application referred to as service change analyser (SCA) has been built up to show the effectiveness of the proposed general methodology of the change management in service-oriented environment. The SCA accepts changes as its input and it can give the detailed analyzed results for the change impact scope and provide suggestions for potentially

used impact patterns. With the help of the SCA, the impact of a specific change becomes transparent and it is not necessary to analyse the impact of changes manually. The time and cost of change management tasks can be dramatically reduced.

1.5 Thesis Organization

The remainder of this thesis is organized as follows. In Chapter 2, we review the existing works about change management in the area of workflow systems and in the SOC environment respectively. In Chapter 3, we propose the service-oriented business process model. This model defines a type of the dependencies between services and business processes that multiple services are supported by a single business process. In Chapter 4, we present the identified change taxonomy related to services and business processes based on the proposed service-oriented business process model. In Chapter 5, we specify the change impact patterns on the foundation of the proposed service-oriented business process model and the identified various types of service changes and process changes. The change impact patterns consist of the change impact patterns associated with service changes and the change impact patterns associated with process changes. Two functions for calculating the impact scopes of a service changes and a process change are also defined in this chapter. In Chapter 6, we discuss how to handle the various types of changes and analyse their cascading effect on services and

business processes based on the specified change impact patterns. We first provide mechanisms for dealing with individual change impact patterns. Then we discuss the issue of change propagation between related services and business processes and present principles for cutting off the change propagation. In Chapter 7, we discuss the design details of our developed prototype tool, Service Change Analyser (SCA), that implements the proposed change management mechanisms for service-based business processes. In Chapter 8, we conclude our work reported in this thesis and outline the future research directions.

Chapter 2

Related Work

Change management is a traditional problem in IT. Without being related to Service Oriented Computing (SOC), change management has been studied in a wide range of research areas such as software engineering [52, 45, 51, 64, 76], distributed systems [95, 50, 30], database management systems [92, 11, 112, 34], and information systems [90, 26]. In particular, the change management for workflow systems, which is an important area closely related to the change management in the context of SOC, has been studied extensively since mid 1990s [97, 19, 40, 41, 77, 79, 78, 85].

The change management for workflow systems is about managing changes in workflow processes and enabling process evolution. The studies in this area can be summarized into the following two categories:

- Evolution of workflow processes. The research on evolution of workflow processes concentrates on evolving process schemas and adapting process instances when their corresponding process schemas are changed. The evolution of process schemas refers to the process of modifying workflow schemas in order to meet changing requirements. Evolving a process schema usually

generates a series of schema versions. The adaptation of process instances refers to the process of managing running workflow instances when their corresponding process schemas evolve to new versions.

- Flexibility of business processes. The flexibility of business processes refers to the ability of business processes to dynamically modify themselves in order to cope with uncertainties including expected and unexpected circumstances. Different to the evolution of processes, the flexibility of processes is mainly about adjusting process schemas at runtime for individual process instances in certain context.

Change management is a challenging issue in the SOC paradigm due to the distributed and dynamic characteristics of services [70, 68]. Service change management is still at its early stages and current research provides partial solutions for overcoming some problems of change management for service-based applications and information systems. Existing research on service change management mainly focuses on analysing compatibility between Web services, managing changes for BPEL processes and Web service based systems, and supporting the evolution of services including service specifications and service protocols. We summarize these research issues and results as follows:

- Service compatibility. Compatibility is a typical issue in distributed systems. Although Web service techniques provide standard specifications that solve compatibility issue at lower levels of abstractions, i.e., messaging, in-

compatibilities between services still exist at the level of service interfaces, protocols and behaviors. For interacting services, the problem of compatibility arises when participating services change their interfaces and/or protocols. In order to achieve interoperability between Web services, the work on service compatibility provides mechanisms to detect incompatibilities of Web services and devise approaches for solving those incompatibilities [12, 16, 17, 28, 27, 31, 47, 60, 62, 67, 75, 74, 84, 83].

- Change management in Web services environments. Researches concerning this topic target on managing changes of Web service compositions including BPEL processes and Web service based systems such as virtual enterprises [14].

A Web services environment is a highly dynamic environment that is facilitated by the Web service technologies including WSDL [1], SOAP [4], and UDDI [3], et al. A BPEL process is a business process implemented by existing WSDL defined Web services offered by different service providers [2, 42]. A BPEL process itself is exposed as a Web service and can be composed with other Web services. Current studies on change management for BPEL processes concentrate on managing changes of interacting BPEL processes in a cross organization context [80, 108]. In these works, alignment between process choreographies and orchestrations of different interacting partners when changes occur are studied. In addition, change

propagation between trading partners are analysed. The research on managing changes of Web service protocols has also been carried out [84, 93]. Some researches focus on detecting, propagating, and reacting to Web service changes in Web service based systems [8, 7, 57, 58, 59]. For instance, Liu and Bouguettaya [57, 58, 59] study the change management for virtual enterprises.

- Service evolution. Service evolution refers to the process of changing services dynamically and consistently in order to meet changing requirements [70]. Service evolution is a challenging issue. Up to now, there still lack mature techniques to support service versioning control. Partial solutions for supporting service evolution in several respects of services such as service interface, service protocol, and service behavior are reported in the literature.

The recent work presented in [18, 44, 43] suggests proposals for supporting the evolution of Web services. Approaches for evolving service specifications are proposed in [9, 10, 70]. Those researches provide mechanisms to generate and control different versions of services and define a set of theories for checking the compatibility between different versions of a service. There is also research work on managing the evolution of service protocols [84, 83]. A service protocol is a specification that describes the information of how a service client can interact with this service. The research on the

evolution of service protocols is focused on enabling dynamic modification of service protocols and managing running protocol instances when their definitions are changed.

In addition to the above mentioned research topics, service adaptation is an important area closely related to the change management in the SOC paradigm. Service adaptation refers to the ability of services to adjust themselves in order to interact with other services if any mismatches between services occur [70]. Current researches on service adaptation are mainly focused on overcoming mismatches of service interfaces [28, 39, 75, 113], service protocols [12, 17, 46, 63, 66, 89, 91], and behaviors of BPEL processes [6, 21, 20, 22, 37, 38, 48, 65, 111, 110, 114]. Mismatch types/patterns related to service interfaces and/or business protocols are identified in the literature. Approaches for designing adapters/operators to solve the identified various types of mismatches are suggested. Adaption of the behaviors of BPEL processes are studied with the purpose to achieve flexible service behaviors in different context. These researches are mainly concentrated on providing extensions to the current BPEL standard by considering different requirements such as contextual information and QoS properties.

The aim of this chapter is to provide an overview of the existing researches on the change management that have been carried out in the field of workflow processes and in the SOC environment summarized as above. The remainder of

this chapter is structured as follows. First, we analyse the main work published in the literature on the change management in the fields of workflow processes and process aware information systems in Section 2.1. These researches are classified into process evolution (Section 2.1.1) and process adaptation (Section 2.1.2). Then we discuss the current research work on the change management in the SOC environment in Section 2.2. In Section 2.2.1, we review the main results that have been achieved to overcome the various types of incompatibilities between Web services and BPEL processes. In Section 2.2.2, we discuss the research work that has been carried out to manage changes of BPEL processes and Web service based systems. In Section 2.2.3, we analyse the latest researches on service evolution including the evolution of WSDL defined Web services, service specifications, and service protocols. We discuss the main results that have been achieved on service adaptation in Section 2.3. These studies are classified into adaptation of service interfaces (Section 2.3.1), adaptation of service protocols (Section 2.3.2), and flexibility of service-based business processes (Section 2.3.3). Finally, we conclude this chapter in section 2.4 by stating the difference of our research to the above work and pointing out the contribution of our research.

2.1 Change Management for Business Processes

The change management in the area of workflow systems has been extensively studied since mid 1990s. These studies are mainly concentrated on evolving work-

flow processes and enabling the flexibility of business processes. The researches on the evolution of workflow processes aim to allow business processes to evolve in a disciplined, controlled, and dynamic manner. Strategies for managing running process instances when their process schemas evolve are also studied in a significant amount of work. The researches on the flexibility of business processes are focused on dynamically modifying process schemas and instances at runtime in order to cope with both expected and unexpected changes. In the following sub sections, we will review the main research work on the evolution of workflow processes and the flexibility of business processes published in the literature respectively.

2.1.1 Evolution of Workflow Processes

The problem of evolving workflow processes has two facets: i) supporting progressively modifying workflow schemas and managing different versions of workflow schemas, and ii) handling active process instances whose schemas have been changed. The above two aspects of process evolution are studied based on different process modeling languages and various mechanisms are provided [97, 19, 40, 41, 77, 49, 79, 78, 85].

Some work focuses on how to dynamically modify workflow process definitions and ensure the consistency and correctness of process definitions. Casati et al. [19] introduce a set of modification primitives to support structurally changing workflow schemas and preserve their syntactical correctness. The declarative

primitives are used to declare variable existence in workflow schemas. The flow primitives are used to change the flow of tasks in workflow schemas such as forking a task or modifying a task condition. The running process instances whose schemas have been changed are managed by a set of evolution policies, which are classified as abort strategy, flush strategy and progressive strategy. Sadiq [85] proposes a framework to deal with active workflow instances when their process models are modified. The proposed approach has three phases as: the phase of defining modification, the phase of conforming to modification, and the phase of enacting modification. In the phase of enacting modification, the concept of compliance graph is defined as a foundation for handling the affected process instances and make them conform to the changed process models.

Version control mechanisms are proposed in the literature to support the evolution of workflow processes. Joeris and Herzog [40, 41] provide mechanisms for managing the evolution of workflow process schemas and the migration of workflow instances. In particular, they focus on the workflow versioning control. Process tasks and workflow schemas are separately defined in their approach. On this basis, the concepts of process task version and workflow process version are proposed. Then the mechanisms for generating and managing different task versions and process versions are devised. Based on the versioning techniques, they present the instance migration rules depending on execution states. These rules enable late binding and local modification of process instances that help achieve dynamically adjustment of workflow instances. Kradolfer and Gep-

pert [49] present a framework for evolving workflow processes based on workflow versions. Different versions of a workflow process are generated by the defined modification operations and those versions form a workflow version tree. The versionning mechanisms are helpful to define the conditions for migrating running workflow instances. In [97], van der Aalst and Basten propose an approach for managing workflow evolution based on the concept of inheritance. In particular, they concentrate on the inheritance of workflow behavior. By representing workflow processes as Petri nets (WF-nets), the authors are able to specify the inheritance of dynamic behavior of workflow processes. They define four types of inheritance relations based on the notion of branching bisimilarity. Then the inheritance-preserving transformation rules that are used for constructing a subclass of a workflow process are devised. These transformation rules are used to deal with both ad hoc and evolutionary changes of workflow processes. Finally, they present ten transfer rules to handle the migration of workflow instances when the corresponding process schemas are changed. Rinderle et al. [79] propose an integrated approach for managing the evolution of process schemas and the migration of running process instances. Similar process changes are reasoned from previous instances and saved for future use such as evolving process schemas.

Some researches focus on propagating process changes to running instances, particularly to long-running instances [77, 78, 32]. Reichert and Dadam [77] investigate the change management for running workflow instances. In particular, they aim to support ad hoc changes of running workflow instances. A complete

and minimal set of change operations that enable structural modification of workflow instances at run time is devised. They allow workflow instances to deviate their schemas by applying the defined change operations on the instances at run time. The correctness properties are provided to assess the applicability of these change operations. In [78], Reichert, Rinderle, and Dadam specify a set of correctness principles and theorems for automatically migrating a large number of long-running process instances to the corresponding workflow schemas that have been changed. Frank, Fong and Lam [32] also study the migration of workflow instances, especially long running workflow instances, when their process schemas are changed. They propose an approach for dynamically adjusting running process instances when their process schemas are modified. Their approach is to treat the migration of large numbers of long running workflow instances as a process itself. The migration process can judge the migration states of running instances and schedule, configure and implement the migration from old workflow process schemas to the evolved schemas without interrupting the availability of processes.

2.1.2 Flexibility of Business Processes

The flexibility of business processes refers to the ability of business processes to change themselves in order to cope with exceptional circumstances including expected and unanticipated changes [106]. The focus of researches on the flexibility of business processes is to provide mechanisms for dynamically changing individ-

ual process schemas and process instances at runtime. Quite a few approaches have been proposed in the literature for achieving flexible business processes [101, 98, 5, 35, 73, 81, 82, 86, 88, 94, 104]. The key idea of these approaches are to permit part of process definitions remain unspecified priori the implementation of business processes. These unspecified process definitions are defined at runtime by taking contextual information into consideration. Techniques such as late binding, late modeling and late composition of process fragments are proposed to achieve process flexibility. For example, in [5], Adams et al. present an approach for supporting flexible workflow instances at runtime. Their approach is to associate a workflow process task with a set of worklets. A worklet is a self-contained sub process. A top-level workflow process specifies the macro process schema. At the runtime, a process task is implemented by a certain worklet from the associated repertoire depending on the contextual information. Weber, Reichert, and Rinderle-Ma [105, 104] address the change management for process-aware information systems (PAIS) [29]. In PAIS, process logic and application codes are separated. They focus on the change management at the level of process logic. They propose a set of change patterns to support process adaptation both at the schema level and at the instance level. The proposed change patterns capture the structural changes of process schemas and are extensions of the workflow patterns [100]. These patterns are described by high level process changes rather than primitive changes. The change patterns consist of adaptation patterns and patterns for changes in predefined regions. The

adaptation patterns support structural adaptation of process schemas whereas the patterns for changes in predefined regions enable built in flexibility such as late modelling and late binding of process fragments.

Using process variants is a typical approach for realising flexible business processes [98, 35, 36, 81, 82]. A process variant is an adaptation of a reference process that meets certain requirements and is valid in a particular context. Generally, for a particular process type, large collections of process variants exist. In [54, 56], techniques for mining process variants are reported for the development of flexible process-aware information systems. In [35, 36], Hallerbach, Bauer, and Reichert present a PROcess Variants by OPTions (Provop) approach for process variants management and reference model development. They address the issues of reference process design, process variants generation, and evolution of reference processes based on process variants. In particular, they focus on the problem of modeling and managing large collections of process variants. They advocate a Provop process variant life cycle management which comprises three iterative phases: process modeling, configuration of variants, and process execution. In the phase of process modeling, a reference process is defined and a set of change options that are a sequence of high level change operations grouped into change options in order to facilitate the process adaptation are specified. In the phase of process configuration, process variants can be derived by applying a sequence of the predefined change options to the reference process. In the phase of process execution, process variants are deployed and implemented by workflow models.

Rosa et al. [81] propose a questionnaire-driven approach for process adaptation. Reference process models are configured for obtaining intended processes in order to meet specific needs. The questionnaire-driven model consists of a set of questions and facts. Questions are associated with variation points in reference process models and facts are answers for questions. Dependencies between questions and constraints for facts are used complementarily to manage questionnaires.

Some work aims to ensure data flow correctness during process adaptation [77, 78, 94]. The adaptation approach proposed in [77] supports dynamically modifying workflow instances at runtime and at the same time maintaining correctness of data flows. They define the flow of data between process tasks and provide rules for ensuring the correctness of data flows during the runtime modification of process instances. A set of correctness principles and theorems are defined in [78] during changing in-progress workflows. Song et al. [94] propose an approach to support process adaptation with a special focus on preserving data flow correctness. Their approach is to preserve data flow correctness during the adaptation process rather than perform model checking after the completion of adaption. To do so, criteria for task deletion, task insertion, and other adaptation operations such as task substitution are specified.

Weidlich, Weske, and Mendling [107] analyse process changes from a different point of view. They concentrate on the change propagation between business processes that are described by different but related process models within one or-

ganization. The related process models refer to process models that are developed at diverse abstract level, serve for different purposes, and have overlapping functions. They assume that related process models are aligned, i.e., correspondence between elements of different models are created. Based on this assumption, they introduce the notion of behavioral profile to handle the change propagation from a source model to a target model. A behavioral profile includes three types of execution relations between elements of a process model: the strict order relation, the exclusiveness relation and the observation concurrency relation. Changes occurred in a source process are described by a behavioral profile. Then those changes are propagated to a related process with the help of correspondence relations between elements of the source process and the related target process. They define two reduction rules to determine the region in a target process of a change in a source process. A change region consists of control flows that describe the boundaries of this change. A change region is useful for process analyst to handle the change propagation.

2.2 Change Management in Service-Oriented Environment

Services are subject to changes and variations from time to time due to various reasons such as environmental conditions, user requirements, technologies, and legal regulations. The change management in the SOC environment is a critical

and challenging issue [70, 68]. A service may change in many aspects and has various level of impact on the entire value chain. For instance, service interfaces, business protocols, and service behaviors may change overtime and have various types and different level of effect on the interacting services and the associated business processes. Classification of service changes is outlined in [70], in which service changes are categorized into *shallow changes* and *deep changes* depending on the effects and side effects they incur. Shallow changes refer to those service changes such as interface changes and business protocol changes that can be restricted locally. Deep changes refer to those service changes such as the behavior changes of composite services that has deep impact and side effects on the entire value-chain. Current researches on the service change management are still at its early stage.

We categorize the existing work concerning the change management in the context of SOC as follows.

- Compatibility of services. Researches on service compatibility are focused on detecting if two services, especially composite services, can interoperate with each other and resolving the incompatibilities. The compatibility of service interfaces, business protocols, and behaviors are investigated in a significant amount of work [12, 16, 17, 28, 27, 31, 47, 60, 62, 67, 74, 75, 84, 83]. In addition, the problem of compatibility of services is also addressed in the research work related to service evolution [9, 10, 70] and service

adaptation [12, 75, 66]. In the study of service evolution, the issues such as version compatibility including backward compatibility and forward compatibility of service interfaces and protocols are addressed. In the study of service adaptation, types of incompatibilities of services relating to interfaces and business protocols are identified and templates for solving those incompatibilities are suggested.

- Change management for Web services. Current researches on managing changes for Web services provide solutions for dealing with changes related to several aspects of Web services. Some researches investigate the change management for choreographies and orchestrations of composite Web services (BPEL processes) [80, 108]. Some work focuses on handling the changes of business protocols in service compositions [84, 93]. There are also studies that provide solutions for managing changes of virtual enterprises in Web services environments [8, 7, 57, 58, 59].
- Service evolution. Service evolution is a challenging issue [70]. Up to now, there are only partial solutions for supporting service evolution and service versioning control. Current researches on service evolution suggest proposals for evolving service specifications [9, 10, 70], service protocols [84, 83], and BPEL process choreographies [80]. There is also works that studies the evolution of WSDL defined Web services and provide mechanisms for controlling different versions of a Web service [18, 44, 43].

In the following sub sections, we review the major work published in the literature on the change management in service-based environments summarized as above.

2.2.1 Compatibility of Services

Compatibility is a typical issue in distributed systems. The SOC paradigm facilitates cross organizational integrations by using services as the fundamental elements in building distributed applications. Services must interact with each other properly to fulfil business tasks. To do so, services must be compatible. Although current Web service standards such as SOAP and WSDL solve the compatibility of Web services at the lower level of abstraction, i.e., messaging, incompatibilities of Web services still exist at higher levels of abstraction, i.e., service interfaces and service protocols. Most of the current researches study the incompatibilities of Web services from the aspects of service signatures and business protocols. The incompatibilities of service signatures refer to the mismatches in relation to service signatures including parameters mismatch and operations mismatch. The incompatibilities of service protocols refer to the mismatches between external messaging behavior of services [27, 70]. In the following, we examine the existing work on the problem of compatibility of Web services from the aspects of service interfaces and service protocols.

Studies on compatibilities between service interfaces target resolving various types of mismatches at the level of service interfaces. Ponnekanti and Fox [75]

investigate the interoperability of independently involving Web services in Web-based applications. Their goal is to enable substitutions of functionality similar Web services that are derived from a common base. They identify four types of incompatibilities between Web services as structural incompatibility, value incompatibility, encoding incompatibility, and semantic incompatibility. They focus on the structural and value incompatibility referred to as SV-incompatibility. The SV-incompatibility is further categorized into five types as missing methods incompatibility, extra fields incompatibility, missing fields incompatibility, facet incompatibility, and cardinality incompatibility. The combination of static and dynamic analysis can determine SV-incompatibility between Web services automatically. The interoperation between Web services then can be realized by semi-automatically generated cross-stubs. In [12], mismatches in operation name, number, order/type of input/output parameters between operations of different Web services but have same functionality are studied. A mismatch pattern called signatures mismatch pattern is specified for mediating the above interface incompatibilities, based on which adapters can be semi-automatically generated. Ontologies are introduced in [113] to resolve mismatching of service at the level service interfaces. A correlation-based approach is proposed for enabling one-to-many semantic matchings between service interfaces.

The work on compatibility of Web service protocols targets mediating mismatches of ordering of messages. A business protocol contains the information of how a service client can interact with the service properly. A number of studies

have been done to mediate mismatches between service protocols. In [12], Benatallah, et al. identify five types of incompatibilities at the protocol level as: message order mismatch, extra message mismatch, missing message mismatch, message split mismatch, and message merge mismatch. Templates for designing adapters are specified for each of the mismatches. Based on these identified types of mismatches, Nezhad et al. [67] present semi-automated support for detecting and solving these types of incompatibilities at the level of interface and protocol by developing adapters. In [84, 83], Ryu et al. have analysed the compatibility of service protocols in the context of service evolution. The forward compatibility and backward compatibility are defined for supporting dynamic evolution of service protocols. Ponge et al. [74] present an approach for supporting automated analysis of compatibility of Web service protocols. In particular, they focus on the timed protocols that include time-related properties.

There is research work that addresses the compatibility in a context of BPEL processes [16, 17, 31, 47, 60, 62, 67]. Gong et al. [33] report a proposal for automatically solving incompatibilities between interacting business processes. They propose methods for discovering interaction mismatches for interacting business processes and solving these mismatches. They consider two major interaction mismatch types: message order mismatch and missing/extra message mismatch which are similar to the protocol mismatch patterns identified in [12]. A cost model is designed to evaluate the complexity of correcting plans for interaction mismatches. The cost is defined and calculated based on high-level change

operations [55]. Brogi and Popescu [17] propose an approach that can automatically design service adapters for overcoming incompatibilities between interacting BPEL processes. In particular, they focus on the behavioral mismatches between communicating BPEL processes. Their approach for developing adapters consists of four phases. The first phase involves transferring BPEL processes into YAWL defined workflows. The second phase is the adapter generation which is realised by service execution trees of the YAWL workflows. An adapter is specified as a workflow process. The adapters created in the second phase are tested for locks in the third phase and deployed in the four phase. Formal notations such as Petri nets are used for analysing BPEL processes. Martens et al. [62, 61] propose a Petri net based method to analyse behavioral compatibility of BPEL processes. They transform BPEL processes into a type of Petri nets, called the BPEL-annotated Petri net. The concepts of communication graph and reachability graph are proposed to check the compatibility of BPEL-annotated Petri nets. Lohmann et al. [60] present an approach for formally analysing interacting BPEL processes by translating BPEL processes into Petri nets. In [47], König et al. extends BPEL abstract process notations based on Petri nets for the purpose of checking the compatibility of BPEL processes. Foster et al. [31] analyse the compatibility of BPEL processes by translating BPEL processes into finite state process notations.

2.2.2 Change Management for Web Services

Web services are loosely coupled software components that operate in a highly dynamic environment. In a service composition, participant services are subject to frequent changes related to the functional or non-functional aspects of these services. For instance, operations of a participant Web service may become unavailable at the runtime. The change management about Web services focus on detecting changes of participant services in service compositions, analysing change impact on service compositions, checking the compatibilities between interacting Web services, and determining proper reactions for handling the changes. The change management for Web services is at its early stages and existing researches provide only partial solutions for change issues in relation to Web services. In the following, we review the recent researches on managing changes in Web services environments.

Rinderle, Wombacher, and Reichert [80] propose a framework to deal with the changes of interacting BPEL processes in a cross organization setting. Trading partners have their private business processes and public processes. They devise mechanisms for automatically modifying public processes according to the changes of the corresponding private processes. They also discuss the issues of change propagation between the processes of different partners. Wombacher [108] proposes an approach for aligning the choreographies and the orchestration automatically when there are occurrence of changes in process choreographies. The

author aims to solve the problem: if the choreography of a partner changes (this change may originate from the associated orchestration of this partner), how this change affects the choreographies of other partners and in turn their orchestrations. An orchestration is the business logic including business critical information of a service composition. An orchestration is represented by the BPEL specification which is further converted into a Nested Word Automata (NWA). The NWA is enriched by invariants to represent the semantic information of an orchestration. A choreography is the public view for a particular partner and is abstracted from the corresponding orchestration, which is described by a finite state automata [109]. An orchestration may have multiple choreographies for all the involved trading partners. Two types of changes, the subtractive changes and the additive changes, are analysed. When a change occurs in a choreography, this change is propagated to the orchestration.

Ryu et al. [84] focus on managing changes of business protocols in Web service environments. As mentioned above, a business protocol of a Web service contains the information about how a client can interact with the Web service. The authors provide a set of change operators for supporting modifying protocols. In addition, they present mechanisms for analysing change impact of business protocols on service compositions. Mechanisms for managing running protocol instances when the corresponding business protocols are changed are devised based on the change impact analysis. In [93], Skogsrud et al. focus on the change management of security protocols in Web services environments.

In particular, they emphasize on changes of trust negotiation protocols. They provide a framework to support the change impact analysis of trust negotiation protocols on the ongoing trust negotiations automatically.

Change management for Web service compositions is also studied in [8, 7, 57, 58, 59]. These work concentrate on the issues of detecting Web service changes and designing proper change reactions. Ontology is used by some approaches for devising change reactions. Akram, Medjahed, and Bouguettaya [8] propose an architecture to support the change management in Web services environments. Especially, they focus on the problem of ensuring service requests in a dynamic Web service environment. The changes of Web services concerning service request are categorized into two types: the internal change and the external change. An internal change refers to the change about the information provided by a Web service. An external change refers to the existence change of service operations and the service itself. The proposed architecture is based on two key ideas: using ontology to organize Web services and using agents to manage changes. The change propagation is also addressed based on the idea of participant list. In [7], Akram and Bouguettaya present an approach for managing changes in virtual enterprises in the context of semantic web. A virtual enterprise is a Web service composition which is formed to achieve a business goal. The changes occurred in virtual enterprises are categorized into three layers: business, ontological, and service. Changes at the business layer are further classified into three types as efficiency change, regulatory change, and development change. Changes at

business layer are mapped to the ontological layer that comprises a collection of Web services organized based on ontologies. Changes at the ontological layer are mapped onto the service layer. Their approach involves three steps as: detecting changes, propagating changes and reacting to the changes.

In [57], Liu and Bouguettaya propose a framework to manage top-down changes in service-oriented enterprises (SOEs). A service-oriented enterprise (SOE) consists of Web services that are composed to achieve a business goal. The business logic of a SOE is defined by the corresponding SOE schema containing a set of Web services ontologies and the control and data flows associated with these Web services. A control flow of a Web service is specified as logic expressions. A data flow of a Web service is represented by a set of data transfers between Web services. A top-down change refers to the change that is initiated by Web service owners or the SOE itself at the SOE schema level. The authors propose a change model for describing a top-down change. A change model is specified by change conditions, functional modification, and quality modification. They provide approaches for handling top-down changes with respect to service matching, data redistribution and process modification. In [58], Liu and Bouguettaya provide a framework for managing changes of SOEs functionalities. The proposed framework consists of a change reaction manager, a domain knowledge provider, a schema container, and a change log manger. The change reaction manager consisting of three components as a change specification manager, a message exchange mediator, and an execution manager is able to modify

the functional schema of a SOE when a change occurs. Liu et al. [59] also address the change issues in SOEs. In particular, they focus on designing change reactions for coping with changes with the help of ontology. They propose the concept of Web service ontology to support the tasks of modifying service composition schemas and selecting candidate Web services. A node in a Web service ontology is service concept that defines a type of Web services within a specific domain. Two types of relationships: inheritance and dependency are used to connect service concepts in a Web service ontology. Algorithms for querying Web service ontologies are provided.

2.2.3 Service Evolution

Service evolution refers to the process of continuously modifying a service through a series of consistent changes [70]. Current Web service technologies do not support service evolution and service versioning control. A few recent work [18, 44, 43] suggest proposals to support the evolution of Web services and provide service versioning mechanisms based on the current Web service technologies such as WSDL and UDDI. Kalali, Alencar, and Cowan [43] address the requirements for building a Web service registry, called service-oriented monitoring registry, that can monitor the changes of Web services and notify service requestors when the requested Web services are changed. In particular, they aim to devise mechanisms that are able to track the interface changes and the availability of Web services. Their focus is on detecting different versions of WSDL interfaces and

notifying these change information to the service requestors. Brown and Ellis [18] address the issue of Web services versioning. They provide several techniques for dealing with service versioning problems. In particular, they advocate the use of version namespace and version numbers in UDDI entry to manage Web service evolution. Their approach achieves backward compatibility by allowing multiple versions of a Web service to support the earlier versions of that service. Kamin-ski, Müller, and Litoiu [44] propose a service design technique, called chain of adapters, for dealing with service versioning problems. They aim to enable Web service evolution and ensure the back compatibility between different versions of Web services. For two successive versions of Web services, an adapter that mediates the differences between the new version and the old version is generated. Their approach produces a chain of adapters for a Web service. Each adapter in the chain resolves the incompatibilities of two versions related to that Web service. The backward compatibility of different versions of a Web service is achieved by using the chain of adapters.

The evolution of service protocols in Web service environments is studied in [84, 83]. Similar to the work on workflow process evolution, the problem of protocol evolution is discussed from a static aspect (changing protocol descriptions) and a dynamic aspect (managing running protocol instances when the corresponding protocol definitions change especially for long conversations). In their papers, Ryu et al [84, 83] propose a framework to support business protocol evolution in a Web service context. In particular, they focus on the problem

of dynamic protocol evolution where ongoing conversations (protocol instances) need to be handled properly when protocols change. They also address the issue of active conversation migration when there is no formal protocol for this conversation. They devise a set of methods to analyse the impact of a protocol change on the active conversation. The forward compatibility and backward compatibility are defined as a foundation to handle the migration of protocol instances when their protocol schemas have evolved. Based on the impact analysis results, how to migrate a protocol instance is determined.

Rinderle et al address the problem of evolving process choreographies in Web services environments [80]. A process choreography contains the information about interactions between partners. In their paper, the authors propose a framework, called DYnamic CHOReographies (DYCHOR) to support the evolution of process choreographies in a cross organizational context. They aim to solve the problem: if a private process of a trading partner changes, how this change will affect the public process of the partner and the private and public processes of other interacting trading partners. In other words, when there is a change in a process of a choreographies, what is the impact on the processes of other partners involved in this conversation. In the DYCHOR framework, a private process is defined by BPEL. Each private process has multiple public views for its partners. Choreographies between trading partners are built on public views. A public view of a private process is described by an annotated finite state automata that is derived from the private process. A public view of a private process describes how

this private process exchanges messages with a particular partner. They consider two types of basic changes, adding a message sequence and deleting a message sequence. When a change occurs in a private process, the public views of the private process are generated. Then this change will be propagated to the public views of its partners to assist the automatic adaptation of public processes. The private processes of other partners can not be automatic changed.

Important theorems and guidelines for service evolution management that abstract from current Web service standards are proposed in the recent work [9, 10, 70]. Papazoglou addresses the challenging issues of service evolution in [70]. Service changes are categorized into shallow changes and deep changes depending on the effect and side effect they cause on the entire business process value-chain. A theoretic approach is introduced for handling shallow service changes such as structural changes and protocol changes. To manage deep service changes such as behavioral service changes, a change-oriented service life cycle methodology is proposed which consists of four major phases. The initial phase involves understanding the causes for a specific service change and determining the change scopes. The second phase is service change analysis that aims to provide in-depth understanding of the service change. The third phase and fourth phase involve service alignment and implementation. In [9], Andrikopoulos et al. provide a theoretic approach for managing service evolution based on a service specification reference model that is independent of current Web service technologies such as WSDL and BPEL and captures the main characteristics of different

service description models and technologies. The proposed service model consists of three layers: an abstract service definition (ASD) layer, a service schema definition (SSD) layer, and an instance of service schema definition (ISD) layer. The ASD contains common concepts and their relationships for a service definition. The ASD is further described by three layers: a regulatory layer, a behavioral layer, and a structural layer. Each layer contains concepts and relationships that describe certain aspects of a service. For example, the regulatory layer includes concepts relating to business rules, policies and requirements. A SSD defines the schema of a particular service by using the concepts in the ASD. Based on this abstract formal service model, evolution of a specific service schema is realised by applying a series of change operations on its service elements. The proposed service specification reference model provides a formal foundation for defining the concept of consistency of service schema evolution and conformance of service schema versions. In [10], Andrikpoulos et al propose the concept of service contract for achieving controlled service evolution. The ultimate goal is to enable independent evolution of services and at the same time preserve the interoperability between two interacting services. The notion of service contract is proposed as a mutual understanding between two interacting parties. A service contract specifies the provided and required functionality of the interacting services. A service contract is established based on two pairs of views of the service: the exposition and expectation views and the required and provided views. The exposition view of a service refers to the offered functionality of the service. The

expectation view refers to the perceived functionality from the perspective of a customer of the service. The required view of a service consists of the input information of the service. The provided view consists of the information generated by the service. Interoperability between interacting services is discussed based on the service contract. The concepts of contractual invariance and contract evolution are defined to handle changes occurred in interoperating services.

2.3 Service Adaptation

In the previous section, we have reviewed the major research work on change management for workflow processes and in the SOC context. In this section, we discuss service adaptation, which is an important area closely related to the change management in the service-oriented environment. Service adaptation refers to the capability of changing a service itself in order to interact with other services when there are mismatches [70]. Current research work on service adaptation focuses on adapting service interfaces, service protocols, and behaviors of complex services business processes, which can be summarized as follows.

- Adaptation of service interfaces. The issue of adapting service interfaces involves adjusting interfaces to overcome mismatches and incompatibilities at the level of service interfaces. A service interface includes both static and behavioral information about the functionality of this service [24]. Depending on the notations for specifying service interfaces, types of mismatches

between service interfaces are identified and approaches are proposed for overcoming those mismatches [28, 39, 75, 113].

- Adaptation of service protocols. A service protocol describes the desired message exchange sequences between the service and its client. The problem of adapting service protocols is about providing methods to cope with mismatches at the level of service protocols. Adapters are typical and effective solutions to achieve the goal of service adaptation and solve mismatches at the protocol level [12, 63, 66, 91].
- Adaptation of service interfaces and protocols. A number of researches deal with the mismatches between service interfaces and protocols at the same time. The classification of mismatch types between service interfaces and protocols are identified and operators/templates are suggested for the development of adapters in order to resolve the various types of mismatches [12, 46, 63, 66].
- Flexibility of service-based business processes. In the SOC context, business processes are complex services that are created by using widely available services offered by multiple business partners. The requirements for process flexibility in the context service-based environment in order to cope with uncertainties are different to those for traditional workflow processes. A large number of current researches on flexibility of service-based business processes focus on extending the BPEL specification. The behavioral as-

pects of BPEL processes and non-functional aspects of business processes have also been investigated.

In the following sub sections, we review the main research works on service adaptation in relation to service interfaces, service protocols, and BPEL processes respectively.

2.3.1 Adaptation of Service Interfaces

The goal of adapting service interfaces is to mediate mismatches between services at the level of service interfaces in order to preserve the interoperability. Approaches for designing templates/operators have been proposed in the literature to solve different types of interface mismatches.

Ontology is used to mediate mismatches of service signatures. Zeng et al. [113] propose a semantic mediation approach for handling mismatches of service signatures. In their work, ontologies are utilised to realise one-to-one service matchings and also one-to-multiple service matchings. In [39], ontologies are used to facilitate service interface adaptation. Their focus is on the static mismatches between signatures of Web services. Sub-ontology is extracted from the original annotation ontology of Web services in order to produce representation ontology for each Web service. Then interface adapters can be automatically generated based on sub-ontologies of interacting Web services to overcome their signature mismatches.

Ponnekanti and Fox [75] deal with the interoperability of Web services in Web-based applications. They identify four types of mismatches between Web services that are structural incompatibility, value incompatibility, encoding incompatibility, and semantic incompatibility. They focus on the structural and value incompatibility referred to as SV-incompatibility. The SV-incompatibility is further categorized into five types as: missing methods incompatibility, extra fields incompatibility, missing fields incompatibility, facet incompatibility, and cardinality incompatibility. Static and dynamic analysis tools are developed to detect the SV-incompatibility between Web services automatically.

Dumas, Spork, and Wang [28] present an approach to reconcile mismatches between service interfaces. In particular, they deal with behavioral service interfaces that contain not only messages but also the order and constraints between these messages. A behavioral interface is described by a set of traces over an alphabet made up of communication actions. They define six algebraic transformation operators to mediate the different types of interface mismatches. An operator takes a service interface as input and generates an interface as output. They also develop a graphical notation of interface transformation based on the defined algebraic operators. A flow operator is devised to transform one action of an interface to another action. A gather operator transforms multiple actions of an interface into a single action. In addition to the definition of the operator, the authors specify the preconditions of using the gather operator. The preconditions imposed on the operator is to achieve reasonable transformation

and prevent deadlock. A scatter operator is defined to transform one action of an interface into multiple actions. A collapse operator can transform a stream of messages generated by a looping execution of a same action into a single message. A burst operator is defined to transform a single message of an interface into a stream of messages. A hide operator is devised to make an action of an interface invisible with the precondition for using this operator that the associated message of the hidden action is not crucial to the operation of the adapted service.

2.3.2 Adaptation of Service Protocols

As mentioned in the previous section, service protocols contain information about how service clients can interact with the service properly. Functionally equivalent or similar services may have different business protocols in the aspects of message format, message content, and message order. The purpose of adapting service protocols is to resolve incompatibilities of service protocols and achieve inter-operability. Similar to the adaptation of service interfaces discussed in the previous sub section, the main method for mediating mismatches of service protocols is developing various types of adapters. Quite a few approaches have been proposed in the literature for devising adapters that can overcome different types and levels of incompatibilities.

