Modeling, Verification and Change Management of Service-based Business Processes

By

Pengbo Xiu

A thesis submitted to Macquarie University

for the degree of Doctor of Philosophy

Department of Computing

July 2018

Examiner's Copy

Typeset in LaTeX 2_ε.

Except where acknowledged in the customary manner, the material presented in this thesis is, to the best of my knowledge, original and has not been submitted in whole or part for a degree in any university.

Pengbo Xiu

Acknowledgments

I would like to express my sincere appreciation to my principal supervisor, Prof. Jian Yang, and Dr. Weiliang Zhao. This work would have been impossible without their constant guidance and encouragement. I am also grateful to all academic and administration staffs in computing department, in particular, Dr. Yan Wang, Ms. Donna Hua, Ms. Fiona Yang, Ms. Sylvian Chow, and Ms. Melina Chan for their kindly help.

I would also like to thank the research students who are in the same research group with me, Lei Han, Yan Mei, Zizhu Zhang, Robertus Nugroho, Yi Tian, Youliang Zhong, and Peiyao Li, for their encouragement and moral support which made me stay and studies in the office 349 more enjoyable.

Special thanks to my beloved wife, Yuxuan Song for her continuous and unfailing love, support, and understanding during my four years pursuit of Ph.D. degree which made the completion of thesis possible. She was always around at times, and she helped me to keep

things in perspective. I greatly value her support and sincerely appreciate her encouragement. I also owe thanks to my mother, Lili Wang, and my father, Shichao Xiu for their love and moral support. I consider myself the luckiest man in the world to have such a lovely and caring family, standing beside me with their love and unconditional support.

I am really grateful to the China Scholarship Council (CSC) and Macquarie University as they offered me a scholarship. This work would not have been possible without the financial support from them.

List of Publications

- Pengbo Xiu, Jian Yang, and Weiliang Zhao. "A Change Management Framework for Service-based Business Process." In *Proceedings of the Australasian Computer Science Week Multiconference (ACSW)*, p. 36. ACM, 2017.

- Pengbo Xiu, Weiliang Zhao, and Jian Yang. "Correctness Verification for Service-Based Business Processes." In *2017 IEEE International Conference on Web Services (ICWS)*, pp. 752-759. IEEE, 2017.

- Pengbo Xiu, Jian Yang, and Weiliang Zhao. "Manage Consistency and Compatibility in Dynamic Service-based Business Processes." In preparing to the conference *2018 IEEE International Conference on Web Services (ICWS)*.

- Pengbo Xiu, Jian Yang, and Weiliang Zhao. "Change Management of Service-based Business Processes." In proceeding of the journal *Service Oriented Computing and Applications (SOCA)*.

Abstract

Service-Oriented Computing (SOC) paradigm and web service technologies are essential enablers for organizations to cooperate or collaborate with each other. The business processes of such organizations are referred to as service-based business processes (SBP). To manage SBPs in their dynamic context, it is vital that the complex dependencies among the internal business process and the exposed external services are correctly analyzed and modeled. This work provides solutions for the challenging issues of modeling, verification, and change management of SBPs.

Different from existing approaches in the research areas of business process management and service management, this Ph.D. research emphasizes the dependencies not only between the internal business process and involved services of an SBP but also between/among their components (e.g., activities, data, and operations). Due to the dependencies between/among the activities and data elements of the internal business process of an SBP, there could be complex control flow and data flow structures which make the SBP modeling and verification

challenging. Due to the dependencies between the internal business process and services involved in an SBP, changes may propagate between them in a cascading manner which makes the change management of SBPs challenging. To deal with this issue, it is necessary to develop a formal model for modeling SBPs at first. Then the correctness properties of SBP should be identified, and corresponding verification methods should be proposed. Finally, for handling changes in SBPs, the change types should be identified, and the change propagation must be under control.

A Petri net based model is proposed for modeling SBPs. Based on the SBP model, a set of correctness properties of SBP are identified, and the verification methods are developed accordingly. A collection of SBP change patterns are identified and classified for translating and implementing changes. With the help of the SBP model, correctness verification, and the change patterns, a change management framework is proposed for managing changes, in particular, the change propagation in SBPs.

Contents

List of Figures

List of Tables

1

Introduction

Service-Oriented Computing (SOC) paradigm and web service technologies are widely adopted by modern organizations to facilitate business cooperation and collaboration. SOC is a paradigm that utilizes services as essential elements for developing applications or solutions [1]. With web service technologies, an organization can encapsulate its business functions

into services for partners (referred to as "provided service"), it also can invoke services supplied by its partners (referred to as "invoked service") [2], the business process is referred to as "internal business process". The provided service, invoked service, and internal business process are the concerned elements of a service-based business process (SBP). As an SBP has complex dependencies among the components and is subject to changes due to its dynamic nature, it is challenging to model the SBP, verify the SBP, and manage changes in this SBP. This work deals with these challenging issues.

In this chapter, the inevitability of SBPs and the necessity of the SBP management are discussed from the historical perspective in Section 1.1. The background of business process management and service management is introduced in Section 1.2. The research problem is discussed in Section 1.3. In Section 1.4, the contributions of this work are summarized. The organizations of this thesis are provided in Section 1.5.

1.1 Service-based Business Process

As a type of information systems, the relevance of SBP systems is introduced from a historical perspective in this section. As discussed in [3, 4], the information systems are classified into various layers according to different features. The first layer consists of operating systems, i.e., the system software that manage computers' hardware/software resources and provide common services for programs. The second layer consists of generic software that can be widely used in different enterprises or different departments of an enterprise (e.g., database management software and text editing software). The third layer consists of domain-specific

software which is only used within specific types of enterprises or departments (e.g., accounting software and HR management software). The fourth layer consists of customized software which is developed for specific enterprises or departments. Business process management (BPM) systems are contained in either the second layer as separate applications (e.g., workflow management systems)or the third layer as integrated components (e.g., enterprise resource planning systems which offer a workflow management module).

Four ongoing trends in information systems have been discussed in [3, 4], which demonstrate the historical context for SBP management systems:

- **Trend 1** - From *data-centric* to *process-aware*: As discussed in [5], the modeling of business processes was often neglected since data modeling was the starting point for developing an information system in 1970s and 1980s. To follow the emerging management trends (e.g., business process re-engineering), the information system developers started to shift their focus from data modeling to business process modeling.

- **Trend 2** - From *coding* to *assembling*: With the rapid increase in hardware performance, software in the first and second layers (e.g., operating systems and database management systems) offer more and more functionality. As a result, the challenge in information system development shifts from the programming of individual modules to assembling multiple existing modules from each of the four layers. To follow this trend, business process management systems are used to customize software without a lot of hard-coding.

- **Trend 3** - From *developing* to *outsourcing*: With the advances of Web service technologies, an enterprise can outsource some parts of its information system to its partners

by invoking the respective services provided by them. Meanwhile, some functions of the enterprise's information system are encapsulated into services for its partners to use directly without developing functions by themselves. To follow this trend, service-oriented architecture must be fully considered during the development period of business process management systems.

- **Trend 4** - From *design* to *re-design*: Due to the frequent changes on nowadays' information systems, fewer of them are developed with a carefully planned design. To follow this trend, the development of business process management systems becomes more dynamic which needs an effective mechanism for dealing with changes.

1.2 Background

In this section, the research background of this research is introduced which contains the research areas of BPM, SOC/SOA/WS, and change management (from the broad sense of change management in software engineering to the narrow sense of change management in service and BPM field).

1.2.1 Business Process Management

As discussed in section 1.1, in the seventies and eighties of the last century, the traditional information systems landscape has been dominated by data-driven approaches which utilize information modeling as the starting point. However, it has been recognized that processes are equally important as data and need to be supported well in a systematic manner. As a result, a set of workflow management systems have been proposed during the last two decades

which aim at the automation of structured processes [6–8]. A workflow management system has been defined in [9] as:

"A system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications."

Note that these traditional workflow systems can only support the process design and the software implementation after design. Few of them can deal with the process analysis works (e.g., process simulation, process verification/validation, and data mining in process log). Different with these traditional workflow systems which are restricted to specific application domains, the concept of the business process management system is in a broader perspective by incorporating different types of analysis and links processes to business and social aspects [10]. Therefore, the BPM system is defined as:

"A system that assists organizations to design, implement, control, and analyze their business processes which involve process information from a broad range of sources (humans, departments, applications, and documents)."

To keep up with the trend of globalization and informatization, an increasing number of enterprises decide to fuel their BPM systems with the service-based technologies which will be discussed in the next subsection.

1.2.2 Service Oriented Computing

Due to the booming development of the Internet, an increasing number of modern enterprises reinforce their information systems with the support of service-based technologies. Service-based information systems are developed by composing existing services for providing some add-value functionalities. Service is a concept of platform-independent software components which utilize external specifications to export their functionalities and properties. The functions of a service can range from answering a simple request from a customer to executing complicated business processes [11]. A service provided by an organization must have a well-defined service description which contains the information of service signature, protocols, QoS (Quality of Service), and necessary behaviors of the services for service publication, discovery, selection, and composition.

As discussed in [1, 12, 13], Service-oriented computing (SOC) is a computing paradigm that supports the development of rapid, low-cost, and easy composition of distributed applications. SOC utilizes services as the fundamental elements. The vision of SOC is a world of cooperating services. In this world, these services are easily assembled into a loosely coupled service network. The aim is to make dynamically flexible business processes and agile applications which are cross-organization or cross-platform.

Different with SOC, service-oriented architecture (SOA) is an architectural style of building software applications. SOA promotes loose coupling between components so that these components are reused and work within a distributed architecture. To differentiate SOA from SOC, SOA is an element of SOC that plays a role to provide a suitable design to achieve the vision of SOC, i.e., it depicts how software applications should be developed in

order to satisfy the goal of SOC.

The Web service technologies provide a technical foundation for building service-based applications and information systems based on the SOC paradigm. As defined by W3C Group [14], Web service is a type of software systems that are designed for supporting machine-to-machine interaction over a network in an interoperable manner. Web services are built up on the basis of a set of Web-related standards. A Web service is defined by the Web Service Description Language (WSDL) [15], published and discovered in service registries by the Universal Description, Discovery, and Integration (UDDI) standards [16], and exchanges information over the Internet by using the Simple Object Access Protocol (SOAP) [17]. Web services defined in WSDL are composed of business processes by utilizing the Business Process Execution Language for Web Services (BPEL4WS or simply BPEL) [18], an XML based specification language. A BPEL process consists of a set of structured activities implemented by respective Web services which are offered and accessed over the Internet.

As discussed in [19], SOC, SOA, and Web service technologies are used to realize process enactment infrastructures. Processes can both implement services and utilize existing services. To keep competitive in today's global business environment, it is necessary for modern BPM systems to provide facilities that can expose defined processes as services and implement activities in a process by directly calling other services.

1.2.3 Change Management

"The only thing that never changes is that everything changes." - Louis L'Amour

Due to the evolving nature of computing programs [20], finding an effective and efficient way to manage changes in the development and maintenance phase of a software system is a traditional issue in software engineering [21, 22]. As a particular type of software systems, BPM systems have been paid a great deal of effort on change management which will be discussed in details in the next chapter. The goal of change management in BPM systems is to make an enterprise gain the ability to react to changes in a quick, effective, and flexible way which is necessary for the enterprise's economic success in today's dynamic business environment [23]. There is two significant research focusing on change management in BPM, which are managing process schema changes (static changes) and managing process instance changes (dynamic changes) [24].

The static change management refers to the problem of modifying the process on the schema level or description level. The target is to provide primitives for refining a process progressively without rewriting it from scratch, and the syntactical correctness of the changed process must be guaranteed.

The dynamic change management refers to the issue of managing running process instances when their process schema has been changed. The target is to provide mechanisms for process designers to 'gently' migrate or adapt running instances to the modified process schema which meets the new business requirements.

Service-based information systems and applications usually work in a distributed and highly dynamic environment. Those systems and applications are subject to changes that can arise from either the internal or the external requirements of the enterprises. As discussed in [25], changes can occur on three significant aspects of a service which are the service's

interface, the service's expected behavior, and the quality of this service (QoS). The interface of a Web service contains the service's sending and receiving messages and the message exchange patterns that may be used [26]. Service interface changes thus refer to those changes occurring on data schema of messages, data types of messages, message exchange patterns, etc. The behavior of a service indicates how it exchanges messages with its clients, i.e., message exchange protocols for the service [27]. Service behavior changes thus refer to those changes occurring on the message exchange protocols for a service. The QoS changes refer to the changes in the QoS properties of a service including execution price, execution duration, and reputation [28].