The problem of adapting service protocols is also referred as behavior-based adaptation [17]. There are some works that deal with behavioral mismatches of BPEL processes by devising adapters [17, 89, 91]. Brogi and Popescu [17] pro-

pose a methodology for automatically creating adapters for resolving mismatches between BEPL processes. Their approach is about transforming BPEL processes into YAWL [99] workflows which serves as an intermediary language for analysis of behavioral mismatches and generating adapters. Shan, kumar and Grefen [91] propose an approach for mediating the communication between client and service processes to overcome the protocol mismatches. The protocol mismatches studied in their paper comprise message mismatches and control flow mismatches. The message mismatches refer to the mismatches of format and content of messages between service consumers and service providers. The control flow mismatches refer to the mismatches in message orders between service consumers and service providers. They propose basic adaptation patterns for mediating the identified mismatch types with respect to messages format, content and order. The adaptation patterns can be combined to meet complex adaptation requirements and they can be extended by users. Seguel, Eshuis, and Grefen [89] describe a method for generating adapters for service protocols to overcome the behavioral incompatibilities. Business protocols are specified as BEPL abstract processes and are transformed into tree structures. They construct interaction analysis matrix based on the behavioral relations of two business protocols. Then a minimal interaction set is found based on the interaction analysis matrix. The minimal set of messages that need to be mediated are discovered from the minimal interaction set. Heuristic information is used to find mismatches between service behaviors. Adapters are generated based on the dependence relations between messages in

the minimal interaction set.

Some current work addresses the adaptation of both service interfaces and business protocols. Benatallah et al. [12] provide an approach to develop adapters for Web services. They focus on solving the incompatibility between functionally similar Web services at the level of interfaces and at the level of business protocols. An adapter behaves as a Web service that coordinates a mismatch between two services and does not affect the functionality of a Web service. They address two main requirements for developing adapters as adaptation for compatibility and adaptation for replaceability. They identify a set of mismatch patterns for service interface and protocol mismatches. The mismatch patterns of service interfaces include a signatures mismatch pattern and a parameter constraints pattern. The mismatch patterns at the protocol level includes: message order mismatch pattern, extra message mismatch pattern, missing message mismatch pattern, message split mismatch pattern, and message merge mismatch pattern. Each impact pattern comprises a pattern name, a mismatch type, template parameters, an adapter template and a sample usage. The use of adapters and message brokers is a typical approach to application integration. Kongdenfha et al. [46] propose an aspect-oriented framework to solve the mismatches between an internal service implementation and standard external specifications at the interface and protocol level. The business logic is treated as the main concern and the adaptation logic is specified as crosscutting concerns. An external specification of a service includes an interface and a business protocol [13]. They

identify the taxonomy of mismatch types of service interfaces and business protocols. Based on the mismatch types, they design a set of templates to handle mismatches of interfaces and business protocols. A template comprises a pointcut and an advice. A pointcut is specified as queries over a business process execution. An advice is described by the BPEL specification to express the actions to be performed when handling a type of mismatch. A tool is developed to support the template instantiation and execution. Nezhad, Xu and Benatallah [66] focus on the problem of service matching with respect to Web service interfaces and business protocols. They deal with the matching problem at the interface level and the business protocol level. They introduce a method to identify a message merge/split type of mismatch between service specifications. They also design two algorithms, depth-based interface matching and iterative reference-based interface matching, for service specification matching by incorporating protocol information. Mateescu, Poizat, and Salaün [63] also deal with mismatches between services at the interface and protocol levels. Their approach relies on a type of process algebra. They represent business protocols as symbolic transition systems which are labeled transition systems extended with value passing. The concept of contract notation is proposed to understand and identify the mismatches between service interfaces and protocols. The automatic generation of service adapters are based on a type of process algebra.

2.3.3 Flexibility of Service-Based Business Processes

Flexibility of business processes is an important area in traditional Workflow systems. As we have discussed in Section 2.2, a lot of approaches have been proposed for enabling the flexibility of Workflow processes in order to cope with uncertain circumstances. The Web service technologies enable business processes to be created by using widely available and standardized Web services from multiple business partners. In the SOC environment, the distributed and dynamic characteristics of business processes demand a higher requirements of process flexibility in order to cope with the various changes and uncertainties arising from the dynamic environments.

BPEL is the de-facto standard for composing Web services into a business process. There are quite a few studies about supporting the flexibility of service-based business processes. Zeng et al. [114] advocate a policy-driven approach for exception management in BPEL processes. The key idea is the separation of the business logic and the exception handling policies. The specified exception handling policies are integrated with business logic at runtime to generate exception-aware process schemas. In [65], Mietzner and Leymann introduce the notion of variability descriptor to customize process based and service-oriented applications. A variability descriptor specifies variability points and dependencies between variability points. Variability descriptors are transformed into BPEL processes. Multiple variability descriptors can be assigned to an application tem-

plate to realise different adaptation according to different customer requirements.

Extensions to BPEL have been proposed in the literature to support flexibility and adaptability of BPEL processes. Koning et al. [48] propose VxBPEL as an extension to BPEL. The VxBPEL is able to model some types of variability for Web service based systems. The concept of variability point and variants are proposed to realise flexible BPEL processes. They demonstrate using the VxBPEL to model three types of variabilities in BPEL processes: service replacement, service parameters and system composition. In [21], Charfi and Mezini provide an aspect-oriented approach to extending the BPEL specification. They aim to improve the modularity and adaptability of BPEL processes from the aspects of logging, persistence, auditing, and security. Agarwal and Jalote [6] extend the BPEL specification by enabling specifying non-functional requirements of Web services.

Contextual information is also considered in some research to enable flexible and adaptable Web service based processes. In [22], Choi et al. propose an adaptation approach for service based processes by taking into the contextual information into account. They consider the changes of user's requirements and context. The proposed method supports modifying the workflow definitions (process schemas) for an individual workflow instance at runtime when a change is required. Jaroucheh, Liu and Smith [38] present an aspect oriented framework for adapting service based processes by taking contextual information into consideration. The adaptation logic is specified as crosscutting concerns. The proposed

framework comprises a process model, a context model, an evolution model and a linkage model. Evolution fragments and evolution primitives are defined in the evolution model to capture changes. The adjustment of a particular process can be realised at both the process schema level and the process instance level.

Some existing work investigates the adaptability of service-based processes from the aspect of non-functional requirements such as QoS properties [20, 37, 111, 110]. Chaffe et al. [20] describe an approach for adapting service-based business processes in case of changing QoS properties. They consider the changes of QoS as variations in service performance, cost, and availability. A concept of value of changed information is proposed to measure QoS changes of participant Web services in a business process. Harney and Doshi [37] present a similar approach for adjusting service based processes to QoS changes. Expiration times are used to reduce the complexity of computation during the process adaptation. Wu and Doshi [110] present a decentralized approach for adjusting service based processes to environmental changes such as service performance and availability. In their paper [6], Agarwal and Jalote address the issues of adapting BPEL processes based on non-functional requirements. They present mechanisms for specifying non-functional requirements of BPEL processes which are extensions to the BPEL specification. Participant Web services are selected and can be substituted as required at runtime to implement activities of BPEL processes based on the specified non-functional requirements.

2.4 Discussion

This chapter provides an overview of the researches on the change management both in the field of workflow systems and in the context of SOC. In addition, service adaptation which is an important area closely related to the problem of change management is analysed. As discussed in this chapter, the change management for workflow systems focuses on business processes and studies the evolution of process schemas and/or migration of process instances. These results provide valuable experience for managing changes in service-based applications and information systems in the context of SOC. However, these researches are inherently inadequate to support change management in service-oriented environment because they focus on workflow processes without taking services into consideration.

Service oriented models are introduced to replace or extend traditional process models in order to develop flexible business processes and to realize inter-organization integration. In the SOC paradigm, business processes and services are coupled with each other when services are used to represent and expose the functionalities of business processes [24, 53, 72]. As discussed in this chapter, the change management in the service-oriented environment has been studied from several aspects including service compatibility, Web service change management, service evolution, and service adaptation. However, those researches on change management concentrate only on either service changes or process changes sepa-

rately. There may be complex and complicated dependence relationships between business processes and services. Changes of business processes and services will affect with each other. Unfortunately, the dependencies between services and business processes have not been fully investigated in the existing work. The research reported in this thesis shows our approach for filling the gaps described above. Our change management solution aims to control the cascading effect of changes of business processes and services. In particular, our approach highlights the case that a business process supports multiple services from view points of different partners of the business process. This research targets the guidelines and generic solutions for the change management of service-based business processes.

Chapter 3

Service-Oriented Business

Process Model

Service-Oriented Computing (SOC) facilitates low-cost and rapid composition of loosely coupled software applications. Service-oriented models are introduced to replace or extend traditional process models in order to develop flexible business processes and to realize inter-organization integration [72]. Services can be integrated in distributed applications across organization boundaries. Business processes and services are subject to change and variation arising from both the external and internal requirements of organizations. A specific change usually makes various level of impact on the business processes and services due to various types of dependencies among business processes and services. The change management, which is a traditional problem in IT, is becoming more challenging in the SOC paradigm [70].

As we have discussed in Chapter 2, a significant amount of research on change management has been done in the context of workflow processes. The research focuses on the processes only without taking services into consideration. They

are inherently inadequate to support change management goals in the service-oriented environment. Quite a few work on change management for Web services and service adaptation has been published in the literature recently. From the analysis we have provided in the previous chapter, these researches concentrate on the features of services without considering the associated business processes.

Services and business processes are coupled with each other in various types of ways in the real world. The dependencies between services and business processes will be crucial for the change management in the service-oriented environment for the reason that changes may introduce various level of impact on services and business processes. In particular, a single business process may support multiple services. Change management becomes complicated due to the dependencies between the business process and different services. In the next section, an example will be given about how a business process supports multiple services. The example shows readers the motivation of this research.

The objective of this chapter is to propose the service-oriented business process model for capturing the major characteristics of change management in the service-oriented context. In particular, our model describes a type of dependencies between services and business processes that multiple services are supported by a single business process. The proposed model is the foundation to build up the taxonomy of changes and generic solution for detection, propagation, and reaction to various types of changes of business processes and services.

This chapter is structured as follows. We first introduce an example, a sales

scenario, which shows the basic characteristics and requirements of a service-based business process model. Then we define the service-oriented business process model in three aspects as the process layer, the service layer, and the relations between the two layers. The process layer consists of business processes that are defined by control flow schemas and information flow schemas. The service layer comprises services that are defined in terms of their behavioral characteristics. The relations between the process layer and the service layer describe how business processes in the process layer and services in the service layer are associated with each other.

3.1 A Motivating Example

Let us consider a typical sales scenario. A sales process can receive an order from a buyer, check the stock availability, and send acknowledgement to the buyer. If an order is accepted, the sales process will send the bill to the buyer. The payment is sales by a finance institute. The buyer will be issued with an invoice after the payment. In the meantime, the sales process handles the shipment of the goods with the support of a shipping company. The buyer will be notified with a shipping schedule. In this scenario, the sales process interacts with three trading partners as a buyer, a finance institute, and a shipping company. In SOC environments, these three partners interact with the sales process by invoking the corresponding services exposed by the sales process. The three services are s_b ,

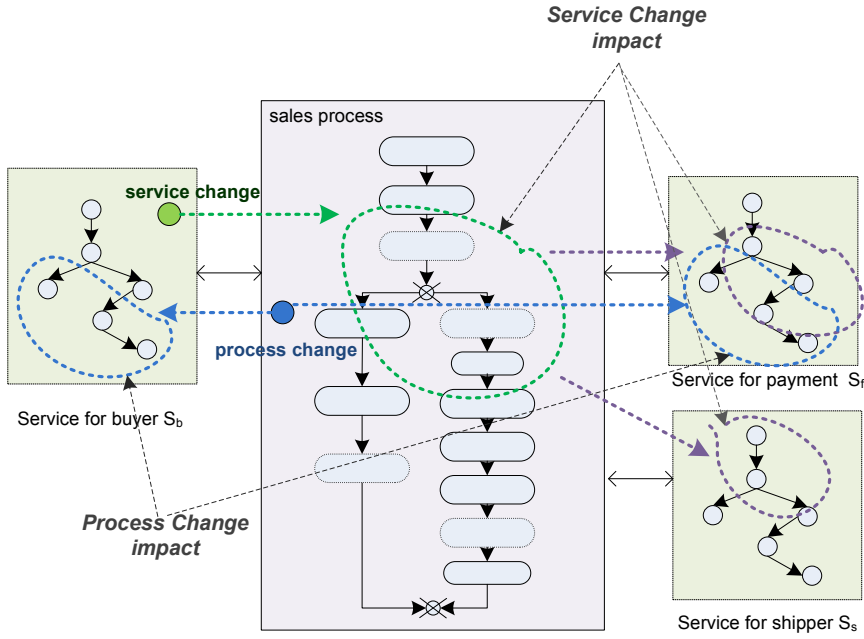


Figure 3.1: A motivating example.

s_f and s_s exposed for interacting with the buyer, the financial institute, and the shipping company (cf. Figure 3.1). Each service is an external view of the sales process from a particular trading partner. For example, the service s_b contains operations that offer functionality such as receiving purchase order, sending order confirmation, sending and receiving billing information as well as sending shipping notice. Private tasks of the sales process such as checking stock availability and processing an invoice are hidden from its partners.

In the real world, there are cases that are similar to the above scenario where multiple services are supported by a single business process. The dependence relation between the services and the business process make change management complicated and challenging. On one hand, when a change occurs in any of the

services, this service change may impact on its internally supporting business process and the other services. As shown in Figure 3.1, a change in the buyer service s_b will impact on the sales process and the payment service s_f as well as the shipper service s_s that are supported by the sales process. On the other hand, when a change occurs in the business process, this process change may impact on the services associated with this business process. The changes of a business process and multiple services will affect with each other. For instance, in Figure 3.1, a change in the sales process will have impacts on its supported services. The above scenario provides the basic requirements and motivation for our approach about the change management for service oriented business processes. The typical case that multiple services are supported by a single business process will be highlighted in this research.

3.2 Service-Oriented Business Process Model

As shown in the above sales scenario, the proposed service-oriented business process model captures a typical case of dependence relations between services and business processes that multiple services are supported by a single business process. In this section, we introduce the proposed model and the basic concepts that will be used in the later chapters. The model consists of two layers: the process layer and the service layer. The process layer contains business processes, which will be referred to as *internal processes* in the following part of the thesis.

The service layer contains services. We define the process layer and the service layer respectively in the following sub sections, and then we discuss the relations between the process layer and the service layer.

3.2.1 Process Layer

The process layer contains business processes which are referred to as internal processes. An internal process is defined by its control flow schema and information flow schema.

3.2.1.1 Control Flow Schema

A control flow schema specifies the control relations between activities. Here activities are categorized into *private activities* (p-activities) and *communication activities* (c-activities) [24, 27, 109]. A *private activity* (p-activity) performs a private task which is invisible to the partners. A *communication activity* (c-activity) is defined for exchanging information with a particular partner. C-activities are further categorized into four types: *receive*, *send*, *receive/reply*, *invoke/recieve*.

Definition 1 (Control flow schema) The control flow schema of an internal process is defined as a 3-tuple: $CFS = (A, C, E)$, where:

- $A = \{a_1, \dots, a_n\}$ is a set of activities. Each activity $a \in A$ is associated with an operation that implement the activity. If a is a c-activity, $a.partner$ refers to the trading partner that a intends to interact with;

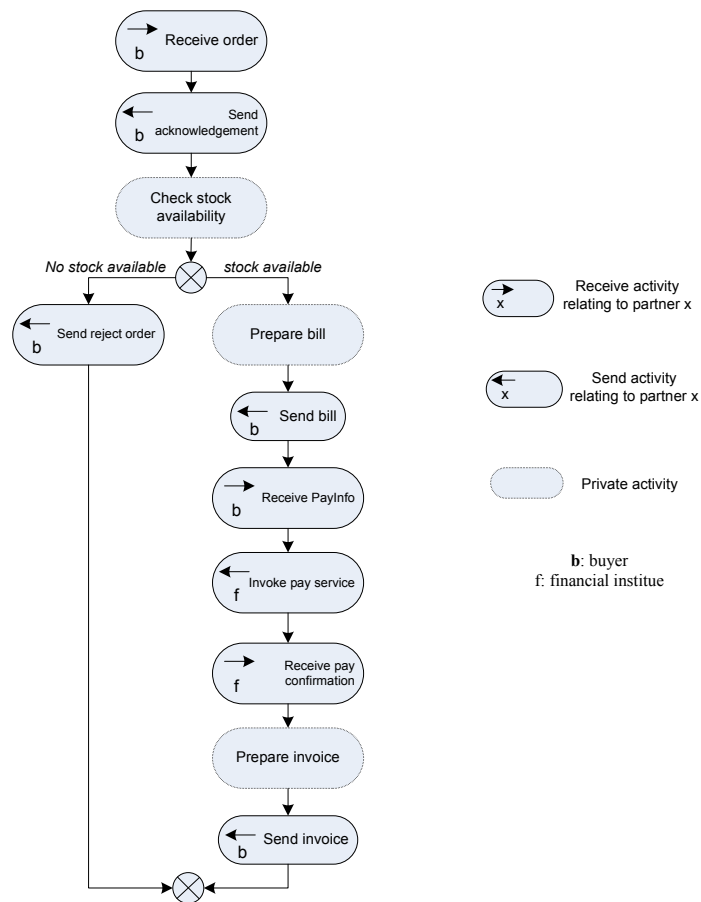


Figure 3.2: Control flow schema of the sales process.

- $C = \{\oplus_{split}, \oplus_{join}, \otimes_{split}, \otimes_{join}\}$ is the set of control connectors, where \oplus represents the *and* connector while \otimes denotes the *xor* connector;
- $E = \{e_1, \dots, e_m\}$ is a set of directed edges associated with the activities and connectors.

Figure 3.2 shows the control flow schema of a sales process which intends to interact with two trading partners: a buyer and a financial institute.

3.2.1.2 Information Flow Schema

The information flow schema of an internal process defines how data is transferred between activities. The information flow is key for understanding the data dependency between activities which is indispensable for analysing change impact. We define the information flow schema of an internal process that is similar to the data flow schema defined in workflow systems [77].

Let $D = \{d_1, \dots, d_n\}$ be a set of data elements associated with the internal process. Every activity a has input parameters, denoting as $InPARs(a)$ and output parameters, denoting as $OutPARs(a)$. A data connection is defined as

$$dc = (d, a, par, mode)$$

where $d \in D$, $a \in A$, $par \in InPARs(a) \cup OutPARs(a)$, and $mode \in \{read, write\}$.

Definition 2 (Information flow schema) Let $CFS = (A, C, E)$ be the control flow schema of an internal process, the information flow schema is defined

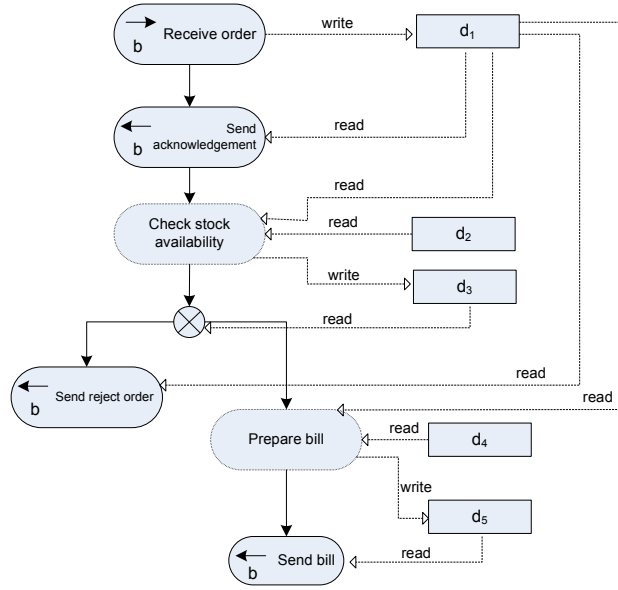


Figure 3.3: Information flow schema of the sales process.

as the set of all data connections:

$$IFS = \{dc_1, \dots, dc_m\}$$

Figure 3.3 shows part of the information flow schema of the sales process. The dashed arrows are data connections. After receiving the order from a buyer, the activity *Receive order* writes d_1 with the information: *customer order*. The data connection is

$$(d_1, \textit{Receive order}, \textit{customer order}, \textit{write})$$

Then *Send acknowledgement* reads from d_1 as input parameter and sends an

acknowledgement to the buyer. The data connection is:

$$(d_1, \textit{Send acknowledgement, customer order, read})$$

Data dependency between activities can be derived from data connections. There exists data dependency between *Receive order* and *Send acknowledgement* because the input of the latter is retrieved from the output of the former. We say *Send acknowledgement* depends on *Receive order* in terms of data.

Definition 3 (Activity data dependency) Let $CFS = (A, C, E)$ be the control flow schema of an internal process, $IFS = \{dc_1, \dots, dc_m\}$ be the information flow schema, and $D = \{d_1, \dots, d_n\}$ be the set of data elements associated with the internal process. For $a_i, a_j \in A$, a_i depends on a_j in terms of data, denoting as $a_i \dashrightarrow^D a_j$ iff:

- $\exists dc_x, dc_y \in IFS$, that $dc_x = (d, a_j, par_s, write)$, $dc_y = (d, a_i, par_t, read)$, where $(d \in D)$, $par_s \in OutPARs(a_j)$ and $par_t \in InPARs(a_i)$;
- a_j precedes a_i in CFS .

To sum up, an internal process IP is defined by a 2-tuple:

$$IP = (CFS, IFS)$$

where CFS is the control flow schema of IP and IFS the information flow schema.

3.2.2 Service Layer

The service layer contains services that are supported by the internal process. Every service is an external view of the internal process from the view point of a specific trading partner.

To achieve effective interaction with its partner services, a service interface exposes the observable behavior rather than a list of operations [15, 23, 87]. We define a service as a set of operations and the invocation relations between these operations.

Definition 5 (Service) A service is defined by a 2-tuple $s = (O, T)$, where:

- $O = \{o_1, \dots, o_n\}$ is a set of operations. Each operation $o_i \in O$ is associated with a c-activity. Every operation has a set of messages;
- $T \subseteq O \times O$ is a set of control relations between operations.

Each transition $t = (o_i, o_j) \in T$ ($o_i, o_j \in O$) denotes the invocation from operation o_i to operation o_j . We call o_i the origin operation of the transition t and o_j the destination operation. For each transition $t \in T$, $c(t)$ denotes the transition constraint on t . A transition constraint is a boolean expression. If $c(t) = \emptyset$, transition t happens immediately after the execution of the origin operation. If $c(t) \neq \emptyset$, transition t occurs when the boolean expression in $c(t)$ is evaluated to be true during execution.

Figure 3.4 shows two services supported by the sales process (cf. Figure 3.2). Figure 3.4(a) is the service s_b for the buyer, which contains six operations and

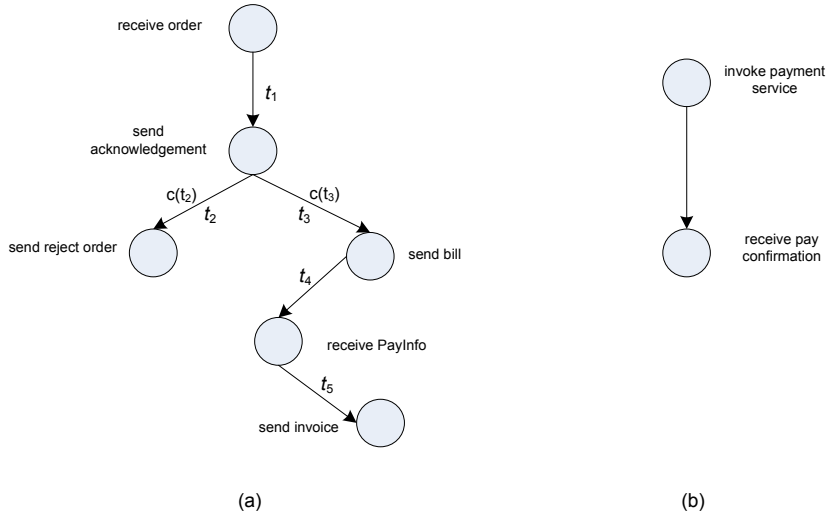


Figure 3.4: Services: (a) the service for buyer s_b ; (b) the service for financial institute s_f .

five transitions. Among the transitions, t_2 and t_3 are governed by two constraints $c(t_2)$ and $c(t_3)$ respectively, which means that after the invocation of operation *send acknowledgement*, whether *send reject order* or *send bill* will be executed depends on the value of $c(t_2)$ and $c(t_3)$. Figure 3.4(b) is the service s_f for the financial institute. For simplicity, messages associated with these operations are not shown in the figure.

3.2.3 Relations Between Process Layer and Service Layer

In this section, we discuss how services and internal processes are related with each other.

An internal process may support multiple services. Every activity in the internal process is associated with an operation that implements the task specified by the activity. Operations that are associated with c-activities are exposed

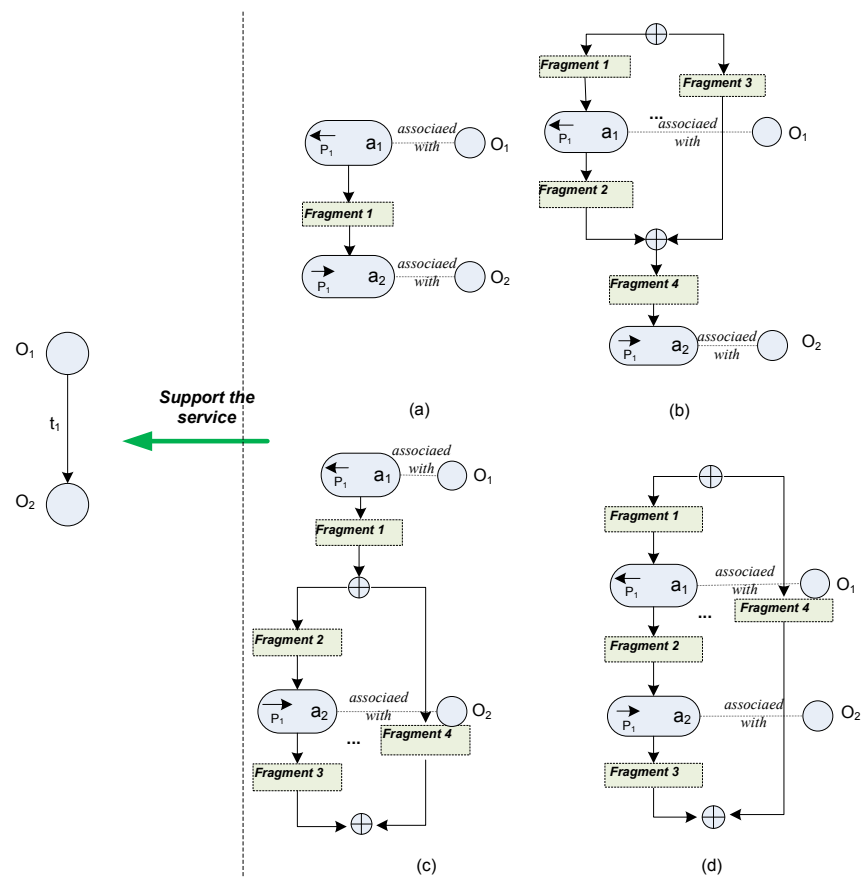


Figure 3.5: Examples of internal processes and service.

to corresponding partners. The operations that are related to one partner are grouped as a service. For example, the service s_b contains six operations relating to the trading partner *buyer*. Transitions between operations are based on the control flows associated with corresponding activities. In the service s_b , transition t_3 is obtained from the control flow between activities *Send acknowledgement* and *Send bill*, which are both c-activities for interacting with a buyer. As the activity *send bill* is in a conditional branch, t_3 is governed by a constraint $c(t_3)$ that is obtained from the conditions of the correspondent xor connector. Thus, the service for a particular trading partner is abstracted from the internal process by exposing operations associated with the c-activities relating to the partner and generating transitions between operations from the control relations between corresponding activities.

A service is an external view of the internal process from the view point of a particular trading partner. Transition sequences of operations reflect the *abstract control relation* between associated activities in the internal process. For example, in Figure 3.4(a), there is a transition sequence *receive PayInfo* t_5 *send invoice* in service s_b . From this transition sequence, we know that the activity *Receive PayInfo* must precede *Send Invoice* in the sales process. Note that there are other activities between the two activities but are invisible to the *buyer*. We summarize the relations between services and internal processes as follows.

- Operations associated with p-activities in the internal process are invisible

to all the trading partners.

- For every c-activity a , if $a.partner = p_1$, the operation associated with a is shown in the service s_{p_1} .
- Transitions between operations in the service s_{p_1} are based on the control relations between the c-activities which intend to interact with p_1 .
- Conditions controlling the execution of c-activities in the internal process are related to the constraints of the corresponding transitions in the services.

We provide an example shown in Figure 3.5 to illustrate the above relations between services and internal processes. As shown in Figure 3.5, there are four internal processes. Each internal process contains the c-activities a_1 and a_2 relating to partner p_1 . We denote the operations associated with a_1 and a_2 as o_1 and o_2 . The process fragments in these internal processes are sub processes containing only the p-activities and the c-activities that are irrelevant to the partner p_1 . The four internal processes shown in the figure support a same service s_{p_1} . That is, given an activity a of a process fragment, a is either a p-activity or a c-activity with $a.partner \neq p_1$. The four internal processes have different control flow schemas. However, according to the relations listed above they support the same service s_{p_1} that contains the transition sequence $o_1 t_1 o_2$. This example also illustrates that a service is an external view of the internal process. The

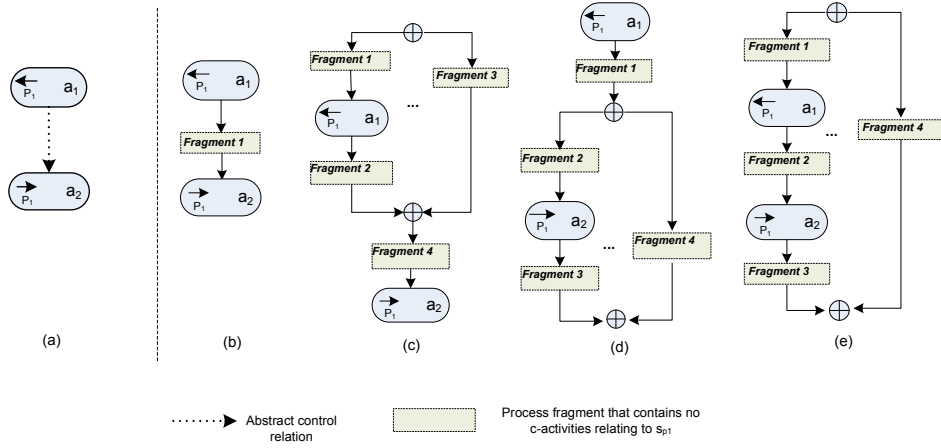


Figure 3.6: (a) Abstract precedence relation; (b)-(e) internal processes.

transitions between operations of a service reflect the control relations between associated c-activities in the internal process.

For this reason, we propose the concept of *abstract control relation* to describe the control relations between c-activities relating to one trading partner. We define three types of abstract control relations: *abstract precedence relation* (cf. Figure 3.6), *abstract parallel relation* (cf. Figure 3.7), and *abstract conditional relation* (cf. Figure 3.8). These abstract control relations will be used in the later chapters for describing the impact on the internal process made by a particular service change.

3.3 Discussion

In this chapter, we have presented a service-oriented business process model that captures a typical type of dependence relations between services and business

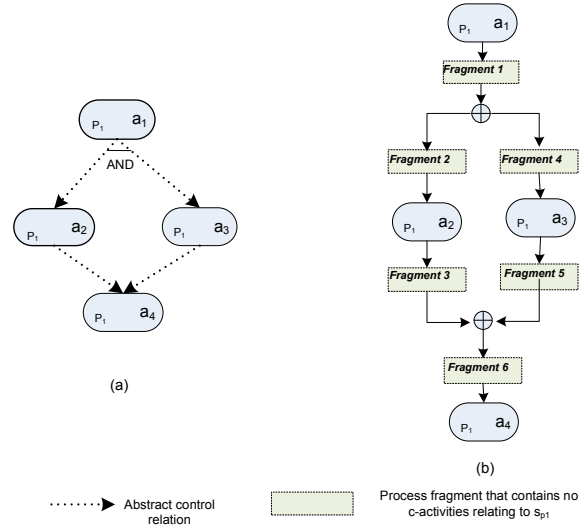


Figure 3.7: (a) Abstract parallel relation; (b) internal process.

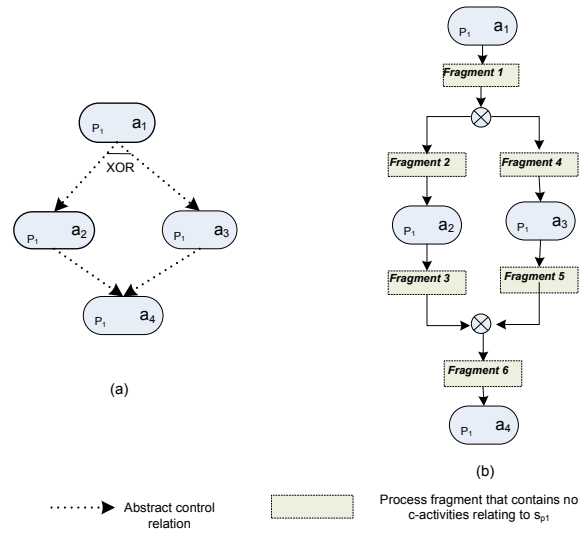


Figure 3.8: (a) Abstract conditional relation; (b) internal process.

processes, where multiple services are supported by a single business process. An internal process is defined by a control flow schema and an information flow schema that describe the control and data aspects of a business process respectively. A service is defined as a set of operations and transitions associated with the operations. This definition captures the behavioral characteristics of a service. A business process may support multiple services. Each service is an external view of the associated internal process from the view point of a particular trading partner. The proposed service-oriented business process model provides the foundation for our research on change management in the SOC context, including identifying the various types of service changes and process changes in service-based business processes and devising approaches for change impact analysis and change reactions.

In the real world, various types of dependence relations between services and business processes exist. The dependence relations between services and business processes are crucial for understanding the impact of a specific change and developing mechanisms for handling changes. Therefore, clarifying these various types of dependence relations are vitally important for identifying the types of changes that may occur in the systems, understanding the impact of a specific change, and devising effective solutions for handling changes. Existing works on change management concentrate on business process changes or service changes separately. The dependencies between services and business processes have not been fully investigated in existing works on change management in the context

of SOC. The research reported in this thesis shows our approach for filling the gaps described above. This chapter presents our service-oriented business process model as the first step for developing the change management mechanisms. This model provides foundation for identifying the various types of changes associated with services and processes and change impact patterns in the next Chapters.

Chapter 4

Change Taxonomy

As discussed in the previous chapter, services and business processes are coupled with each other in the real world. The proposed service-oriented business process model describes a type of dependencies between services and business processes when multiple services are supported by a single business process. Due to various reasons such as business regulations and application environments, services and business processes need to change from time to time. A service may need to change in many aspects such as operation granularity, operation invocation sequences, and operation existence et al. in order to engage in business collaboration. A business process may also need to change in many aspects of its structure such as changing the existence of activities or embedding activities in a conditional branch et al. The changes of services and business processes will affect with each other due to their coupling relations. Moreover, different types of service changes and process changes may have distinct impact on the associated services and business processes. Understanding the various types of service changes and process changes is a priori for developing effective and efficient mechanisms for analysing the impact caused by a specific service change and a process change

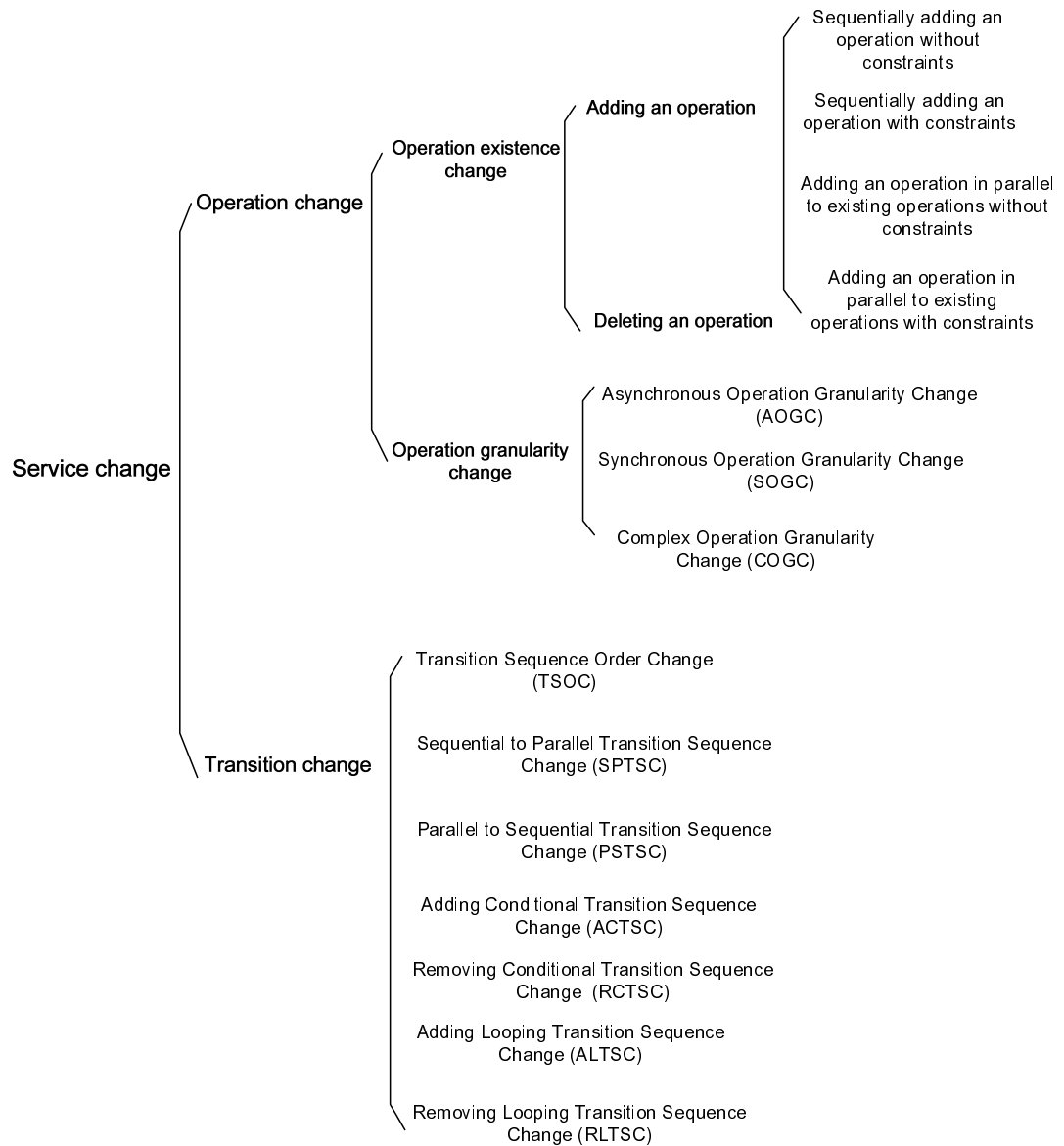
and controlling those changes.

The objective of this chapter is to present a taxonomy of changes associated with services and business processes that provides the foundation for change impact analysis and change reaction for service-based business processes [103]. We identify two major categories of changes as service changes and process changes based on the proposed service-oriented business process model defined in the previous chapter. The operation changes and transition changes are classified as the two major types of service changes. The operation changes are further categorized into existence changes and granularity changes. The types of process changes are identified based on the activities and their control relationships.

This chapter is structured as follows. First we present the taxonomy of service changes in Section 4.1. Two broad categories of service changes as *the operation changes* and *the transition changes* will be introduced in Section 4.1.1 and Section 4.1.2 respectively. Then we present the classification of process changes in Section 4.2. Finally, we conclude this chapter in Section 4.3.

4.1 Service Changes

Two major types of service changes are identified: the operation changes and the transition changes. The operation changes are further categorized into operation existence changes and operation granularity changes (cf. Figure 4.1).

**Figure 4.1:** Taxonomy of service change.

4.1.1 Operation Existence Changes

An operation existence change occurs due to adding or removing operations in a service. There are four possible ways of adding an operation as shown in Figure 4.2: *sequentially adding an operation without constraints*, *sequentially adding an operation with constraints*, *adding an operation in parallel to existing operations without constraints*, and *adding an operation in parallel to existing operations with constraints*.

- *Sequentially adding an operation without constraints* refers to the change that an operation is added in between two operations unconditionally. Figure 4.2(a) exemplifies this type of operation existence change.
- *Sequentially adding an operation with constraints* refers to the change that an operation is added in between two operations conditionally. Figure 4.2(b) shows an example of this type of change. The operation o_x is between o_i and o_j with constraints c_{x1} and c_{x2} which are mutually exclusive. If c_{x1} is satisfied, o_x is invoked. Otherwise, c_{x2} is evaluated to be true and o_x is skipped.
- *Adding an operation in parallel to existing operations without constraints* refers to the change that an operation is added in parallel to existing operations unconditionally. Figure 4.2(c) shows that o_x is added in parallel to o_j without any constraints. After the execution of o_i , o_j and o_x are invoked simultaneously.

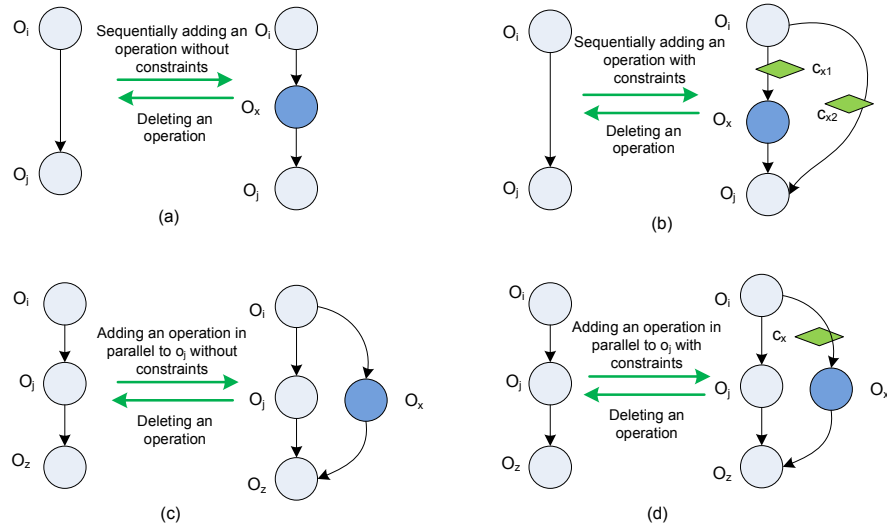


Figure 4.2: Operation existence change.

- *Adding an operation in parallel to existing operations with constraints* refers to the change that an operation is added in parallel to existing operations conditionally. Figure 4.2(d) shows that operation O_x is added in parallel to O_j with a constraint C_x . This means that after the execution of O_i , O_j and O_x are executed. However, O_x will be invoked only if C_x is evaluated to be true.

The *deleting an operation* change refers to that an operation is removed from a service. The examples in Figure 4.2 also illustrate this type of change.

4.1.2 Operation Granularity Changes

Operation granularity change refers to the change when existing operations are reorganized into different grained operations. Changing granularity of operations is a service design concern in order to meet business requirements from both the

organization and the partners.

We consider *asynchronous operation* with only input messages and *synchronous operation* with both input and output messages [42, 80]. The input messages of an operation are called the input parameter, and the output messages the output parameter. We assume in the following discussion that two operations that take the same parameter as input and generate the same output parameter perform the same functionalities [96]. Based on this assumption, the operation granularity change is discussed by analyzing the change of input and output parameters. We focus on the change of information structure that an operation can process. The information structure of an operation refers to the basic data types an operation can handle. Two functions are defined to retrieve the basic data types from the input and output parameters of an operation. Suppose *dataType* is the basic XML data types used by operation definition, the functions are defined as

$$InInfo : O \rightarrow \wp(dataType)$$

and

$$OutInfo : O \rightarrow \wp(dataTypes)$$

InInfo takes an operation as the input and generates the set of basic data types of the input parameter, whereas *OutInfo* takes an operation as the input and generates the set of basic data types of the output parameter.

We have identified three major types of operation granularity change: the

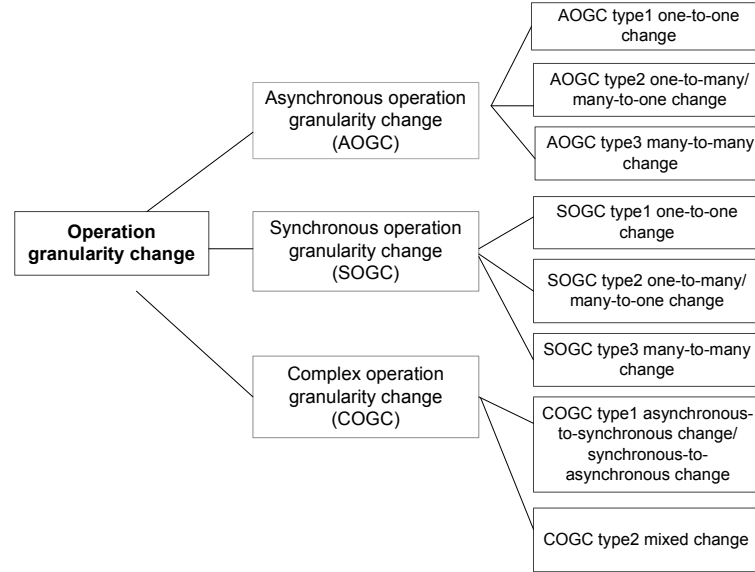


Figure 4.3: Operation granularity changes.

asynchronous operation granularity change (AOGC), the synchronous operation granularity change (SOGC) and the complex operation granularity change (COGC). Figure 4.3 presents the sub types of the three types of operation granularity change. We discuss these three types of operation granularity change in more detail in the following sub sections.

4.1.2.1 Asynchronous Operation Granularity Change

The asynchronous operation granularity change (AOGC) refers to the granularity change of asynchronous operations. We classify AOGC into: *AOGC type 1 one-to-one change*, *AOGC type 2 one-to-many/ many-to-one change*, and *AOGC type 3 many-to-many change*.

AOGC type 1 one-to-one change describes that one asynchronous operation

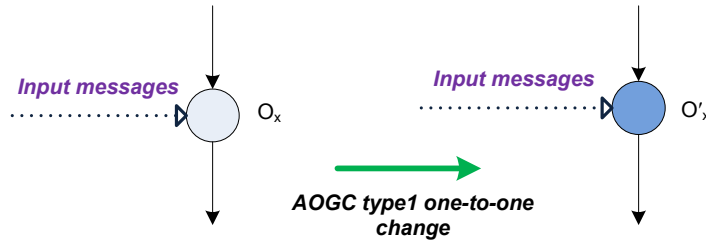


Figure 4.4: AOGC type 1 one-to-one change.

o_x is modified to another asynchronous operation o'_x (cf. Figure 4.4).

The following relations between o_x and o'_x exist:

-

$$InInfo(o_x) \subset InInfo(o'_x)$$

which means that o_x is modified to o'_x by accepting more data types as its input parameter.

-

$$InInfo(o_x) \supset InInfo(o'_x)$$

which means that o_x is modified to o'_x by requiring less data types as its input parameter.

- the above two conditions do not hold and

$$InInfo(o_x) \cap InInfo(o'_x) \neq \emptyset$$

This relation means that o'_x accepts some of the data types that are accepted

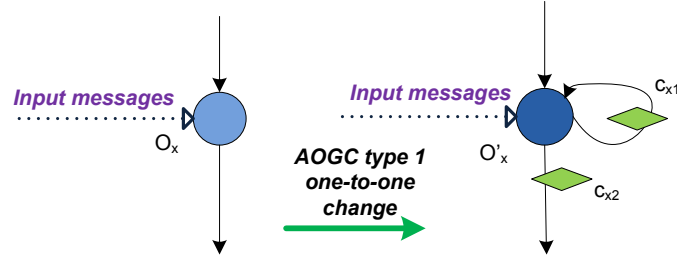


Figure 4.5: An example for AOGC type 1 one-to-one change.

by o_x and o'_x also accepts data types that are not accepted by o_x .

Operation granularity change involves transition sequences changes. These transition sequence changes indicate how the modified operations are organized in a service. For example, in Figure 4.5, the asynchronous operation o_x is changed to another asynchronous operation o'_x . Observe that o'_x is embedded in a looping transition sequence, which enables o'_x to be invoked multiple times.

AOGC type 2 one-to-many/ many-to-one change defines the granularity change between an asynchronous operation and a set of asynchronous operations. The *one-to-many change* covers the case that an operation is split and modified into a set of operations. The *many-to-one change* covers the case that multiple operations are merged and modified into one operation (cf. Figure 4.6). We discuss the *one-to-many change* in detail. The *many-to-one change* can be similarly defined.

Let o_x be an asynchronous operation, o_x is split into a set of operations $O_Y = \{o_{y1}, \dots, o_{yt}\}$, where $\forall o_{yj} \in O_Y (j = 1, \dots, t)$, $InInfo(o_x) \cap InInfo(o_{yj}) \neq \emptyset$.

The relations between o_x and o_Y are listed below.

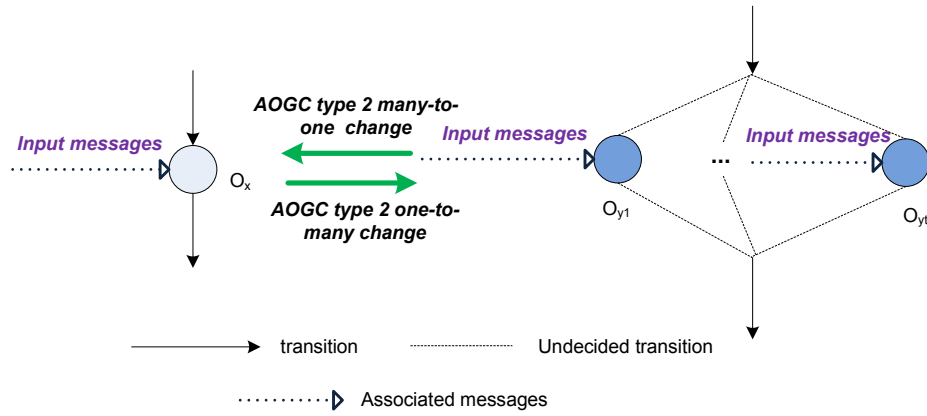


Figure 4.6: AOGC type 2 one-to-many/many-to-one change.

•

$$InInfo(o_x) = InInfo(O_Y)$$

which means that operation o_x is split into functionally equivalent finer operations.

•

$$InInfo(o_x) \subset InInfo(O_Y)$$

which means that operation o_x changes to a set of operations O_Y which is capable of processing more data types in its information structure than the operation o_x .

•

$$InInfo(o_x) \supset InInfo(O_y)$$

which means that operation o_x changes to a set of operations O_Y that

accepts less data types as its input parameters than the operation o_x .

- the above three relations do not hold and

$$InInfo(o_x) \cap InInfo(O_y) \neq \emptyset$$

This relation describes that O_Y covers only part of the functionality of o_x . Moreover, O_Y is capable of processing more data types that are not accepted by o_x .

Note that the operations o_{y1}, \dots, o_{yt} can be organized in various ways. We use dotted lines to represent the undecided transitions between these operations (cf. Figure 4.6). Figure 4.7 gives examples of the various ways to organize the changed operations during the *one-to-many change*. In this example, we consider an operation o_x is split into two operations o_{x1} and o_{x2} . The operations o_{x1} and o_{x2} can be invoked sequentially without constraints (cf. Figure 4.7(a)) or with constraints (cf. Figure 4.7(b)). They can be invoked in parallel without constraints (cf. Figure 4.7(c)) or with constraints (cf. Figure 4.7(d)).

AGOC type 3 many-to-many change describes the granularity change between two sets of asynchronous operations. Let $O_X = \{o_{x1}, \dots, o_{xs}\}$ be a set of asynchronous operations, O_X is modified into a set of operations $O_Y = \{o_{y1}, \dots, o_{yt}\}$, where $\forall o_{xi} \in O_X$ ($i = 1, \dots, s$), $\exists o_{yj} \in O_Y$ ($j = 1, \dots, t$), such that $InInfo(o_{xi}) \cap InInfo(o_{yj}) \neq \emptyset$ (cf. Figure 4.8).

The following relations between O_X and O_Y exist:

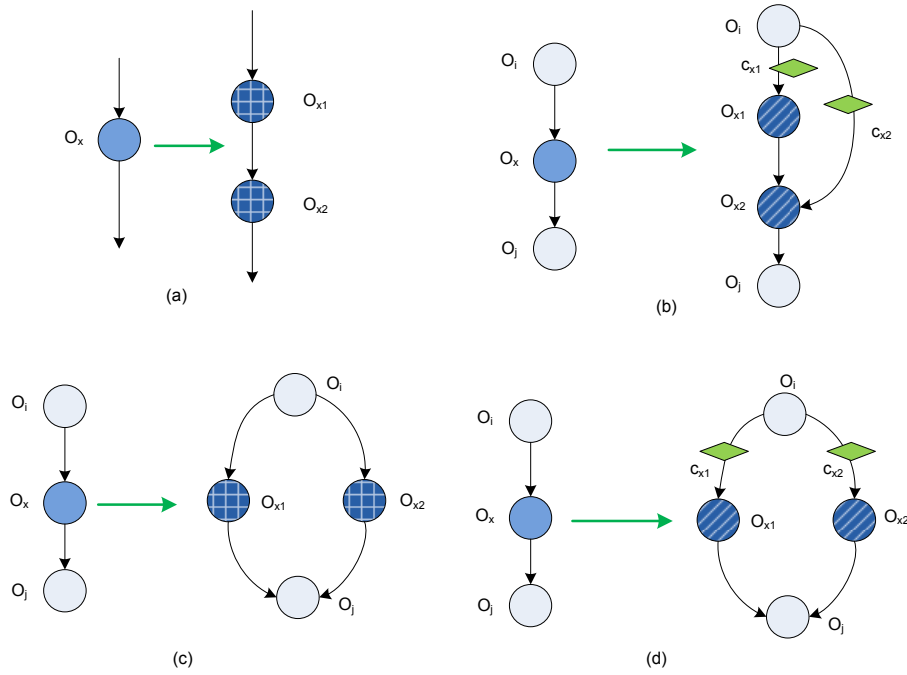


Figure 4.7: Examples for AOGC type 2 one-to-many change.

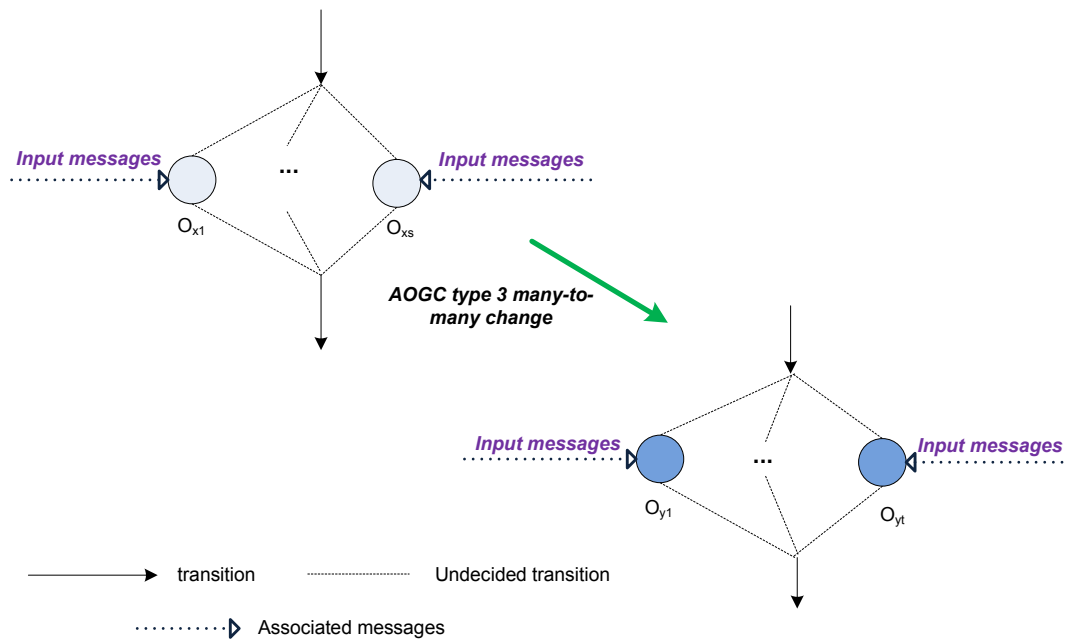


Figure 4.8: AOGC type 3 many-to-many change.

-

$$InInfo(O_X) = InInfo(O_Y)$$

which means that operations in O_X are redesigned into a set of operations O_Y . Although O_X and O_Y remain functionally equivalent, each operation o_{xi} in O_X ($i = 1, \dots, s$) is different with any operation o_{yj} in O_Y ($j = 1, \dots, t$).

-

$$InInfo(O_X) \subset InInfo(O_Y)$$

which means that O_Y is capable of processing more data types than O_X .

-

$$InInfo(O_X) \supset InInfo(O_Y)$$

which means that O_Y accepts less data types as the input parameters than O_X .

- the above relations do not hold and

$$InInfo(O_X) \cap InInfo(O_Y) \neq \emptyset$$

This relation between O_X and O_Y describes that O_Y retains only part of the functionality of O_X and has functionality that is not provided by O_X .

4.1.2.2 Synchronous Operation Granularity Change

The synchronous operation granularity change (SOGC) refers to the granularity change of synchronous operations. SOGC is classified into three types: *SOGC type 1 one-to-one change*, *SOGC type 2 one-to-many/ many-to-one change*, and *SOGC type 3 many-to-many change*.

SOGC type 1 one-to-one change describes the granularity change between two synchronous operations. Let $o_x \in O$ be a synchronous operation, o_x can be changed to another synchronous operation o'_x by modifying its input and output parameters (cf. Figure 4.9). For instance,

$$InInfo(o_x) = InInfo(o'_x)$$

and

$$OutInfo(o_x) \subset OutInfo(o'_x)$$

which indicates that o'_x accepts the same input as o_x but generates output with more data types in its information structure.

SOGC type 2 one-to-many/ many-to-one change describes the granularity change between a synchronous operation o_x and a set of synchronous operations $O_Y = \{o_{y1}, \dots, o_{yt}\}$, where $\forall o_{yj} \in O_Y (j = 1, \dots, t)$ such that $(InInfo(o_x) \cup$

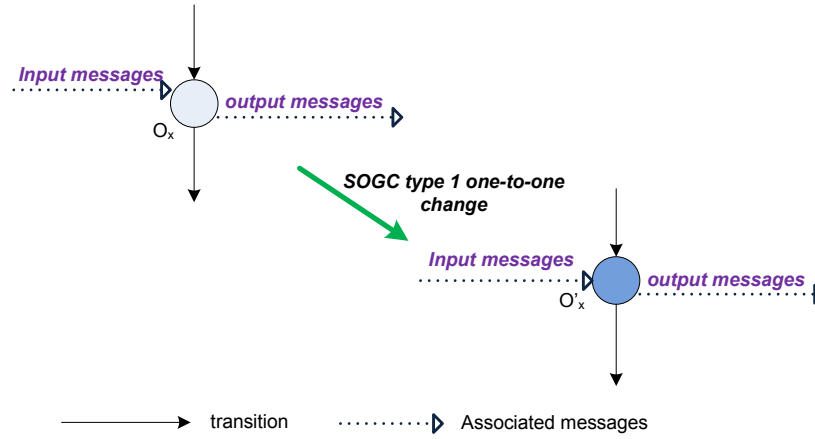


Figure 4.9: SOGC type 1 one-to-one change.

$OutInfo(o_x) \cap (InInfo(o_{yj}) \cup OutInfo(o_{yj})) \neq \emptyset$ (cf. Figure 4.10). For instance,

$$InInfo(o_x) = InInfo(O_Y)$$

and

$$OutInfo(o_x) \subset OutInfo(O_Y)$$

This relation indicates that o_x and O_Y accept the same input parameters whereas O_Y generates output with more data types in its information structure than o_x .

SOGC type 3 many-to-many change describes the granularity change between two sets of synchronous operations (cf. Figure 4.11). Let $O_X = \{o_{x1}, \dots, o_{xs}\}$ be a set of synchronous operations, O_X is redesigned into a set of synchronous operations $O_Y = \{o_{y1}, \dots, o_{yt}\}$, where $\forall o_{xi} \in O_X$ ($i = 1, \dots, s$), $\exists o_{yj} \in O_Y$

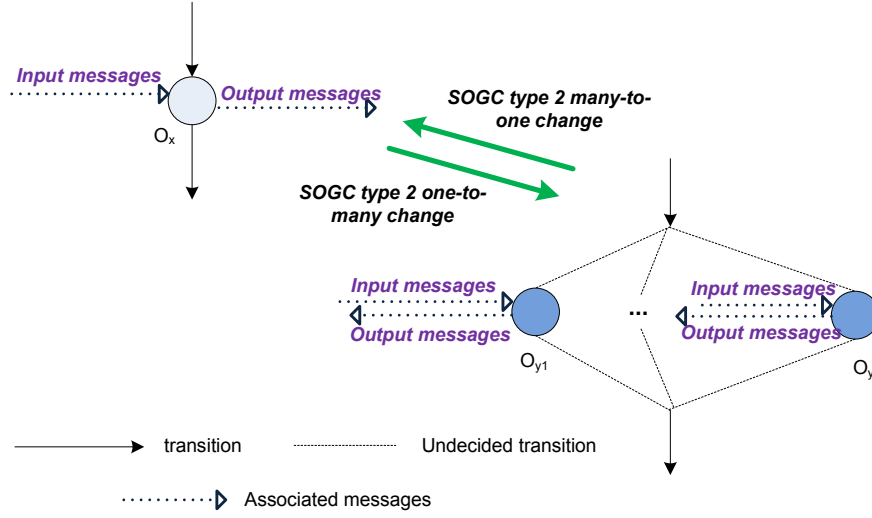


Figure 4.10: SOGC type 2 one-to-many / many-to-one change.

($j = 1, \dots, t$), such that

$$(InInfo(o_{xi}) \cup OutInfo(o_{xi})) \cap (InInfo(o_{yj}) \cup OutInfo(o_{yj})) \neq \emptyset$$

4.1.2.3 Complex Operation Granularity Change

The complex operation granularity change (COGC) refers to the granularity change that involves both synchronous and asynchronous operations. COGC is classified into: *COGC type 1 asynchronous-to-synchronous / synchronous-to-asynchronous change* and *COGC type 2 mixed change*.

COGC type 1 asynchronous-to-synchronous / synchronous-to-asynchronous change describes the granularity change between asynchronous operations and synchronous operations and vice versa (cf. Figure 4.12). We define the granularity change from asynchronous to synchronous operations. The *synchronous-to-asynchronous*

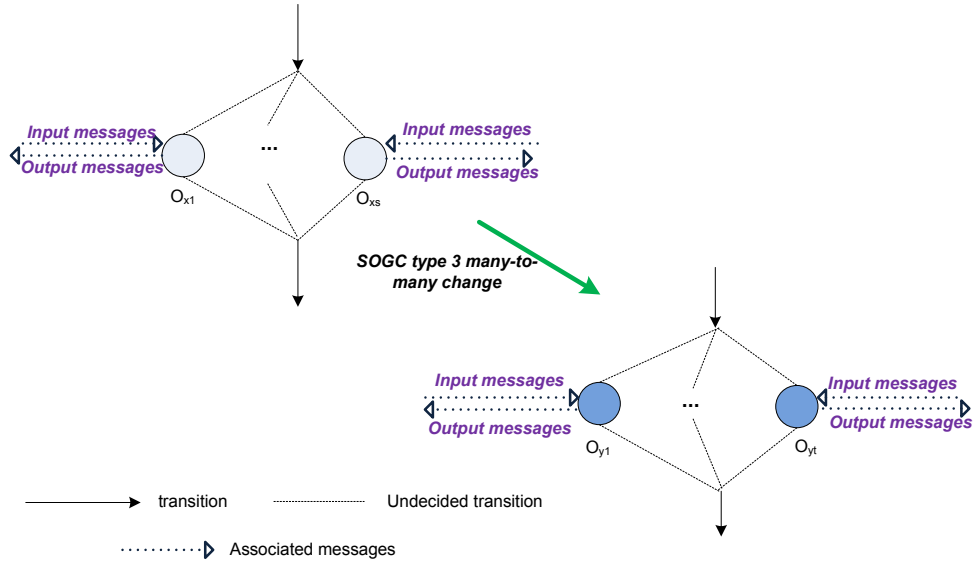


Figure 4.11: SOGC type 3 many-to-many change.

change is similarly defined. Let $O_X = \{o_{x1}, \dots, o_{xs}\}$ be a set of asynchronous operations and $O_Y = \{o_{y1}, \dots, o_{yt}\}$ be a set of synchronous operations. There is a *COGC type 1 asynchronous-to-synchronous change* iff $\forall o_{xi} \in O_x$ ($i = 1, \dots, s$), $\exists o_{yj} \in O_Y$ ($j = 1, \dots, t$), such that $InInfo(o_{xi}) \cap (InInfo(o_{yj}) \cup OutInfo(o_{yj})) \neq \emptyset$ (cf. Figure 4.12). For instance,

$$InInfo(O_X) \subset (InInfo(O_Y) \cup OutInfo(O_Y))$$

which means that O_Y covers all the functionality of O_X and provides extra functionality than O_X .

Figure 4.13 gives an example of this type of operation granularity change. A synchronous operation o_x is modified to three asynchronous operations o_{y1} , o_{y2} and o_{y3} . The operation o_{y1} can be invoked in parallel to the operations o_{y2} and

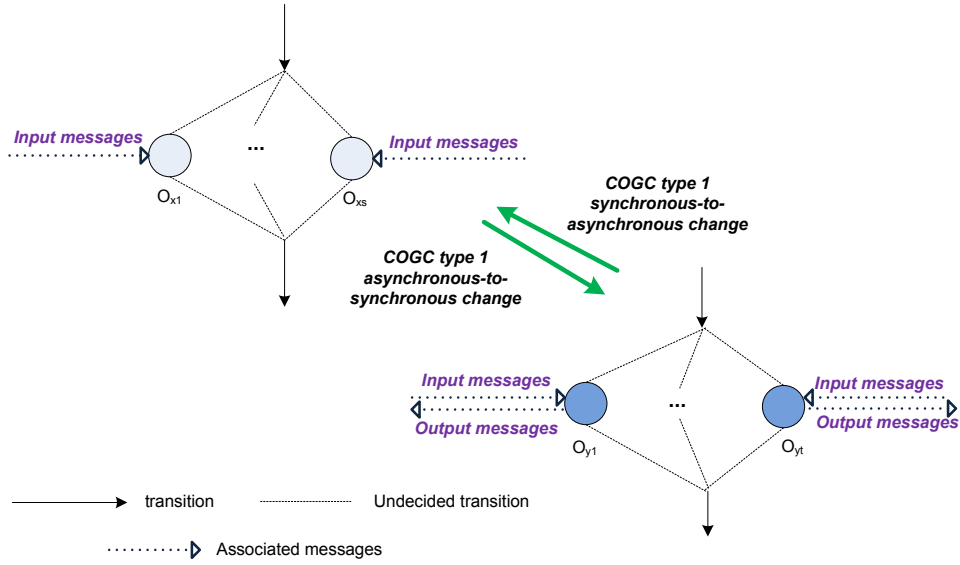


Figure 4.12: COGC type 1 asynchronous-to-synchronous change.

o_{y3} . In addition, o_{y2} is followed by o_{y3} which is embedded in a looping transition.

COGC type 2 mixed change describes the change between two sets of operations: O_X and O_Y . Each set of operations consists of both synchronous and asynchronous operations. Let $O_X = \{o_{x1}, \dots, o_{xs}\}$ and $O_Y = \{o_{y1}, \dots, o_{yt}\}$ be two sets of operations. O_X is categorized into two sets: O_X^a and O_X^s , where O_X^a consists of the set of asynchronous operations and O_X^s contains the synchronous operations. Similarly, O_Y can be classified into O_Y^a and O_Y^s . There is a COGC type 2 mixed change iff all the following conditions are satisfied:

- $\forall o_{xi} \in O_X^a, \exists o_{yj} \in O_Y$, such that

$$InInfo(o_{xi}) \cap InInfo(o_{yj}) \neq \emptyset (o_{yj} \in O_Y^a)$$

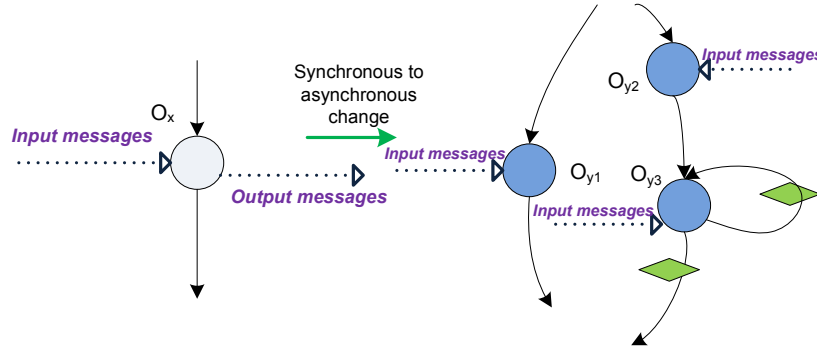


Figure 4.13: An example for COGC type 1 synchronous-to-asynchronous change.

or

$$InInfo(o_{xi}) \cap (InInfo(o_{yj}) \cup OutInfo(o_{yj})) \neq \emptyset (o_{yj} \in o_Y^s)$$

- $\forall o_{xi} \in O_X^s, \exists o_{yj} \in O_Y$, such that

$$(InInfo(o_{xi}) \cup OutInfo(o_{xi})) \cap InInfo(o_{yj}) \neq \emptyset (o_{yj} \in O_Y^a)$$

or

$$(InInfo(o_{xi}) \cup OutInfo(o_{xi})) \cap (InInfo(o_{yj}) \cup OutInfo(o_{yj})) \neq \emptyset (o_{yj} \in O_Y^s)$$

4.1.3 Transition Changes

A transition change refers to the modification of transitions between operations. Rather than discuss primitive changes such as adding or removing a transition, we identify seven types of high level transition changes (cf. Figure 4.1). These

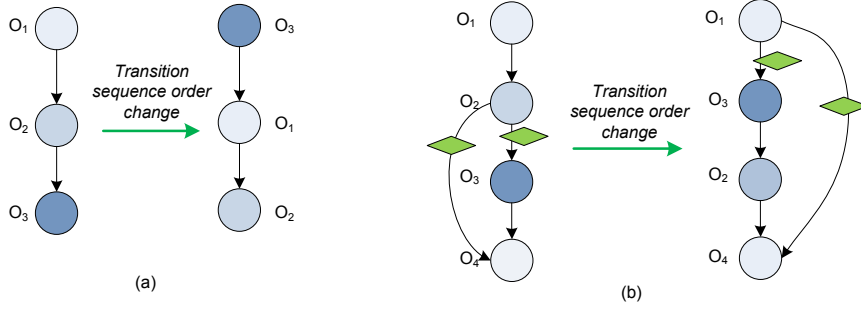


Figure 4.14: Transition sequence order change (TSOC).

high level transition changes can be accomplished by applying primitive changes. We believe the high level transition changes are more meaningful in describing transition changes of a service in the real world. These high level transition changes are discussed in detail below.

A **Transition Sequence Order Change (TSOC)** describes the change that operations are invoked in a different order. Figure 4.14 (a) and (b) are examples for TSOC.

B **Sequential to Parallel Transition Sequence Change (SPTSC)** describes the change that a transition sequence is split into multiple parallel transition sequences. For example, in Figure 4.15 (a), the transition sequence $o_1t_1o_2t_2o_3t_3o_4$ is changed to two transition sequences $o_1t'_1o_2t'_3o_4$ and $o_1t'_2o_3$. Figure 4.15 (b) is another example for SPTSC involving conditional transition sequences. Before the change there are two alternatively executed transition sequences: $o_1t_1o_2t_2(c_{x1})o_3t_3o_5$ (if c_{x1} is evaluate to be true) and $o_1t_1o_2t_4(c_{x2})o_4t_5o_5$ (if c_{x2} is evaluate to be true). After the SPTSC, the two alternative transition sequences are changed into three

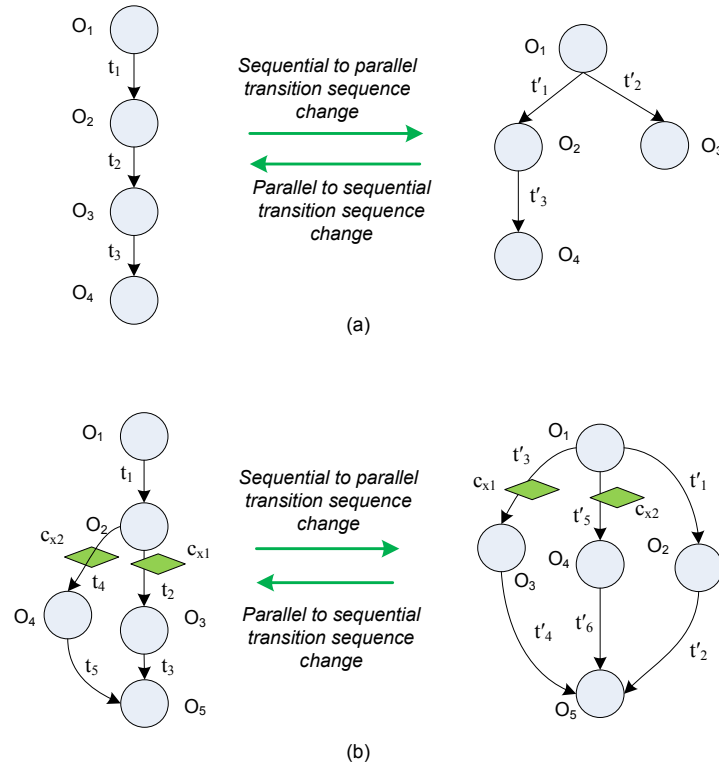


Figure 4.15: Sequential (parallel) to parallel (sequential) transition sequence change (SPTSC, PSTSC).

transition sequences: $o_1 t'_1 o_2 t'_2 o_5$, $o_1 t'_3(c_{x1}) o_3 t'_4 o_5$ (if c_{x1} is evaluate to be true), and $o_1 t'_5(c_{x2}) o_4 t'_6 o_5$ (if c_{x2} is evaluate to be true).

C Parallel to Sequential Transition Sequence Change (PSTSC) describes the change that transition sequences that can be executed in parallel are merged into a single transition sequence (cf. Figure 4.15(a) and (b)).

D Adding Conditional Transition Sequence Change (ACTSC) refers to the change that unconditional invocation of operations are controlled by constraints. In Figure 4.16 the invocation of o_2 is changed to be conditional by introducing a constraint c_{x1} to the transition t_1 . In addition, a transition t_3 with

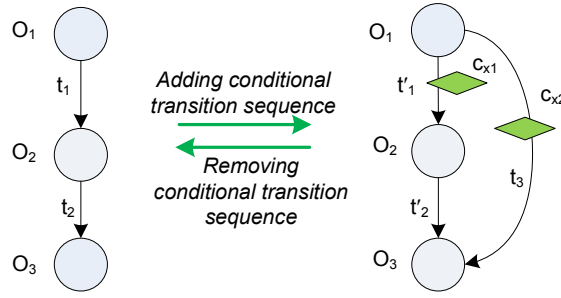


Figure 4.16: Adding (removing) conditional transition sequence change (ACTSC (RCTSC)).