The existing works dealing with the changes in service-based context are categorized into three research aspects, which are service evolution, service adaptation, and change management in service composition:

- **Service evolution** refers to the issue of continuously modifying a service in a dynamical and consistent manner [29]. The target of service evolution is to develop a versioning strategy to support multiple versions of services and business protocols.

- **Service adaptation** refers to the problem of adjusting a service to eliminate the incompatibilities with its clients caused by interface changes or behavior changes [30]. The target of service adaptation is to design an adapter for dealing with the various types of mismatches related to service interfaces and service behaviors.

- **Change management in service composition** refers to the issue of managing the changes in composited services [31]. The target of change management in service

composition is to develop mechanisms for detecting, analyzing, and finally propagating the changes of participant services in a service composition.

1.3 Research Problem

1.3.1 A Scenario of Service-based Business Processes

To keep competitive in the environment of business globalization, realizing inter-organizational cooperation and collaboration becomes an inevitable trend for nowadays enterprises. With the advances of SOC paradigms and Web service technologies, an enterprise's business process can achieve dynamic cooperation with partners. As discussed in [32], for such kind of enterprises, BPM can define not only the processes within their service implementations but also the orchestration of the use of shared services for defining their enterprise operation. For realizing the global business, such a type of enterprise encapsulates some functions of its business process into services for its partners to invoke, and at the same time, this enterprise may request services provided by its partners [2]. From this enterprise's perspective, in this work, the former type of services is named as "provided service", the latter is named as "invoked service", and its business process is named as "internal business process". This scenario is referred to as an SBP in this work.

1.3.2 Problem Definition

In order to manage SBPs, it is vital that the complex dependencies among the internal business process and the exposed external services are correctly developed and maintained.

Due to the distributed and dynamic nature, the SBP of an enterprise is subject to frequent changes arising from both its internal and external requirements (e.g., environment, policy, regulation, and technology). These changes can happen in either the internal business process or the involved services of the SBP. To minimize the risks and losses caused by the changes in SBP, there is a need for an effective and robust mechanism to manage the changes which include the following features:

1. **An SBP model with formal definition**

   As discussed in [33], modeling has always been at the core position of Information Systems design and development. Decision-makers utilize modeling technologies as enabling tools for filtering out the irrelevant complexities of the real world, thereby directly facing the most critical parts of the system being studied. A formally defined model can provide a solid foundation for developing the correctness verification and the change management mechanism for SBPs. Therefore, before dealing with changes in an SBP, it is better to formally described each SBP component being concerned.

2. **The support of complicated structures**

   As discussed in [34], because a business process may contain very complicated structures for meeting business requirements, there is an arising need for modeling a broad range of business processes on a recurring basis and describing them in an imperative way. The change management mechanism must be provided with the capability for handling changes in an SBP with complicated structures.

3. **The capability for describing various types of dependencies within an SBP**

There are various types of dependencies existing within an SBP, e.g., the dependencies between activities (control flow), the dependencies between data and the process (data flow), and the dependencies between the process and involved services. As discussed in [35], there are complicated dependencies between the internal business process and involved services of an SBP, which means that the changes occurring on one side may affect others and even may propagate in the whole SBP like the cascading effect [36–38]. Complete descriptions of these dependencies provide enabling tools for analyzing the change impact and propagation in an SBP.

4. **A taxonomy of changes with formal description**

For effectively dealing with changes in SBP, it is necessary to classify these changes by their different characteristics. A taxonomy of typical changes in SBP enables developers to develop respective change management solution for each specific change type. As a result, a reusable basis for change management in SBP is provided meaning that there is no need to deal with each change in an ad-hoc manner. Respective patterns and features of the change types in the taxonomy should be formally described by the model language of SBP. As discussed in [39], these change patterns should ensure that changes are performed in a correct, consistent, and efficient way.

5. **The capability for analyzing change impact**

Due to the complicated dependencies existing between different components of an SBP, a change on one component may affect the other components in various degrees [36–38]. Before managing the change, developers must be clear of three questions: which

components will be affected, how are they affected, and is there any further propagated impact? After analyzing the change impact by answering these three questions, corresponding management strategies are developed. Therefore, for an effective and efficient change management solutions for SBP, it is crucial to be able to analyze change impact.

6. **Capability for control change propagation**

   As a result of the impact analysis of a change occurring on one component of an SBP, the impact triggered by this change on other components is identified. For adapting this primary change on the SBP correctly, it is inevitable to have some further changes on those affected components which are used to reduce the negative impact. Naturally, as these changes also may affect the other components of the SBP in certain degrees, the further changes for adapting these "further changes" will occur and the changes will propagate in the SBP like the cascading effect. Therefore, it is crucial to have a method to control the change propagation.

7. **Capability for verifying the correctness of the changed SBP**

   It is error-prone to manage changes on SBPs which are in distributed and dynamic environment, due to the complicated dependencies among SBP components. Managing changes in an SBP is a risky work, as any errors on a component of the changed SBP may cause a series of chain effect which even may lead to the system breakdown. For making sure that a changed SBP is operated without any errors, it is crucial for the change management mechanism to include a correctness verification for the SBP. The correctness properties of an SBP should be identified, and the corresponding

verification methods should be proposed.

1.3.3 Research Gaps and Challenges

Current studies in existing works provide limited support for dealing with modeling, verification, and change management in the SBP scenario which is only utilized as partial solutions. A set of research gaps are identified as follows:

1. There is a lack of support for modeling the complicated dependencies between the internal business process and involved services of an SBP. Several existing model languages can model business processes and services separately, but few of them have paid consideration to the complicated dependencies between processes and supporting services. Without a formal model language which is able to model an SBP with its inner dependencies between each component, the change management for SBP cannot even succeed from the very beginning stage.

2. In some existing change management works on business process area, and web service area, a set of change types or patterns have been classified and identified. However, there is a lack of comprehensive taxonomy on varies types of changes in SBP scenario, which leads the change management of SBP to be carried out without a reusable basis.

3. The change impact propagation between the business process and involved services of an SBP needs to be analyzed in-depth and efficiently controlled. Complicated dependencies exist between services and the business process of an SBP. A specific process change usually affects the associated services and change occurring on a service often

has various levels of impact on the process. Even though some existing works have discussed the change propagation in the components of a business process or a service, the change propagation between a process and its involved services are rarely discussed with the consideration on the coupling relations between them. Without an effective method for analyzing and controlling the change propagation between the internal business process and involved services of an SBP, the compatibility and consistency between the process and services are influenced which may lead to the system failure.

4. Any errors on a component of a changed SBP can lead to a series of chain effect which even can result in the system failure. Therefore, the correctness of the update SBP has to be verified before implementation. There is a lack of identifications on the correctness properties of SBP as well as the corresponding verification methods. Current related studies usually focus on verifying the correctness of a single business process, a collaboration of business processes, a single service, and a composition of several services. Few works have discussed the correctness verification on an SBP with consideration on the complicated dependencies between the business process and involved services. By using the existing verification methods, even if the internal business process and involved services of an SBP are verified as correct, the whole SBP cannot be guaranteed correct, e.g., a communication conflict may exist between the process and a service which can lead to the system failure.

The challenges for filling in the gaps are listed as follows:

1. **To deal with the complexity of components in an SBP**

For managing an SBP, each component of the SBP must be formally described at the first stage. There are a variety of components contained by an SBP including both the elements from the internal business process and the elements from the services involved in the SBP. Different from the existing process models or service models, the expected SBP model must have the ability to formally describe the features of both the processes and the services.

2. **To deal with the complex dependencies among SBP components**

   There are complex dependencies that are not only among the elements of the internal business processes and the elements of the services involved in an SBP but also between the internal business process and the involved services. Due to the existence of these dependencies, a change occurs on one component can affect another one which may lead to further changes as change propagation if it is necessary. As the foundation for managing changes in SBPs, it is necessary to have the ability to describe these dependencies by the expected SBP model formally. For the dependencies among the components of the internal business process of an SBP, the control flow of the activities and the data flow of the data elements belonging to this process must be described by the expected model. For the dependencies among the components of a service involved in an SBP, the interface behaviors of the operations must be described by the expected model. For the dependencies between the internal business process and the services involved in an SBP, the interaction relations between the process and the invoked services as well as the mapping relations between the process and the provided services.

3. **To identify and classify the diverse change types in an SBP**

   There is a variety of SBP changes due to the different change requirements. For handling the changes, the first step of the SBP change management is to identify and classify the common change types or change patterns, so that these change types or patterns are utilized as reusable tools for SBP designers to transform the ad-hoc change requirements into general changes and decompose complex changes into several change primitives.

4. **To control the change propagation in multiple modes**

   In the context of SBP, there are complex dependencies between the internal business process and involved services; therefore, the changes occurring in one side may affect the other one in certain degrees and can propagate through the whole SBP in multiple modes like the cascading effect which makes the change management in SBP challenging. A service change may require further changes of one or multiple activities of the internal business process. A change of an activity belonging to the internal business process may affect other activities and related services of the process. Change propagation refers to that a single change of an activity or a service causes a series of changes associated with activities and/or services like the cascading effect. Therefore, change management in the context of SBP is a challenging issue due to the complex dependencies between internal business processes and services involved. To deal with the complicated situations of SBP changes, it is necessary to have a mechanism for managing the changes, in particular to ensure the change propagation to be under control.

5. **To guarantee the absence of errors when managing changes**

   Due to the complex dependencies among the components of SBPs and the following multiple change propagation modes, it is error-prone to manage SBPs, in particular to deal with changes in SBPs. In order to avoid the potential risks of the system breaking down, the absence of errors must be guaranteed in both the design and maintenance phases of SBPs. As there are various types of potential errors in each component of an SBP, it is challenging to identify the correctness properties for each component to avoid the errors. Due to the different features of different components in SBPs, it is challenging to develop verification methods for verifying the correctness properties of respective components.

1.4 Contributions

For addressing the research problem and filling up the gaps, a set of results are contributed by this work in the SBP management area. These contributions mainly cover three aspects of SBP management which are modeling, correctness verification, and change management. The contributions of this work are summarized as follows:

1. An SBP model is defined for capturing the characteristics of different types of components and the dependencies among them. Petri net language is employed for building up the SBP model due to its formal semantic, graphical representation, state-based features, and the support by a rich set of existing analysis tools. The proposed model can be utilized as a foundation for further analysis and management work.

2. Two classes of SBP design patterns are identified which are control flow patterns and process-service relation patterns supported by the SBP model in this work. These SBP design patterns provide re-usable solutions for SBP designers to deal with recurrent problems when building up SBP models.

3. A set of correctness properties of SBP are specified, and verification methods for verifying respective correctness properties are developed. These identified correctness properties and corresponding verification methods can be utilized as enablers by SBP developers for guaranteeing the absence of errors during the design and maintenance phases of SBPs.

4. Fourteen internal business process change patterns and eleven service change patterns are identified on the basis of the SBP model. For handling complex SBP changes, these SBP change patterns can be employed to decompose complex changes into primitive changes. These SBP change patterns can assist the designers to analyze and implement complex change requirements on the original SBP model which is the first and the most crucial step for analyzing the changes impact and managing the changes.

5. A change management framework for handling SBP changes is proposed based on the SBP model, design patterns, correctness verification methods, and change patterns. This change management framework provides guidelines for SBP designers to manage changes arising in SBPs.

1.5 Thesis Organizations

The remainder of this thesis is organized as follows. In Chapter 2, the existing works are reviewed in the area of process and service modeling, verification, and change management. In Chapter 3, an SBP model is proposed which can formally describe the components of the SBP and the dependencies among them. In Chapter 4, two classes of SBP design patterns are identified which are control flow patterns and process-service relation patterns. In Chapter 5, four correctness properties of SBPs are identified, and corresponding verification methods are proposed. The correctness of the control flow structures of internal business processes and services of SBPs is referred to as the control flow soundness. The correctness of the data flow structures of internal business processes of SBPs is referred to as the data flow soundness. The correctness properties of the relations between internal business processes and different types of involved services of SBPs are referred to as the consistency (for provided services) and the compatibility (for invoked services). In Chapter 6, a set of SBP change patterns are identified with the formal description of how to implement each pattern on an SBP. Chapter 7 proposes a change management framework based on the technical components in Chapter 3&4 (SBP modeling), Chapter 5 (SBP correctness verification), and Chapter 6 (SBP change patterns). The relationship among different technical components used in the framework is discussed as well. In Chapter 7, the method for managing the changes on internal business processes of SBPs is introduced at first, followed by the method for managing changes on services involved in SBPs. In Chapter 8, the results of this thesis are concluded, and the future work is discussed.