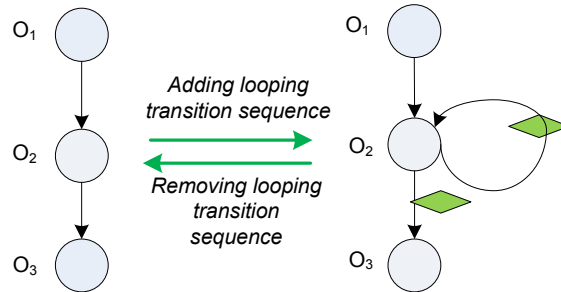


Figure 4.17: Adding (removing) looping transition sequence change (ALTSC (RLTSC)).

a constraint c_{x2} is added from o_1 to o_3 . ACTSC describes a typical change that unconditionally invoked operations are changed to be conditionally executed.

E Removing Conditional Transition Sequence Change (RCTSC)

refers to the change that conditionally invoked operations are changed to be invoked without constraints. Figure 4.16 is also an example of RCTSC.

F Adding Looping Transition Sequence (ALTSC) refers to that operations are changed to be executed multiple times. For instance, in Figure 4.17 the operation o_2 is embedded in a looping transition.

G Removing Looping Transition Sequence Change (RLTSC) de-

scribes the change that operations are moved out from looping transitions. Figure 4.17 is an example, where o_2 can only be executed once after the change.

4.2 Process Changes

In this section, we present the identified process changes. Figure 4.18 shows the taxonomy for process changes in our work. These types of process changes are classified with the goal to facilitate the change impact analysis for service-based business processes. That is, these identified types of process changes provide foundations for analysing the change impact on internal processes and services.

Change classification of business processes has been studied in the workflow systems [100, 104]. In those works, the high level classification of process changes are proposed. The basic element in the change classification is *process fragment*. A process fragment is a sub process that has a single node in and a single node out. However, the impact of process changes on individual services can not be analysed when using process fragments as the basic elements in the change classification. To facilitate the change impact analysis in service based business processes, our work uses *activity* as the basic element in classifying changes of internal processes.

We provide an example to illustrate the reason for using activities as the basic elements in the change classification than using process fragments. As Figure 4.19 shows that *process fragment* 1 is inserted between two successively

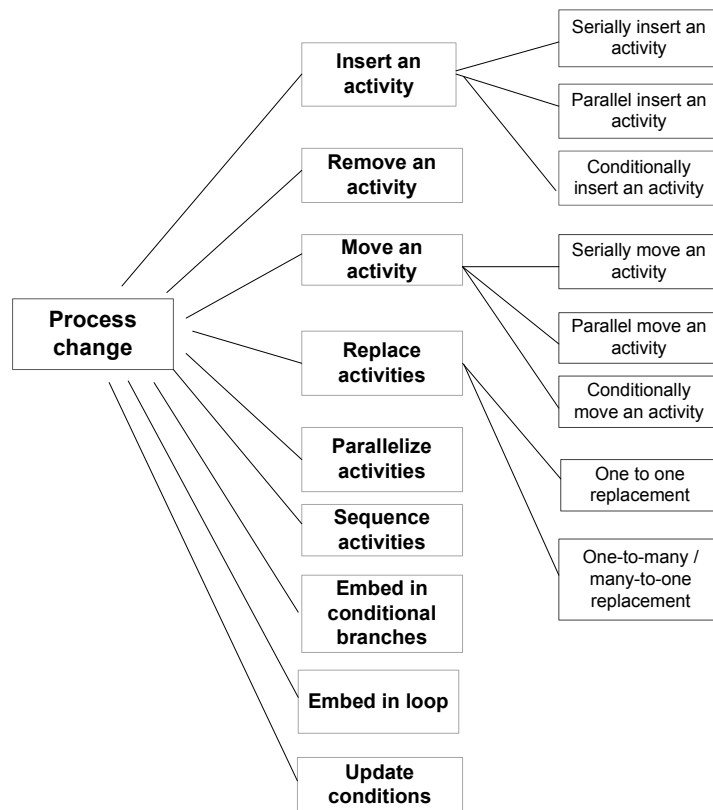


Figure 4.18: Taxonomy of process changes.

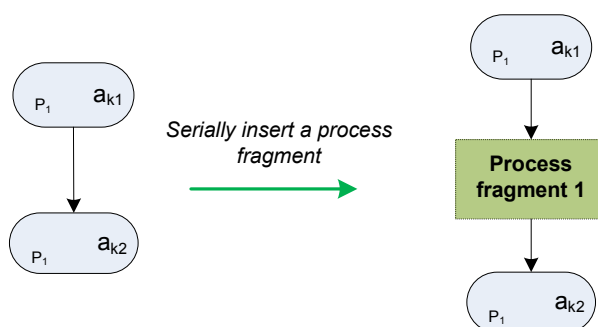


Figure 4.19: Serially insert a process fragment.

executed c-activities a_{k1} and a_{k2} . To understand the impact on the associated services made by this process change, we must know the details inside *process fragment 1*. Figure 4.20 shows four cases of the *process fragment 1*. In Figure 4.20(a), *process fragment 1* contains only a c-activity a_x relating to the partner p_1 . In this case, the process change requires that an operation associated with a_x must be inserted between operations associated with activities a_{k1} and a_{k2} . In Figure 4.20(b), *process fragment 1* contains an xor structure with two conditional branches. One branch contains two p-activities a_i and a_j and a c-activity a_x relating to the partner p_1 . In this case, the process change requires that an operation associated with a_x must be inserted conditionally between operations associated with activities a_{k1} and a_{k2} in the service s_{p_1} . In Figure 4.20(c), *process fragment 1* contains two parallel branches with four activities. In one parallel branch, a p-activity a_i , a c-activity a_x relating to partner p_1 , and a p-activity a_j are executed successively. Another parallel branch contains a c-activity a_{k3} relating to partner p_1 . In this case, the process change requires that operations associated with a_x and a_{k3} must be added in parallel to each other in the service s_{p_1} . In Figure 4.20(d), *process fragment 1* contains two parallel branches. One parallel branch has two conditional branches. In one conditional branch, a p-activity a_i , a c-activity a_x relating to partner p_1 , and a p-activity a_j are executed successively. Another parallel branch contains only a c-activity a_{k3} relating to partner p_1 . In this case, the process change requires that the operation associated with a_x must be added with constraints between the operations associated with

a_{k1} and a_{k2} in the service s_{p1} . In addition, the operation associated with a_{k3} must be added in parallel to the operation associated with a_x . In the above four cases of *process fragment 1*, the process change *insert a process fragment* has different impact on the associated services. Therefore, change classification needs to be discussed based on *activities* rather than process fragment.

In the following, we discuss the identified process changes as shown in Figure 4.18 in detail.

Insert an Activity

The *Insert an activity* change is further classified into three sub types as follows.

A Serially insert an activity The *serially insert an activity* change describes that an activity is added to between two directly succeeding activities. Figure 4.20 are examples of this type of process change.

B Parallel insert an activity The *parallel insert an activity* change describes that an activity is added in parallel to another activity. Figure 4.21 shows an example, where a_x is inserted in parallel to a_i .

C Conditionally insert an activity The *conditionally insert an activity* refers to that an activity is added to the internal process with conditions. Figure 4.22 shows an example, where a_x is inserted in between a_i and a_j with conditions.

Remove an Activity

The *remove an activity* change describes that an existing activity is deleted from the internal process.

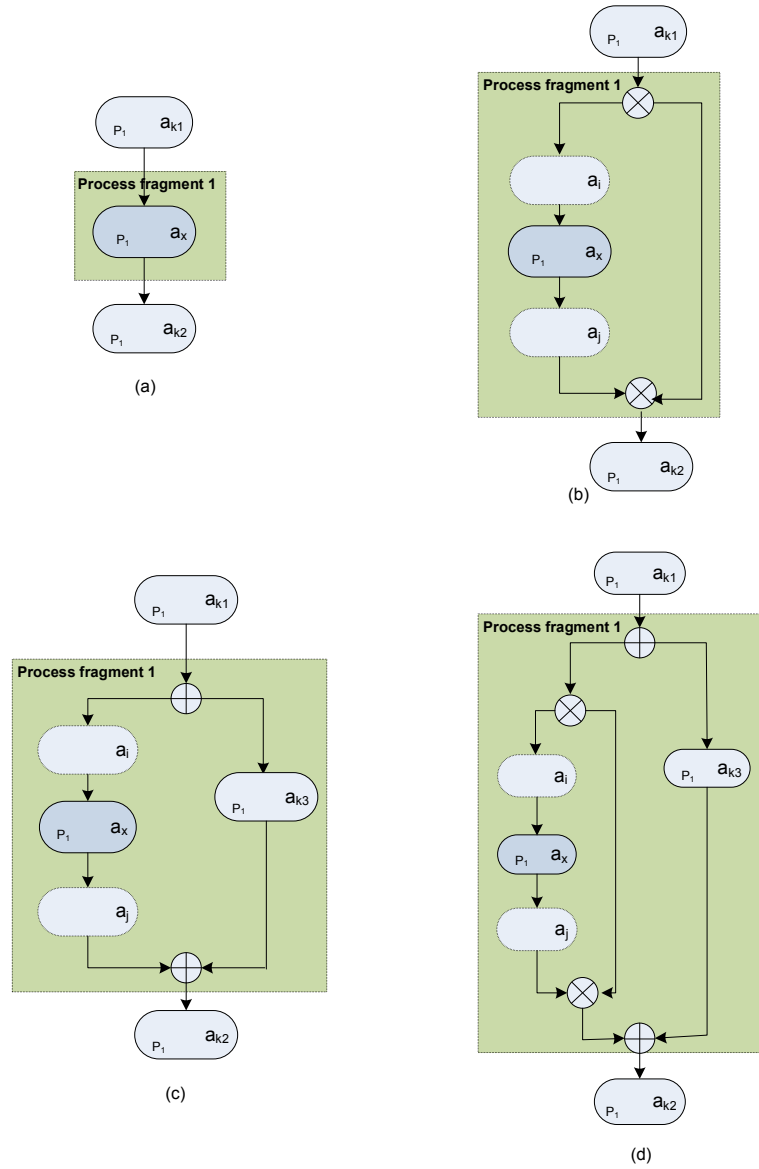


Figure 4.20: Examples of serially inserting an activity.

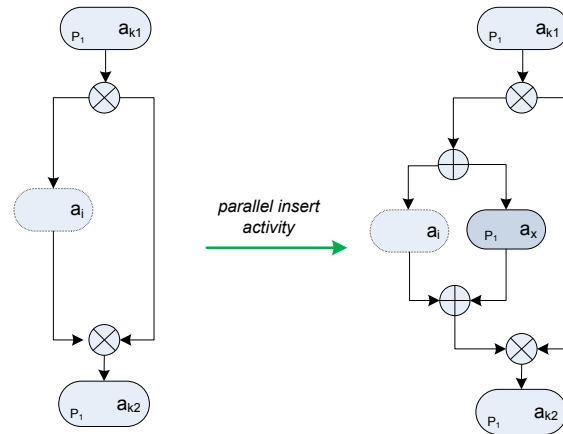


Figure 4.21: Parallel insert an activity.

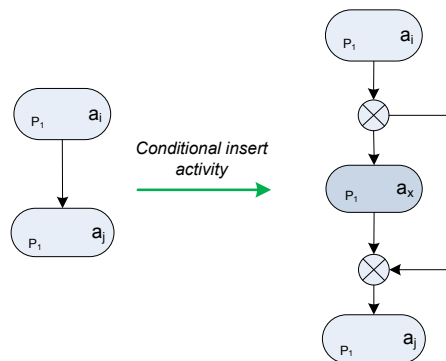


Figure 4.22: Conditionally insert an activity.

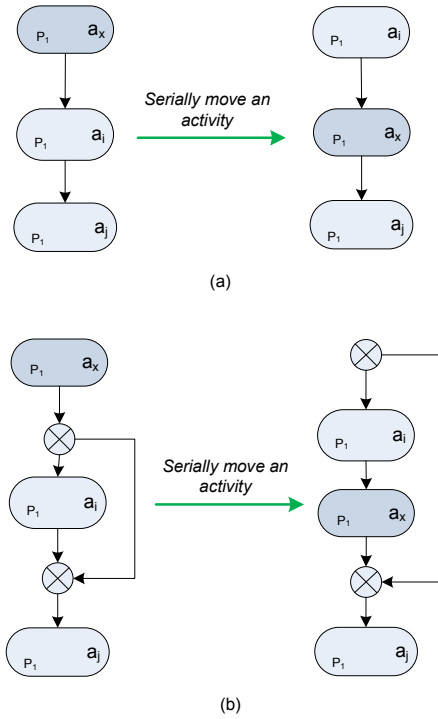


Figure 4.23: Serially move an activity.

Move an Activity

The *move an activity* change describes that an activity is relocated to another position in the internal process. This type of change is further classified into three sub types of change: *serially move an activity*, *parallel move an activity* and *conditionally move an activity*, which are discussed in detail below.

A Serially Move an Activity The *serially move an activity* change describes that an activity is re-inserted between two directly succeeding activities. Figure 4.23 are two examples for this type of change, where a_x is moved to the position between a_i and a_j .

B Parallel Move an Activity The *parallel move an activity* change de-

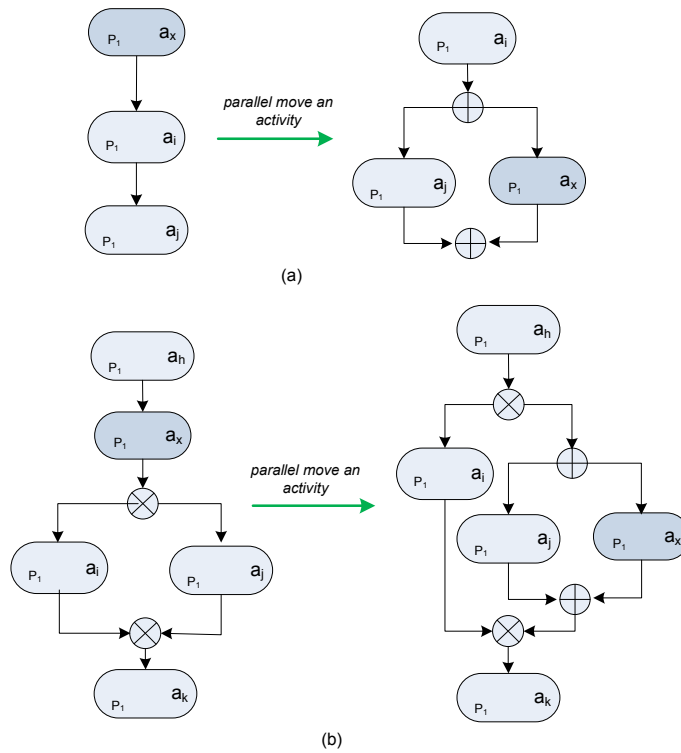


Figure 4.24: Parallel move an activity.

scribes that an activity is moved from its current position to a position that is in parallel to an existing activity. Figure 4.24 shows two examples.

C Conditionally Move an Activity The *conditionally move an activity* describes the change that an activity is re-inserted between two activities with conditions. For example, a_x is moved into an xor construct as shown in Figure 4.25. Before the change, a_x is executed without conditions. After the change, a_x is conditionally executed.

Replace Activities

The *replace activities* change describes that an activity is replaced by a set of new activities. This type of change is further classified into two sub types as

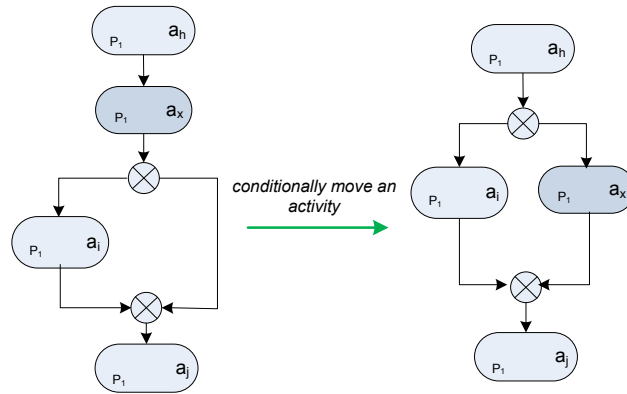


Figure 4.25: Conditionally move an activity.

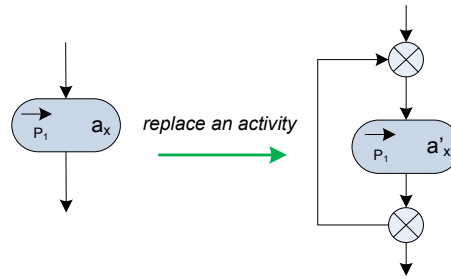


Figure 4.26: One-to-one replacement.

follows.

A One-to-one replacement The *one-to-one replacement* change describes that one activity is replaced by another activity. Figure 4.26 shows an example, where a receive activity a_x is replaced by another receive activity a'_x . In addition, a'_x is embedded in a loop construct. This change means that instead of receiving information from the partner p_1 only once, the internal process repeatedly receives information from the partner p_1 until certain condition is satisfied.

B One-to-many/ many-to-one activities replacement The *one-to-many/*

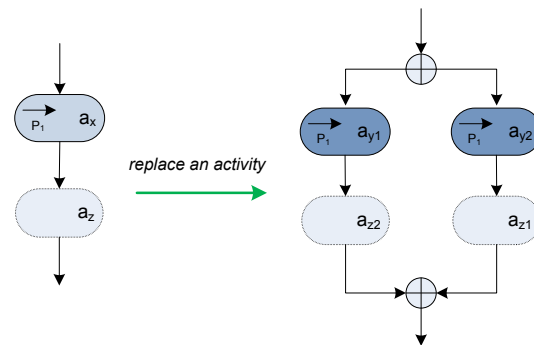


Figure 4.27: One-to-many activities replacement.

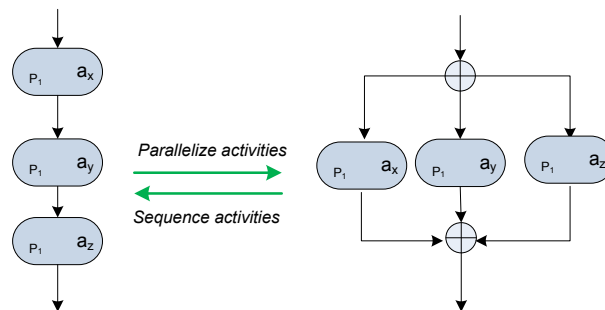


Figure 4.28: Parallelize (sequence) activities.

many-to-one activities replacement change describes that an activity is replaced by a set of activities or vice versa. Figure 4.27 gives an example, where a receive activity a_x is replaced by two parallel executed receive activities a_{y1} and a_{y2} . This change indicates that instead of receiving the information from the partner p_1 once, the internal process can receive the required information in parallel.

Parallelize Activities

The *parallelize activities* describes the change that a set of sequentially executed activities are changed to be executed in parallel (cf. Figure 4.28).

Sequence Activities

The *sequence activities* change describes that a set of parallel executed activ-

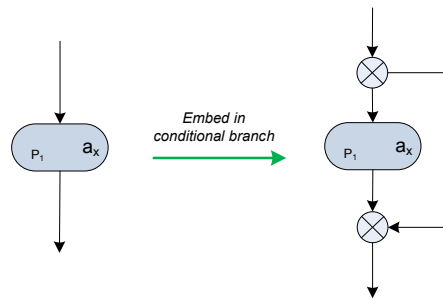


Figure 4.29: Embed an activity in conditional branch.

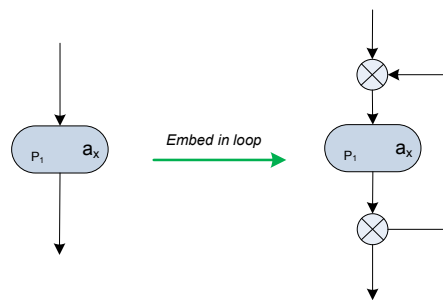


Figure 4.30: Embed an activity in conditional branch.

ities are modified to be performed in sequence. Figure 4.28 shows an example.

Embed in Conditional Branches

The *embed in conditional branches* change describes that an activity is wrapped in a conditional construct (cf. Figure 4.29).

Embed in Loop

The *embed in loop* change describes that an activity is wrapped in a loop (cf. Figure 4.30).

Update Conditions

The *update conditions* change describes that the conditions of an xor connector are modified.

4.3 Discussion

In this chapter, we have presented the taxonomy for the changes associated with services and business processes based on the proposed service-oriented business process model. Service changes are categorized into operation changes and transition changes. The operation changes are further classified into operation existence changes and operation granularity changes. The operation existence changes capture the types of changes of adding or removing operations in a service, which are further categorized into *sequentially adding an operation without constraints*, *sequentially adding an operation with constraints*, *adding an operation in parallel to existing operations without constraints*, *adding an operation in parallel to existing operations with constraints*, and *deleting an operation*. The operation granularity changes comprise the types of variations related to operation granularity. We have identified three sub types of operation granularity change as: *asynchronous operation granularity change* (AOGC), *synchronous operation granularity change* (SOGC), and *complex operation granularity change* (COGC). Seven types of high level transition changes that describe the changes related to service transitions are identified. Process changes are discussed on the basis of the control flow schemas of internal processes. Nine major types of process changes are identified as: *insert an activity*, *remove an activity*, *move an activity*, *replace activities*, *parallelize activities*, *sequence activities*, *embed in conditional branches*, *embed in loop*, and *update conditions*. These various types of service changes and process

changes provide a solid foundation for analysing the impact caused by service changes and process changes on service-based business processes and developing generic change management solutions for controlling these changes.

As discussed in Chapter 2, process change patterns have been presented in the literature for supporting the change management of workflow processes and enabling the flexibility of workflow processes [100, 104]. Those change classifications are presented with the purpose to manage changes of business processes without considering the characteristics of services. In [25], a classification of process differences between business processes is presented, which are classified from the aspects of authorization, activities, and control flows. The purpose of the process difference classification is to support process development. We have a focus on the change analysis in service-based business processes where services and business processes have complicated coupling relationships. Our classification of process changes are identified with the goal to facilitate the change impact analysis and change management solutions for associated services and business processes.

In the SOC paradigm, change management has been studied for Web service compositions [8, 7, 57, 58, 59, 80, 108]. In [80, 108], two types of service changes related to BPEL processes are analysed as: the subtractive changes and the additive changes. In [57, 58, 59], the top-down changes and bottom-up changes in virtual enterprises are discussed. In the field of service adaptation, which is closely related to the area of change management, mismatch types at the

levels of service interfaces and service protocols are identified for the purpose of developing adapters [12, 46]. The classification of service changes presented in the above mentioned researches does not consider the coupling relations between services and business processes. Moreover, the diversity of service changes is not adequately addressed in the above works. Our change classification related to services and business processes is based on the proposed service-oriented business process model and with the goal to facilitate the change management for services and business processes where complicated dependencies exist.

Chapter 5

Change Impact Analysis

Change management is challenging in service-based environments. As we exemplified in the sales scenario in Chapter 3, services and business processes are related to each other when multiple services are supported by a single business process. Due to various reasons such as business regulations and application environments, services and business processes may change from time to time. A specific change usually makes various level of impact on business processes and services because of the dependencies between business processes and services. When a change occurs in a service, this change may affect the business process and may have impact on the other services associated with this business process. When a change occurs in a business process, this change may affect the services that are associated with this business process. Analysis of the change impact on service-based business processes is crucial for the development of effective and efficient change management solutions in the context of SOC.

Current researches on service change management are mainly concentrated on managing changes for BPEL processes [80, 108] and Web services [9, 10, 57, 59] respectively. The complicated dependencies between services and business pro-

cesses have not been fully studied in the existing works on change management. Based on the service-oriented business process model proposed in Chapter 3 and the types changes identified in Chapter 4, we present our approach for change impact analysis in this chapter with the goal to control these various types of changes and their cascading effect on services and business processes.

Our change analysis approach is based on the identified change impact patterns [102]. Each change impact pattern captures a specific type of change effect that is described by the information as the direct impact scope of the change, the cause of the change, and the effect on the services and the associated business process. The direct impact scopes of a specific service change and a process change are calculated by the defined functions. The change impact patterns provide rich intermediate results in the analysis process which help reduce the complex tasks of managing the various types of changes and controlling the cascading effect in service-based business processes. Moreover, these change impact patterns can be reused in the development and maintenance of service-based applications and information systems.

This chapter is structured as follows. First, we provide an overview of the identified change impact patterns and discuss the structure of a change impact pattern in Section 5.1. Then we discuss the issue of calculating the direct impact scope for a specific service change and a process change in Section 5.2. Two functions and the corresponding algorithms are defined for calculating direct impact scopes of service changes and process changes. In Section 5.3, we present

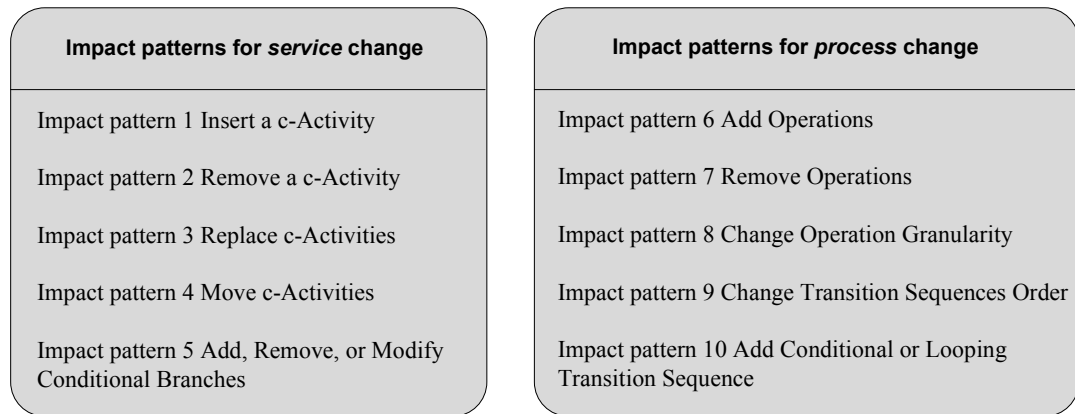


Figure 5.1: Overview of change impact patterns.

the ten identified change impact patterns. Examples are provided to illustrate the different types of change impact described in the change impact patterns. Finally, we conclude this chapter in Section 5.4.

5.1 Overview of Change Impact Patterns

Figure 5.1 shows the overview of our identified change impact patterns and Figure 5.2 is the list of these patterns. The impact patterns are classified into two categories: *impact patterns for service change* and *impact patterns for process change*. The *impact patterns for service change* capture the impact on internal processes made by service changes. The *impact patterns for process change* describe the impact on services made by process changes.

The structure of an impact pattern is shown in Figure 5.3. Each of these impact patterns includes the following elements:

Change Impact Patterns		
<i>Name</i>	<i>Pattern description</i>	<i>Cause</i>
Impact pattern 1 Insert a c-Activity	a c-activity needs to be added to the internal process	adding an operation to a service
Impact pattern 2 Remove a c-Activity	c-activities need to be removed from the internal process	deleting operations in a service
Impact pattern 3 Replace c-Activities	c-activities need to be replaced by another c-activity or a set of structured c-activities	changing operation granularity in a service
Impact pattern 4 Move c-Activities	c-activities need to be reordered	operation transition sequence change, including TSOC, SPTSC and PSTSC
Impact pattern 5 Add, Remove or Modify Conditional Branches	xor structures need to be modified or new xor structures need to be created	transition sequence change, such as ACTSC, RCTSC, ALTSC, and RLTS
Impact pattern 6 Add Operations	operations need to be added to corresponding services	inserting a c-activity or replacing a c-activity in the internal process
Impact pattern 7 Remove Operations	operations need to be deleted from services	deletion of c-activities or the replacement of c-activities in the internal process
Impact pattern 8 Change Operation Granularity	operation granularity needs to be modified	replacement of c-activities in the internal process
Impact pattern 9 Change Transition Sequence	transition sequences of the corresponding services need to be reordered	moving activities, parallelizing activities or sequencing activities in the internal process
Impact pattern 10 Add Conditional or Looping Transition Sequence	constraints and extra transition sequences need to be added between operations	embedding activities in conditional branches

Figure 5.2: Change impact patterns.

-
- Name of the impact pattern.
 - Pattern description. A pattern description gives a summary of the impact captured by this change impact pattern. The detailed impact will be specified in the *Effect on internal process/services*.
 - Cause of the impact. This element gives a brief description of the changes that may make this type of impact. For instance, the cause of impact captured by *impact pattern 1 Insert a c-Activity* is adding an operation to a service.
 - Direct impact scope. A direct impact scope refers to the affected region in the service-oriented business process relating to a particular service change or process change before any reactions are taken to handle this change. The direct impact scope is crucial for understanding the impact of a specific change. The *direct* in the term direct impact scope is used to differentiate the *actual* impact scope that will be discussed in the next chapter. We will discuss the functions and algorithms for calculating the direct impact scopes for a service change and a process change in the next section.
 - Effect on an internal process (impact patterns for service change)/services (impact patterns for process change). The effect on an internal process is described by *abstract control relations* associated with c-activities which are defined in Chapter 3. For instance, in Figure 5.3, the dashed arrows linking c-activities a_i , a_j , and a_x are abstract control relations. These abstract

control relations reflect the change effect on the control flow schema of the internal process caused by a specific type of service change. The effect on services are described by a specific type of service changes based on the change taxonomy presented in Chapter 4. For instance, in Figure 5.3, the process change, embed in a conditional branch, has the impact on the corresponding service: *Adding conditional transition sequence change (ACTSC)*.

5.2 Direct Impact Scope

Before presenting the identified change impact patterns, we discuss the direct impact scopes of a service change and a process change first. The direct impact scope of a change is crucial for understanding and specifying the impact of the change. As stated in the previous section, the direct impact scope of a specific change is the affected region in a service-oriented business process before any reactions are taken to handle this change. In the following sub sections, we will define two functions for calculating the direct impact scopes for a service change and a process change respectively.

5.2.1 Direct Impact Scope of a Service Change

The direct impact scope of a service change includes the elements of the internal process, called process elements, that are affected by this service change. A

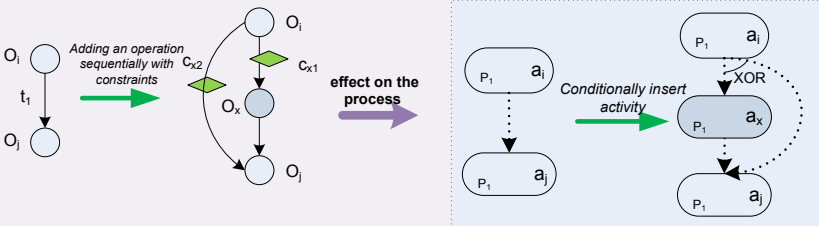
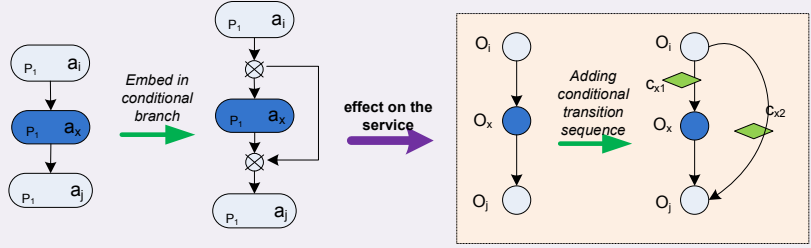
<i>Name</i>	Name of the change impact pattern
<i>Pattern description</i>	A description of the change impact captured by this pattern
<i>Cause</i>	A description of the types of change which may cause the impact captured by this pattern
<i>Direct impact scope</i>	The change region of a service change or a process change before any treatment is taken for handling the change.
<i>Effect on internal process or services</i>	<p>- For the impact caused by a service change, the effect on the internal process is described by the abstract control relations between c-activities.</p>  <p>(Example illustrated for impact pattern 1)</p> <p>- For the impact caused by a process change, the effect on the services is described by a specific type of service changes.</p>  <p>(Example illustrated for impact pattern 10)</p>

Figure 5.3: Structure of the change impact pattern

process element refers to an activity, a control connector, or a data connection. In order to derive the direct impact scope of a service change, we define the function *FuncDISS* which accepts a service-oriented business process and a service change as its input and generates the affected process elements by this service change as its output.

Definition 1 *FuncDISS* Let A be a set of activities, $IFS = \{dc_1, \dots, dc_m\}$ be the information flow schema, and $S = \{s_1, \dots, s_n\}$ be a set of services. *FuncDISS* is the function: $schange \rightarrow PE$. The input of the function includes a service change *schange* with a set of involved operations $O_c = \{o_1, \dots, o_r\}$. The output of the function *FuncDISS* is a set of process elements: $PE = \{pe_1, \dots, pe_r\}$, where pe_i ($i = 1, \dots, r$) consists of:

- the c-activity a that is associated with o_i ;
- the set of activities: $A_{depend} = \{a_1, \dots, a_s\} \subseteq A$, where $\forall a_j \in A_{depend}$ ($j = 1, \dots, s$) such that $a \dashrightarrow^D a_j$;
- the set of data connections: $DC \subseteq IFS$, where $\forall dc \in DC$ such that dc is the data connection linking to the activity a or any activity in A_{depend} .

Algorithm 1 is designed based on Definition 1 that calculates the direct impact scope of a service change. This algorithm accepts a service change *schange* as its input and it returns a set of process fragments PE . In this algorithm, if a is associated with an operation in O_c , a is included into pe and all the data connections linking to a are included into DC (lines 6-12). Then, all the

activities that are depended by a are included in A_{depend} (lines 14-16). All the data connections linking to the activities in A_{depend} are added into DC (lines 17-21). The process fragment pe consists of activity a , the related activities A_{depend} and the associated data connections DC (line 24).

We provide the following example to illustrate the direct impact scope of a service change computed by *FuncDISS*.

Example *Direct Impact Scope of a Service Change* We take the sales process as an example. Figure 5.4(a) shows a service change, transition sequence order change (TSOC), in the service s_b . The operation *send invoice* is moved before the operation *send bill*. Here the set of operations that are involved in this service change are

$$O_c = \{send\ bill, receive\ PayInfo, send\ invoice\}$$

Figure 5.4(b) shows the process elements of the sales process that are affected by this service change. In this figure, activities *Send bill*, *Receive PayInfo*, and *Send invoice* are associated with the operations involving in the service change TSOC. The activities that have data dependency with the three activities are *Prepare bill* and *Prepare invoice*. The data connections linking to these activities are shown as thick dashed lines. With definition 1, the direct impact scope of the service change is:

$$PE = \{pe_1, pe_2, pe_3\}$$

where

$$pe_1 = \{Send\ bill, Prepare\ bill, dc_1, dc_2, dc_3, dc_4\}$$

Algorithm 1 *FuncDISS*

```

1: Input schange
2: Output PE
3: Let  $O_c = \{o_1, \dots, o_s\}$  be the set of operations involved in schange
4:  $PE \leftarrow \emptyset$ 
5: for all  $a \in A$  do
6:   if  $a$  is the c-activity associated with  $o_i$  ( $i = 1, \dots, s$ ) then
7:      $DC \leftarrow \emptyset, pe \leftarrow \{a\}$ 
8:     for all  $dc_j \in IFS$  do
9:       if  $dc_j$  is associated with  $a$  then
10:         $DC \leftarrow DC \cup \{dc_j\}$ 
11:      end if
12:    end for
13:     $A_{depend} \leftarrow \emptyset$ 
14:    for all  $a_k \in A$  do
15:      if  $a \dashrightarrow^D a_k$  then
16:         $A_{depend} \leftarrow A_{depend} \cup \{a_k\}$ 
17:        for all  $dc_j \in IFS$  do
18:          if  $dc_j$  is associated with  $a_k$  then
19:             $DC \leftarrow DC \cup \{dc_j\}$ 
20:          end if
21:        end for
22:      end if
23:    end for
24:     $pe \leftarrow pe \cup A_{depend} \cup DC$ 
25:  end if
26:   $PE \leftarrow PE \cup \{pe\}$ 
27: end for
28: return  $PE$ 

```

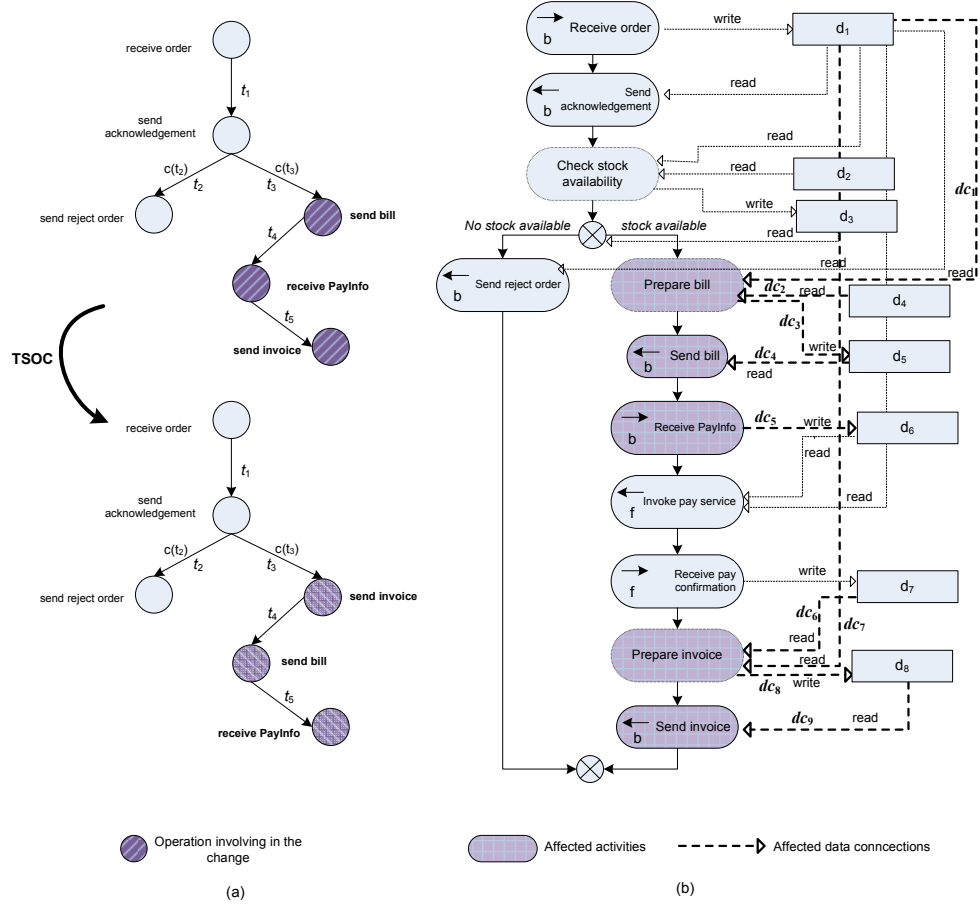


Figure 5.4: (a) Service change: TSOC in service s_b ; (b) direct impact scope of the service change.

$$pe_2 = \{Receive PayInfo, dc_5\}$$

$$pe_3 = \{Send invoice, Prepare invoice, dc_6, dc_7, dc_8\}$$

5.2.2 Direct Impact Scope of a Process Change

The direct impact scope of a process change includes the affected elements of services associated with the internal process, called service elements. A service

element refers to an operation or a transition. In order to derive the direct impact scope of a process change, we define the function *FuncDISP*, which accepts a service-oriented business process and a process change as its input and generates a set of service elements affected by this process change as its output.

Definition 2 *FuncDISP* Let A be a set of activities, and $S = \{s_1, \dots, s_n\}$ be a set of services. *FuncDISP* is the function: $pchange \rightarrow SF$. The input of the function includes: a process change $pchange$ with a set of directly affected operations O_c . The output of the function *FuncDISP* is a set of service fragments

$$SF = \{sf_i | i \in \{1, \dots, n\}\}$$

where a service fragment sf_i consists of all the affected service elements in service s_i :

- the set of operations $O_c^i \subseteq O_c$, where O_c^i consists of all the operations that belong to both O_c and service s_i ;
- the set of transitions T_c^i , where $\forall t \in T_c^i$, t takes an operation in O_c^i as the origin operation or the destination operation;
- the set of operations O_x^i , where O_x^i consists of all the operations that are associated with transitions in T_c^i and are not in O_c^i .

Algorithm 2 calculates the direct impact scope of a process change based on Definition 2. This algorithm accepts a process change $pchange$ as its input and it generates a set of service fragments SF as its output. First, the affected operations relating to service s_i are put in O^i (lines 4-9), where n denotes the number of services. The service fragment sf_i contains all the affected elements in service s_i . Lines 10-20 calculate each sf_i based on Definition 2. All the non-

Algorithm 2 *DISP*

```

1: Input  $pchange$ 
2: Output  $SF$ 
3: Let  $A_c$  be the set of activities involved in  $pchange$ 
4:  $O^i \leftarrow \emptyset (i = 1, \dots, n)$ 
5: for all  $a \in A_c$  do
6:   if  $a$  is the c-activity relating to  $p_i (i = 1, \dots, n)$  then
7:      $O^i \leftarrow O^i \cup \{o\}$  ( $o$  is associated with  $a$ )
8:   end if
9: end for
10:  $sf_i \leftarrow O^i (i = 1, \dots, n)$ 
11: for all  $sf_i (i = 1, \dots, n)$  do
12:   for all  $o \in O^i$  do
13:     for all  $t_j$  that associated with  $o$  do
14:        $sf_i \leftarrow sf_i \cup \{t_j\}$ 
15:       if  $o_x$  is associated with  $t_j$  &&  $o_x \neq o$  then
16:          $sf_i \leftarrow sf_i \cup \{o_x\}$ 
17:       end if
18:     end for
19:   end for
20: end for
21:  $SF \leftarrow \emptyset$ 
22: for all  $sf_i (i = 1, \dots, n)$  do
23:   if  $sf_i \neq \emptyset$  then
24:      $SF \leftarrow SF \cup \{sf_i\}$ 
25:   end if
26: end for

```

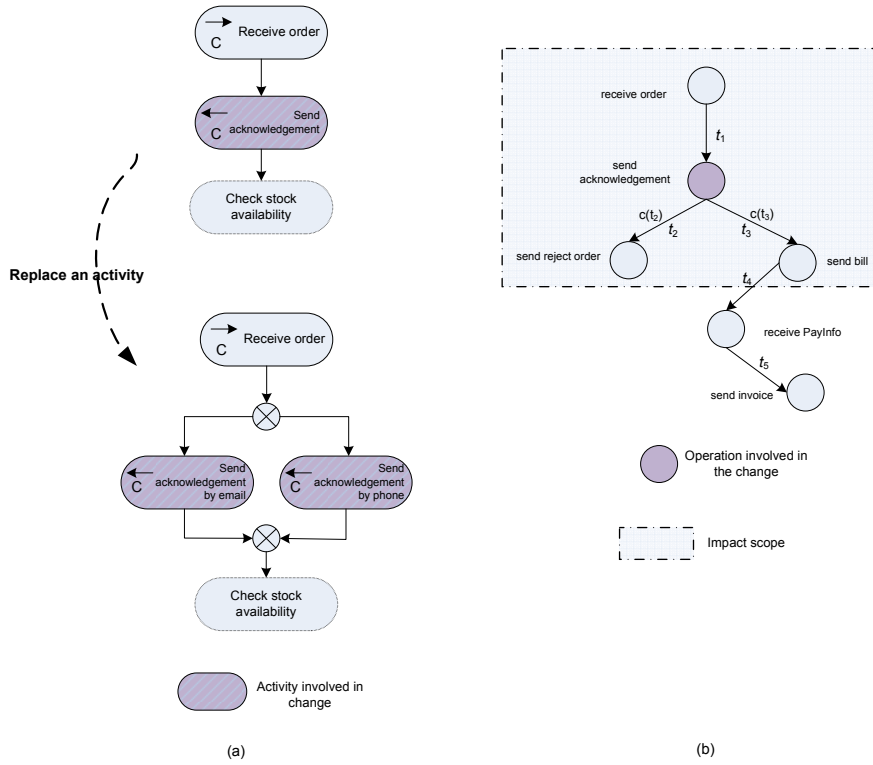


Figure 5.5: (a) Process change: replace activities; (b) direct impact scope of the process change.

empty set of service fragments are included into SF (lines 21-26). The following example illustrates the direct impact scope of a process change calculated by *FuncDISP*.

Example *Direct Impact Scope of a Process Change* We take the sales process as an example. Figure 5.5(a) shows a process change: replace an activity, where *Send acknowledgement* is replaced by an xor construct with two activities: *Send acknowledgement by email* and *Send acknowledgement by phone*. This process change enables the sales process to send acknowledgement to a buyer by the preferred way specified by the buyer. Figure 5.5 (b) shows the direct impact scope of the process change. In this figure, *send acknowledgement* is the operation associated with the activity *Send acknowledgement*, which is involved in the process change. With definition 2, transitions t_1 , t_2 , and t_3 , and operations

receive order, *send reject order*, and *send bill* are included into the direct impact scope of this process change. Therefore, the direct impact scope of the process change is: $SF = \{sf\}$, where

$$sf = \{\textit{receive order}, t_1, \textit{send acknowledgement}, t_2, t_3, \textit{send reject order}, \textit{send bill}\}$$

5.3 Change Impact Patterns

In this section we present the identified change impact patterns. We discuss the impact patterns for service change and impact patterns for process change respectively in the following sub sections.

5.3.1 Change Impact Patterns for Service Change

The change impact patterns for service change include the impact patterns that capture the effect on internal processes made by service changes. The types of change effect on internal processes are described by *abstract control relations* associated with c-activities.

Impact pattern 1 Insert a c-Activity The *Insert a c-Activity* pattern (cf. Figure 5.6) describes the change impact that a c-activity needs to be inserted to the internal process. The cause of this type of impact is adding an operation to a service. The effect on the internal process is categorized into four types:

- a c-activity needs to be serially inserted into the internal process between two successively executed c-activities without conditions. Figure 5.6 (1) shows an example. An operation o_x is added in between two operations o_i and o_j . The impact of this service change is that a c-activity a_x associated with o_x is inserted between activities a_i and a_j that are associated with

operations o_i and o_j respectively.

- a c-activity needs to be serially inserted into the internal process between two successively executed c-activities with conditions. Figure 5.6 (2) shows an example. An operation o_x is added between operations o_i and o_j with constraints c_{x1} and c_{x2} . The impact of this service change is that a c-activity a_x associated with o_x needs to be inserted conditionally between activities a_i and a_j that are associated with operations o_i and o_j . The conditions for controlling the execution of a_x are derived from the constraints c_{x1} and c_{x2} .
- a c-activity needs to be inserted into the internal process in parallel to existing activities without conditions. Figure 5.6 (3) shows an example. An operation o_x is added in parallel to the operation o_j . The change effect on the internal process is that an activity a_x associated with o_x is inserted in parallel to the activity a_j associated with operation o_j .
- a c-activity needs to be inserted into the internal process in parallel to existing activities with conditions. Figure 5.6 (4) shows an example. An operation o_x is added in parallel to the operation o_j with a constraint c_x . The change effect on the internal process is that an activity a_x associated with o_x needs to be inserted in parallel to the activity a_j associated with o_j conditionally. The condition for controlling the execution of a_x is derived from the constraint c_x .

Impact pattern 2 Remove a c-Activity The *Remove a c-Activity* pattern (cf. Figure 5.7) describes the impact that c-activities need to be removed from the internal process. This type of impact is caused by deleting an operation in a service. Figure 5.7 gives an example. The conditionally invoked operation o_x

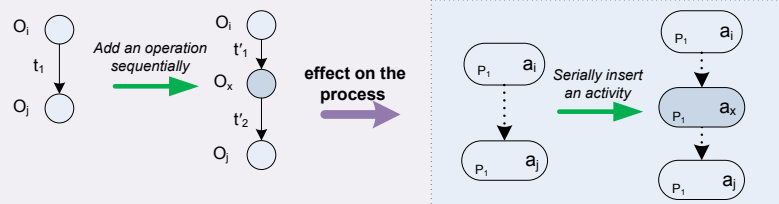
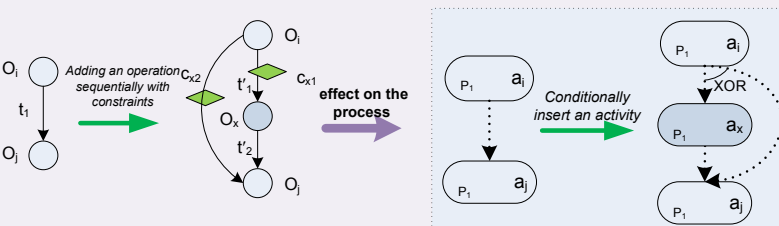
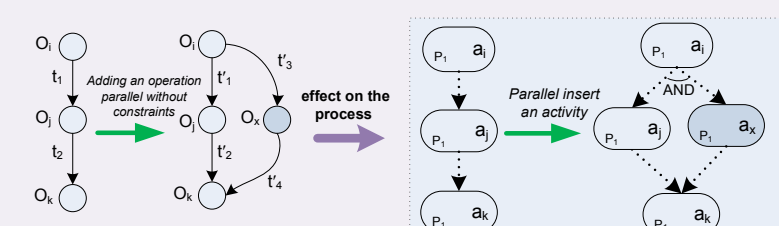
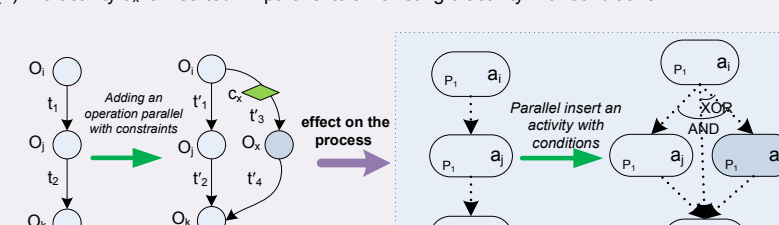
Name	Impact pattern 1 Insert a c-Activity
Pattern description	This impact pattern describes that a c-activity needs to be added to the internal process.
Cause	This type of impact is caused by adding an operation to a service.
Effect on the internal process	<p>The change effect on the internal process is categorized into the following four types.</p> <p>(1) A c-activity a_x is serially inserted between two successively executed c-activities without conditions.</p>  <p>(2) A c-activity a_x is serially inserted between two successively executed c-activities with conditions.</p>  <p>(3) A c-activity a_x is inserted in parallel to an existing c-activity without conditions.</p>  <p>(4) A c-activity a_x is inserted in parallel to an existing c-activity with conditions.</p> 

Figure 5.6: Change impact pattern 1 Insert a c-Activity.

Name	Impact pattern 2 Remove a c-Activity
Pattern description	This impact pattern describes that c-activities need to be removed from the internal process.
Cause	This type of impact is caused by deleting operations in a service.
Effect on the internal process	<p>Let o_x be the operation that needs to be deleted from the service s_{p1}, and a_x be the c-activity Which is associated with o_x. The c-activity a_x needs to be removed from the internal process.</p>

Figure 5.7: Change impact pattern 2 Remove a c-Activity.

between operations o_i and o_j is deleted. The effect on the internal process is that activity a_x associated with o_x needs to be removed from the internal process.

Impact pattern 3 Replace c-Activities The *Replace c-Activities* pattern (cf. Figure 5.8) describes the impact that existing c-activities need to be replaced by a c-activity or a set of c-activities. This type of impact is caused by changing granularity of operations in a service. The effect on the internal process is complicated due to the diversity of operation granularity changes. We classify this type of change impact into the following four sub types:

- an existing c-activity needs to be replaced by another c-activity. Figure 5.8 (1) shows an example. An asynchronous operation o_x is changed to another asynchronous operation o'_x . In addition, the transition sequence associated with o_x is modified, where o'_x is in a looping transition sequence controlled by the constraints c_{x1} and c_{x2} . The effect on the internal process is: the activity a_x associated with o_x must be replaced by another activity a'_x and a'_x is executed conditionally. The conditions for controlling the execution of a'_x are derived from the constraints c_{x1} and c_{x2} .
- an existing c-activity needs to be replaced by a set of c-activities. Figure

5.8 (2) shows an example. A synchronous operation o_x is modified to three asynchronous operations o_{y1} , o_{y2} , and o_{y3} . The operation o_{y1} is in parallel to o_{y2} and o_{y3} . In addition, the operation o_{y3} is in a looping transition sequence. This operation granularity change incurs that the c-activity a_x associated with o_x is replaced by two *send* type c-activities a_{y1} and a_{y2} and a *receive* type c-activity a_{y3} . The c-activity a_{y1} is executed in parallel to the other two c-activities. The c-activity a_{y3} is repeatedly executed with conditions. The conditions for controlling the execution of a_{y3} are derived from the constraints c_{x1} and c_{x2} .

- a set of c-activities need to be replaced by a c-activity. This type of impact is similar to the second type of effect described above.
- a set of c-activities need to be replaced by another set of c-activities. Figure 5.8 (4) shows an example. Three asynchronous operations o_{x1} , o_{x2} , and o_{x3} are changed to two asynchronous operations o_{y1} and o_{y2} . In addition, o_{y1} and o_{y2} are invoked sequentially. This granularity change has the impact on the internal process that three c-activities that are associated with operations o_{x1} , o_{x2} , and o_{x3} are replaced by two c-activities a_{y1} and a_{y2} that are associated with operations o_{y1} and o_{y2} .

Impact pattern 4 Move c-Activities The *Move c-Activities* pattern (cf. Figure 5.9) describes the impact that existing c-activities need to be reordered. This type of impact is caused by operation transition sequence changes, including transition sequence order change (TSOC), sequential to parallel transition sequence change (SPTSC), and parallel to sequential transition sequence change (PSTSC). The type of change impact is categorized into the following two sub types:

- c-activities need to be serially moved. Figure 5.9 (1) shows two examples

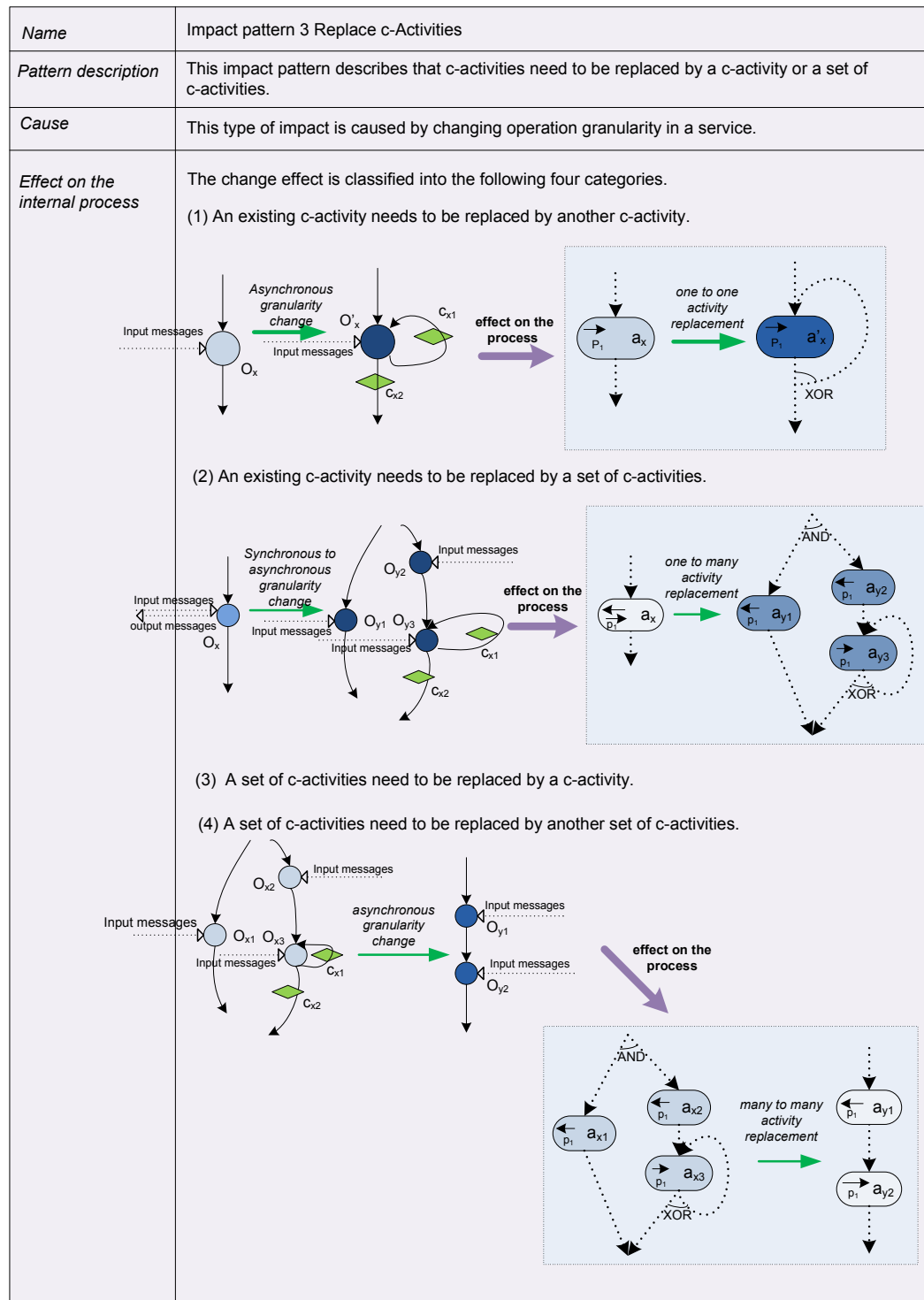


Figure 5.8: Change impact pattern 3 Replace c-Activities.

for this type of impact. In the first example, a transition sequence $o_i t_1 o_j t_2 o_k$ is changed its order as $o_k t'_1 o_i t'_2 o_j$. This TSOC has the effect on the internal process that the activity a_k associated with o_k is serially moved to a position that precedes a_i . The second example describes a TSOC involving constraints. Before the change, operation o_j is invoked conditionally after o_i . After the change, o_j is moved before o_i . The constraints that control the invocation of o_j are also retained. This service change has the impact that the activity a_j associated with o_j is moved before a_i . The conditions that control the execution of a_j are also retained;

- c-activities need to be parallel moved. Figure 5.9 (2) shows two examples for this type of impact. In the first example, a transition sequence $o_l t_1 o_i t_2 o_j t_3 o_k$ is split into two parallel transition sequences: $o_l t'_1 o_j t'_3 o_k$ and $o_l t'_2 o_i$. This SPTSC has the impact on the internal process that the activity a_i associated with o_i is moved in parallel to the activity a_j associated with o_j . The second example describes the case of SPTSC involving constraints. Before the change, there are two conditional transition sequences: $o_l t_1 o_i t_2 (c_{x2}) o_s t_3$ and $o_l t_1 o_i t_4 (c_{x1}) o_j t_5 o_k$, which indicates that the operations o_s and o_j are invoked conditionally depending on the value of the constraints c_{x1} and c_{x2} . After the service change, the two conditional transition sequences are modified to three transition sequences as: $o_l t'_1 o_i t'_4 o_k$, $o_l, t'_2 (c_{x2}) o_s t'_4 o_k$, and $o_l t'_3 (c_{x1}) o_j t'_5 o_k$. This SPTSC is caused by enabling o_i to be invoked in parallel to o_j and o_s . This change has the effect that the activity a_i associated with o_i is moved to the position that is in parallel to the activity a_j and a_k .

Impact pattern 5 Add, Remove or Modify Conditional Branches The *Add, Remove or Modify Conditional Branches* pattern (cf. Figure 5.10) describes

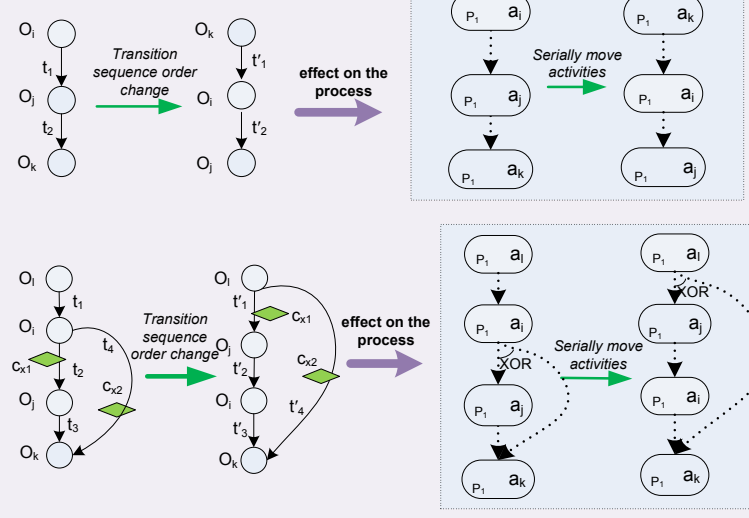
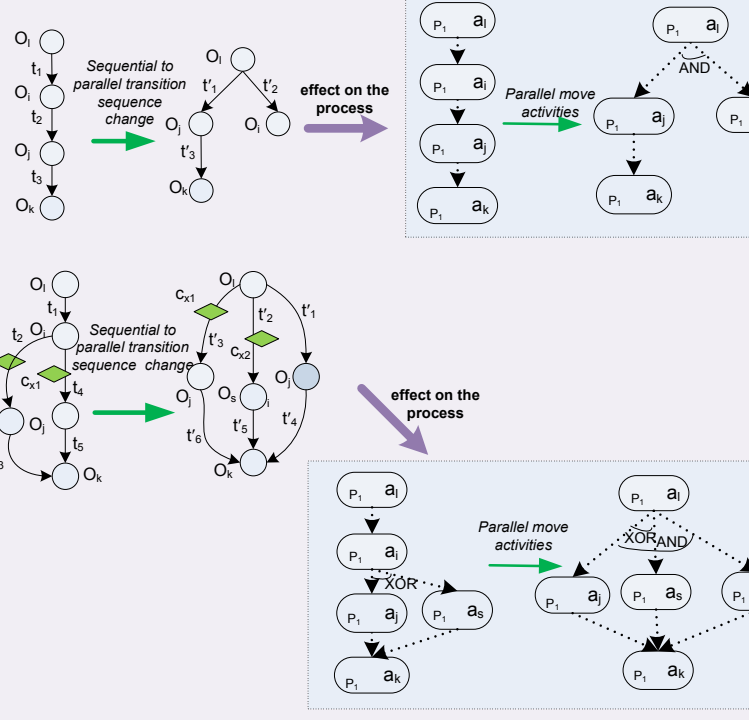
<i>Name</i>	Impact pattern 4 Move c-Activities
<i>Pattern description</i>	The impact pattern describes that existing c-activities need to be reordered.
<i>Cause</i>	This type of impact is caused by changing operation transition sequences, including transition sequence order change (TSOC), sequential to parallel transition sequence change (SPTSC), and parallel to sequential transition sequence change (PSTSC).
<i>Effect on the internal process</i>	<p>The effect on the internal process is classified into the following two types.</p> <p>(1) C-activities need to be serially moved.</p>  <p>(2) C-activities need to be parallel moved.</p> 

Figure 5.9: Change impact pattern 4 Move c-Activities.

the impact that xor structures need to be modified or new xor structures need to be created. This impact is caused by transition sequence changes including adding conditional transition sequence change (ACTSC), removing conditional transition sequence change (RCTSC), adding looping transition sequence change (ALTSC), and removing looping transition sequence change (RLTSC). This type of impact is further classified into the following two types:

- c-activities need to be embedded in or removed from a conditional branch. Figure 5.10 (1) gives an example, where conditional transition sequences are added. This service change means that the operation o_j is invoked conditionally depending on the value of the constraints c_{x1} and c_{x2} . The effect on the internal process is that the activity a_j associated with o_j is executed with conditions that are derived from the constraints c_{x1} and c_{x2} ;
- c-activities need to be embedded in or removed from a looping branch. Figure 5.10 (2) gives an example, where a looping transition sequence is added. This service change means that the operation o_j is invoked repeatedly depending on the value of the constraints c_{x1} and c_{x2} . The effect on the internal process is that the activity a_j associated with o_j is repeatedly executed with conditions that are derived from the constraints c_{x1} and c_{x2} .

5.3.2 Change Impact Patterns for Process Change

The change impact patterns for process change include the patterns that capture the effect on the services associated with internal processes made by process changes. These types of effect on services are described by the various types of service changes that have been defined in Chapter 4.

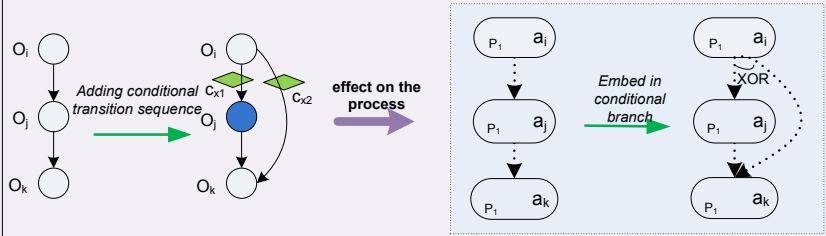
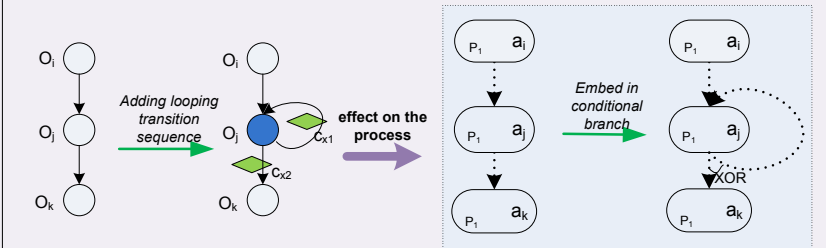
Name	Impact pattern 5 Add, Remove or Modify Conditional Branches
Pattern description	This impact pattern describes that a new conditional branch must be added, a conditional branch must be removed, or existing Conditional branches must be modified.
Cause	This type of impact is caused by adding/removing conditional transition sequence (ACTS/RCTS) or adding/removing looping transition sequence (ALTS/RLTS).
Effect on the internal process	<p>The effect on the internal process is classified into the following two types.</p> <p>(1) C-activities must be embedded in or removed from a conditional branch.</p>  <p>(2) C-activities must be embedded in or removed from a looping branch.</p> 

Figure 5.10: Change impact pattern 5 Add, Remove or Modify Conditional Branches.

Impact pattern 6 Add Operations The *Add Operations* pattern (Figure 5.11) describes the impact that operations need to be added to services. This type of impact is caused by inserting a c-activity or replacing an existing c-activity with c-activities in the internal process. We discuss the type of change impact that a single operation must be added. The type of change impact that multiple operations must be added can be similarly defined. This type of change effect is classified into the following four sub types:

- an operation needs to be added sequentially between operations without constraints. Figure 5.11 (1) shows an example of this type of change impact. In the internal process, a c-activity a_x is inserted between two successively executed c-activities a_i and a_j . These three c-activities are all related to the partner p_1 . This process change has the impact on the service s_{p_1} that an operation o_x associated with a_x is added sequentially between o_i and o_j ;
- an operation needs to be added sequentially between operations with constraints. Figure 5.11 (2) shows an example of this type of change impact. In the internal process, a c-activity a_x relating to the partner p_1 is inserted between two successively executed p-activities a_i and a_j . After the process change, the three activities a_x , a_i , and a_j are in a conditional branch, which is between two c-activities a_{k1} and a_{k2} relating to the partner p_1 . This process change has the impact on the service s_{p1} that an operation o_x associated with a_x is added sequentially between o_i and o_j with constraints that are derived from the conditions of the corresponding *xor* connector;
- an operation needs to be added in parallel to existing operations without constraints. Figure 5.11 (3) shows an example of this type of change impact. In the internal process, a c-activity a_x relating to the partner p_1 is inserted

in parallel to the c-activity a_{k2} that also relates to the partner p_1 . Note that the *and* branches containing a_x and a_{k2} is between c-activities a_{k1} and a_{k2} that also relate to the partner p_1 . This process change has the impact on the service s_{p1} that the operation o_x associated with a_x is added in parallel to the operation o_{k2} associated with a_{k2} without constraints;

- an operation needs to be added in parallel to existing operations with constraints. Figure 5.11 (4) shows an example of this type of change impact. A c-activity a_x relating to the partner p_1 is inserted in a conditional branch that contains no other c-activities relating to p_1 . This conditional branch is in an *and* branch which is in parallel to the c-activity a_{k2} relating to p_1 . This process change has the impact on the service s_{p1} that the operation o_x associated with a_x is added in parallel to o_{k2} with a constraint derived from the conditions of the corresponding *xor* connector.

Impact pattern 7 Remove Operations The *Remove Operations* pattern (cf. Figure 5.12) describes the impact that operations need to be deleted from services. The cause of this impact is the deletion of c-activities or replacement of existing c-activities in the internal process. For example, in Figure 5.12 a c-activity a_x is deleted. This process change has the impact on the service that the operation o_x associated with a_x needs to be removed from the corresponding service. If multiple c-activities are deleted or replaced, the associated operations need to be removed from the corresponding services.

Impact pattern 8 Change Operation Granularity The *Change Operation Granularity* pattern (cf. Figure 5.13) describes the impact that operation granularity needs to be modified. This type of impact is caused by replacing existing c-activities with other c-activities. We classify this type of change effect into the following three sub types:

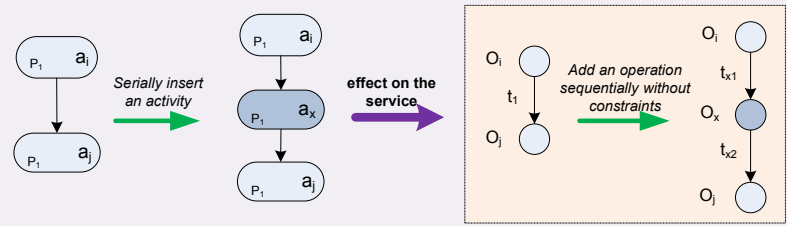
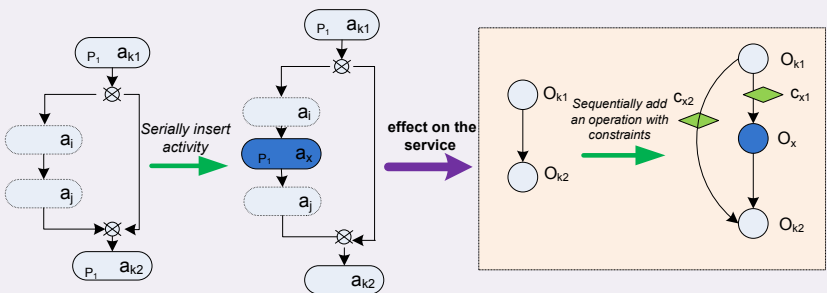
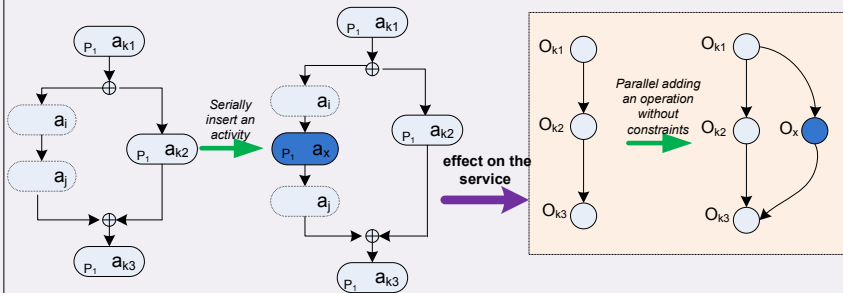
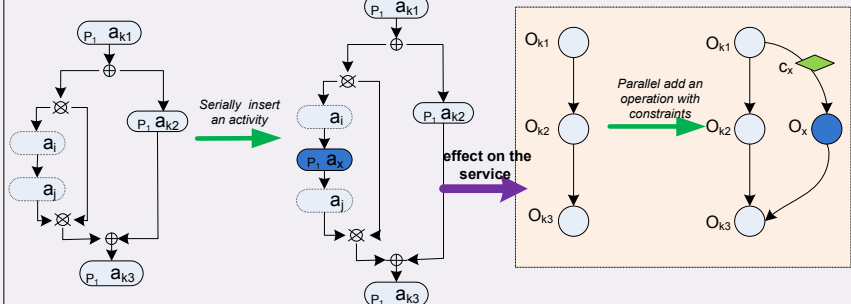
Name	Impact pattern 6 Add operations
Pattern description	This impact pattern describes that operations need to be added to the corresponding services.
Cause	This type of impact is caused by inserting a c-activity or replacing an existing c-activity in the internal process.
Effect on the services	<p>The effect on services is classified into the following four types.</p> <p>(1) An operation needs to be added sequentially between existing operations without constraints.</p>  <p>(2) An operation needs to be sequentially added between existing operations with constraints.</p>  <p>(3) An operation needs to be added in parallel to existing operations without constraints.</p>  <p>(4) An operation needs to be added in parallel to existing operations with constraints.</p> 

Figure 5.11: Change impact pattern 6 Add an Operation.

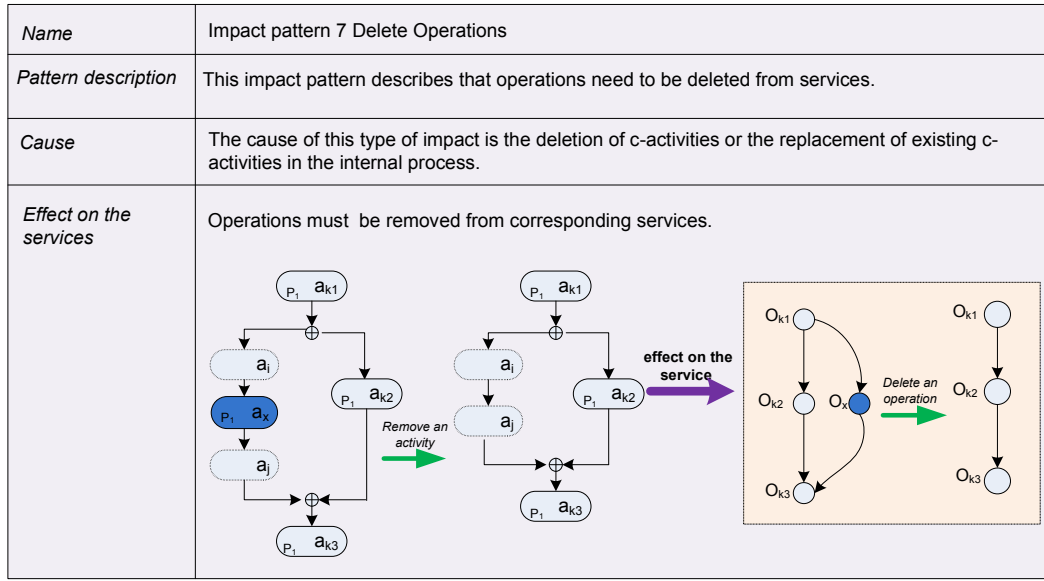


Figure 5.12: Change impact pattern 7 Remove Operations.

- granularity of asynchronous operations must be modified. Figure 5.13 (1) gives an example of this type of impact. In the internal process, a *receive* type of c-activity a_x relating to p_1 is replaced by another *receive* type of c-activity a_y . In addition, the c-activity a_y is embedded in a conditional branch. This process change has the impact on the service s_{p_1} that the asynchronous operation o_x associated with a_x must be changed to another asynchronous operation o_y associated with a_y . Moreover, the operation o_y is in a looping transition sequence. The constraints that control the invocation of o_y are derived from the conditions of the corresponding *xor* connector;
- granularity of synchronous operations must be modified. Figure 5.13 (2) shows an example of this type of effect. In the internal process, a *send/receive* type of c-activity a_x relating to the partner p_1 is replaced by another *send/receive* type of c-activity a_y . In addition, a_y is embedded in a condi-

tional branch. This process change has the impact on the service s_{p_1} that the synchronous operation o_x associated with a_x is modified in terms of granularity into another synchronous operation o_y . Moreover, o_y is in a looping transition sequence. The constraints that control the invocation of o_y is derived from the conditions of the corresponding *xor* connector;

- granularity of both asynchronous and synchronous operations must be modified. Figure 5.13 (3) gives an example. In the internal process, a *send/receive* type of c-activity a_x relating to the partner p_1 is replaced by two *send* type of c-activities a_{y1} and a_{y2} and a *receive* type of c-activity a_{y3} . The operation a_{y1} is invoked in parallel to a_{y2} and a_{y3} . In addition, a_{y2} is succeeded by a_{y3} that is repeatedly executed. This process change has the impact on the service s_{p_1} that the synchronous operation o_x associated with a_x is changed to three asynchronous operations o_{y1} , o_{y2} , and o_{y3} . Moreover, o_{y1} is in parallel to o_{y2} and o_{y3} that can be repeatedly invoked. The constraints for controlling the invocation of o_{y3} are derived from the conditions of the corresponding *xor* connector.