2

RELATED WORK

This chapter provides an overview of the existing research related to the modeling, verification, and change management of SBP. The chapter is structured as follows. Firstly, the research on the process modeling is introduced in Section 2.1. The Petri net based process modeling languages are highlighted in this section. Then, in Section 2.2, the research related to the correctness verification of SBP is discussed. Finally, the research related to the change

management of SBP is discussed in Section 2.3.

2.1 Process Modeling with Petri Nets

Business process modeling is one of the major focuses in the research field of the information system. As discussed in [40], the modeling of processes plays a key role in business process management that can be utilized to plan, design, simulate and automatically execute a business process. Therefore, it is essential to choose a suitable modeling language to represent an organization's process. As identified in [19], there are three types of process modeling languages as follows. To study business processes by abstracting them from their "implementation details", formal modeling languages are used as theoretical models which have unambiguous semantics and allow for analysis. Commonly used formal languages for process modeling include Petri nets, state machines, transition systems, temporal logic, process algebras, etc. Due to the low-level and inflexible nature of formal modeling languages, conceptual modeling languages are often used for roughly describing the desired behavior of processes which are high-level languages without the formal semantics. Common conceptual modeling languages include BPMN (Business Process Modeling Notation) [41–43], EPCs (Event-Driven Process Chains) [44–46], UML activity diagrams [47–49], etc. For process enactment, execution modeling languages are used in practice which is more technical than the former two. One of the most common execution modeling languages is BPEL (Business Process Execution Language) [50–52]. The target of this work is to model SBPs, thereby analyzing them and managing changes for them. Therefore, following sections only focus on existing formal model languages related to this work which have well-defined semantics and

allow for analysis.

2.1.1 Petri Nets Background

Classsical Petri Nets

A class of modeling languages is named as Petri nets which were invented by *Carl Adam Petri* [53]. The use of Peti net is for the description of distributed systems. Petri nets not only are developed with a mathematical foundation which is well-defined, but also have graphical features which make them user-friendly. A Petri net is directed graph which contains three types of components: *places*, *transitions*, and *arcs*. A place represents a state of the system which is often shown as a circle graphically. A transition represents an event which may change the state of a system, often drawn as a rectangle or a bar. An arc represents the flow relation between a place and a transition which is shown as an arrow connecting the place and the transition. Arcs of a Petri net cannot connect two nodes in the same type. For representing the dynamic behavior of a system, tokens (graphically shown as dots) are used which stay in places and can be "consumed" and "produced" by transitions. In [54], the author has presented a formal definition of classical Petri nets.

A Petri net is a tuple, $PN = (P, T, F, M_0)$ where:

- P is a finite set of places.

- T is a finite set of transitions.

- $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs or flow relations between places and transitions.

- M_0 is the initial marking of the Petri net representing the initial state of the system.

A marking $M = \{M(p_1), M(p_2), M(p_3), ...\}$ represents the system state. $M(p) : p \rightarrow \mathbb{N}$

($\mathbb{N}$ is the set of all non-negative integers) indicates how many tokens in place p. The markings of a Petri net can be changed based on the firing rules or transition rules as follows.

1. A transition t is *enabled* (it may fire) if there is at least one token in each input place of it.

2. Firing a transition t consumes one token from each input place of t, and produces one token in each output place of t.

Extended Petri Nets

As same as other formal modeling languages, Petri nets are "extensible". For modeling specific situations in the real world, there are a plenty of existing extension works on Petri nets that add new properties to classical Petri nets. In [55], the distinction between tokens are allowed that each token can be attached with a data value named "color", the extended Petri nets are called "colored Petri nets". The development of colored Petri nets provides an enable tools for such kind of modeling works that the manipulation of system's data values and the representation of system's hierarchical structure must be concerned. For modeling and analyzing the temporal behavior of systems, the property of time has been extended on Petri nets in several works [56–59]. The notion of time can be represented in Petri nets in four ways which are tokens holding durations of places [56], firing durations of transitions [57], enabling durations of transitions [58], and timestamps as token colors of colored Petri nets [59].

2.1.2 Petri-net-based Modeling Techniques

As indicated in [60], there are three benefits of process modeling languages which are developed based on Petri Nets. The first benefit is that by using Petri nets, process logic can be represented by a formal and graphical modeling language. The formal semantics of Petri nets provide a reusable basis for modeling business process, and the graphical feature eases the understanding of the modeled processes. The second benefit is that Petri nets make it possible to model the state of a system explicitly, different from many other process modeling languages which are event-based. Event-based languages model processes' transitions (tasks) explicitly and the states before or after each transition (task) implicitly. Event-based languages can only be used in the circumstance where the tasks can only be triggered by the process management system. However, in many realistic situations, the activation of a task is affected by the environment instead of the restrictions of the system. To deal with these situations, there is a need for using a state-based modeling language instead of an event-based one. The third benefit is that there is an abundance of formal analysis techniques developed based on Petri nets. By the assistance of these techniques, Petri nets-based process modeling languages can not only deal with the structural modeling but also apply a wide range of analysis, verification, and validation of the modeled processes.

Modeling Control FLows of Processes

One of the most significant works of using Petri nets to model and analyze business processes is the Workflow Net. For the purpose of verifying the correctness of the control-flows of business processes, the notion of Workflow Net was firstly proposed in [61]. Due to the third

benefit of process modeling languages based on Petri Nets which has been mentioned before, the correctness of a business process has been shown that it can be represented by such a Petri net and can be verified by using standard Petri-net-based techniques. In a Workflow net, each transition represents a corresponding task of a process. A place represents the pre-condition or post-condition of a task. The flow relations of the process are represented by arcs.

Modeling Data FLows of Processes

In the process modeling research area, early works mainly focus on the process structure but neglect the data-flow inside. As discussed in [62], a process's data-flow and control-flow should not be completely independent of each other. Interactions exist between the data-flow and the control-flow of a process, e.g., the data-flow's execution order can be determined by the control-flow, the control-flow's execution order can be influenced by the result of a data operation in the data-flow. It is necessary to integrate the control-flow and data-flow when modeling a process for analyzing it or managing changes in it. Some approaches have been proposed by recent works that extend the existing control-flow models to show the data-flow at the same time.

A formal foundation for integrating the data-flow and control-flow was firstly proposed in the field of embedded system modeling [63]. The authors have proposed a Petri-net-based model that is named as Dual Flow Net (DFN). In a DFN, there are three types of elements, i.e., reactive units, storage units, and transformational units.

The DFN has been extended in [64] for workflow modeling and verification. The authors have proposed a model language named Dual Flow Net (DFN), which could model the

control-flow, the data-flow, and the interactions between them explicitly.

Another extension on Workflow nets for supporting data-flow modeling has been proposed in [65]. The Workflow nets with input/output data (WFIO-net) have been proposed in this work, which extended each activity of the control-flow with its input and output data sets. In this model, each data set is represented by a place which is connected with its associated activity transitions by arcs. The different directions of these arcs represent corresponding input/output operations by connected activities.

Modeling Services

Not only for process modeling, but Petri net techniques also have been used for web service modeling. As discussed in [66], for facilitating the monitoring and verification of web services-oriented software systems, an architectural modeling language named Web Service Net (WS-Net) has been presented. WS-Net is developed based on the Colored Petri Net (CPN) semantics, which defines the architectural components of a web service in three layers: interface net, interconnection net, and interoperation net. An automatic translation engine which can translate from WSDL to the WS-net has also been developed for handling the real-life applications and eliminating translation errors by manual work.

As a single web service cannot satisfy the increasing users' requirements, there are new challenges for nowadays' web services on composing or collaborating. As a solution, many effective approaches by using the workflow technique have been provided for composing complex web applications [31, 67, 68]. By using the workflow technique, there are also many Petri-net-based methods for facilitating web service composition [69–76].

In [69], a Petri-net-based algebra has been proposed for modeling and supporting the

performance analysis of web service composition from the control-flow's perspective. In this work, the behavior of a web service is treated as a partially ordered set of service operations which can be directly mapped into a Petri net. The authors modeled the service behavior by mapping the operations into transitions, the state of the service into places, the causal relations between places and transitions into arcs. The composition of a set of web services is modeled by an algebra which uses the Petri net of each service as building blocks and represents the control-flow relations (e.g., sequence, alternative, iteration, arbitrary sequence, parallel with communication, discriminator, selection, and refinement) between the composited services by corresponding algebra operators.

In [72], the authors have presented an approach based on hierarchical colored Petri nets for verifying the reliability of web services composition. The data-flow of web service composition has been considered in this work by using the token colors to represent the data elements.

In [75], a colored-Petri-net-based method has been proposed. By transforming the BPEL descriptions into a colored-Petri-net-based modeling language, the compatibility of two composited services can be analyzed.

2.2 Verification Techniques

2.2.1 Correctness of Control-Flow

A property named *soundness* and its alternative notions are identified in several works for verifying the correctness of the control-flow of a business process. Analyzing the *soundness*

property of a control-flow enables a system designer to detect the absence of deadlocks, livelocks, and other anomalies without domain knowledge of the process [77]. In [61], a Petri-net-based model named workflow nets has been proposed that can be used to model the control-flow of a business process. The *soundness* property of workflow nets also has been defined as a correctness criteria in this work. A workflow net is *sound* if and only if all the following three conditions hold: (i) *option to complete*: for every reachable state M from the initial state, there is an existing execution route which leads from state M to the end state, (ii) *proper completion*: the end state is the only state reachable from the initial state, (iii) *no dead transition*: by following the appropriate execution route, an arbitrary activity of the net can be executed. The verification of *soundness* property of a workflow net has been proved to be equivalent to the problem of *liveness* and *boundedness* of a circuited net that connects the initial state and end state of the workflow net. As discussed in [78], the classical *soundness* means *1-soundness* based on their alternative notion named *k-soundness*. A workflow net is *k-sound* if an execution is starting with k tokens in the initial place, it always properly terminates with k tokens in the final place.

In [79], a property named *weak soundness* has been defined. The requirements of *weak soundness* are a subset of the requirements of the original *soundness* which allows the existence of "dead transitions" (activities never been executed in any possible behaviors of the process).

In [80], *relaxed soundness* has been defined which is also implied by the notion of original *soundness*. *Relaxed soundness* does not require proper termination in all possible execution cases of a Petri net, only requires that for each transition of the net, there is an existing

firing sequence taking the initial state to the end state. Livelocks, deadlocks, or still active transitions while reaching the final state might be contained in some cases of a *relaxed sound* Petri net.

In [81], *lazy soundness* has been defined. A *lazy sound* process allows that some "lazy-activities" of the process might still be active or never been active when the final state is reached. The livelocks and deadlocks are prohibited by *lazy soundness*. **Figure1** shows the classification for different alternative notions of *soundness*. As shown in the figure, the requirements of *lazy soundness* is a subset of the requirements of *weak soundness*.

2.2.2 Correctness of Data-Flow

The correctness verification from the control-flow perspective has received much more attention than the other perspectives including data-flow, resources, and temporal features of business processes. The data-flow perspective has not been considered enough in the works done in the dimension of process analysis.

In [82], a set of potential data validation issues that can occur in the data-flow of a business process are identified, which are: lost data, missing data, redundant data, inconsistent data, mismatched data, misdirected data, and insufficient data. Three data-flow implementation models are also introduced briefly. The first one, explicit data-flow, describes the transition of data elements from one activity to another. The second one, implicit data-flow through control-flow, utilizes the control-flow to pass data from one activity to another. The third one, implicit data-flow through process data store, describes the data-flow by building a repository of process data that is utilized by activities for writing and retrieving data.

Because the authors have not proposed a formal model for modeling the data-flow, there is a lack of formal methods for detecting the identified types of data-flow errors in this work.

As the data-flow performs how the data elements are transferred between activities in a business process, for better describing and analyzing the data-flow, it is better to model the data-flow together with the control-flow. Another idea is to extend existing control-flow models so that data-flow can be modeled and verified at the same time [63–65, 83, 84].

In [83], the authors provide a method for detecting a set of data-flow anomalies. Although there is a formal definition of an integration of control-flow and data-flow in this work, the data-flow anomalies are only described by scenarios and examples. There is no formal specification for data-flow anomalies, and a well-defined detection algorithm becomes impossible. Both data-flow and data-flow anomalies are defined formally in [65]. The authors propose a Petri-net-based approach for formulating the data-flow modeling and verification. The detection algorithms for data flow anomalies in [65, 83] can only give the coarse granularity result of a detected anomaly. These algorithms give the result such as "there is a data initialization anomaly" but can not tell the difference between "delayed initialization anomaly" and "missing input data anomaly".

In [63], a formal approach is proposed in the aspect of embedded-systems that can describe both the control-flow and data-flow, and then applied to the aspect of the business process in [64]. The proposed approach is developed based on Petri net language by extending the two building blocks of classical Petri net to three: the storage unit, the reactive unit, and the transformational unit. This approach extends the notion of classical soundness from [61] for supporting the cases when data-flow can influence control-flow. No explicit

correctness property of data-flow is considered independently in this approach.

For verifying the data-flow correctness of a business process, an extended Workflow net, named Workflow net with data (WFD-net), has been proposed in [84]. The authors extended Workflow net with data elements and defined four types of operations (read, write, destroy, and guard) by tasks on these data elements. A series of data-flow anti-patterns are identified, which are: missing data, strongly redundant data, weakly redundant data, strongly lost data, weakly lost data, inconsistent data, never destroyed data, twice destroyed, and data not deleted on time.

In order to guarantee the data flow correctness during the adaptation of a business process, some methods for checking the correctness of data flow in process change management area are proposed in [85–87]. In [85], the data flow correctness of workflow is maintained dynamically during the modification of workflow instances. The data flow is defined as the exchange of data elements between tasks which read or write these data elements. A set of restrictions are established for governing the data flow correctness. As an extension of [85], in [86], a formal framework is developed for migrating workflow instances to a changed workflow model without affecting the correctness. A set of correctness principles and formal theorems for both control flow and data flow are imposed for designing adaptive workflow models. In [87], a method that guarantees the data flow correctness during the adaptation of a business process rather than performing model checking after the adaption. A set of criteria for different types of adaptation operations are proposed that can assist the developers to adapt the process model correctly. For the operations that introduce data flow anomalies, these criteria also can provide further remedy suggestions, i.e., further adaptation

operations.

2.2.3 Compatibility & Consistency

In the research areas of business process and web service, the notion *compatible* means that there is no conflict or deadlock when two cooperating processes or two interacting services communicate with each other. The notion *consistent* means that two processes or two services have the same behavior at certain abstract level. Quite a few works are dealing with the verification of compatibility and consistency in web services and business processes areas [2, 88–91].

A Petri-net-based approach is proposed in [2]. The author defines a property named semantic compatibility between two interactive web services. According to this definition of compatibility, if two services can interact without any deadlocks or conflicts, they are compatible with each other. To verify the compatibility, another property named as weak soundness is defined in this work, so that the verification of the compatibility between two services is converted into the verification of the weak soundness of the composition net of these two services that are fully supported by CPN Tools [92]. For non-interactive services, a property named backward compatibility between different versions of a service is defined and analyzed in [88]. In [89], a finite-state-automata-based method is proposed for solving the similarity problem of business processes.

In [90], an automatic method is proposed for checking consistency between executable and abstract business processes. A Petri-net-based model is developed to model business processes, and a data structure named communication graph is used for checking the property

of consistency. An approach based on Refined Process Structure Tree is proposed in [91] for handling the change propagation in business process choreographies. For calculating the change impact from a change occurring on a private process to its public process, a method for abstracting the control-flow in a private process is developed. However, these works can only handle the process with simple control-flow structures and provide limited support for dealing with the complicated network structures.

2.3 Change Management

Due to the evolution nature of computing programs [20], finding effective and efficient ways to manage changes in development and maintenance phases of software systems is a traditional research area in software engineering [22, 93–96], database management system [97–100], distributed system [101–103], and information system [104, 105]. A plenty of works also have been done on the change management of business processes or the change management of web services.

2.3.1 Business Process Change Management

In the research area of the business process change management, quite a few works have been done particularly in the change management for workflow systems since the end of last century. The works are categorized into two classes which are managing changes during design time and managing changes during runtime.

Managing Changes During Design Time

For change management of business processes during design time that these processes are not executed yet, the research is similar to software version and configuration management (SCM) [106–109]. There is a broad range of works on business process model versioning.

In [110] the prototype of a supporting tool for merging different versions of a business process is proposed which can be used as a plug-in for IBM WebSphere Business Modeler [111]. As a further extension of this work, an approach for detecting and analyzing the differences between different versions of a process model without a change log is proposed in [112]. This approach utilizes the concept of an SESE fragment decomposition of process models as an enabling tool for detecting and the visualizing the differences based on the structure of process models.

In [110, 112], the differences between different versions of a process model can be visualized and analyzed by applying change operations iteratively on the original version that reconnect the control flow of the process automatically.

In [113], an approach for language-independent change management of business processes is proposed. In this work, the concept of an intermediate representation for process models is identified. Based on the concept, the differences between different process models can be computed.

For comparing business process models in multi-developer environments on a semantic level, an approach is proposed in [114]. This approach can identify the equivalence between business process models based on the detection of equivalent process fragments. This

approach is extended in [115], which improves the detection mechanism by further considerations on the semantics of different process modeling languages. Several analysis strategies are proposed in [116] for the detected semantic conflicts by using approaches in [114, 115].

Managing Changes During Run Time

For change management of running business processes, based on the different research goals, the existing works can be summarized into two categories as process evolution and process flexibility.

Process evolution aims to evolve running business processes in a dynamic and disciplined manner. Quite a few approaches have been proposed for managing instances of running instances during the evolution of their business process models.

Some works aim to develop mechanisms that can dynamically modify business process models, meanwhile, ensure the correctness of these models and their consistency with running instances. In [24], for preserving the syntactical correctness during workflow models changes, two types of change primitives (declarative primitives and flow primitives) supporting workflow models changes are identified. The declarative primitives are used to modify the declaration of workflow variables which can be added, removed, or reset default values. The flow primitives are used to modify the flow structure of a workflow which can modify the execution condition, predecessors, or successors of a task. For ensuring the consistency between changed workflow models and their running instances, a set of evolution policies are proposed in [24] which are abort strategy, flush strategy, and progressive strategy. The abort strategy requires that all running instances of the original workflow model are aborted, and the newly created instances follow the changed workflow model. The flush strategy requires

that no new instance starts until all running instances of the original workflow model terminate, then all new instances start following the changed workflow model. The progressive strategy requires that multiple versions of a workflow model for all instances exist at the same time according to corresponding history or state of each instance. In [117], a three-phase framework for dealing with running instances of a changed business process is proposed. The first phase is defining modification; the second phase is conforming to modification; the third phase is enacting modification that handles the affected process instances for conforming to the changed process models by defining a concept of compliance graph as a foundation.

Not only for business process change management in design time, but version control mechanisms are also developed for assisting process evolution at runtime. In [118, 119], a version control mechanism for evolving workflow models and migrating running instances is proposed. In this work, a workflow and its tasks are defined separately, so that the task version and workflow version can be managed separately as well. Then a set of instance migration rules are identified that depend on corresponding execution states. The proposed workflow version control mechanism and instance migration rules provide an enabling tool for dynamically adjustment of workflow instances by late binding and local modification techniques of workflow instances. In [120], a framework for workflow evolution is proposed based on version control techniques. In this work, a set of change operations are defined firstly. Then a version tree can be generated from different versions of a workflow caused by some change operations. The version tree can be utilized to determine the conditions of instances migration of the modified workflow.

Some works focus on the changes propagation from business processes to their running

instances [85, 86, 121]. In [85], a formal foundation is proposed to support dynamically modifying running instances of a workflow. A conceptual and graph-based workflow model is built with formal defined syntax and semantics. Based on this model, a "complete and minimal set of change operations" are developed. By using these change operations, running instances of a workflow can be modified by preserving the structure correctness and their consistency with the workflow. Different with permanent structural changes of workflow instances, when managing temporary changes, this work proposes some precautions for enabling a running workflow system to undo temporary changes in case of backward operations. As an extension, in [86], a set of correctness properties and corresponding theorems are proposed for assisting automatic migration of a plenty of long-running process instances to a modified workflow. In [121], an approach is proposed for managing the migration of large numbers of long-running process instances to a changed workflow. A continuous instances migration model is built to keep an instance continuing to run during its migration to a changed process. The state of migrating running instances can be determined by the proposed method. Based on the result of the determination, the migration process from the old version to the evolved version of the concerned workflow can be scheduled, configured, and implemented without affecting the workflow's availability. A demonstration of how to utilize the proposed approach to a business process system is presented in this work.

Different to the process evolution that focuses on evolving a process model with the consideration of all running instances, the process flexibility focuses on adjusting a process model for individual instances in a particular context. Aiming to make fast and effective reactions on either expected or unexpected changes during runtime, the research on process

flexibility tries to find out the possible dynamic modification of both business process models and their instances.

There is quite limited literature discussing the change management problem for SBPs. In [36, 37], a classification of different types of changes and a set of change impact patterns in SBP are identified. The authors also develop a method for analyzing the change impact propagation between/within different components of an SBP. Without a formal defined SBP model, the proposed approach has limited capability to analyze the details of the complex dependency between the internal business process and involved services.

2.3.2 Service Change Management

The target of the change management in service-based applications is to make the changes be handled automatically in the in the development and maintenance phases of these applications. In this subsection, existing research on service change management is categorized into three classes which are service adaption, service evolution, and change management in service composition.

Service adaption

The research on service adaptation mainly focuses on the ability of a service to adjust itself on the interface behavior in order to keep compatible during the interaction with other services [29]. For adapting services, the standard approach is to design adapters to mediate the mismatches between the services to be adapted and the changed interacting services. For developing adapters, some existing methods are proposed based on the mismatches between protocols or interfaces of services on the technical level and some are based on the formal

service models on the abstract level.

In [122], the interoperability with substituted services of Web-based applications is discussed. Four types of incompatibilities arising during the interoperation are categorized which are structural incompatibility, value incompatibility, encoding incompatibility, and semantic incompatibility. These incompatibilities are addressed from three aspects. Firstly, a set of static and dynamic analysis tools are proposed to estimate the compatibility between an application and a substituted service. Secondly, a set of middle-ware components named 'cross-stubs' are generated semi-automatically which can resolve the incompatibilities and enable interoperation with substituted services. Thirdly, a mechanism named 'multi-option types' are proposed to enable applications to be designed in an interoperation-friendly manner from the very beginning stage.

In [123], a general guideline for managing service adaptation is proposed. Firstly, for defining the adaptation of web services on the business protocol level, different types of incompatibilities that need to be resolved for services adaptation are identified and classified. The types of incompatibilities are the signature mismatch, the message order mismatch, the missing/extra message mismatch, and the message split/merge mismatch. Then, a methodology for developing adapters for services is proposed based on the identified incompatibilities patterns and service composition technologies.

Based on the types of incompatibilities between services identified in [123], as the following work, an aspect-oriented framework is proposed in [124] for dealing with the differences between standard external specification and internal service implementation at the interface and protocol level. The framework consists of three parts, which are: i) a taxonomy is

categorized for the different mismatch types, ii) a repository of aspect-based templates is proposed for automating the mismatch handling task, and iii) a supporting tool is developed for the instantiation of templates and the execution together with respective service implementations.

Based on the mismatch patterns identified in [123], as another following work, a semi-automated mechanism is proposed in [125] for generating adapter specification to mediate the incompatibilities between protocols and service interfaces. All kinds of service order mismatches are identified, and a tree named as 'mismatch tree' is generated in this work. Furthermore, a semi-automated support for resolving the mismatches are provided by analyzing the mismatch tree.

As an extension to [125], in [126], the matching and adaptation of services focus on not only the functional description including the interfaces and data but also the behavioral description of services. A semi-automated matching method is proposed in this work with the consideration of both types of service descriptions. Two algorithms are developed for matching protocol-aware service interface, one is depth-based interface matching algorithm, and the other one is iterative reference-based interface matching algorithm. These algorithms can be utilized to refine the results of service interface matching by compositing the ordering constraints of service operations and business protocols.