Impact pattern 9 Change Transition Sequence The *Change Transition Sequence* pattern (cf. Figure 5.14) describes the impact that transition sequences need to be reordered. This type of impact is caused by process changes including moving activities, parallelizing activities, and sequencing activities. We classify this type of change effect into the following three sub types:

- transition sequences must be reordered. Figure 5.14 (1) shows an example. In the internal process, the c-activity a_x relating to the partner p_1 is serially moved into a conditional branch in a position succeeding a_x . This process change has the impact on the service s_{p_1} that the transition sequences

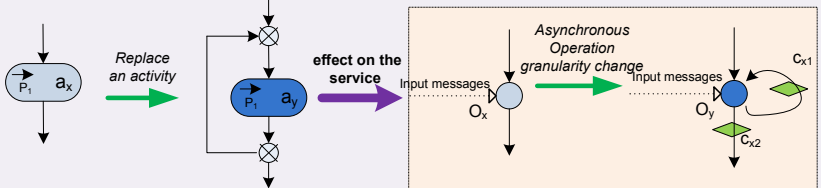
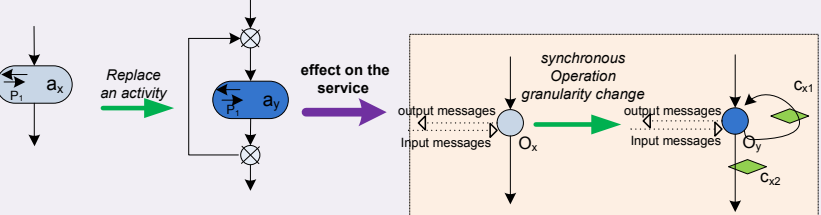
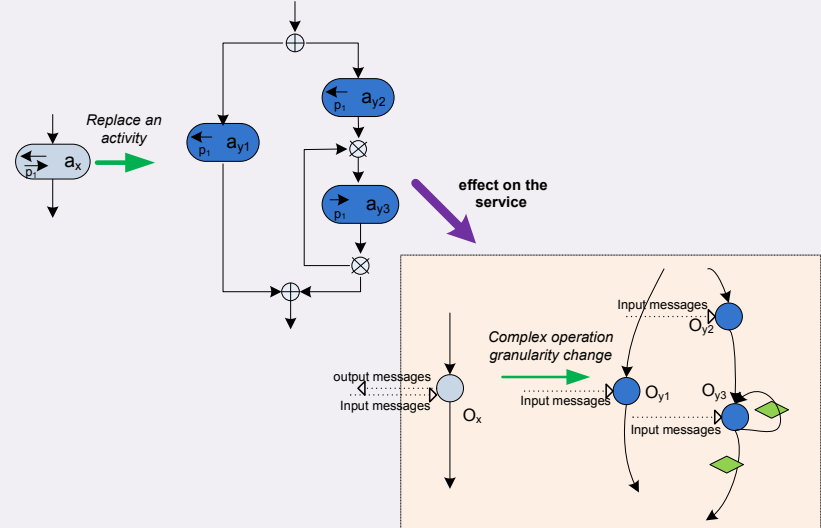
Name	Impact pattern 8 Change Operation Granularity
Pattern description	This impact pattern describes that operation granularity needs to be modified.
Cause	The cause of this type of impact is the change of replacing existing c-activities with new c-activities in the internal process.
Effect on the services	<p>The impact on services is categorized into the following three types.</p> <p>(1) Asynchronous operation granularity change (AOGC).</p>  <p>(2) Synchronous operation granularity change (SOGC).</p>  <p>(3) Complex operation granularity change (COGC).</p> 

Figure 5.13: Change impact pattern 8 Change Operation Granularity.

related to the operation o_x which is associated with a_x must be reordered accordingly;

- sequential transition sequences must be changed to parallel transition sequences. Figure 5.14 (2) shows an example. In the internal process, the c-activity a_x relating to the partner p_1 is moved to the position in parallel to the c-activity a_i that also relates to the partner p_1 . Note that the c-activities a_i and a_x are in a conditional branch after this process change. The change impacts on the service s_{p_1} that the transition sequences related to the operation o_x that is associated with a_x must be changed to parallel transition sequences. As shown in the figure, the conditional transition sequences: $o_h t_1 o_x t_2 (c_{x1}) o_i t_3 o_j$ and $o_h t_1 o_x t_4 (c_{x2}) o_j$ are changed to the transition sequences: $o_h t'_1 (c_{x1}) o_i t'_2 o_j$, $o_h t'_3 (c_{x1}) o_x t'_4 o_j$, and $o_h t'_5 (c_{x2}) o_j$. The invocation of o_i and o_j are controlled by the constraint c_{x1} . Both the constraints c_{x1} and c_{x2} are derived from the conditions of the corresponding *xor* connector;
- parallel transition sequences must be changed to sequential transition sequences. Figure 5.14 (2) is also an example of this type of impact.

Impact pattern 10 Add Conditional or Looping Transition Sequence

The *Add Conditional or Looping Transition Sequence* pattern (cf. Figure 5.15) describes the impact that conditional or looping transition sequences need to be added to operations. This type of impact is caused by embedding activities in conditional branches. We classify this type of change effect into the following two sub types:

- conditional transition sequences need to be added in services. Figure 5.15 (1) shows an example of this type of impact. In the internal process, the

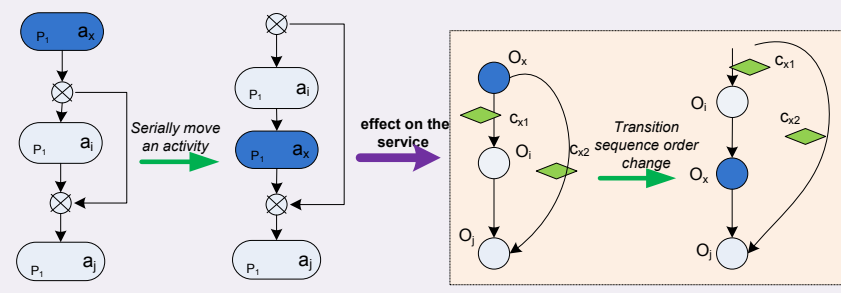
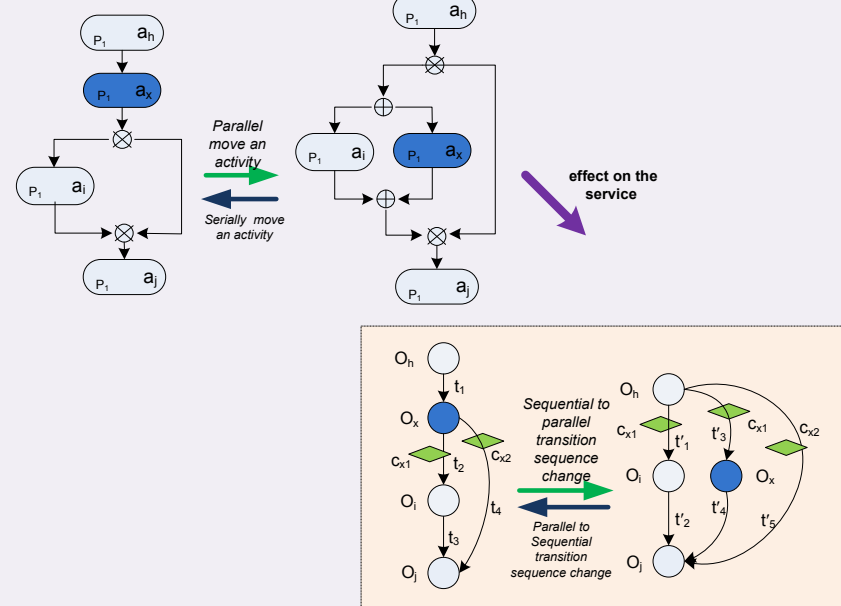
Name	Impact pattern 9 Change Transition Sequence
Pattern description	This impact pattern describes that transition sequences need to be reordered.
Cause	The cause of this type of impact is the changes of moving activities, parallelizing activities and sequencing activities in the internal process.
Effect on the services	<p>We classify the impact into the following three categories.</p> <p>(1) Transition sequence order change (TSOC).</p>  <p>(2) Sequential to parallel transition sequence change (SPTSC).</p>  <p>(3) Parallel to sequential transition sequence change (PSTSC).</p>

Figure 5.14: Change impact pattern 9 Change Transition Sequence.

c-activity a_x relating to partner p_1 is embedded in a conditional branch. This process change has the impact on the service s_{p_1} that the transition from o_i to o_x is changed by adding a constraint c_{x1} . In addition, a transition is added from o_i to o_j with a constraint c_{x2} . Both the constraints c_{x1} and c_{x2} are derived from the conditions of the corresponding *xor* connector;

- looping transition sequences need to be added in services. Figure 5.15 (2) shows an example of this type of impact. In the internal process, the c-activity a_x relating to partner p_1 is embedded in a conditional branch which enables repeated execution of a_x . This process change has the impact on the service s_{p_1} that the transition from operation o_x to o_j is changed by adding a constraint c_{x2} . In addition, a transition is added from o_k to o_k with a constraint c_{x2} which enables the repeated invocation of o_x . Both the constraints c_{x1} and c_{x2} are derived from the conditions of the corresponding *xor* connector.

5.4 Discussion

This chapter reports our work on change impact analysis for service-based business processes where multiple services are supported by a single business process. Ten change impact patterns are specified for capturing the different types of effect caused by service changes and process changes. The change impact patterns are categorized into impact patterns of service changes and impact patterns of process changes. Each change impact pattern contains the following elements: a pattern name, a pattern description, a description stating the cause of the change effect, the direct impact scope calculated by the defined functions, and the effect on services/internal process. The specified change impact patterns provide rich

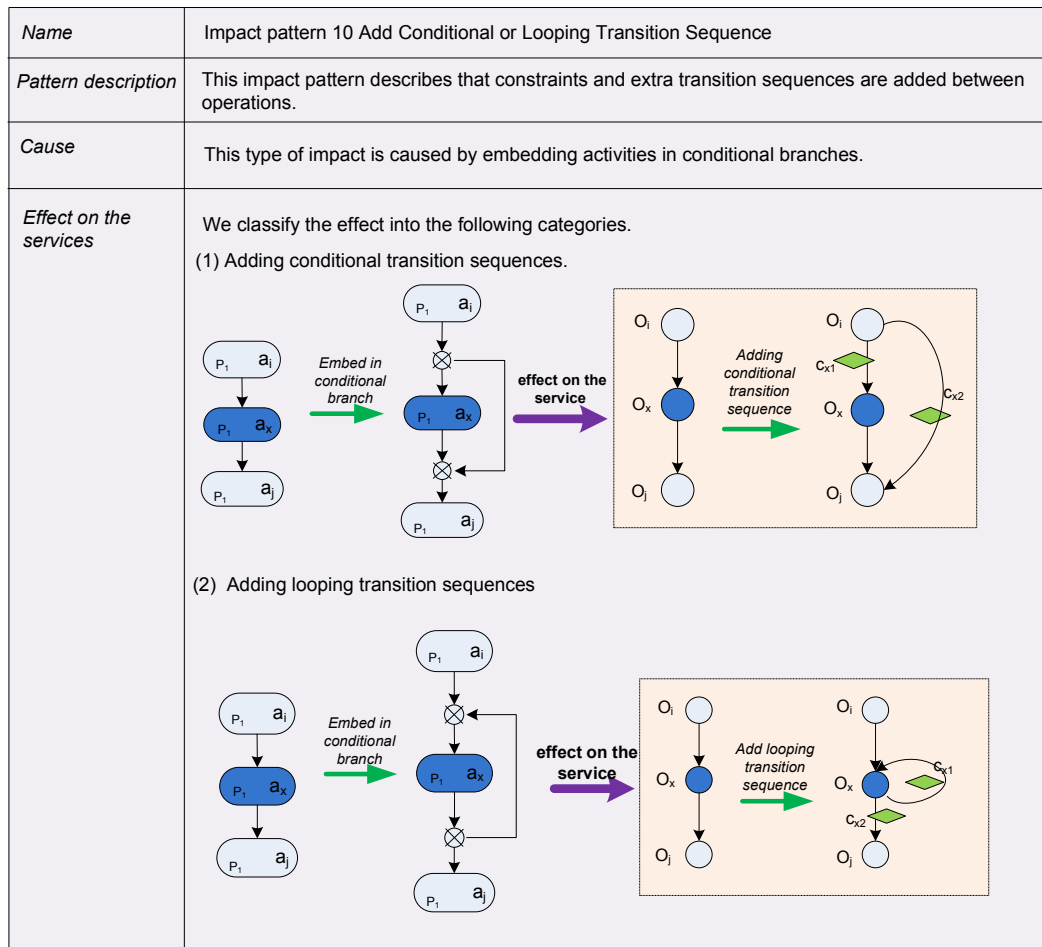


Figure 5.15: Change impact pattern 10 Add Conditional or Looping Transition Sequence.

intermediate results for the change impact analysis process and are helpful to reduce the complex tasks of change management. These impact patterns can be reused in the development and maintenance of service-based applications and information systems. The proposed change impact patterns provide solid foundations for handling different types of change impact and controlling the change propagation in service-based applications and information systems.

Change impact analysis is crucial for developing effective and efficient mechanisms for handling the various types of changes and cutting off the change propagation. Existing works about change management in the service-oriented context suggest proposals for managing changes for BPEL process orchestration and choreography [80, 108], dealing with changes in Web services environments [57, 59], and supporting the evolution of service specifications [9, 10] and business protocols [83]. In the field of service adaptation which is closely related to the service change management, approaches for adapting service interfaces and business protocols are proposed [12, 28, 46]. In the real world, there are various types of dependencies between services and business processes in service-based applications and information systems. Change analysis and change reactions are challenging issues due to the complicated dependencies between services and business processes. The existing works mentioned above concentrate on changes related to services or business processes respectively. The dependencies between services and business processes are neglected by those researches. Our work fills this gap by presenting the approach for change impact analysis in service-based business processes where multiple services are supported by a single business process.

Chapter 6

Change Handling

As discussed in the previous chapter, a change can have various level of impact on business processes and services due to the complicated dependencies between business processes and services. Let us go over the sales scenario given in Chapter 3 as an example. A sales process receives an order from a buyer, checks the stock availability, and sends confirmation to the buyer. If an order is accepted, the sales process sends the bill to the buyer. The payment is processed by a finance institute. The buyer is issued with an invoice after the payment. The sales process handles the shipment of the goods with the support of a shipping company. In this scenario, the sales process interacts with three partners as a buyer, a finance institute, and a shipping company. In the service-oriented environment, the three partners interact with the sales process by invoking the corresponding services exposed by this sales process. Each service is an external view of the sales process from the view point of a specific partner. Private tasks of the sales process, such as checking stock availability and processing invoices are hidden from its partners. This sales scenario exemplifies a type of the coupling relations between services and business processes when multiple services are supported by a single business process. If a change occurs in any of the services, this change usually affects the business process and may have impact on the other services associated with this business process. If a change occurs in the business process, this change may

affect the associated services. To manage changes in the context of SOC, it is crucial to identify the various types of changes, analyse the change impact, and decide effective and efficient mechanisms for dealing with those changes.

In this chapter, we discuss how to deal with various types of changes. The end goal is to provide effective mechanisms for controlling changes and their cascading effect on other services and business processes. First, we discuss how to handle the individual types of changes based on the change impact patterns. Then we analyse the change propagation in business processes and services. The concept of actual impact scopes for a specific change will be defined on the basis of the direct impact scopes and the analysis of change propagation. Finally, we will discuss the issue of *change isolation* in service-based applications and information systems. Change isolation refers to the action of cutting off the cascading effect caused by a specific change with the goal to minimize the change impact on the entire system.

This chapter is organized as follows. In Section 6.1, we provide the mechanisms for dealing with the changes based on the impact patterns. For individual types of service changes, we will discuss how to handle them in the internal process at the activity level and at the operation level respectively. For process changes, we will discuss how to handle each type of changes in affected services. In Section 6.2, we discuss how to analyse change propagation in business processes and services. We will provide an example for showing the change propagation of a service change. The actual impact scopes of a specific change will be defined based on the analysis of change propagation. In Section 6.3, we discuss the issue of change isolation. Finally, we conclude this chapter in Section 6.4.

6.1 Handling Individual Changes

This section discusses the possible solutions for dealing with each type of the changes based on the change impact patterns. For every type of service changes, the change handling mechanisms are discussed at two levels: the activity level and the operation level.

6.1.1 Dealing with Service Change: Add an Operation

Assume an operation o_x is added to a service. The change impact is described in the change impact pattern 1 *Insert a c-Activity*. Based on this impact pattern, a c-activity needs to be inserted to the internal process.

Managing Change at Activity Level

Managing this type of change at the activity level involves three steps.

- The c-activity a_x associated with the operation o_x is created.
- Data connections relating to a_x must be established. This step may involve modifying existing p-activities and c-activities. Moreover, new c-activities and p-activities may be created to assist the tasks specified by a_x .
- The c-activity a_x and other activities that are created in the second step must be inserted into the internal process properly.

Managing Change at Operation Level

We discuss how to deal with the change *Add an Operation* at the operation level by the following four cases.

- The operation o_x exists but is invisible to the partner. In this case, o_x can be exposed to the partner by associating this operation with the c-activity a_x .

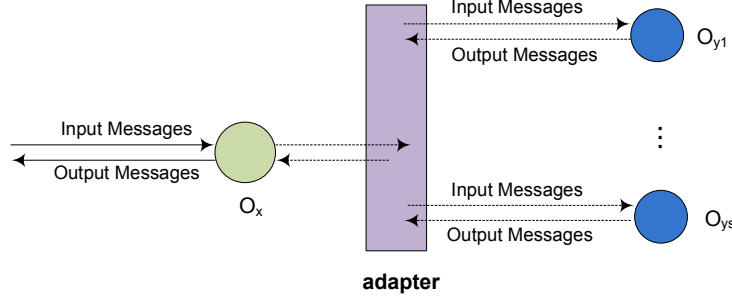


Figure 6.1: Adapter for solving *type A* case at operation level.

- There exists an operation or a set of operations that has similar functionality to o_x . Let $\{o_{y1}, \dots, o_{ys}\}$ be the operations that has similar functionality to o_x . We classify this case into two sub types according to the relations between o_x and $\{o_{y1}, \dots, o_{ys}\}$.
 - *Type A* $o_x \subset \{o_{y1}, \dots, o_{ys}\}$. This relation describes that a set of existing operations cover the functionality specified by o_x . The operation o_x can be created by wrapping the existing operations with an adapter (cf. Figure 6.1). The function of the adapter is to generate the input (and output) messages of the operation o_x by using the existing operations. The adapter accepts the input messages of o_x and assigns these messages to the set of operations $\{o_{y1}, \dots, o_{ys}\}$. The output messages of these operations are collected and transformed by the adapter as the output of o_x . The adapter makes the set of operations $\{o_{y1}, \dots, o_{ys}\}$ behave like o_x . We call o_x an *abstract operation*.
 - *Type B* The condition in *type A* does not hold and $o_x \cap \{o_{y1}, \dots, o_{ys}\} \neq \emptyset$. This relation means that the existing operations cover some functionality of o_x . In such a case, an operation o_z is created. The functionality of o_z is decided by $o_z = o_x \setminus \{o_{y1}, \dots, o_{ys}\}$. That is, o_z covers

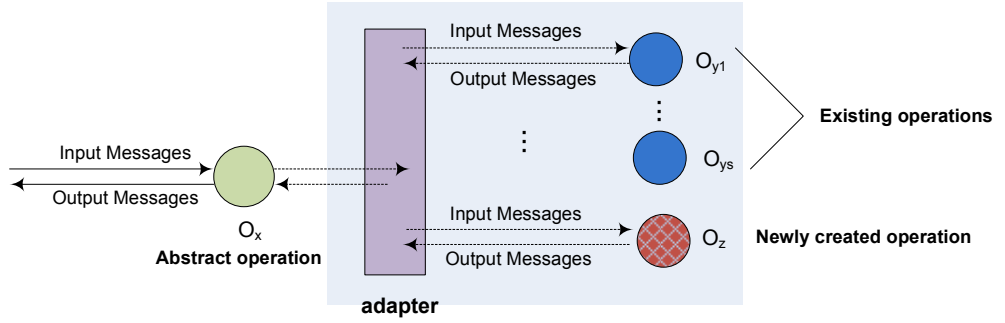


Figure 6.2: Adapter for solving *type B* case at operation level.

the functionality that is required by o_z but not provided by the existing operations. Similar to *Type A*, an adapter is used to generate an abstract operation o_x by the set of operations $\{o_{y1}, \dots, o_{ys}, o_z\}$ (cf. Figure 6.2).

- For other cases that are not covered by the above discussion, a new operation o_x is created.

6.1.2 Dealing with Service Change: Delete an Operation

Let us assume that the operation o_x is to be deleted. This change has the effect on the internal process that a c-activity needs to be deleted which is described in the change impact pattern 2 *Remove a c-Activity*. For this type of change, no actions are required at the operation level. In the following, we discuss the mechanisms for managing this type of change at the activity level.

Managing Change at Activity Level

Let activity a_x be the c-activity associated with the operation o_x . We identify two cases when dealing with this type of change at the activity level.

- The c-activity a_x is not depended by any other activities in the internal process in terms of data. For example, suppose a_x is a *send* type of c-

activity, which sends a notification to the corresponding partner. In the internal process, there exist no activities that depend on a_x in terms of data. In this case, a_x and the associated data connections can be removed from the internal process.

- The c-activity a_x is depended by some activities in the internal process in terms of data. For example, suppose a_x is a *receive* type of c-activity, which accepts information from a partner. The received data is processed by the internal process and then send to another partner. In this case, before deleting a_x modifications on the internal process are required because the deletion of a_x will affect the activities that have data dependency on a_x .

6.1.3 Dealing with Service Change: Modify Operation Granularity

Modifying operation granularity in a service has the change impact that c-activities must be replaced in the internal process. As described in the change impact pattern 3 *Replace c-Activities* there are four types of impact caused by this type of service change. We discuss how to handle one sub type of operation granularity change in this sub section. The other sub types of operation granularity change can be similarly handled. Suppose an existing operation o_x is changed to another operation o'_x . Based on the impact pattern, the effect caused by this service change is that an existing c-activity needs to be replaced by another c-activity. Let a_x be the c-activity associated with the operation o_x .

Managing Change at Activity Level

There are three steps to manage this type of change at the activity level.

- The c-activity a_x is replaced by a c-activity a'_x that is associated with the

operation o'_x .

- Data connections relating to a'_x are established. In this step, some existing p-activities and c-activities may be modified. Moreover, new p-activities and c-activities may be created to assist the tasks specified by a'_x .
- The c-activity a'_x and the activities that are created in the second step are inserted into proper positions in the internal process.

Managing Change at Operation Level

We identify the following three cases for dealing with this type of change.

- $o'_x \subset o_x$, which means that the operation o'_x offers part of the functionality of o_x . In this case, we apply an adapter to the operation o_x , which makes the operation o_x behaves like o'_x . The function of this adapter is similar to the adapter introduced for managing the change impact *Insert a c-Activity* (cf. Figure 6.1).
- The above condition does not hold and $o'_x \cap o_x \neq \emptyset$. This relation describes the situation that o'_x includes all the functionality provided by o_x or o'_x contains part of the functionality provided by o_x and has some functionality that is not offered by o_x .

– If there exist a set of operations $\{o_{y1}, \dots, o_{ys}\}$, such that

$$o'_x \subset (o_x \cup o_{y1} \cup \dots \cup o_{ys})$$

In such a case, an abstract operation o'_x is created by adding an adapter to the operations $o_x, o_{y1}, \dots, o_{ys}$. The function of this adapter is similar to the adapter introduced for managing the change impact *Insert a c-Activity* (cf. Figure 6.2).

-
- If the above condition is not satisfied, a new operation o_z is created:
 $o_z = o'_x \setminus o_x$. Similar to the above case, an abstract operation is created by adding an adapter to the operations o_x and o_z .
 - For other cases, a new operation o'_x must be created.

6.1.4 Dealing with Service Change: Reordering Transition Sequences

Reordering transition sequences refers to the defined service transition changes: transition sequence order change (TSOC), sequential to parallel transition sequence change (SPTSC), and parallel to sequential transition sequence change (PSTSC). The impact caused by these types of service transition changes are captured by the change impact pattern 4 *Move c-Activities*. Based on the impact pattern, corresponding c-activities in the internal process need to be reordered. For these types changes, no actions are required at the operation level. In the following, we discuss how to manage these types of transition changes at the activity level.

Managing Change at Activity Level

We provide two alternative approaches for dealing with the types of changes described above at the activity level.

- The first approach is to move the affected c-activities and relevant activities to the required positions directly based on the requirements specified in the impact pattern. The relevant activities here include those activities that depend on the directly affected c-activities in terms of data. We illustrate this approach by using the example shown in Figure 6.3. The c-activity *Send invoice* needs to be moved before the activity *Send bill*. *Send invoice*

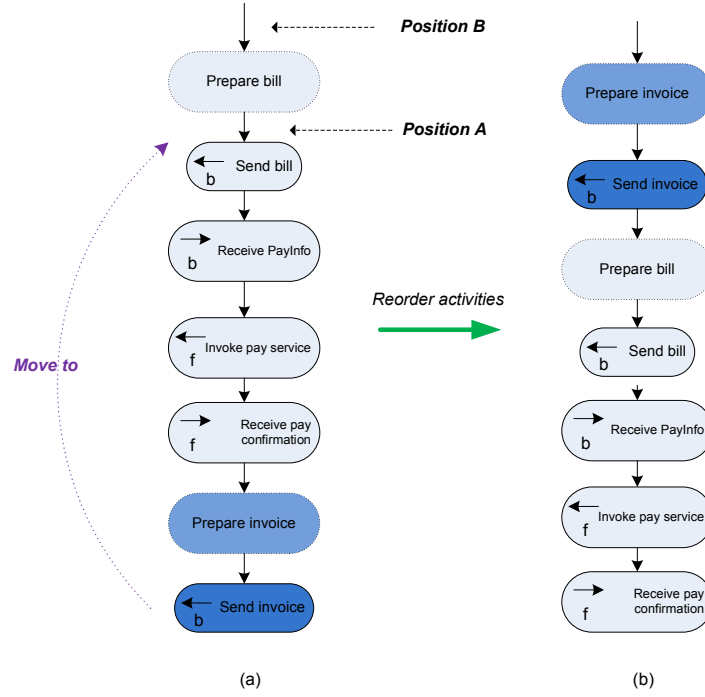
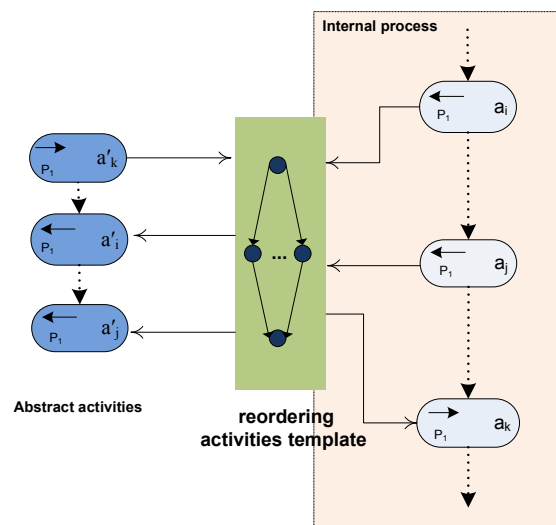


Figure 6.3: Reordering activities.

depends on the p-activity *Prepare invoice* in terms of data. When moving *Send invoice*, *Prepare invoice* must be moved as well to ensure the correctness of the information flow schema. Observe that multiple positions for inserting the activities *Send invoice* and *Prepare invoice* exist as denoted in the Figure 6.3 (a) *position A* and *B*. Figure 6.3 (b) shows the results that the two activities are moved to position B.

- The second approach is to use a *reordering template* for adapting the affected activities to the required control order. Table 6.1 shows the structure of the reordering template. The *Scope* specifies the change region where the adaptation is applied. The *Activities* are the directly affected c-activities that need to be reordered. The *Instructions for reordering* specifies the requirements of the control flow relations between the activities. These requirements are described by the *abstract control relations*. Figure 6.4 gives

Adaptation Template	
Name	Reordering activities
Scope	A set of process elements
Activities	A set of activities that needs to be reordered
Instructions for reordering activities	The abstract control relations between the involved activities

Table 6.1: Structure of reordering template.**Figure 6.4:** Using reordering template.

an example of applying the reordering template to three c-activities a_i , a_j and a_k . In the internal process, the c-activity a_i proceeds a_j which is followed by a_k . Due to a service change, the three c-activities must be changed to a required control sequence: a_k must precede a_i and a_i is followed by a_j . The template mediates the inputs and outputs of these activities and generate the required control sequence. The positions of the three activities remain unchanged in the internal process.

It should be noted that applying reordering templates to the internal process may not be applicable in some cases because of the occurrence of deadlocks. Fig-

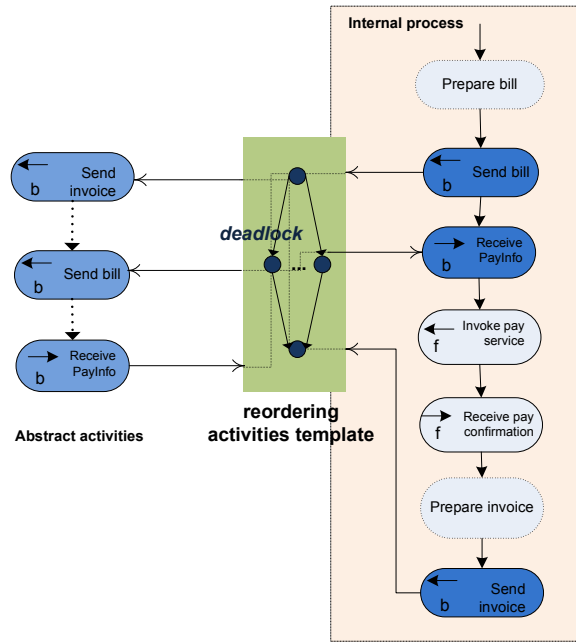


Figure 6.5: An example of deadlock when using reordering template.

Figure 6.5 shows an example where a deadlock occurs when using the reordering template. The deadlock happens when the internal process expects the information from a buyer (*Receive PayInfo*), whereas the buyer is waiting for the message from the internal process (*Send invoice*).

In such circumstances, the affected activities and relevant activities must be directly reordered according to the requirements.

6.1.5 Dealing with Service Change: Modify Conditional and Looping Transition Sequences

Modifying conditional transition sequences refers to the defined service transition changes: adding conditional transition sequence change (ACTSC), removing conditional transition sequence change (RCTSC), adding looping transition sequence change (ALTSC), and removing conditional transition sequence change

(RLTSC). The effect caused by these types of service transition changes are described by the change impact pattern 5 *Add, Remove or Modify Conditional Branches*. Based on the impact pattern, new conditional branches need to be added, existing conditional branches need to be deleted, or existing conditional branches need to be modified. No actions are required at the operation level for handling these types of changes. In the following we discuss the mechanisms for managing these types of changes at the activity level.

Managing Change at Activity Level

When a new conditional branch needs to be added in the internal process, the related activities that must be embedded into the conditional branch are identified. The conditions of the conditional branch are derived from the corresponding transition constraints. In this step, new activities may be created in order to realize the required conditional branch. For instance, c-activities need to be added to acquire necessary information from a particular business partner. The relevant activities are embedded in an *xor* structure accordingly.

When an existing conditional branch needs to be removed, the activities in the affected conditional branch that need to be removed from this conditional branch must be identified. In addition, related data connections associated with the conditional branch need to be removed.

6.1.6 Dealing with Process Changes

In this section, we discuss how to handle different types of changes occurred in internal processes based on the change impact patterns 6-10.

The change effect caused by the process change: *insert an activity* is described in the change impact pattern 6 *Add Operations*. Based on the impact pattern, operations must be added to the corresponding services. In addition, transitions

that are associated with the newly added operations must be established based on the control relations of the corresponding c-activities in the internal process.

The change effect caused by the process change: *remove an activity* is described in the change impact pattern 7 *Delete Operations*. Based on the impact pattern, corresponding operations need to be deleted from services as well as the associated transitions.

The change effect caused by the process change: *replace activities* is described in the change impact pattern 8 *Change Operation Granularity*. Based on the impact pattern, operation granularity must be modified accordingly. Depending on the sub types of this process change, three major types of operation granularity change as: asynchronous operation granularity change, synchronous operation granularity change, and complex operation granularity change will be made.

The change effect caused by the process changes: *move an activity*, *parallelize activities*, and *sequence activities* is described in the change impact pattern 9 *Change Transition Sequence*. Based on the impact pattern, transition sequences in the corresponding services need to be reordered. Depending on the specific process change, three types of transition changes as: transition sequence order change, sequential to parallel transition sequence change, and parallel to sequential transition sequence change will be made during dealing with the process change.

The change effect caused by the process changes: *embed in conditional branches* and *embed in loop* is described in the change impact pattern 10 *Add Conditional or Looping Transition Sequence*. Based on the impact pattern, constraints and transition sequences must be added between operations. Depending on the specific process change, two types of transition changes as adding conditional transition sequence change and adding looping transition sequence change will be

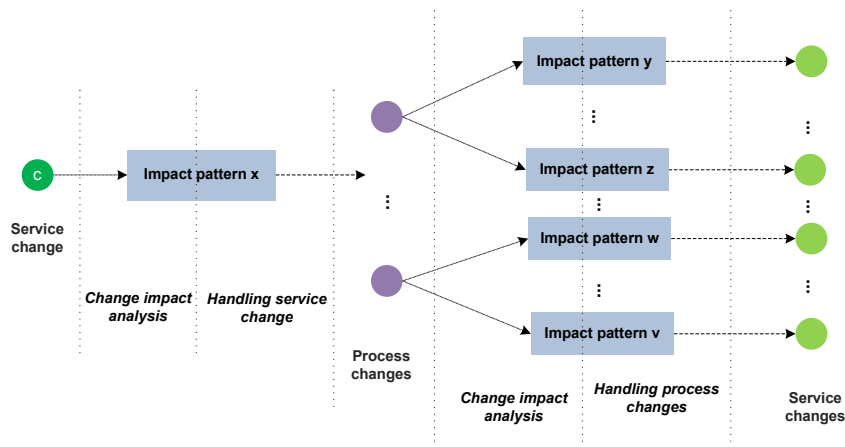


Figure 6.6: Propagation of a service change.

made when dealing with the change.

6.2 Handling Change Propagation

In this section, we provide a brief discussion about how to deal with the change propagation in service-based business processes. The change propagation refers to the process that the occurrence of a change affects the associated services and internal processes. Analysing change propagation is crucial for understanding the actual impact of a specific change and controlling the impact on service-based applications and information systems. Based on the proposed service-oriented business process model, the types of service changes and process changes, the change impact patterns, and the approaches for handling individual change impact patterns, we identify the following two cases of change propagation in a service-oriented business process.

- When a service change occurs, the change impact on the internal process is analysed based on the proposed change impact patterns. This service change is managed by taking actions on the internal process which cause a

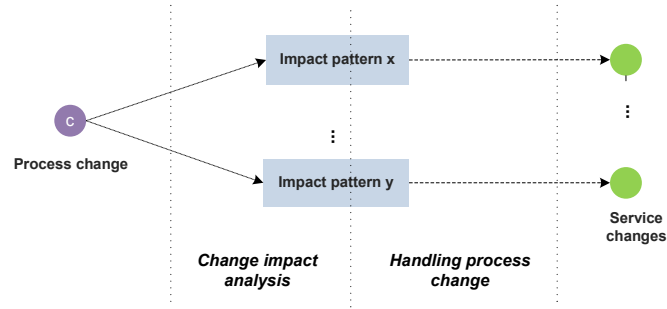


Figure 6.7: Propagation of a process change.

series of process changes. For each of these process changes, the change impact on the associated services is determined by the corresponding change impact pattern. These process changes are then managed by taking some actions on the associated services which incur further changes on the associated services. Figure 6.6 illustrates this change propagation process.

- When a process change occurs, the change impact on the associated services is analysed based on the proposed change impact patterns. This process change may have impact on multiple services. With the change analysis results, some actions are taken to manage this process change which cause a series of changes on the associated services. Figure 6.7 illustrates this change propagation process.

We provide an example as follows to illustrate the propagation of a service change in a service-oriented business process.

Example a service change propagation

As shown in Figure 6.8 (a), a service change: *Adding Conditional Transition Sequences Change* (ACTSC) occurs in the service s_{p_1} . Operation o_2 is changed to be invoked conditionally by adding two conditional transition sequences $t'_1(c_{x1})$ and $t'_3(c_{x2})$. Figure 6.8(b) is the internal process IP that supports the service

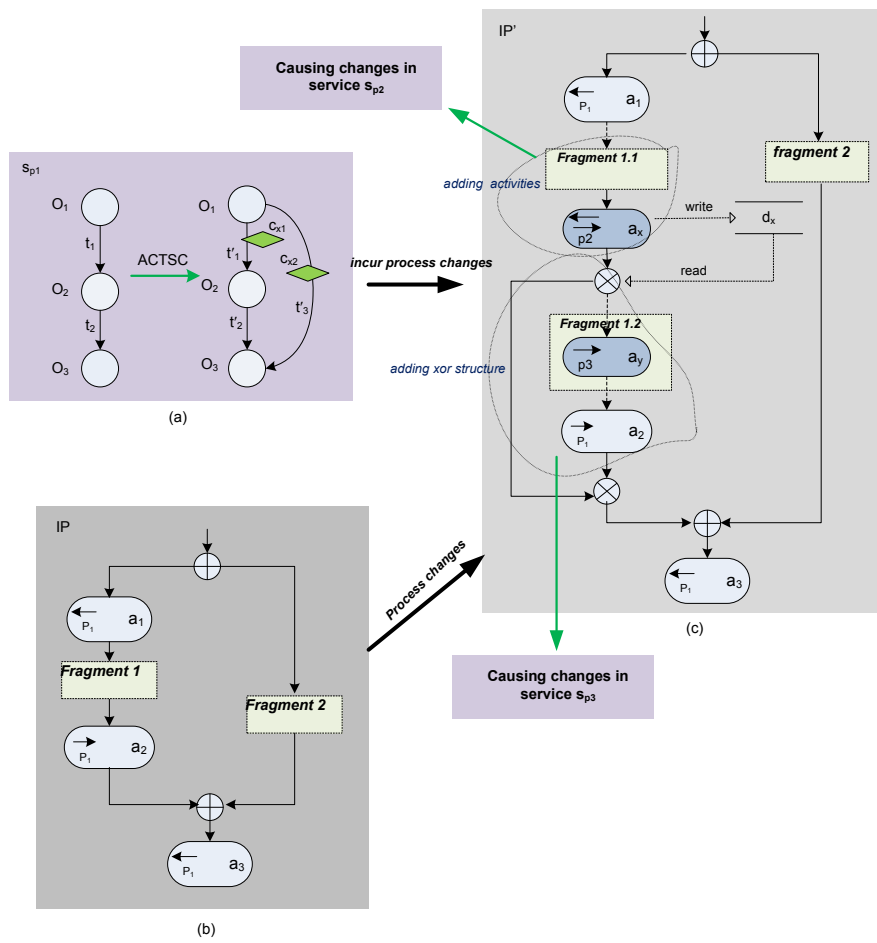


Figure 6.8: An example for propagation of a service change.

s_{p_1} before the change occurs. The c-activities a_1 , a_2 , and a_3 are associated with operations o_1 , o_2 , and o_3 respectively. The *fragment 1* and *fragment 2* are sub processes that contain only p-activities and c-activities that are not related to partner p_1 . The impact on the internal process of this service change is captured by the change impact pattern 5 *Add, Remove or Modify Conditional Branches*. As described in the impact pattern 5, the c-activity a_2 needs to be embedded in a conditional branch. The conditions for controlling the execution of a_2 is derived from constraints c_{x1} and c_{x2} . To handle the service change, an *xor* structure that includes c-activity a_2 is added to the internal process (cf. Figure 6.8 (c)). To support this action, *fragment 1* is split into two parts: *fragment 1.1* and *fragment 1.2*. *Fragment 1.2* is included in the *xor* structure. Suppose *fragment 1.2* contains a *receive* type of c-activity a_y relating to the partner p_3 . In addition, a *send/receive* type of c-activity a_x relating to partner p_2 is inserted before the *xor* structure, which acquires the necessary information used by the *xor* connector. We observe that a series of process changes happen in order to manage the service change ACTSC occurring in service s_{p_1} . Among these process changes, the process change: inserting a c-activity a_x relating to partner p_2 will incur changes in service s_{p_2} . In addition, the process change: embedding the c-activity a_y relating to the partner p_3 in a conditional branch will cause changes in the service s_{p_3} . It is clear that the service change ACTSC in service s_{p_1} is propagated to the services s_{p_2} and s_{p_3} through the internal process.

With the propagation of a service change and a process change, we are able to define the *actual* impact scope of a specific change. As discussed in section 5.2, the *direct* impact scope of a change defines the impact region of this change before any solutions are taken to manage the change. The *actual* impact scope of a change includes the direct impact scope of this change and the impact region of

the further changes caused by actions for handling this change. From the above discussion of the change propagation, it is clear that the actual impact scope of a process change is the direct impact scope of this change (cf. Figure 6.7). In contrast to the actual impact scope of a process change, the actual impact scope of a service change includes:

- the direct impact scope of the service change
- the direct impact scopes of the process changes that are caused by handling the service change.

In order to calculate the actual impact scope of a service change, we define the the function *FuncAISS* as follows.

Definition 1 (*FuncAISS*) Let A be a set of activities and $S = \{s_1, \dots, s_n\}$ be a set of services. *FuncAISS* is the function: $schange, PCHANGE \rightarrow PE, SFS$. The input of the function includes: (i) a service change *schange*, and (ii) a set of process changes $PCHANGE = \{pchange_1, \dots, pchange_r\}$ that are caused by the reactions for handling the service change *schange*. The output of *FuncAISS* includes:

- a set of process elements PE , where

$$PE = FuncDISS(schange)$$

- a list of service fragments $SFS = \{SF_1, \dots, SF_h\}$, where

$$SF_i = FuncDISP(pchange_i)(i = 1, \dots, h)$$

6.3 Change Isolation

In the context of SOC, business processes are usually cross organizational boundaries and involve multiple business partners. In such inter-organizational cooperation, each business partner fulfils parts of the functionalities of the entire value-chain. A single change induced by one business partner may affect multiple partners and this impact may cause further changes to the entire value-chain. Effective mechanisms must be devised to cut off the cascading effect of changes and maintain the stability of the collaboration and cooperation. Our goal of the change management for service-based applications and information systems is to analyse and control cascading effects in services and business processes. A specific change may have cascading effect on services and business processes due to the dependencies between services and business processes. The example *propagation of a service change* provided in the previous section shows how a service change impacts the other two services associated with the same internal process. The objective of this section is to discuss the mechanisms of *change isolation* based on the change taxonomy and the change impact patterns.

Change isolation in the context of service-based business processes refers to restricting and confining a specific change in a minimum scope and cutting off its impact on the related services and business processes. Our research provides foundations for understanding the impact of a specific service change and process change in service-based business processes. When a service change occurs, this change affects the internally supporting internal process. The direct impact on the internal process is captured by the corresponding change impact pattern. The impact analysis results contain the direct impact scopes of the service change that comprises the affected process elements and the change effect on the internal process that is described in terms of *abstract control relations*. Based on the

change impact analysis and the provided mechanisms for handling individual changes, the desired treatment can be taken to cope with this change impact on the internal process. In the following, we discuss how change isolation can be achieved when handling the various types of service changes.

Based on the defined service-oriented business process model and the change impact patterns, we propose the following two criteria for restricting and confining the impact of a service change in order to cut off cascading effects.

- The treatment applied on the internal process does not change the *abstract control relations* of the c-activities related to each of the associated services. This criterion is based on that if the *abstract control relations* of c-activities are maintained unchanged when handling the change, the services associated with the internal process can not be affected.
- The treatment applied on the internal process does not change data connections related to the c-activities of the associated services. This criterion is based on that if the data connections associated with c-activities are maintained unchanged when handling the change, the services associated with the internal process can not be affected.

We provide two examples to show that if the above criteria are violated, changes occurring in one service can not be isolated and will have cascading effects on other services.

The example: *propagation of a service change* (cf. Figure 6.8) in Section 6.2 illustrates the first criterion. To handle the transition change: ACTSC in service s_{p1} , a series of changes are made on the internal process. Among those process changes, adding c-activity a_x that is associated with partner s_{p2} and c-activity a_y that is associated with partner s_{p3} to the internal process affects the

abstract control relations related to service s_{p2} and s_{p3} respectively. Therefore, the two services s_{p2} and s_{p3} must be changed based on those process changes. This example shows that the service change ACTSC in service s_{p1} causes impacts on services s_{p2} and s_{p3} because handling ACTSC affects *abstract control relations* related to those two services.

We use the sales example to illustrate the second criterion. Suppose the shipping service changes by providing a new operation *express shipping* in addition to its existing operation *standard shipping*. To handle this service change, the sales process needs to make a series of changes in order to realise the function of selecting the proper shipping method for customer goods. Suppose the sales process needs to acquire additional information from the buyer about its shipping preference, which requires modifying existing data connections or adding new data connections related to the buyer service. Therefore, the buyer service must be changed based on the changes in the sales process. This example shows that the change occurred in the shipping service causes impact on the buyer service because handling the change occurred in the shipping service affects the data connections related to the buyer service.

6.4 Discussion

In this chapter, we have discussed how to handle changes based on the types of changes and the change impact patterns. We discuss in detail how to deal with the individual changes first and then analyse the change propagation. We also discuss the issue of change isolation in service-based applications and information systems and propose criteria for cutting off the cascading effect.

The problem of change management that has been studied extensively in

the field of workflow systems focuses on handling process changes of individual workflow processes [19, 41, 77, 86]. The change management for service-based applications and information systems is more complex and challenging because of the distributed and dynamic nature of services and business processes. In the SOC environment, it is important to understand the change impact of a specific change, provide effective mechanisms to handle the various types of change impact, and more importantly, to isolate the change and control the cascading effect when complicated dependencies between services and business processes exist. Important guidelines for managing service changes in service-based business processes are proposed in [70], in which a change-oriented life cycle is introduced to handle deep changes. Existing works about service change management address some change issues related to services such as the evolution of service specification [9, 10] and business protocols [83] and some change issues related to BPEL processes such as alignment of process choreography and orchestration [80, 108]. Unfortunately, the dependencies between services and business processes are not discussed in those researches. Moreover, the approaches for analysing and handling the change propagation in service-based business processes have never been proposed by the current research works. Our research provides mechanisms for handling individual change impact patterns and analysing the change propagation in service-based business processes. Our change handling approach is based on the identified change impact patterns. The separation of change impact analysis and change impact handling reduces the complex tasks of change management for service-based applications and information systems. Moreover, the specified change impact patterns and mechanisms for handling individual change impact patterns can be reused in the development and maintenance of service-based applications and information systems.

Chapter 7

Service Change Analyser—A Prototype

In this chapter, we provide the design details of our developed prototype tool referred to as **Service Change Analyser (SCA)**. The SCA is a JAVA based tool that implements the proposed change management mechanisms for analysing the various types of change impacts in service-based business processes. In particular, this tool focuses on a type of dependencies that multiple services are supported by a single business process. The SCA accepts a specific service change as its input and provides the impact analysis results for the change impact scope as well as the potentially used change impact pattern. With the help of the change analyser, the impact of a specific change becomes transparent and it is not necessary to analyse the impact of changes manually. This tool provides developers a standard practice to change the complicated change management tasks into a series of simple standard procedures. The SCA is implemented by the Netbeans IDE 6.9.1 with JDK 1.6.0-22 and MySQL 5.1.

The rest of this chapter is structured as follows. In Section 7.1, we introduce the high level architecture for the SCA. In Section 7.2, we describe the data structures of the SCA. In Section 7.3, we discuss the components of the SCA and discuss the functions of each component. In Section 7.4, we provide two running

examples of our prototype tool. Finally, we conclude this chapter in Section 7.5.

7.1 Architecture

Figure 7.1 shows the architecture of the SCA. The architecture comprises three major procedures as: *Select a service to be changed*, *Choose a service change and specify change*, and *Analyse service change impact*. The *Select a service to be changed* module provides an interface for users to browse the existing services. Users can browse the details of a service including its operations and transitions. The *Choose a service change and specify the change* module can i) provide the types of service changes for users and ii) when a specific type of service change is chosen the corresponding interface is presented for the user to specify the details of the service change. The *Analyse service change impact* module can provide the results of service change impact analysis based on the specified service change. The results of change impact analysis include the direct change impact scope of this service change and the corresponding change impact pattern associated with this service change. The direct impact scope of a service change contains the process elements of the affected business process and is calculated by using Algorithm 1 *FuncDISS* defined in Chapter 5. The associated change impact pattern presented captures the specific type of change effect caused by the service change.

The *Service-based business processes* shown in Figure 7.1 are the databases that store the data of services and business processes. The data of a service includes the operations and transitions associated with these operations. The data of a business process includes the control flow schema and the information flow schema that define this business process. The databases are accessed by the

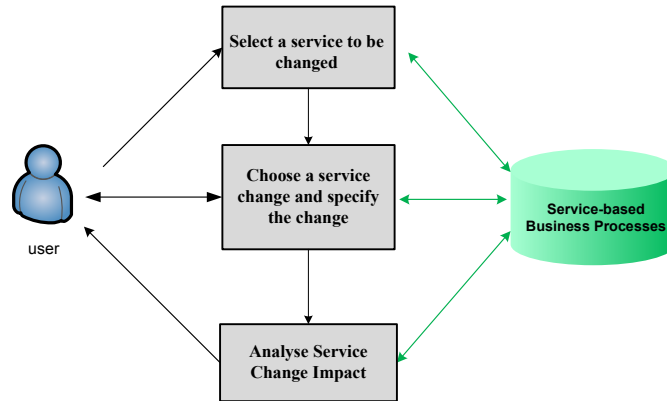


Figure 7.1: High level architecture for SCA.

above mentioned three modules of the SCA during the execution of this tool.

We summarize the features of the SCA as follows:

- The SCA enables the change management for service-based business processes where services and business processes have complex dependence relations. In particular, the developed prototype tool focuses on a type of dependencies that multiple services are supported by a single business process.
- The SCA is developed based on our change management approach including the change taxonomy and the change impact patterns. These change types and change impact patterns can be reused and extended for the development and maintenance of service-based applications and information systems.
- The SCA provides rich results of change impact analysis including direct impact scopes and change effect on associated business processes. These analysis results for a specific service change are helpful to analyse the change

impact in service-oriented business processes and provide the foundation for determining proper change treatment.

7.2 Data Structure

In this section, we describe the development of our databases: service-based business processes. We use MySQL 5.1 as the database server. We design and develop the databases of service-based business processes by using the visual tool: MySQL Workbench 5.2 CE.

Based on the definitions for service-oriented business processes in Chapter 3, we have designed nine tables as *processes*, *business partners*, *activities*, *control connectors*, *controlflowschemas*, *dataelements*, *informationflowschemas*, *operations*, and *servicetransitions* to store the data of service-based business processes (cf. Figure 7.2). We discuss the design details of these tables and their relations as follows. We will use the example provided in Chapter 3 to illustrate these tables.

The *processes* table has two columns as: *process_id* and *process_name*. This table is used to store the existing business processes. For our example, the business process: sales process needs to be stored in this table.

The *business partners* table has three columns as: *partner_id*, *partner_name*, and *associate_process* that stores the business partners associated with business processes. For our example, three business partners are stored in this table as: buyer, payment, and shipper.

The *activities* table has seven columns as: *activity_id*, *activity_name*, *activity_type*, *partner*, *InPARs*, *OutPARs*, and *associate_operation*. The column of *activity_type* contains the type of an activity. Based on the definitions in Chapter

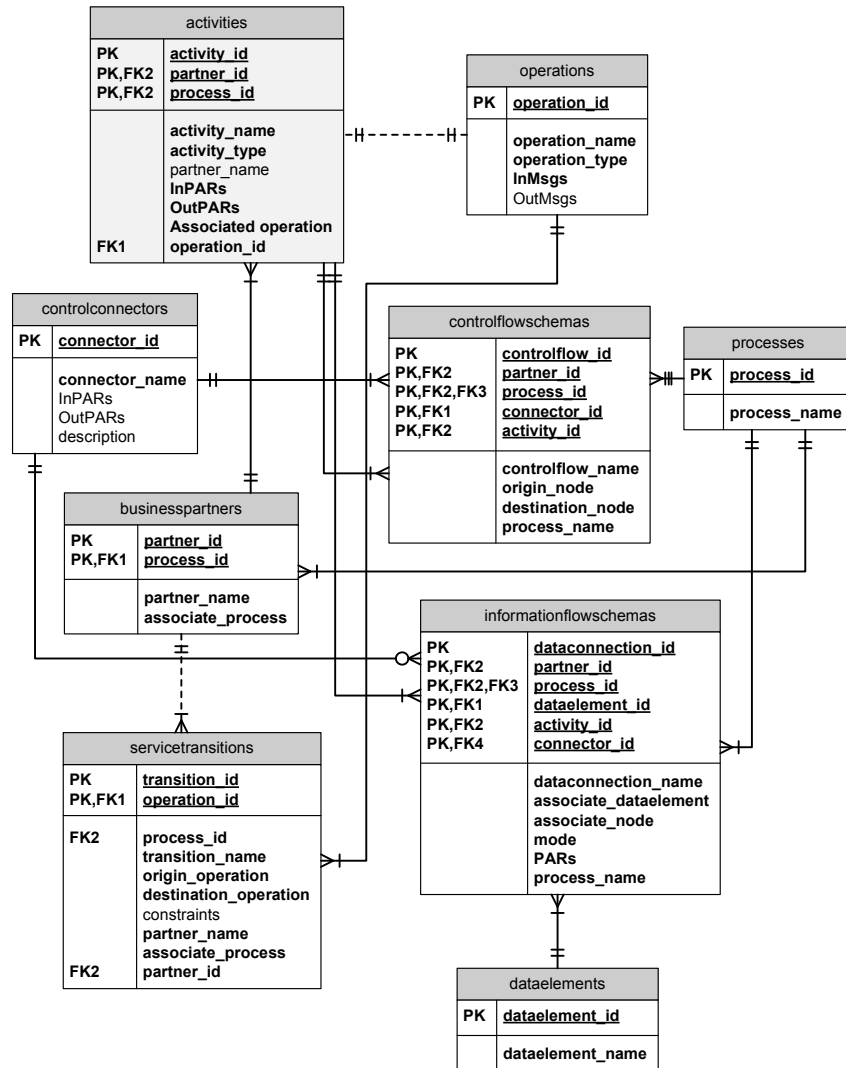


Figure 7.2: Entity relationship diagram.

3, activities of a business process are categorized into *private* activity (p-activity) and *communication* activity (c-activity). C-activities are further classified into *Send* activity, *Receive* activity, *Send/Receive* activity, and *Receive/Send* activity. The column of *partner* stores the corresponding business partner that a c-activity intends to interact with. For a p-activity, the content for this column is *null*. The columns of *InPARs* and *OutPARs* store the input and output parameters of an activity respectively. The column of *associate_operation* stores the operation associated with a specific activity. For example, the activity *Get shipping quote* is saved as:

(A13, *Getshippingquote*, *Send/Receive*, *shipper*, *shippingInfo*,
shippingquote, *getshippingquote*)

The *controlconnectors* table has five columns as: *connector_id*, *connector_name*, *InPARs*, *OutPARs*, and *description*.

The *controlflowschemas* table consists of five columns as: *controlflow_id*, *controlflow_name*, *origin_node*, *destination_node*, and *associate_process*. The *controlflowschemas* table stores the control relations between activities. The columns of *origin_node* and the *destination_node* record the start node (an activity or a control connector) and the end node (an activity or a control connector) of the corresponding control relation respectively.

The *dataelements* table has two columns as: *dataelement_id* and *dataelement_name*. This table stores the data elements that are read or written by process activities.

The *informationflowschemas* table has seven columns as: *dataconnection_id*, *dataconnection_name*, *associate_dataelement*, *associate_activity*, *PARs*, *mode*,

associate_process. This table stores the information flow schemas of business processes which are defined in Chapter 3(cf. 3.2.1). Each row of this table records a data connection.

The *operations* table has five columns as: *operation_id*, *operation_name*, *operation_type*, *InMsgs*, and *OutMsgs*. This table stores information about an operation. For example, the operation *get shipping quote* is stored as:

(O13, *getshippingquote*, *Synchronous*, *shippingInfo*, *shippingquote*)

The *servicetransitions* table has six columns as: *transition_id*, *transition_name*, *origin_operation*, *destination_operation*, *constraints*, and *partner*. This table stores the transitions of services. Each row in this table records a specific service transition.

Figure 7.2 presents the entity relationship diagram which defines the structures of tables and the relationships between these tables. For instance, the relation between the activities table and the operations table is that each activity is associated with an operation. The relation between the activities table and the businesspartners table is that each c-activity is associated with a specific business partner and each business partner may relate to more than one c-activities. P-activities of business processes are not associated with any business partners. For example, the c-activity Receive order in the sales process is associated with the buyer partner. The p-activity Check stock availability in the sales process is not associated with any business partner. The relation between the activities table and the controlflowschemas table is that each control relation involves at most two activities and one activity may be associated with more than one control relations. That is, based on our definition of control flow schema of business

processes, a control relation has an origin node and a destination node. A node refers to an activity or a control connector. Similarly, the relation between the controlconnectors table and the controlflowschemas table is that each control relation involves at most two control connectors and one control connector may be associated with one or more control relations.

7.3 Components of Service Change Analyser

In this section, we discuss the components of the SCA. As shown in Figure 7.3, the function of the SCA: *Service Change Analysis* is divided into two broad modules as *Operation based Analysis* and *Transition based Analysis*. We discuss the two functional modules in the following sub sections respectively.

7.3.1 Operation Based Analysis

As shown in Figure 7.3, the operation based analysis module is realized by two major functional modules: change operation existence and change operation granularity.

The change operation existence module is further divided into three sub modules as: sequentially add an operation, add an operation in parallel to an existing operation, and delete an operation. The sequentially add an operation module accepts a service change with the type of sequentially adding an operation and generates the impact analysis results of this service change. The add an operation in parallel to an existing operation module accepts a service change with the type of parallel adding an operation and generates the impact analysis results of this service change. The delete an operation module accepts a service change with the type of deleting an operation and generates the impact analysis results

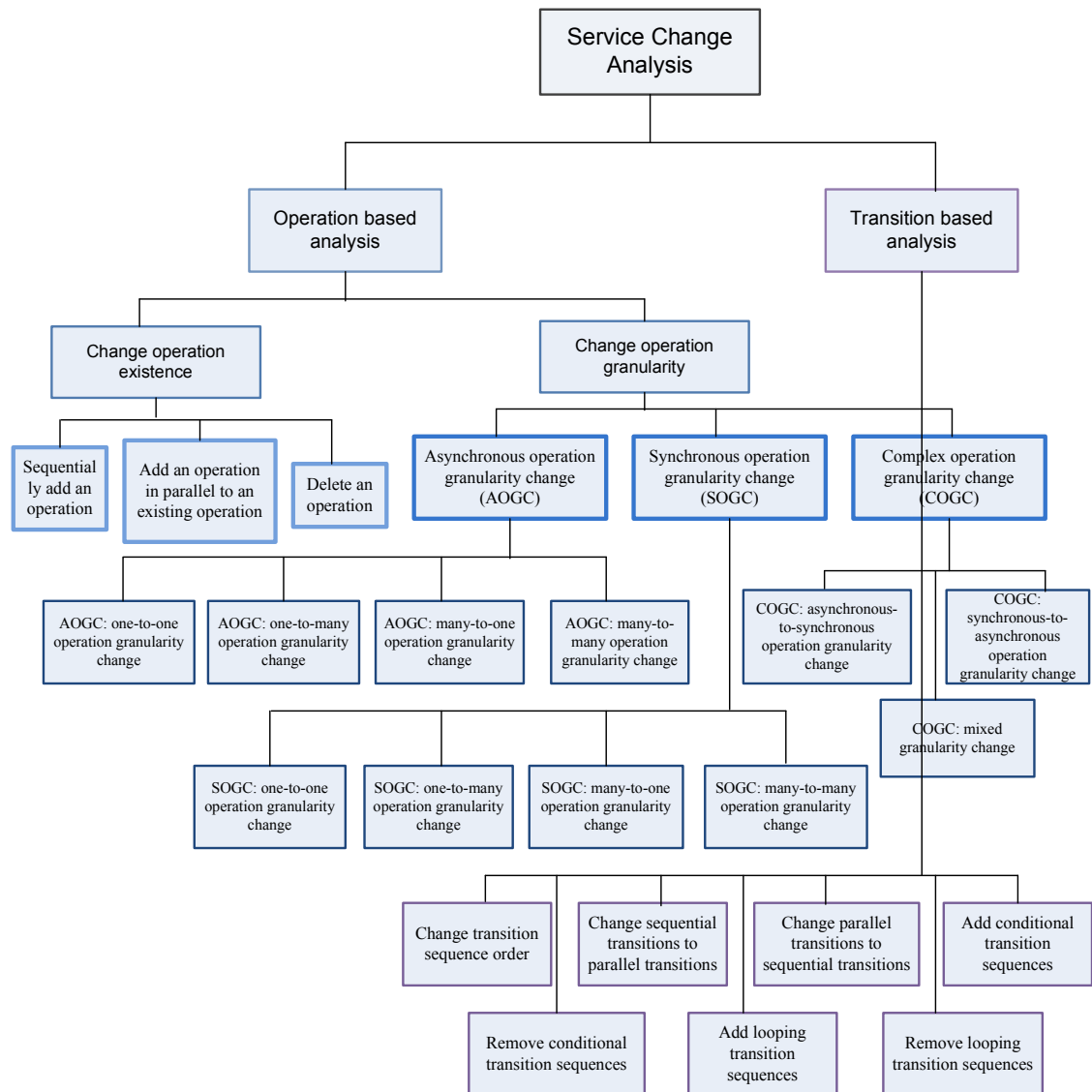


Figure 7.3: Hierarchy diagram.

of this change.

The change operation granularity module consists of three major sub modules as asynchronous operation granularity change (AOGC), synchronous operation granularity (SOGC), and complex operation granularity change (COGC).

The AOGC module is further divided into four modules as: AOGC one-to-one operation granularity change, AOGC one-to-many operation granularity change, AOGC many-to-one operation granularity change, and AOGC many-to-many operation granularity change. These modules deal with the impact analysis of asynchronous operation granularity changes.

The SOGC module is further divided into four modules as: SOGC one-to-one operation granularity change, SOGC one-to-many operation granularity change, SOGC many-to-one operation granularity change, and SOGC many-to-many operation granularity change. These modules deal with the impact analysis of synchronous operation granularity changes.

The COGC module is further divided into three modules as: COGC asynchronous -to-synchronous operation granularity change, COGC synchronous-to-asynchronous operation granularity change, and COGC mixed operation granularity change. These modules deal with the impact analysis of operation granularity changes involving both asynchronous operations and synchronous operations.

7.3.2 Transition Based Analysis

As shown in Figure 7.3, the transition based analysis is divided into seven sub modules as: change transition sequence order, change sequential transitions to parallel transitions, change parallel transitions to sequential transitions, add conditional transition sequences, remove conditional transition sequences, add looping transition sequences, and removed looping transition sequences. These seven

modules accept the corresponding service transition changes and provide the change impact analysis results. For instance, the change transition sequence order module accepts a service change with the type of transition sequence order change and generates the impact analysis for this service change.

7.4 Running Examples

In this section, we provide two running examples to show the effectiveness of our change analyser. The two examples cover the operation based analysis and transition based analysis respectively.

7.4.1 Example for Operation Based Analysis

In this sub section, we present an example for operation based change impact analysis. We will show how a user can specify a type of operation change using the SCA and what is the actual output.

First, a user needs to select the service that he/she wants to change. The SCA provides an interface for users to browse existing services. As shown in Figure 7.4, the interface contains a service list and a tabbed panel: Operation and Transition. The service list shows the existing services that are retrieved from the databases of service-based business processes. In our example, three services as: buyer service, payment service, and shipper service exist and are listed. When a service is selected, its operations and transitions will be displayed in the tabbed panel below as trees. In this example, the buyer service is selected and its operations are shown as a tree (cf. Figure 7.4). The root node is the selected service and the operations are displayed as children nodes. Each operation node has three children nodes as operation type, input messages, and output messages. As shown in Figure

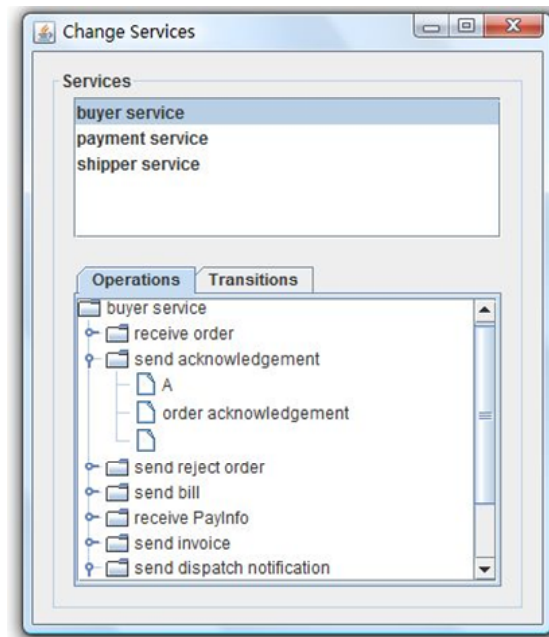


Figure 7.4: Browse service operations.

7.4, the operation send acknowledgement is an asynchronous operation denoted as “A”. The input messages of this operation is “order acknowledgement”.

When a user wants to change the selected service, he/she can right click the mouse in the area of panels showing service operations and transitions. When the user right clicks the mouse in the service operation area, the defined types of service change related to operations are popped out as menus (cf. Figure 7.5).

In Chapter 4, we have presented the taxonomy of changes for services (cf. Figure 4.1). Based on this service change classification, the change types associated with service operations provided for users are: *Sequentially add an operation*, *Add an operation in parallel to an existing operation*, *Delete an operation*, *Change granularity of asynchronous operation*, *Change granularity of synchronous operation*, and *Complex operation granularity change*. Note that the *Change granularity of asynchronous operation* menu has four submenus as:

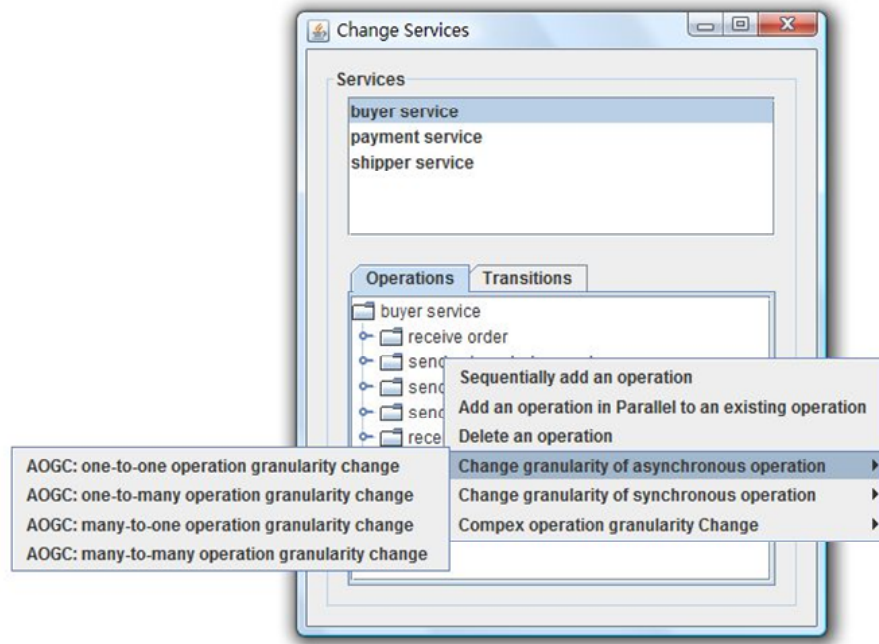


Figure 7.5: Choose change types of service operations.

AOGC one-to-one granularity change, AOGC one-to-many granularity change, AOGC many-to-one granularity change, and AOGC many-to-many granularity change. Similarly, the *Change granularity of synchronous operation* menu also has four submenus as: SOGC one-to-one granularity change, SOGC one-to-many granularity change, SOGC many-to-one granularity change, and SOGC many-to-many granularity change. For the *Complex operation granularity change*, three submenus are provided as: COGC asynchronous-to-synchronous operation granularity change, synchronous-to-asynchronous operation granularity change, and mixed operation granularity change.

The user can choose a change type he/she wants to apply on the selected service. When a specific change type is chosen, the SCA will provide the corresponding interface for the user to specify the change. In this example, the *Add an operation in parallel to an existing operation* change is selected and the

Figure 7.6: Add an operation in parallel to an existing operation.

corresponding interface is presented (cf. Figure 7.6).

Suppose a user wants to add an operation *send dispatch notification* to the buyer service. The new operation must be invoked after the operation *send acknowledgement* and before the *receive PayInfo*. In addition, the new operation can be executed in parallel with the operation *send bill*. The constraints for executing the new operation *send dispatch notification* is “order is confirmed and the goods are dispatched”. The operation *send dispatch notification* is an asynchronous operation and its input messages include “customer order” and “dispatch notice from shipper”. To make the above operation change, the user needs to choose the item *Add an operation in parallel to an existing operation* of the popped up menu. Then the SCA will provide an interface for the user to specify this service operation change (cf. Figure 7.6). There are two parts

of information that need to be specified by the user: indicating where the new operation needs to be inserted and specifying the details of the new operation. As shown in Figure 7.6, the user can select the origin operation and destination operation from the drop-down lists. In addition, the user must specify which operation the new operation can be executed in parallel. If the invocation of this new operation is conditional, the user can specify the conditions in the constraints textfield. The name, type, input and output messages of the new operation need to be specified in the corresponding textfields. Figure 7.6 shows the above mentioned service change.

After the user input the information of a specific change, he/she can click the “Analyse Change Impact” button at the bottom of the frame. The function of analysing service change impact is based on the proposed mechanisms for change impact analysis in Chapter 5. The results of the change impact analysis of the SCA include the direct impact scope of the input service change, the associated change impact pattern, and a description for the change effect based on the specified change information. The direct impact scope of a service change is defined as a set of process elements and is calculated using Algorithm 1 *FuncDISS* (cf. Chapter 5, Section 5.2.1). For the specified service change, the expected output of the SCA consists of the change impact scope and the associated change impact pattern. The affected business process is the sales process. The affected activities include:

- The activities that associated with the operations involved in the service change: *Send acknowledgment, Receive PayInfo, Send bill.*
- The activities that depend on the activities in (i): *Invoke pay service.*
- The activities that are depended by the activities in (i): *Receive order,*

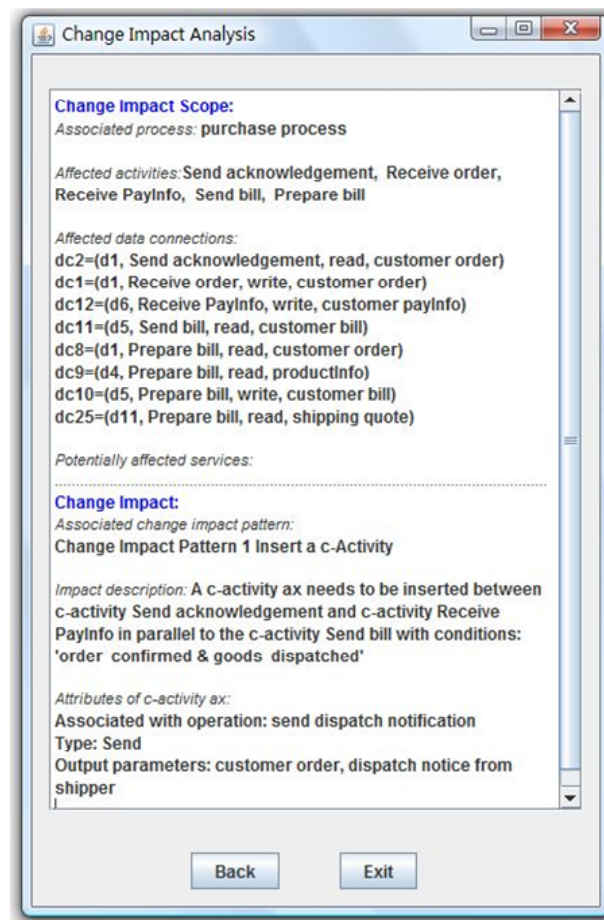


Figure 7.7: Impact analysis for the change of adding operation in parallel to an existing operation.

Prepare bill.

The affected data connections are those data connections associated with the activities specified in the above affected activities. The affected service is “payment” service. The change impact associated with the specified service is *Change Impact Pattern 1 Insert a c-Activity*.

Figure 7.7 shows the actual output of our tool: the results of the change impact analysis for the specified change “Add an operation in parallel to an existing operation”. The change impact pattern that captures this type of change

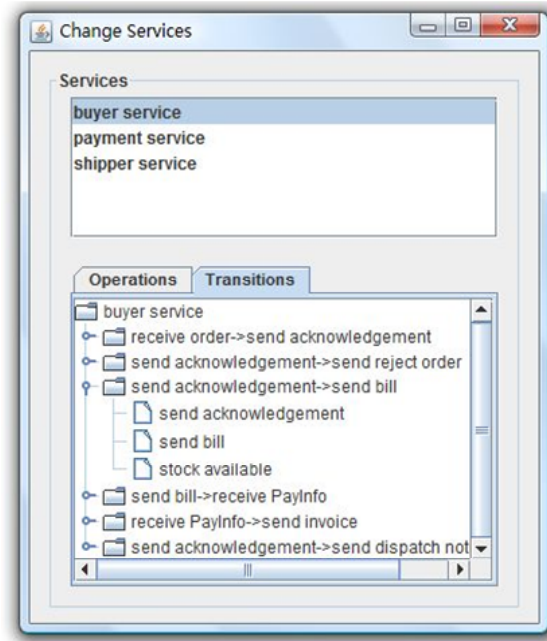


Figure 7.8: Browse service transitions.

effect is *Change Impact Pattern 1 Insert a c-Activity*. The impact description provided by the SCA indicates that a new c-activity must be inserted to the sales process between the activity *Send acknowledgement* and *Receive PayInfo* and in parallel to the c-activity *Send bill*. The new c-activity is conditionally executed. The details of the c-activity that needs to be inserted to the sales process are also provided based on the information about the new operation specified by the user.

7.4.2 Example for Transition Based Analysis

In this sub section, we provide an example for transition based change impact analysis. We will show how a user can specify a type of transition change using the SCA and what is the actual output.

Transitions of a service are displayed as a tree. As shown in Figure 7.8, the

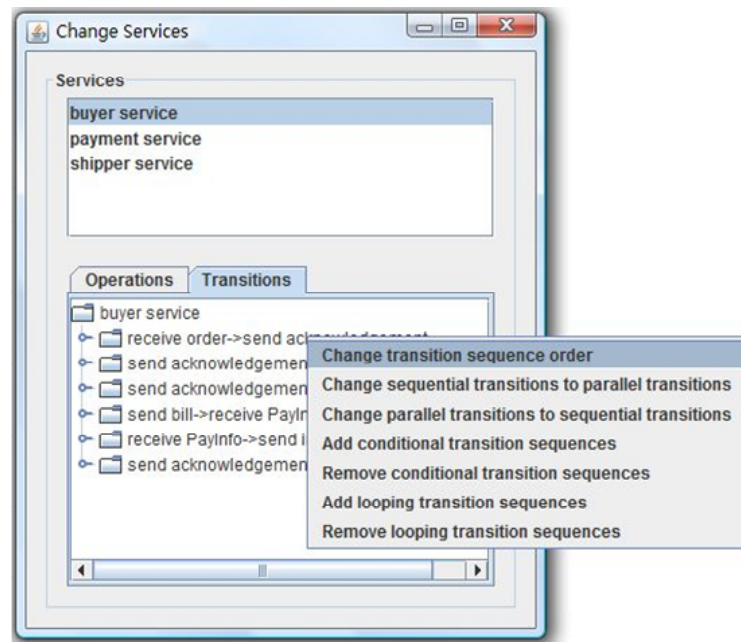


Figure 7.9: Choose change types of service transitions.

buyer service has six transitions. The root node of a transition tree is the selected service and the children nodes are its transitions. Each transition node has three children nodes as origin operation, destination operation, and constraints.