Some research about service adaption is based on the formal service models such as Petri nets, process algebra, and finite state machine. In [2, 127], for checking and managing the compatibility between two services, the formal modeling language, Petri nets are employed

for translating a BPEL process into a Petri nets based service model, so that the compatibility can be analyzed both formally and automatically. The work in [128, 129] also utilize Petri nets to translate BPEL. The problem of analyzing the interaction between WS-BPEL processes are characterized. A technology chain is proposed in these work. The technology chain firstly translates a WS-BPEL process into a Petri nets based model. Secondly, the 'controllability' of the process (the existence of a partner process which can be interacted with correctly) is analyzed. Then the operating guideline is generated. Finally, for dealing with the processes in realistic size (which can be quite massive), a flexible model generation method is proposed for generating the compact Petri nets based models.

Service evolution

Service evolution is about the capability to keep consistency when dynamically changing services [29]. Two aspects can be classified for the existing research on service evolution which are service version control and service evolution on the abstract level.

Service versioning focuses on handling the different versions of services. In [130], the functional requirements for a Service-Oriented Monitoring Registry (SOMR) are provided. An SOMR can send a notification to the clients of a service when the version of the service interface changes or when the service becomes disabled. Four architectural views of an SOMR are presented which are: the overall skeleton of the SOMR system, four tiers of the SOMR and the interconnection among these tiers, details of each tier, and the deployment of the SOMR in an executable environment.

The work in [131] focuses on deploying multiple versions of a web service simultaneously for the unsupervised clients who are independently developed. For addressing this problem,

a solution is proposed by developing a technique named as 'Chain of Adapters'. According to the arguments in this work, this approach can keep a right balance for dealing with the different kinds of requirements.

In [132], a version-aware service model is proposed by extending the existing architecture of WSDL and UDDI. This work aims to address the version management of Web services. WSDL is enhanced in this work for describing the versions of services. UDDI is enhanced in this work for utilizing the attributes of versions in service directory with an event-based notification/subscription mechanism. A proxy is developed for dynamically updating the client application instance during the runtime of a service.

In [133], for recognizing changes in the WSDL specification of a service accurately, an empirical study on the analysis of WSDL evolution is presented. Firstly, an algorithm called 'VTracker' is developed for differentiating XML files which can recognize changes in WSDL precisely. Then, the changes occurring on the subsequent versions of a set of services are analyzed. Finally, the potential impact on the maintainability of these services is discussed.

The research about service evolution on the abstract level focuses on providing theorems or guidelines for managing service evolution which is abstracted from the technical standards of services. In [134], the notion of service evolution management is introduced by providing an understanding of service changes impact analysis, service changes control, service versions tracking/auditing, and service status monitoring. In order to address the problem, a formal model is proposed for managing service evolution. The formal model captures the major features of various of service description models in practical used and is independent of existing technical standards.

In [135], an approach for managing service evolution by leveraging the loosely-coupled nature of the service-oriented context is proposed. The goal of this work is to enable independent evolution of loosely coupled interacting services in a transparent manner which can preserve the interoperability between these interacting services at the same time. The concept of service contract is defined in this work to ensure the service evolution under control. A service contract covers the different types of functionalities of two interacting services which are provided functionalities and requested functionalities. Based on the concept of the service contract, the notion of interoperability of interacting services is introduced.

Change management in service composition

Due to the dynamic nature of services, participant services in a service composition may change from time to time. The changes on one participant service can affect other services in this composition and further changes on other services may be implemented due to the change propagation. The research on change management in service composition focuses on detecting participant service changes in a composition, analyzing the changes impact, simulating and facilitating the changes propagation, and finally generating the proper reactions for handling the changes.

Change management in cross-organizational business processes is studied in quite a few research which is a similar problem as change management in services composition. In [136], the problem of the process choreographies evolution is defined as the change of interactions between partners' processes in a cross-organizational setting. Modifications applied in one partner's side may lead to inconsistencies or even errors in the process choreography. To address this problem, a framework called Dynamic Choreographies (DYCHOR) is proposed

which can help process developers to analyze the change impact propagation in the choreography. In particular for recognizing the participant process changes from a choreography change, in [137], an approach is proposed to enrich the description of the private processes of a choreography for enabling the automated generation of public process. Two types of changes are discussed in this work which are subtractive changes and additive changes. Subtractive changes are referred to as the change of deleting a message sequence. Additive changes are referred to as inserting a message sequence in the respective public views.

In [138], a framework is proposed for managing the evolution of business protocols in web services. The goal of this work is to handle the ongoing instances started based on the old protocols which have already been changed. The proposed framework can support the service administrators to manage the evolution of business protocols. A set of change operators are identified for supporting the protocols modification. This work can assist the analysis on the impact of business protocol changes on service compositions.

In [139], a framework and corresponding tools are proposed for facilitating the change management in security protocol, in particular, the trust negotiation protocols. This framework can automatically determine the consequences of the protocol changes which can apply to the ongoing trust negotiations. For managing the change process, a GUI tool which is database-backed is implemented in this work.

In [140], an integrated framework is proposed for managing changes in long-term composited services which focuses on automating the change reaction process. A tree-structured service ontology is proposed which can provide semantic support for change reaction. Then

a set of algorithms are developed for querying semantics from the ontology. The service ontology can support the modification of the service composition and participant services. As an extension, the following work [141] propose a system to realize the framework proposed in [140] which is named as Evolution of Long-term Composed Services (Ev-LCS). This system can address the change management issues in long-term composed services. Firstly, a formal model is proposed which provides the semantic foundations to automate the process of change management. Then, a set of change operators are presented which can identify a change in a precise and formal manner. Thirdly, a strategy for implementing changes is proposed. Finally, a prototype system is developed to demonstrate the result of this work.

2.4 Discussion

An overview of the research related to the SBP management is provided by this chapter. The reviewed literature in this chapter provides valuable experience and solid foundations for modeling, verification, and change management of SBPs. However, the works focusing on business processes are always without consideration on services, and then the works focusing on services rarely take the business process into consideration. As discussed in this chapter, few works focus on the research set of SBP which can be treated as an integration of the features of business processes, services, and the new features resulted by the dependencies between business processes and services. For filling the research gap of SBP management, an SBP model, a set of correctness properties and corresponding verification methods, and a change management framework are reported in the following chapters.

3

A Model for Service-based Business Processes

Service-Oriented Computing (SOC) paradigm and web service technologies are widely adopted by modern organizations to realize collaboration and cooperation. SOC is a computing paradigm that utilizes services as essential elements for developing applications or solutions [1]. With web service technologies, an organization can encapsulate its business functions

into services for partners (referred to as "provided service"), it also can invoke services supplied by its partners (referred to as "invoked service")[90], the business process is referred to as "internal business process". The provided services, invoked services, and the internal business process are the concerned elements of a service-based business process.

The rest of this chapter is organized as follows. Section 3.1 presents an example to illustrate the SBP of a travel agency. Section 3.2 proposes a Petri-net-based model of SBP. In this section, the internal business process model of an SBP is proposed firstly in Sub Section 3.2.1. Then the involved service model of an SBP is proposed in Sub Section 3.2.2. Finally, the SBP model is proposed in Sub Section 3.2.3 which consists of the internal business process model, the involved service model and the dependency relations between process and services. Section 3.3 summarizes this chapter.

3.1 An Example of Service-based Business Processes

An SBP example of a travel agency is shown in Figure 3.1. Figure 3.1A shows the internal business process of the travel agency. Rounded rectangles in Figure Figure 3.1A represent the activities of the process.

Figure 3.1B shows that the travel agency provides a service "***Flight Inquiry & Booking***" for customers to inquire and book flights. Figure 3.1C shows that the "***Flight***" service from an airline company is invoked by the travel agency in order to collect available flights information for customers. Figure 3.1D shows that the "***Payment***" service from a financial institution is invoked by the travel agency for processing the payment by the assist from a third party financial institute. In Figure 3.1B, C, and D, circles represent the operations

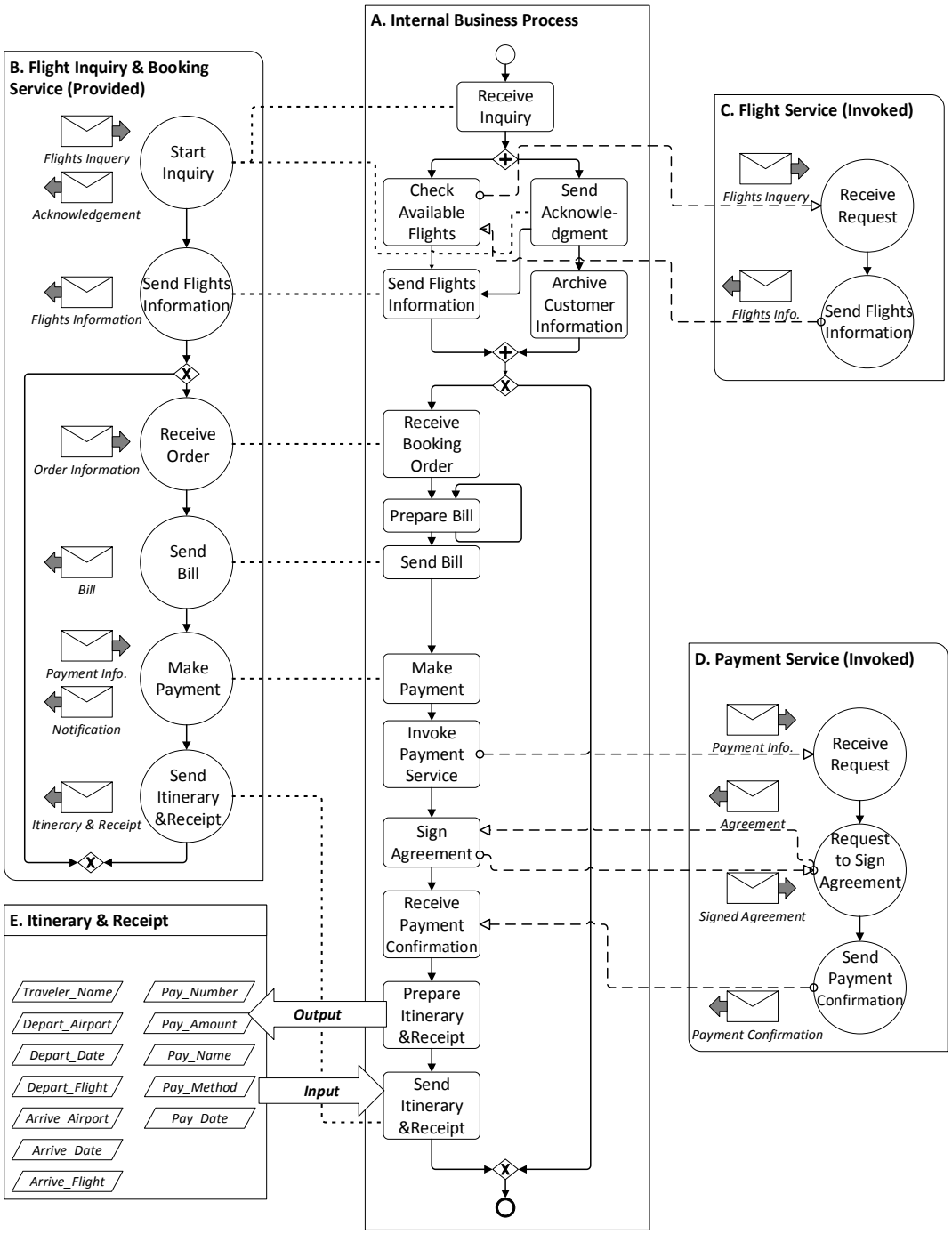

Figure 3.1: An SBP Example of a Travel Agency

of these services, envelope icons represent the messages of operations, and the arrows on messages are used for determining the message types (input message or output message).

The relations between operations of the provided services (Figure 3.1B) and activities of the internal business process are shown by dotted lines. The relations between operations of the invoked services (Figure 3.1C and D) and activities of the internal business process are shown by dashed arrows. Activities associated with provided services support their functions. Activities associated with invoked services utilize their functions.