When the user wants to make a change of service transitions, he/she can right click the mouse in the area of service transitions. Figure 7.9 shows the popped out menu of transition change types. Based on the taxonomy of service changes, we have designed seven menus items as: *Change transition sequence order*, *Change sequential transitions to parallel transitions*, *Change parallel transitions to sequential transitions*, *Add conditional transition sequences*, *Remove conditional transition sequences*, *Add looping transition sequences*, and *Remove looping transition sequences*.

Suppose a user wants to change the order of the transition sequence of the buyer service: (*send bill*, *receive PayInfo*, *send invoice*) to (*send invoice*, *send*

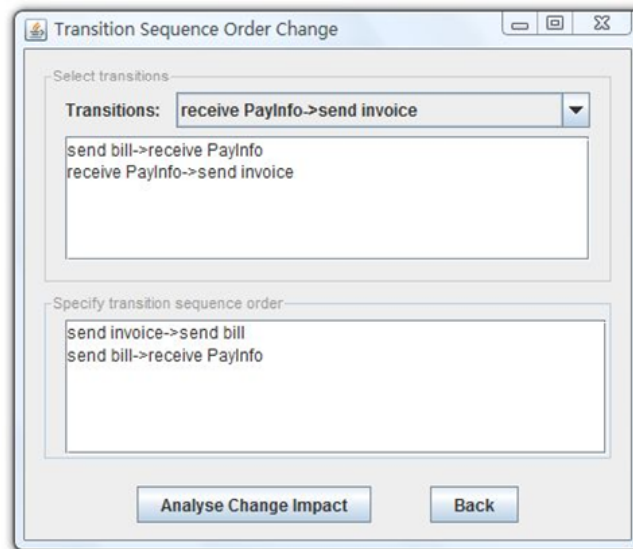


Figure 7.10: Change transition sequence order.

bill, *receive PayInfo*). To make this transition change, the user needs to choose the item *Change transition sequence order* of the popup menu. Then the SCA will provide an interface for the user to specify this type of transition change (cf. Figure 7.10). The user can select the transition sequence he/she wants to change from the drop-down list. The selected transitions will be displayed in the below testarea. The user must specify the new order of the selected transition sequence in the corresponding textarea. Figure 7.10 shows the above described transition sequence order change.

The expected output of our change analyser consists of the change impact scope and the associated change impact pattern. The change impact scope includes the affected activities and data connections in the internal process and the services associated with the internal process. The affected activities include:

-
- The activities that associated with the operations involved in the service change: *Send bill, Receive PayInfo, Send invoice*.
 - The activities that depend on the activities in (i): *Invoice pay service*.
 - The activities that are depended by the activities in (i): *Prepare bill, Prepare invoice*.

The affected data connections are those data connections associated with the activities specified as above. The affected service is “payment” service. The change impact associated with the specified service is *Change Impact Pattern 4 Move c-Activities*.

Figure 7.11 shows the results of the change impact analysis for the specified change “Change transition sequence order” in the Step 2. The affected business process is the sales process. The change impact pattern that captures this type of change effect is *Change Impact Pattern 4 Move c-Activities*. The impact description provided the SCA indicates that the c-activities *Send bill, Receive PayInfo*, and *Send invoice* must be reordered. The *abstract control relations* of the affected c-activities are decided from the transition sequence of operations specified by the user.

7.5 Discussion

In this chapter, we have presented a prototype tool referred to as Service Change Analyser (SCA) that is developed to validate the concepts and mechanisms of service change management proposed in this thesis. The SCA accepts changes as its input and it can give the detailed analysed results for the change impact scope and provide suggestions for potentially used change impact patterns. With the

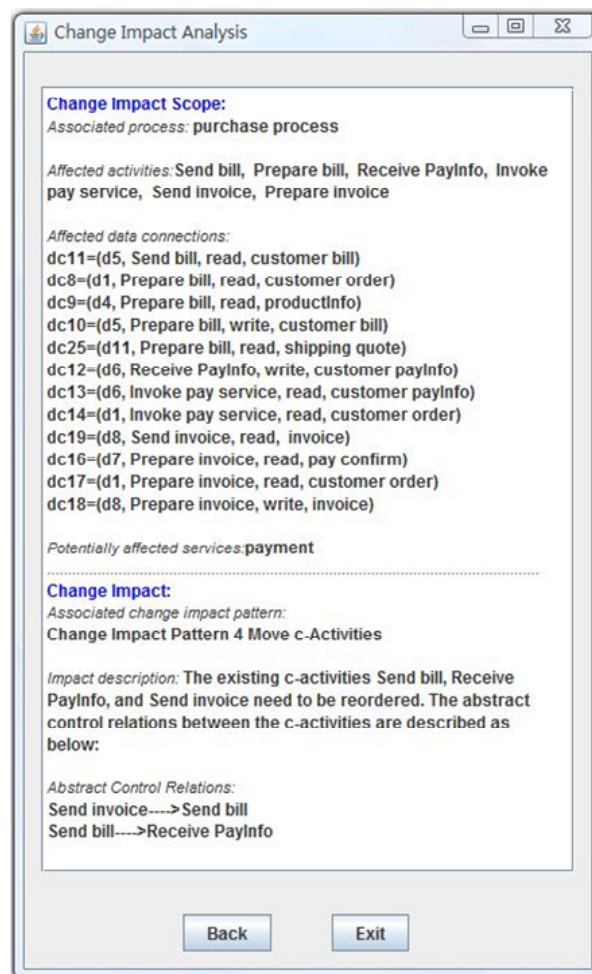


Figure 7.11: Impact analysis for change transition sequence order.

help of the SCA, the impact of a specific change becomes transparent and it is not necessary to analyse the impact of changes manually. Even as a prototype, the current change analyser has already have user-friendly graphic user interfaces.

The prototype is implemented with IDE NetBeans 6.9.1 with JDK 1.6.0-22 for Windows XP and MySQL 5.1 and MySQL Workbench 5.2 CE. The major functionalities of the SCA are realized by two major modules as: operation based analysis and transition based analysis. We have discussed the design details of this tool including the data structures and the components of this tool. Two running examples are presented which show how to analyse impact of operation changes and transition changes respectively.

The SCA supports the change management for service-based business processes and has the following features. First, the SCA advocates reusability in the change analysis and change reaction processes because the change taxonomy associated with services and business processes and the change impact patterns can be reused during managing changes for service-based business processes. Second, the SCA provides rich change impact analysis information. These intermediate results are helpful to reduce the complex change management tasks. In addition, they can help users make proper decisions in determining change reactions.

Chapter 8

Conclusions and Future Work

In this chapter, we summarize the contribution of this thesis and discuss some future research directions on the foundation of this work.

8.1 Concluding Remarks

The service-oriented computing (SOC) offers a promising computing paradigm for realising seamless integration of applications and information systems across organization boundaries. Service-based applications and information systems need to operate correctly in a fast changing environment due to the dynamic and distributed features of services. In this thesis, we have presented an approach for dealing with the challenging issue of change management for service-based business processes. Current works on service change management provide solutions for handling changes of business processes or services separately. Our research shows that complicated dependencies between services and business processes exist and these dependence relations are crucial for the change management in service-based applications and information systems. The proposed approach for change management focuses on the dependencies between business processes and services. A type of dependencies that one business process supports multiple services is highlighted in this thesis. We summarize the contribution of our research

reported in this thesis as follows:

- We have defined a service-oriented business process model for capturing the major characteristics of the change management issues in the service-oriented environment. The proposed model describes dependencies between services and business processes that multiple services are supported by a single business process. Two layers as the process layer and the service layer are defined in the service-oriented business process model. The process layer contains internal processes which are invisible for business partners. The service layer consists of services that are supported by the internal processes. Functionalities of internal processes are represented and exposed as services. Business partners interact with internal processes by invoking the corresponding services. The proposed model provides the foundation for building up the change taxonomy associated with services and business processes, change impact analysis, and reaction to various types of changes.
- We have identified the various types of changes associated with services and business processes on the basis of the service-oriented business process model. Service changes are categorized into two major types as the operation change and the transition change. Based on the definition of services, operation changes are further classified into the operation existence change and the operation granularity change. The operation existence change identifies the various ways of inserting an operation to a service. The operation granularity change discusses grain changes related to operations including asynchronous operation granularity change, synchronous operation granularity change, and complex operation granularity change that involves both asynchronous and synchronous operations. The transition change describes variation related to transitions associated with operations. Process changes

are discussed based on the control flow schemas of internal processes. We have identified nine major types of process changes as insert an activity, remove an activity, move an activity, replace activities, parallelize activities, sequence activities, embed in conditional branches, embed in loop, and update conditions. The presented taxonomy of changes related to services and business processes provide a solid foundation for analysing the impact caused by service changes and process changes on service-based business processes and developing generic change reactions for controlling these changes.

- We have presented the ten change impact patterns to support the change impact analysis in service-based business processes. The change impact patterns are categorized into two major types: the impact patterns related to service changes and the impact patterns related to process changes. Each change impact pattern describes the direct impact scope of a change, the cause of the change, and the effect of the change on the services and the associated business process. The change impact patterns provide intermediate results in the analysis process and they can be reused in the development and maintenance of service-based information systems. The separation of change impact analysis and change reactions by using the change impact patterns is helpful to reduce the complexity of change management tasks.
- We have provided mechanisms for handling various types of changes based on the specified change impact patterns. With the help of the change impact patterns and the mechanisms for dealing with individual changes, we are capable of analysing the change propagation in service-based business processes. The actual impact scopes of a specific service change and a

process change are calculated on the basis of the analysis of change propagation. We have also discussed the important issue of change isolation and provided principles for cutting off the change propagation in service-based applications and information systems.

- We have developed a prototype tool referred to as service change analyser that implements the proposed change management mechanisms. This prototype application shows the effectiveness of the proposed general methodology of the change management in the service-oriented environment. With the help of the SCA, the impact of a specific change becomes transparent and it is not necessary to analyse the impact of changes manually. The time and cost of change management tasks can be dramatically reduced. Even as a prototype, the current change analyser has already have user-friendly graphic user interfaces.

To summarize, the research work reported in this thesis provided our approach for dealing with the change management for service-based business processes. We have defined a service-oriented business process model that captures a type of dependencies between services and business processes where multiple services are supported by a single business process. Based on the proposed service-oriented business process model, we have identified a taxonomy of changes associated with services and business processes. We have presented the ten change impact patterns to support the change impact analysis. We have shown how to analyse the change propagation and control the cascading effect of changes in service-based business processes. Finally, we have built up a prototype that implements our proposed change management mechanisms.

8.2 Future Directions

Change management is critical and challenging in the service-oriented environment. Existing researches feature only some aspects of change management issues and provide partial solutions for dealing with changes in service-based applications and information systems. This thesis reports the first stage of our research about the change management of service-based business processes. In the following, we discuss our future research directions in relation to service change management.

- As we have stated in this thesis, complex dependence relations exist between services and business processes and they are crucial for the development of effective mechanisms for handling various types of changes in service-based systems. Our research reported in this thesis has highlighted the typical case that one business process supports multiple services. We plan to identify more typical types of dependencies between business processes and services and develop the corresponding change management mechanisms.
- The SOC paradigm aims to enable inter-organization cooperation and collaboration by dynamically integrating business processes belonging to different business partners. An organization relies on its partners to fulfil business tasks. For each of the partners, the internal process and associated services are subject to changes from time to time. Business processes belonging to different partners that are dynamically orchestrated to accomplish a goal need to operate correctly when the participant processes and services change due to various factors. In the context of multi-party cooperation and collaboration, the change management becomes more critical and challenging. Our research reported in this thesis provides solutions for man-

aging changes associated with services and business processes within one organization. The presented mechanisms support identifying, analysing, and reacting to the various types of changes related to services and business processes that occur in one organization. This research provides a solid foundation for studying the challenging issue of change management in the cooperation and collaboration context. Current researches on the change management in this context is very limited. We intend to tackle the change management for inter-organization cooperation and collaboration from the following aspects:

- Providing approaches for analysing the impact of changes that affects participant processes and services. A change arising from one party has different impact on the business processes and services of its partners. Understanding the various types of change impact is the first step to develop effective change reactions.
- Devising mechanisms for controlling changes. Changes must be isolated and controlled with the aim to reduce their impact on the entire system to a minimum. The cascading effect of changes must be cut off.
- Developing strategies for change reactions. When a change is introduced by a partner, the business partners involved in the cooperation and collaboration must take proper change reactions in order to ensure the successful execution of the cooperation and collaboration. The reaction strategies need to support multi-party negotiation when a change incurred by one partner affects the related business processes and services of other partners.

Bibliography

- [1] Web Service Description Language (WSDL) 1.1. <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>, 2001.
- [2] Business Process Execution Language for Web Services Version 1.1. <http://www.ibm.com/developerworks/library/ws-bpel/>, 2003.
- [3] Universal Description, Discovery and Integration (UDDI), Version 3, 2004.
- [4] Simple Object Access Protocol (SOAP) Version 1.2, 2007.
- [5] M. Adams, A. H. M. ter Hofstede, D. Edmond, and W. M. P. van der Aalst. Worklets: A service-oriented implementation of dynamic flexibility in workflows. In *Proceedings of OTM Conferences (1)*, pages 291–308, Montpellier, France, October 29 - November 3 2006.
- [6] V. Agarwal and P. Jalote. From specification to adaptation: An integrated QoS-driven approach for dynamic adaptation of web service compositions. In *Proceedings of the 2010 IEEE International Conference on Web Services (ICWS 2010)*, pages 275–282, Miami, Florida, USA, July 5-10 2010.
- [7] M. S. Akram and A. Bouguettaya. Managing changes to virtual enterprises on the semantic web. In *Proceedings of the 5th International Conference on Web Information Systems Engineering (WISE)*, pages 472–478, Brisbane, Australia, 2004.
- [8] M. S. Akram, B. Medjahed, and A. Bouguettaya. Supporting dynamic changes in web service environment. In *proceedings of 1st International*

- Conference on Service Oriented Computing*, pages 319–334, Trento, Italy, 2003.
- [9] V. Andrikopoulos, S. Benbernou, and M. P. Papazoglou. Managing the evolution of service specifications. In *Proceedings of the 19th International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 359–374, Montpellier, France, 2008.
- [10] V. Andrikopoulos, S. Benbernou, and M. P. Papazoglou. Evolving service from a contractual perspective. In *Proceedings of the 20th International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 290–304, Amsterdam, the Netherlands, 2009.
- [11] J. Banerjee, W. Kim, H.-J. Kim, and H. F. Korth. Semantics and implementation of schema evolution in object-oriented databases. In *Proceedings of the 1987 Annual Conference on Association for Computing Machinery Special Interest Group on Management of Data*, pages 311–322, San Francisco, California, May 27-29 1987.
- [12] B. Benatallah, F. Casati, D. Grigori, H. R. Nezhad, and F. Toumani. Developing adapters for web services integration. In *Proceedings of the 17th International Conference on Advance Information Systems engineering (CAiSE)*, pages 415–429, Porto, Portugal, 2005.
- [13] B. Benatallah, F. Casati, and F. Toumani. Representing, analysing and managing web service protocols. *Data & Knowledge Engineering*, 58:327–357, 2006.
- [14] B. Benatallah, B. Medjahed, A. Bouguettaya, A. K. Elmagarmid, and J. Beard. Composing and maintaining web-based virtual enterprises. In

-
- First VLDB Workshop on Technologies for E-Services*, pages 155–174, 2000.
- [15] D. Berardi, D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Mecella. Automatic service composition based on behavioral descriptions. *Int. J. Cooperative Inf. Syst.*, 14(4):333–376, 2005.
- [16] L. Bordeaux, G. Salaün, D. Berardi, and M. Mecella. When are two web services compatible? In *Proceedings of the 5th International Workshop on Technologies for E-Services*, pages 15–28, Toronto, Canada, August 29-30 2004.
- [17] A. Brogi and R. Popescu. Automated generation of BPEL adapters. In *Proceedings of the 4th International Conference Service-Oriented Computing*, pages 27–39, Chicago, IL, USA, December 4-7, 2006 2006.
- [18] K. Brown and M. Ellis. Best practices for web services versioning. IBM Technical library, January 30 2004.
- [19] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow evolution. *Data & Knowledge Engineering*, 24:211–238, 1998.
- [20] G. Chaffle, P. Doshi, J. Harney, S. Mittal, and B. Srivastava. Improved adaptation of web service compositions using value of changed information. In *Proceedings of 2007 IEEE International Conference on Web Services*, pages 784–791, Salt Lake City, Utah, USA, July 9-13, 2007 2007.
- [21] A. Charfi and M. Mezini. AO4BPEL: An aspect-oriented extension to BPEL. In *World Wide Web*, pages 309–344, 2007.

- [22] J. Choi, Y. Cho, K. Shin, and J. Choi. A context-aware workflow system for dynamic service adaptation. In *Proceedings of the 2007 International Conference on Computational Science and Its Applications*, pages 335–345, Kuala Lumpur, Malaysia, August 2007.
- [23] L. de Alfaro and T. A. Henzinger. Interface automata. In *ESEC / SIGSOFT FSE*, pages 109–120, 2001.
- [24] R. Dijkman and M. Dumas. Service-oriented design: A multi-viewpoint approach. *International Journal of Cooperative Information Systems*, 13(4):337–368, 2004.
- [25] R. M. Dijkman. A classification of differences between similar business processes. In *Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference*, pages 37–50, Annapolis, Maryland, 15-19 October 2007 2007.
- [26] C. Dorn and S. Dustdar. Interaction-driven self-adaptation of service ensembles. In *Proceedings of the 22nd International Conference on Advanced Information Systems Engineering*, pages 393–408, Hammamet, Tunisia, June 6-9 2010.
- [27] M. Dumas, B. Benatallah, and H. R. M. Nezhad. Web service protocols: Compatibility and adaptation. *IEEE Data Eng. Bull.*, 31(3):40–44, 2008.
- [28] M. Dumas, M. Spork, and K. Wang. Adapt or perish: Algebra and visual notation for service interface adaptation. In *Proceedings of the 4th International Conference on Business Process Management*, pages 65–80, Vienna, Austria, 2006.

-
- [29] M. Dumas, W. M. van der Aalst, and A. H. ter Hofstede. *Process Aware Information Systems: Bridging People and Software Through Process Technology*. Wiley-Interscience, 2005.
- [30] H. Evans and P. Dickman. Drastic: A run-time architecture for evolving, distributed, persistent systems. In *Proceedings of the 11th European Conference on Object-Oriented Programming*, pages 243–275, Finland,, June 9C13 1997. Springer.
- [31] H. Foster, S. Uchitel, J. Magee, and J. Kramer. Compatibility verification for web service choreography. In *Proceedings of the 2004 IEEE International Conference on Web Services*, pages 738–741, San Diego, California, June 6-9 2004.
- [32] D. Frank, L. Fong, and L. Lam. A continuous long running batch orchestration model for workflow instance migration. In *Proceedings of the 2010 IEEE International Conference on Services Computing*, pages 226–233, Miami, Florida, USA, July 5-10 2010.
- [33] S. Gong, J. Xiong, Z. Liu, and C. Zhang. Correcting interaction mismatches for business processes. In *Proceedings of the 2010 IEEE International Conference on Services Computing*, pages 457–465, Miami, Florida, USA, July 5-10 2010.
- [34] M. Grossniklaus, S. Leone, A. de Spindler, and M. C. Norrie. Dynamic metamodel extension modules to support adaptive data management. In *Proceedings of the 22nd International Conference on Advanced Information Systems Engineering*, pages 363–377, Hammamet, Tunisia, June 7-9 2010.
- [35] A. Hallerbach, T. Bauer, and M. Reichert. Managing process variants in

- the process life cycle. In *Proceedings of the 10th International Conference on Enterprise Information Systems*, pages 154–161, Barcelona, Spain, June 12-16 2008.
- [36] A. Hallerbach, T. Bauer, and M. Reichert. Capturing variability in business process models: The provop approach. *Software Process: Improvement and Practice*, (Accepted for Publication), 2010.
- [37] J. Harney and P. Doshi. Speeding up adaptation of web service compositions using expiration times. In *Proceedings of the 16th International Conference on World*, pages 1023–1032, Banff, Alberta, Canada, May 8-12 2007.
- [38] Z. Jarouchech, X. Liu, and S. Smith. Apto: A MDD-based generic framework for context-aware deeply adaptive service-based processes. In *Proceedings of the 2010 IEEE international Conference on Web Services*, pages 219–226, Miami, Florida, USA, 2010.
- [39] L. Jin, J. Wu, J. Yin, Y. Li, and S. Deng. Improve service interface adaptation using sub-ontology extraction. In *Proceedings of the 2010 IEEE International Conference on Services Computing*, pages 170–177, Miami, Florida, USA, July 5-10 2010.
- [40] G. Joeris and O. Herzog. Managing evolving workflow specifications. In *Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems*, pages 310–321, New York City, New York, USA, August 20-22 1998.
- [41] G. Joeris and O. Herzog. Managing evolving workflow specifications with schema versioning and migration rules. Technical report, TZI Technical

-
- Report, Center for Computing Technologies (TZI), University of Bremen, 1999.
- [42] M. B. Juric. A hands-on introduction to BPEL. <http://www.oracle.com/technetwork/articles/matjaz-bpel1-090575.html>.
- [43] B. Kalali, P. Alencar, and D. Cowan. A service-oriented monitoring registry. In *Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative research (CASCON '03)*, pages 107–121, 2003.
- [44] P. Kaminski, H. Mller, and M. Litoiu. A design for adaptive web service evolution. In *Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems*, pages 86–92, Shanghai, China, 2006.
- [45] Y. Kataoka, M. D. Ernst, W. G. Griswold, and D. Notkin. Automated support for program refactoring using invariants. In *Proceedings of the International Conference on Software Maintenance*, pages 736–743, 2001.
- [46] W. Kongdenfha, R. Saint-Paul, B. Benatallah, and F. Casati. An aspect-oriented framework for service adaptation. In *Proceedings of the 2nd International Conference on Service-Oriented Computing (ICSOC)*, pages 15–26, New York, USA, 2006.
- [47] D. König, N. Lohmann, S. Moser, C. Stahl, and K. Wolf. Extending the compatibility notion for abstract ws-bpel processes. In *Proceeding of the 17th international conference on World Wide Web (WWW '08)*, pages 785–794, New York, NY, USA, 2008. ACM.
- [48] M. Koning, C. ai Sun, M. Sinnema, and P. Avgeriou. VxBPEL: Supporting variability for web services in BPEL. *Information & Software Technology*, 51(2):258–269, 2009.

- [49] M. Kradolfer and A. Geppert. Dynamic workflow schema evolution based on workflow type versioning and workflow migration. In *Proceedings of the 4th IFCIS International Conference on Cooperative Information Systems*, pages 104–114, Edinburgh, Scotland, September 2-4 1999.
- [50] J. Kramer and J. Magee. The evolving philosophers problem: Dynamic change management. *IEEE Transactions on Software Engineering*, 16(11):1293–1306, 1990.
- [51] M. Lanza and S. Ducasse. Understanding software evolution using a combination of software visualization and software metrics. in *Proceedings of L’OBJET*, 8(1-2):135–149, 2002.
- [52] M. M. Lehman. Program evolution. *Information Processing & Management*, 20(1):19–36, 1984.
- [53] F. Leymann, D. Roller, and M.-T. Schmidt. Web services and business process management. *IBM Systems Journal*, 41(2):198–211, 2002.
- [54] C. Li, M. Reichert, and A. Wombacher. Mining process variants: Goals and issues. In *Proceedings of the IEEE International Conference on Services Computing*, pages 573–576, Honolulu, Hawaii, 8-11 July 2008.
- [55] C. Li, M. Reichert, and A. Wombacher. On measuring process model similarity based on high-level change operations. In *Proceedings of the 27th International Conference on Conceptual Modeling*, pages 248–264, Barcelona, Spain, October 20-24 2008.
- [56] C. Li, M. Reichert, and A. Wombacher. Discovering reference models by mining process variants using a heuristic approach. In *Proceedings of*

-
- the 7th International Conference on Business Process Management*, pages 344–362, Ulm, Germany, 2009.
- [57] X. Liu and A. Bouguettaya. Managing top-down changes in service-oriented enterprises. In *Proceedings of the 2007 IEEE International Conference on Web Services*, pages 1072–1079, Salt Lake City, Utah, USA, July 9-13 2007.
- [58] X. Liu and A. Bouguettaya. Reacting to functional changes in service-oriented enterprises. In *Proceedings of the 3rd International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 264–270, White Plains, New York, USA, November 12-15 2007.
- [59] X. Liu, C. Liu, M. Rege, and A. Bouguettaya. Semantic support for adaptive long term composed services. In *Proceedings of the 2010 IEEE International Conference on Web Services (ICWS 2010)*, pages 267–274, Miami, Florida, USA, July 5-10 2010.
- [60] N. Lohmann, P. Massuthe, C. Stahl, and D. Weinberg. Analyzing interacting WS-BPEL processes using flexible model generation. *Data Knowl. Eng.*, 64(1):38–54, 2008.
- [61] A. Martens. Analyzing web service based business processes. In *Proceedings of the 8th International Conference Fundamental Approaches to Software Engineering*, pages 19–33, Edinburgh, UK, 2005.
- [62] A. Martens, S. Moser, A. Gerhardt, and K. Funk. Analyzing compatibility of bpm processes. In *Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web*

- Applications and Services*, page 147, Guadeloupe, French Caribbean, 19-25 February 2006.
- [63] R. Mateescu, P. Poizat, and G. Salaün. Adaptation of service protocols using process algebra and on-the-fly reduction techniques. In *Proceedings of the 6th International Conference on Service-Oriented Computing (ICSOC 2008)*, pages 84–99, Sydney, Australia, December 1-5 2008.
- [64] T. Mens and T. Tourwe. A survey of software refactoring. *IEEE Transactions on Software Engineering*, 30(2):126–139, February 2004.
- [65] R. Mietzner and F. Leymann. Generation of BPEL customization processes for saas applications from variability descriptors. In *Proceedings of 2008 IEEE International Conference on Services Computing*, pages 359–366, Honolulu, Hawaii, US, 8-11 July 2008.
- [66] H.-R. Motahari-Nezhad, G. Y. Xu, and B. Benatallah. Protocol-aware matching of web service interfaces for adapter development. In *Proceedings of the 19th International Conference on World Wide Web*, pages 731–740, Raleigh, North Carolina, USA, April 26-30 2010.
- [67] H. R. M. Nezhad, B. Benatallah, A. Martens, F. Curbera, and F. Casati. Semi-automated adaptation of service interactions. In *Proceedings of the 16th International Conference on World Wide Web*, pages 993–1002, Banff, Alberta, Canada, May 8-12 2007.
- [68] E. D. Nitto, C. Ghezzi, A. Metzger, M. P. Papazoglou, and K. Pohl. A journey to highly dynamic, self-adaptive service-based applications. *Autom. Softw. Eng.*, 15(3-4):313–341, 2008.

-
- [69] M. P. Papazoglou. What's in a service? In *Proceedings of 1st European Conference on Software Architecture*, pages 11–28, Aranjuez, Spain, 2007.
- [70] M. P. Papazoglou. The challenges of service evolution. In *Proceedings of the 20th International Conference Advanced Information Systems Engineering*, pages 1–15, Montpellier, France, June 16-20 2008.
- [71] M. P. Papazoglou and D. Georgakopoulos. Service-oriented computing. *Communications of the ACM*, 46(10):25–28, October 2003.
- [72] M. P. Papazoglou and W.-J. van den Heuvel. Service-oriented design and development methodology. *Int. J. Web Eng. Technol.*, 2(4):412–442, 2006.
- [73] M. Pesic, M. H. Schonenberg, N. Sidorova, and W. M. P. van der Aalst. Constraint-based workflow models: Change made easy. In *Proceedings of OTM Conferences (1)*, pages 77–94, Vilamoura, Portugal, November 25-30 2007.
- [74] J. Ponge, B. Benatallah, F. Casati, and F. Toumani. Fine-grained compatibility and replaceability analysis of timed web service protocols. In *Proceedings of the 26th International Conference on Conceptual Modeling (ER 2007)*, pages 599–614, Auckland, New Zealand, November 5-9 2007.
- [75] S. Ponnekanti and A. Fox. Interoperability among independently evolving web services. In *Proceedings of the 2004 ACM/IFIP/USENIX International Middleware Conference*, pages 331–351, Toronto, Canada, October 18-20, 2004 2004.
- [76] V. Rajlich. A model for change propagation based on graph rewriting. In *Proceedings of International Conference on Software Maintenance*, pages 84–91, Bari, Italy, 1-3 October 1997.

- [77] M. Reichert and P. Dadam. Adept_{flex}-supporting dynamic changes of workflows without losing control. *J. Intell. Inf. Syst.*, 10(2):93–129, 1998.
- [78] M. Reichert, S. Rinderle, and P. Dadam. On the common support of workflow type and instance changes under correctness constraints. In *Proceedings of the 2003 OTM Confederated International Conferences, CoopIS, DOA, and ODBASE*, pages 407–425, Catania, Sicily, Italy, November 3-7 2003.
- [79] S. Rinderle, B. Weber, M. Reichert, and W. Wild. Integrating process learning and process evolution - a semantics based approach. In *Proceedings of the 3rd International Conference (BPM 2005)*, pages 252–267, Nancy, France, September 5-8 2005.
- [80] S. Rinderle, A. Wombacher, and M. Reichert. Evolution of process choreographies in dychor. In *Proceedings of the 2006 OTM Confederated International Conferences on CoopIS, DOA, GADA, and ODBASE*, pages 273–290, Montpellier, France, October 29 - November 3 2006.
- [81] M. L. Rosa, J. Lux, S. Seidel, M. Dumas, and A. H. M. ter Hofstede. Questionnaire-driven configuration of reference process models. In *Proceedings of the 19th International Conference on Advanced Information Systems Engineering*, pages 424–438, Trondheim, Norway, June 11-15 2007.
- [82] M. Rosemann and W. M. P. van der Aalst. A configurable reference modelling language. *Inf. Syst.*, 32(1):1–23, 2007.
- [83] S. H. Ryu, F. Casati, H. Skogsrud, B. Benatallah, and R. Saint-Paul. Supporting the dynamic evolution of web service protocols in service-oriented architectures. *ACM Transactions on the Web*, 2(2):Article 13, April 2008.

-
- [84] S. H. Ryu, R. Saint-Paul, B. Benatallah, and F. Casati. A framework for managing the evolution of business protocols in web services. In *Proceedings of the 4th Asia-Pacific Conference on Conceptual Modelling (APCCM2007)*, pages 49–59, Ballarat, Victoria, Australia, January 30 - February 2 2007.
- [85] S. W. Sadiq. Handling dynamic schema change in process models. In *Proceedings of the 2000 Australasian Database Conference*, pages 120–126, 2000.
- [86] S. W. Sadiq, M. E. Orlowska, and W. Sadiq. Specification and validation of process constraints for flexible workflows. *Inf. Syst.*, 30(5):349–378, 2005.
- [87] G. Salaün, L. Bordeaux, and M. Schaerf. Describing and reasoning on web services using process algebra. In *Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, pages 43–52, San Diego, California, USA, June 6-9 2004.
- [88] H. Schonenberg, B. Weber, B. F. van Dongen, and W. M. P. van der Aalst. Supporting flexible processes through recommendations based on history. In *Proceedings of the 6th International Conference on Business Process Management*, pages 51–66, Milan, Italy, 2008.
- [89] R. Seguel, R. Eshuis, and P. W. P. J. Grefen. Generating minimal protocol adaptors for loosely coupled services. In *Proceedings of IEEE International Conference on Web Services*, pages 417–424, Miami, Florida, USA, July 5-10 2010.
- [90] E. Serral, P. Valderas, and V. Pelechano. Supporting runtime system evolution to adapt to user behaviour. In *Proceedings of the 22nd International*

- Conference on Advanced Information Systems Engineering*, pages 378–392, Hammamet, Tunisia, June 7-9 2010.
- [91] Z. Shan, A. Kumar, and P. W. P. J. Grefen. Towards integrated service adaptation. In *Proceedings of the IEEE International Conference on Web Services*, pages 385–392, Miami, Florida, USA, July 5-10 2010.
- [92] A. H. Skarra and S. B. Zdonik. The management of changing types in an object-oriented database. In *Proceedings of the 1986 conference on Object-oriented programming systems, languages, and applications*, pages 483 – 495, 1986.
- [93] H. Skogsrud, B. Benatallah, F. Casati, and F. Toumani. Managing impacts of security protocol changes in service-oriented applications. In *Proceedings of the 29th International Conference on Software Engineering (ICSE 2007)*, pages 468–477, Minneapolis, MN, USA, May 20-26 2007.
- [94] W. Song, X. Ma, S. Cheung, H. Hu, and J. Lü. Preserving data flow correctness in process adaptation. In *Proceedings of the 2010 IEEE International Conference on Services Computing*, pages 9–16, Miami, Florida, USA, July 5-10 2010.
- [95] G. Taentzer, M. Goedicke, and T. Meyer. Dynamic change management by distributed graph transformation: Towards configurable distributed systems. In *Proceedings of the 6th International Workshop on Theory and Application of Graph Transformations*, pages 179–193, Paderborn, Germany, November 16-20 1998.
- [96] E. Toch, A. Gal, and D. Dori. Automatically grounding semantically-enriched conceptual models to concrete web services. In *Proceedings of*

-
- the 24th International Conference on Conceptual Modeling*, pages 304–319, Klagenfurt, Austria, 2005.
- [97] W. M. P. van der Aalst and T. Basten. Inheritance of workflows: an approach to tackling problems related to change. *Theor. Comput. Sci.*, 270(1-2):125–203, 2002.
- [98] W. M. P. van der Aalst, M. Dumas, F. Gottschalk, A. H. M. ter Hofstede, M. L. Rosa, and J. Mendling. Correctness-preserving configuration of business process models. In *Proceedings of the 11th International Conference on Fundamental Approaches to Software Engineering*, pages 46–61, Budapest, Hungary, 2008.
- [99] W. M. P. van der Aalst and A. H. M. ter Hofstede. YAWL: yet another workflow language. *Inf. Syst.*, 30(4):245–275, 2005.
- [100] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
- [101] W. M. P. van der Aalst, M. Weske, and D. Grünbauer. Case handling: a new paradigm for business process support. *Data Knowl. Eng.*, 53(2):129–162, 2005.
- [102] Y. Wang, J. Yang, and W. Zhao. Change impact analysis for service based business processes. In *IEEE International Conference on Service-Oriented Computing and Applications, SOCA 2010, 13-15 December 2010, Perth, Australia*, pages 1–8, 2010.
- [103] Y. Wang, J. Yang, and W. Zhao. Managing changes for service based business processes. In *the 5th IEEE Asia-Pacific Services Computing Confer-*

- ence, *APSCC 2010, 6-10 December 2010, Huangzhou, China*, pages 75–82, 2010.
- [104] B. Weber, M. R. Reichert, and S. Rinderle-Ma. Change patterns and change support features - enhancing flexibility in process- aware information systems. *Data and Knowledge Engineering*, 66(3):438–466, May 2008.
- [105] B. Weber, S. Rinderle, and M. Reichert. Change patterns and change support features in process-aware information systems. In *Proceedings of the 19th International Conference on Advanced Information Systems Engineering*, pages 574–588, Trondheim, Norway, June 2007.
- [106] B. Weber, S. Sadiq, and M. Reichert. Beyond rigidity - dynamic process lifecycle support. *Computer Science - R&D*, 23(2):47–65, 2009.
- [107] M. Weidlich, M. Weske, and J. Mendling. Change propagation in process models using behavioural profiles. In *Proceedings of the 2009 IEEE International Conference on Services Computing*, pages 33–40, Bangalore, India, 21-25 September 2009.
- [108] A. Wombacher. Alignment of choreography changes in BPEL processes. In *Proceedings of the International Conference on Services Computing (SCC)*, pages 1–8, Bangalore, India, 2009.
- [109] A. Wombacher, P. Fankhauser, and E. J. Neuhold. Transforming BPEL into annotated deterministic finite state automata for service discovery. In *Proceedings of the IEEE International Conference on Web Services*, pages 316–323, San Diego, California, USA, June 6-9 2004.
- [110] Y. Wu and P. Doshi. Regret-based decentralized adaptation of web processes with coordination constraints. In *Proceedings of 2007 IEEE Interna-*

-
- tional Conference on Services Computing*, pages 262–269, Salt Lake City, Utah, USA, 9-13 July 2007 2007.
- [111] Y. Wu and P. Doshi. Making BPEL flexible: adapting in the context of coordination constraints using WS-BPEL. In *Proceedings of the 17th International Conference on World Wide Web*, pages 1199–1200, Beijing, China, April 21-25 2008.
- [112] C. Yu and L. Popa. Semantic adaptation of schema mappings when schemas evolve. In *Proceedings of the 31st VLDB Conference*, pages 1006–1017, Trondheim, Norway, 2005.
- [113] L. Zeng, B. Benatallah, G. Xie, and H. Lei. Semantic service mediation. In *Proceedings of the 4th International Conference on Service-Oriented Computing*, pages 490–495, Chicago, IL, USA, December 4-7 2006.
- [114] L. Zeng, H. Lei, J.-J. Jeng, J.-Y. Chung, and B. Benatallah. Policy-driven exception-management for composite web services. In *Proceedings of the 7th IEEE International Conference on E-Commerce Technology*, pages 355–363, München, Germany, 19-22 July 2005.