Figure 3.1E shows one of the I/O parameters of the internal business process. The parameter "***Inquiry & Booking***" is the output of activity "*Prepare Inquiry & Booking*" and the input of activity "*Send Inquiry & Booking*". The parallelograms in Figure 3.1E represent the data elements contained in this parameter.

3.2 Model Specification

This section proposes a formal SBP model represented with Petri net [53]. Petri net provides not only a well-defined formal language but also has a user-friendly graphical representation. As indicated in [60], there are three reasons to describe processes with a Petri net as (1)it provides the formal language with a graphical representation; (2)it has the capability to capture complex and complicated states; (3)there are a rich set of analysis tools. In the proposed SBP model, there are three upper components (i.e., the internal business process model, a set of involved services models, and the relations between these services models and the process model) which are discussed in the following subsections respectively.

3.2.1 Internal Business Process

In this subsection, the control flow model of the internal business process of an SBP is defined firstly (Definition 1). Then, for model the data flow of the internal business process, the data flow is defined combined with the control flow model (Definition 2). Finally, the internal business process model is defined that contains activities, data elements, control flow relations, and data flow relations (Definition 3).

Control Flow Model

A labeled Petri net is employed for building up the control flow model of the internal business process of an SBP. The control flow model is named as a control flow net (C-net) that describes the control flow relations between activities of the internal business process (Definition 1). In a C-net, transitions are labeled with activities that represent corresponding activities of an internal business process, places represent the pre-conditions or post-conditions of activities, and arcs represent the control flow relations in the process.

Definition 1. ***(C-net)*** *A control flow net (C-net) is a tuple* $CN = (P^C, T^C, F^C, L^C, M_0^C)$*, where:*

- $P^C = \{i^C\} \cup \{o^C\} \cup P_L^C$ *is a finite set of places of a C-net.*

  i^C *and* o^C *are a pair of special places which represent the initial and ending state of the C-net (i.e.,* $\bullet i^C = \phi$*,* $o^C \bullet = \phi$*).*

  P_L^C *is a finite set of logic places which represent the pre-conditions or post-conditions of activities.*

- T^C *is a finite set of transitions.*

- $F^C \subseteq \big(P^C \times T^C\big) \cup \big(T^C \times P^C\big)$ *is a finite set of arcs which represent the flow relations between the places and the transitions of the C-net*

- $L^C : T^C \rightarrow Act \cup \{\tau^C\}$ *is a labeling function where Act is a set of activities of the process. It is assumed that* $\tau^C \notin Act$ *denotes a silent activity which is not an actual activity of the process but only for logical requirements.*

- M_0^C *is the initial marking of the C-net.*

  A marking $M(p^C) : P^C \rightarrow \mathbb{Z}^+$ *represents the number of tokens in place* p^C *(*$\mathbb{Z}^+$ *is the set of all non-negative integers). The initial marking* M_0^C *is only one token in place* i^C*.*

Figure 3.2 shows the C-net that represents the control flow of the internal business process of the example in Figure 3.1. The labeling function of this C-net is shown in Table 3.1.

Data Flow Model

The data flow performs how the data elements are transferred between activities in the internal business process of an SBP. As discussed in [62], the data flow and the control flow of a process should not be completely independent of each other. There are interactions between the data elements and the activities of a process, e.g., the operation order of the data elements can be determined by the control flow, and the execution order of the activities can be influenced by the result of a data operation in the data flow. Therefore, it is necessary to integrate the control flow and data flow for modeling the data flow of a process. A "combined

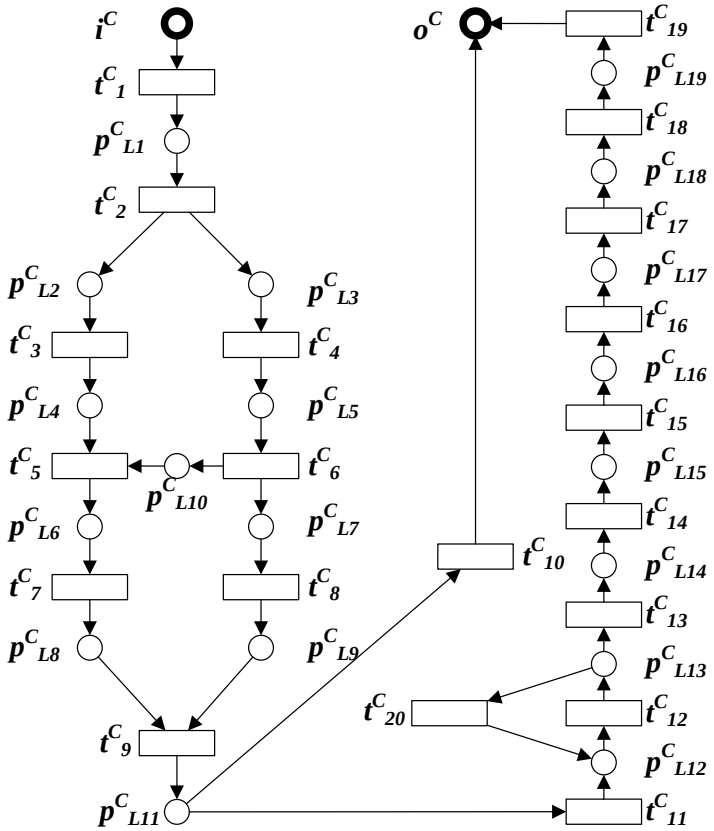

Figure 3.2: The C-net of the Travel Agency's SBP

control&data flow net" (CD-net) is presented in this subsection (Definition 2). Referring to the data flow schema defined in [85], it can be depicted that the transfer of data elements in the process is reflected by activities reading or writing respective parameters which contain the data elements. In the scenario of this work, each activity of the internal business process of an SBP has an input parameter containing all the input data elements and an output parameter containing all the output data elements. An example is shown in Figure 3.1E, the output parameter of activity "*Prepare Itinerary & Receipt*" and the input parameter of activity "*Send Itinerary & Receipt*" are same which contains the same set of data elements

Table 3.1: The Labeling Function of The C-net

t^C	$L^C(t^C)$	t^C	$L^C(t^C)$
t_1^C	*Receive Inquiry*	t_{11}^C	*Receive Booking Order*
t_2^C	τ^C	t_{12}^C	*Prepare Bill*
t_3^C	*Check Available Flights*	t_{13}^C	*Send Bill*
t_4^C	*Send Acknowledgment*	t_{14}^C	*Make Payment*
t_5^C	τ^C	t_{15}^C	*Invoke Payment Service*
t_6^C	τ^C	t_{16}^C	*Sign Agreement*
t_7^C	*Send Flights Information*	t_{17}^C	*Receive Payment Confirmation*
t_8^C	*Archive Customer Information*	t_{18}^C	*Prepare Itinerary & Receipt*
t_9^C	τ^C	t_{19}^C	*Send Itinerary & Receipt*
t_{10}^C	τ^C	t_{20}^C	τ^C

about itinerary and receipt information.

Definition 2. ***(CD-net)*** *A combined control & data flow net (CD-net) is a tuple* $CDN = (P^\Omega, T^\Omega, F^\Omega, L^C, L^D, M_0^\Omega)$*, where:*

- $P^\Omega = P^C \cup P^D \cup P^I \cup P^O \cup \{i^\Omega\} \cup \{o^\Omega\}$ *is a finite set of places.*

  P^C *is the set of places from the C-net.*

  P^D *is a set of data places representing the data elements.*

  P^I *is a set of input parameter places representing the input parameters of respective activities.*

  P^O *is a set of output parameter places representing the output parameters of respective*

activities.

i^{Ω} and o^{Ω} are a pair of special places which represent the initial and ending state of the CD-net (i.e., $\bullet i^{\Omega} = \phi$, $o^{\Omega} \bullet = \phi$).

- $T^{\Omega} = T^{C} \cup T^{I} \cup T^{O} \cup \{t^{leg}\} \cup \{t^{tar}\}$ *is a finite set of transitions.*

  T^{C} is the set of transitions from the C-net.

  T^{I} is a set of input parameter transitions which are used to collect data elements for input parameters of respective activities.

  T^{O} is a set of output parameter transitions which are used to distribute data elements for output parameters of respective activities.

  $\{t^{leg}\}$ is a transition for initializing the legacy data elements of the internal business process.

  $\{t^{tar}\}$ is a transition for gathering the target data elements of the internal business process.

- $F^{\Omega} \subseteq \big(P^{\Omega} \times T^{\Omega}\big) \cup \big(T^{\Omega} \times P^{\Omega}\big)$ *is a finite set of arcs which represent the flow relations between the places and the transitions of the CD-net. To make the data elements available for multiple readings, each arc connecting a p^{D} and a t^{I} is bidirectional.*

- L^{C} *is the labeling function of the C-net.*

- $L^{D} : P^{D} \rightarrow Data$ *is a labeling function where Data is a set of data elements of the internal business process.*

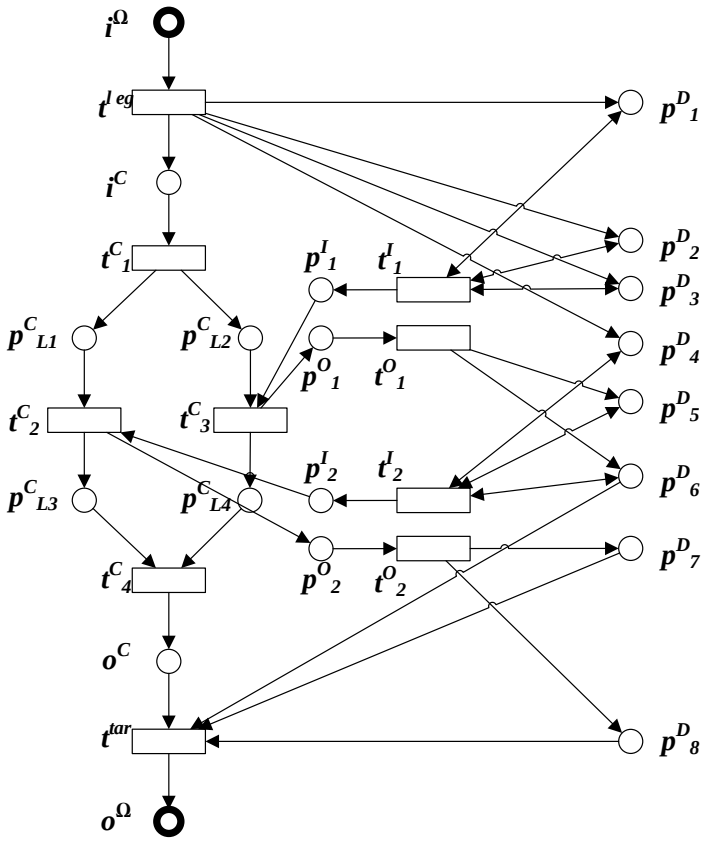

Figure 3.3: The Complete CD-net of Another Process

- *M_0^{Ω} is the initial marking of a CD-net. For each CD-net, the initial marking M_0^{Ω} is only one token in place i^{Ω}.*

Due to the oversize CD-net of the internal business process of the travel agency example in Figure 3.1, for illustrating a complete CD-net as an alternative, Figure 3.3 shows the CD-net of another business process with smaller size. In the control flow of this process, two tasks, t_2^C and t_3^C, are executed in parallel. There also are eight data elements involved in the data flow, four of them (p_1^D, p_2^D, p_3^D, p_4^D) are external data, the rest (p_5^D, p_6^D, p_7^D, p_8^D) are produced by t_2^C and t_3^C, and three of them (p_6^D, p_7^D, p_8^D) are the target data of the process.

Internal Business Process Model

An internal business process of an SBP is considered as a combination of a set of activities, a set of data elements, and their relations (control flow and data flow). In this subsection, the internal process model IP is introduced (Definition 3).

Definition 3. ***(IP)*** *The internal business process of a service-based business process is a tuple* $IP = (Act, Data, CN, CDN)$*, where:*

- *Act is a set of activities of the internal business process.*

  An activity $a \in Act$ *is a tuple* $a = (aId, InPara, OutPara, aType)$ *where* $aId(a)$ *is a function that returns the identification of the activity* a*,* $InPara(a)$ *is a function that returns the set of data elements contained in the input parameter of the activity* a*,* $OutPara(a)$ *is a function that returns the set of data elements contained in the output parameter of the activity* a*, and* $aType(a)$ *is a boolean function that returns a result whether the activity* a *is a communication activity or an internal activity. An activity being indicated as a communication activity means that there are involved service(s) associated by this activity.*

- *Data is a set of data elements of the internal business process.*

- CN *is the C-net of the internal business process which aims to describe the control flow structure of the process.*

- CDN *is the CD-net of the internal business process which aims to describe the data flow structure of the process.*

3.2.2 Service

In this subsection, the service behavior model that is describing the control flow of the involved services of an SBP is defined firstly (Definition 4). Then, the service model is defined (Definition 5).

Service Behavior Model

From a service requester's perspective, it is necessary that the expected behavior of the service is observable, which implies that the requester can see both what operations are in this service and how to invoke them [36, 37]. As discussed in [69], the behavior of a web service can be treated as a partially ordered set of service operations which can be directly mapped into a Petri net. For describing the behavior of a service, the operations of the service can be mapped into transitions, the state of the service can be mapped into places, and the causal relations between places and transitions into arcs.

A labeled Petri net is employed for building up the behavior model of the services involved in an SBP. The behavior model is named as a service behavior net (S-net) that describes the invocation relations associated with the operations of a service (Definition 4). In an S-net, transitions are labeled with operations that represent corresponding operations of the respective involved service, places represent the pre-conditions or post-conditions of operations, and arcs represent the control flow relations in the service behavior model.

Definition 4. ***(S-net)*** *An operations transition net (S-net) of a service is a tuple* $SN = (P^S, T^S, F^S, L^S, M_0^S)$*, where:*

- $P^S = \{i^S\} \cup \{o^S\} \cup P_L^S$ *is a finite set of places.*

i^S and o^S are a pair of special places which represent the initial and ending state of the S-net (i.e., $\bullet i^S = \phi$, $o^S \bullet = \phi$).

P_L^S is a finite set of logic places which represent the pre-conditions or post-conditions of operations.

- *T^S is a finite set of transitions.*

- *$F^S \subseteq \left(P^S \times T^S\right) \cup \left(T^S \times P^S\right)$ is a finite set of arcs which represent the flow relations between the places and the transitions of the S-net*

- *$L^S : T^S \rightarrow Op \cup \{\tau^S\}$ is a labeling function where Op is a set of operations of a service. It is assumed that $\tau^S \notin Op$ denotes a silent operation which is not an actual operation of the service but only for the logical requirements.*

- *M_0^S is the initial marking of an S-net. For each S-net, the initial marking M_0^S is only one token in place i^S.*

The S-nets of the three services involved in the SBP example (Figure 3.1) are shown in Figure 3.4. The labeling functions of these S-nets are shown in Table 3.2.

Service Model

A service involved in an SBP is considered as a combination of a set of operations, control flow relations between operations, the name of the service, and the type of the service. Because a service model can represent either a provided service or an invoked service of an SBP, the type property of the service can indicate what type the service is. In this subsection, the service model s is introduced (Definition 5).

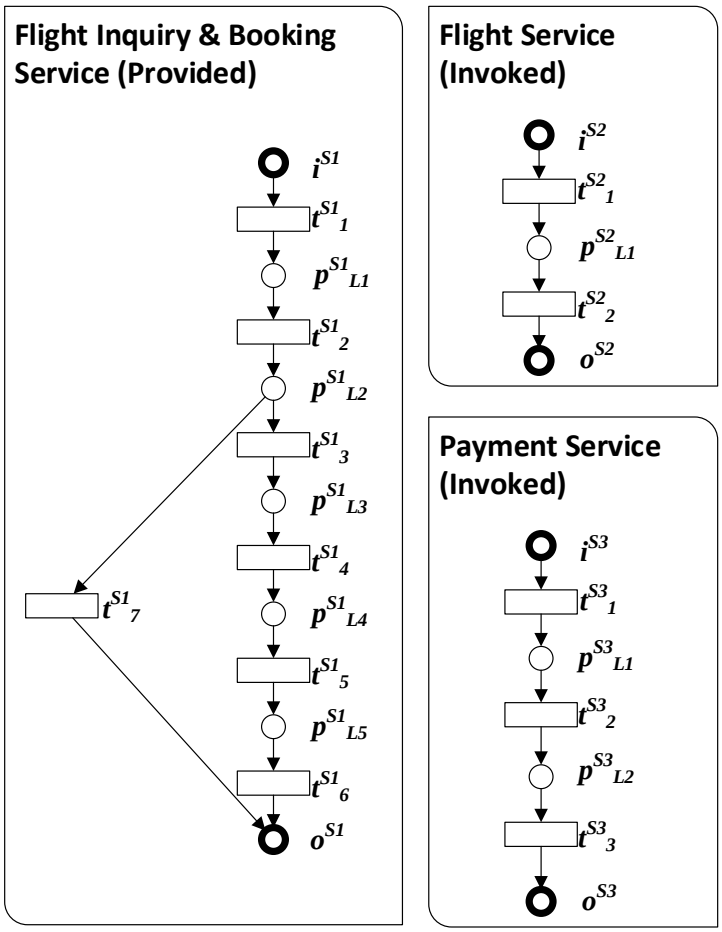

Figure 3.4: The S-nets of the Travel Agency's SBP

Definition 5. ***(Service)*** *A service S is tuple $S = (sId, Op, SN, sType)$, where:*

- *sId is the identification of the service.*

- *Op is a set of operations of the service.*

  An operation $o \in Op$ is a tuple $o = (oId, InMsg, OutMsg, oType)$ where $oId(o)$ is is a function that returns the identification of the operation, $InMsg(o)$ is a function that returns the input message of the operation, and $OutMsg(o)$ is a function that returns the output message of the operation.

Table 3.2: The Labeling Function of The S-nets

$\boldsymbol{t^{S1}}$	$\boldsymbol{L^{S1}(t^{S1})}$	$\boldsymbol{t^{S2}}$	$\boldsymbol{L^{S2}(t^{S2})}$
t_1^{S1}	*Start Inquiry*	t_1^{S2}	*Receive Request*
t_2^{S1}	*Send Flights Information*	t_2^{S2}	*Send Flights Information*
t_3^{S1}	*Receive Order*	$\boldsymbol{t^{S3}}$	$\boldsymbol{L^{S3}(t^{S3})}$
t_4^{S1}	*Send Bill*	t_1^{S3}	*Receive Request*
t_5^{S1}	*Make Payment*	t_2^{S3}	*Request to Sign Agreement*
t_6^{S1}	*Send Itinerary & Receipt*	t_3^{S3}	*Send Payment Confirmation*

$oType(o) : o \rightarrow \{$*One-way, Request-response, Solicit-response, Notification*$\}$ *is a mapping function which returns to a message transmission type of an operation, indicating which type from the four types the operation belongs to.*

- SN *is the service behavior net of the service.*

- $sType(S) : S \rightarrow \{$*Provided, Invoked*$\}$*is a boolean mapping function returning a result which indicated whether the service is a provided service or an invoked service.*

Note that as introduced in [15], there are four possible types of message transmission patterns for an operation, which are:

1. $One - way$ - The operation receives a message. As shown in Figure 3.1D, the operation "*Receive Request*" of the invoked service "***Payment Service***" is an $One - way$ operation which only receives a message of request information.

2. $Notification$ - The operation sends a message. In Figure 3.1D, the operation "*Send*

Payment Confirmation" of the invoked service "***Payment Service***" is an $Notification$ operation which only sends a message of payment confirmation.

3. $Request - response$ - The operation receives a message, then sends a correlated message. In Figure 3.1B, the operation "*Start Inquiry*" of the provided service "***Flight Inquiry & Booking***" is a $Request - response$ operation which receives a message of flight inquiry and then replies an acknowledgment.

4. $Solicit - response$ - The operation sends a message, then receives a correlated message. A $Solicit - response$ operation is similar to a $Notification$ operation; however, the difference is that the $Solicit - response$ operation expects a correlated response from the requester. In Figure 3.1D, the operation "*Request to Sign Agreement*" of the invoked service "***Payment Service***" is a $Solicit - response$ operation which sends a message of agreement and then receives the signed agreement.

3.2.3 Service-based Business Process

The SBP model is used to specify the details of an SBP on the top of the internal business process model, the set of involved service models, and the relations between the internal business process and the services. Combined with the definitions in former sections, an SBP model is defined as follows (Definition 6).

Definition 6. ***(SBP)*** *A service-based business process is a tuple* $SBP = (IP, \Sigma, R_S, R_O)$*, where:*

- IP *is the internal business process of the SBP.*

- $\Sigma = \{S_1, S_2, ..., S_n\}$ *is a set of services involved in the SBP.*

- $R_S \subseteq Act \times \Sigma$ *is a set of relations between activities and services.*

  For example, a relation $r_S \in R_S$ *and* $r_S = \langle a_n, S_m \rangle$,

  1. *if* $sType(S_m) = Provided$, *it means that activity* a_n *provides a function of provided service* S_m.

  2. *if* $sType(S_m) = Invoked$, *it means that activity* a_n *invokes a function of invoked service* S_m.

- $R_O \subseteq Act \times \Omega$ *is a set of relations between activities of the internal business process and operations of services involved in the SBP.*

  $\Omega = Op_1 \cup Op_2 \cup, ..., \cup Op_n$ *is the set of all operations of services involved in the SBP.*

  For example, a relation $r_O \in R_O$ *and* $r_O = \langle a_n, o_m \rangle$, o_m *belongs to service* S_m,

  1. *if* $sType(S_m) = Provided$, *it means that the activity* a_n *"carries out" the function of the operation* o_m.

  2. *if* $sType(S_m) = Invoked$, *it means that the activity* a_n *"utilizes" the function of the operation* o_m.

3.3 Discussion

In this chapter, a Petri-net-based model is defined for modeling SBPs. The proposed SBP model provides a formal basis for describing SBPs, verifying SBPs, simulating SBPs, analyzing SBPs, and managing changes in SBPs.

A motivating example of a travel agency is presented for capturing the typical features of SBP scenario in this work. In the SBP scenario in this work, an SBP consists of an internal business process, the provided service(s) by the process, the invoked service(s) by the process, and the dependency relations between these services and the internal business process.

The internal business process of an SBP is defined as a composition of a C-net, a CD-net, a set of activities, and a set of data elements. The C-net is defined for representing the control flow of the internal business process of an SBP. The CD-net is defined for modeling the data flow of the internal business process of an SBP.

A service involved in an SBP is defined as a composition of the identification of the service, the type of the service, operations of the service, and an S-net. The S-net is defined for representing the operations transition behavior of the service that can be considered as the control flow relation of these operations.

For modeling a broad range of real-world SBP cases on a recurring basis and describing them in an imperative way, corresponding design patterns are identified in the next chapter. These patterns provide enabling tools for describing SBPs in a unified and reusable manner.

4

Design Patterns of Service-based Business Processes

The notion of pattern is attributed to [142] generally, which means a classification of recurring problems and respective solutions in a specific domain. Early pattern-based studies mainly

focus on the domain of architecture. The notion of patterns has generally been applied in a broad range of other domains. A set of pattern-based studies have been proposed in the domain of information technology for designing software systems [143, 144] and business processes [39, 145–147].

For modeling a broad range of real-world SBP cases on a recurring basis and describing them in an imperative way, corresponding design patterns are identified in the following sections. In Section 4.1, a set of control flow patterns of SBP are identified. In Section 4.2, a set of process-service relation patterns of SBP are identified. Section 4.3 summarizes this chapter.

4.1 Control Flow Patterns

The complicated control flow structure makes it challenging to describe, simulate, and analyze an SBP. It also increases the complexity of change management for an SBP. In this section, seven common design patterns for modeling the control flow structure of real-world SBP cases are identified based on the SBP model which are sequence pattern, AND-split pattern, AND-join pattern, XOR-split pattern, XOR-join pattern, loop pattern, and control dependency pattern. These control flow patterns provide a unified and recurring basis in order to deal with the concerns of the complicated control flow structures of SBP.

4.1.1 Sequence Pattern

- ***Description:*** An activity is enabled after the completion of executing a preceding activity in the internal business process of an SBP.

- ***Example:*** For the SBP example in Figure 3.1, as the activity "*Send Itinerary & Receipt*" is enabled after the completion of the activity "*Prepare Itinerary & Receipt*", they are in a sequence control flow pattern.

- ***Demonstration:*** Figure 4.1 is the demonstration of sequence pattern by C-net. To represent the sequence pattern by C-net, a place is used to connect two transitions, one of which is labeled by an activity and the other one is labeled by its preceding activity.

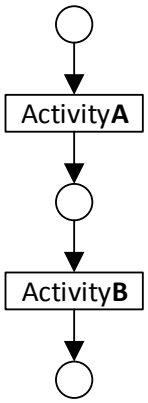

Figure 4.1: Sequence Pattern in C-net

4.1.2 AND-split Pattern

- ***Description:*** The control flow of one branch diverges into multiple parallel branches, these parallel branches are executed concurrently.

- ***Example:*** An example of AND-split pattern is shown in Figure 3.1, after the completion of the activity "*Receive Inquiry*", the activity "*Check Available Flights*" and

the activity "*Send Acknowledgment*" are enabled concurrently, they are in a AND-split control flow pattern.

- ***Demonstration:*** Figure 4.2 is the demonstration of AND-split pattern by C-net. To represent the AND-split pattern by C-net, the divergence to parallel branches is achieved by a transition which has one input arc and two or more output arcs. This transition is labeled with a silent activity τ^C.

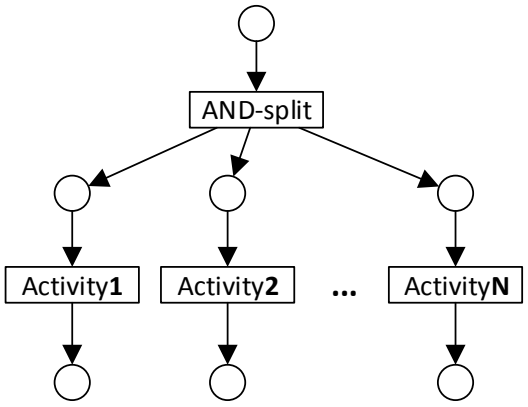

FIGURE 4.2: AND-split Pattern in C-net

4.1.3 AND-join Pattern

- ***Description:*** The control flows of two or more branches converge into a single consecutive branch. The consecutive branch is enabled after the completion of all its input branches.

- ***Example:*** An example of AND-join pattern is shown in Figure 3.1, the thread of control is passed to the consecutive branch after the completion of the activity "*Send*

Flights Information" and the activity "*Archive Customer Information*", they are in an AND-join control flow pattern.

- ***Demonstration:*** Figure 4.3 is the demonstration of AND-join pattern by C-net. To represent the AND-join pattern by C-net, the convergence from parallel branches is achieved by a transition which has two or more input arcs and one output arc. This transition is labeled with a silent activity τ^C.

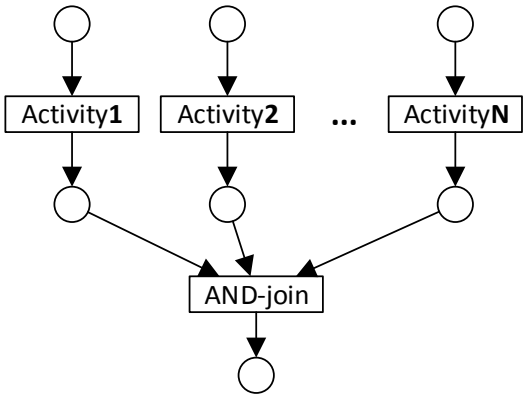

Figure 4.3: AND-join Pattern in C-net

4.1.4 XOR-split Pattern

- ***Description:*** One branch diverges into multiple mutually exclusive branches which are enabled immediately after the completion of the incoming branch. Only one of these outgoing branches can be executed depending on a selection mechanism that can select one of them to be executed. The selection can either be made by users or be value-based.

- ***Example:*** An example of the XOR-split pattern is shown in Figure 3.1, the thread of control diverges into two branches, one is to execute the activity "*Receive Booking Order*" firstly, the other one is to do nothing. A selection has to be made from these two branches depending on whether the customer decides to continue to book a flight from the travel agency after inquiry.

- ***Demonstration:*** Figure 4.4 is the demonstration of XOR-split pattern by C-net. To represent the XOR-split pattern by C-net, the divergence to mutually exclusive branches is achieved by a place which has up to one input arc and two or more output arcs.

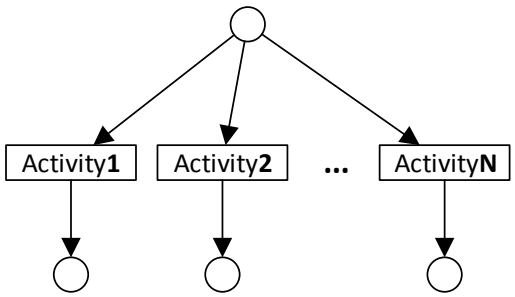

FIGURE 4.4: XOR-split Pattern in C-net

4.1.5 XOR-join Pattern

- ***Description:*** Multiple mutually exclusive branches converge into one branch which is enabled immediately after the completion of one of the incoming branches.

- ***Example:*** An example of XOR-join pattern is shown in Figure 3.1, the threads of

control converge into a single branch after the possible completion of one of the incoming two branches, one is the completion of the activity "*Send Itinerary & Receipt*" finally, the other one is to do nothing.

- ***Demonstration:*** Figure 4.5 is the demonstration of XOR-join pattern by C-net. To represent the XOR-join pattern by C-net, the convergence from multiple mutually exclusive branches is achieved by a place which has two or more input arcs and up to one output arc.

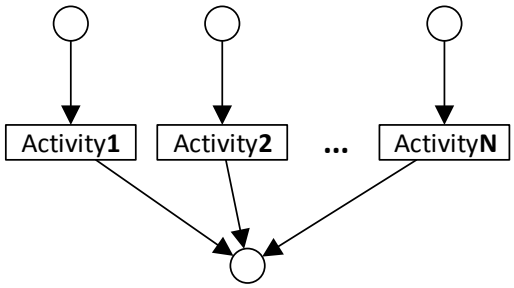

Figure 4.5: XOR-join Pattern in C-net

4.1.6 Loop Pattern

- ***Description:*** A fragment of control flow is executed repeatedly. There is a post-test condition after the completion of the fragment to evaluate whether to repeat it or to continue its consecutive branch.

- ***Example:*** For the SBP example in Figure 3.1, the activity "*Prepare Bill*" is executed repeatedly. Its consecutive activity "*Send Bill*" can be executed only after a satisfactory result comes out from the post-test of this loop.

- ***Demonstration:*** Figure 4.6 is the demonstration of loop pattern by C-net. To represent the loop pattern by C-net, a transition is used to connect from the output place to the input place of a fragment. This transition is labeled with a silent activity τ^C.

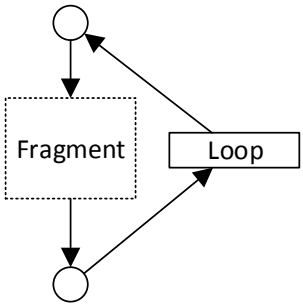

FIGURE 4.6: Loop Pattern in C-net

4.1.7 Control Dependency Pattern

- ***Description:*** Two activities in two parallel branches, whether an activity is enabled or not depends on the completion of the other one.

- ***Example:*** For the SBP example in Figure 3.1, the activity "*Send Flights Information*" and the activity "*Send Acknowledgment*" are in two parallel branches. The activity "*Send Flights Information*" is enabled after not only the completion of its preceding activity "*Check Available Flights*" in same branch, but also the completion of the activity "*Send Acknowledgment*" in other parallel branch.

- ***Demonstration:*** Figure 4.7 is the demonstration of control dependency pattern by C-net. To represent the control dependency pattern by C-net, a transition after the

output place of the preceding activity transition is used to diverge the control thread into the same branch and the consecutive activity transition in another parallel branch. This transition has the same function as an AND-split transition and is labeled with a silent activity τ^C. Another transition before the input place of the consecutive activity transition is used to converge the control threads from the same branch and another parallel branch. This transition has the same function as an AND-join transition and is labeled with a silent activity τ^C either. These two silent activity transitions are connected by a place.

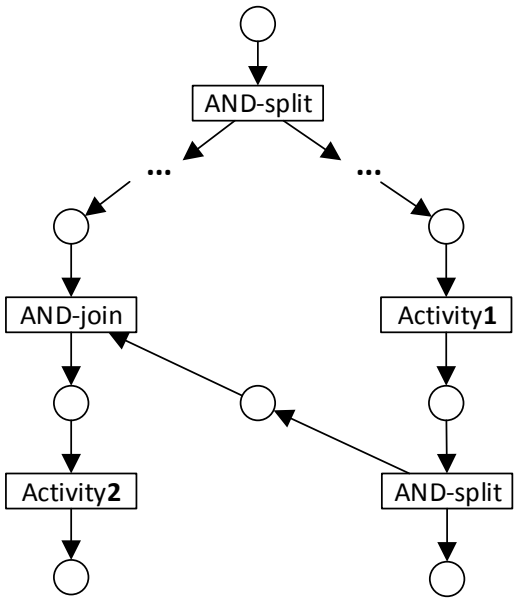

Figure 4.7: Control Dependency Pattern in C-net

4.2 Process-service Relation Patterns

As discussed in [148], one of the most important characteristics of software interaction is whether the interaction is synchronous or asynchronous. For synchronous interactions, when a software system calls another one during an execution thread, the thread cannot proceed until the response comes back. Even though the design of synchronous interactions is simple and intuitive, for some specific types of applications, there is no need for them to interact synchronously. As an alternative, the design of synchronous interactions is applied. E-mail is a remarkable example of asynchronous interactions. It is not necessary for the sender of an e-mail to wait until a reply is received. The sender can work on other things before receiving a reply. Therefore, sending and receiving e-mails are synchronous interactions.

In the domain of web service, a "$Request-response$" or "$Solicit-response$" operation receiving/sending a message, and then sending/receiving a correlated message is referred to as a *synchronous* operation; a "$One-way$" or "$Notification$" operation only receiving/sending a message is referred to as an *asynchronous* operation.

According to different granularity levels of operations from a service involved in an SBP and their related activities from the internal business process, four process-service relation patterns for R_O (relations between activities and operations of services involved) are supported by the SBP model which are described as follows. These patterns are 1A-to-1syncO (one activity to one synchronous operation) pattern, 1A-to-1asyncO (one activity to one asynchronous operation) pattern, 2A-to-1syncO (two activities to one synchronous operation) pattern, and 1A-to-2asyncO (one activity to two asynchronous operations) pattern.

4.2.1 1A-to-1syncO Pattern

- ***Description:*** One activity is related with one synchronous operation. For a provided service, a synchronous operation is supported by one activity. For an invoked service, an activity invokes a synchronous operation synchronously.

- ***In Formal:*** For an SBP, $r_O = \langle x, y \rangle$ ($r_O \in R_O$, x is an activity, y is an operation) belongs to a 1A-to-1syncO Pattern, iff :

  i. $oType(y) = Request - response$ *or* $oType(y) = Solicit - response$,

  ii. $\nexists r'_O = \langle x', y' \rangle$ $(r'_O \in R_O)$ $s.t.$ $x' = x$ *or* $y' = y$.

- ***Example:*** As shown in Figure 3.1, the activity "*Make Payment*" of the internal business process and the operation "*Make Payment*" of the provided service "***Flight Inquiry & Booking***" are in a 1A-to-1syncO pattern. The activity carries out the operation by receiving the payment from the customer followed by sending a notification for a successful payment.

4.2.2 1A-to-1asyncO Pattern

- ***Description:*** One activity is related with one asynchronous operation. For a provided service, an asynchronous operation is supported by one activity. For an invoked service, an activity invokes an asynchronous operation asynchronously.

- ***In Formal:*** For an SBP, $r_O = \langle x, y \rangle$ ($r_O \in R_O$, x is an activity, y is an operation) belongs to a 1A-to-1asyncO Pattern, iff :

i. $oType(y) = One - way \ or \ oType(y) = Notification$,

ii. $\nexists r'_O = \langle x', y' \rangle \ (r'_O \in R_O) \ s.t. \ x' = x \ or \ y' = y.$

- ***Example:*** As shown in Figure 3.1, the activity "*Invoke Payment Service*" of the internal business process and the operation "*Receive Request*" of the invoked service "***Payment Service***" are in a 1A-to-1asyncO pattern. The activity calls the operation by sending the payment information.

4.2.3 2A-to-1syncO Pattern

- ***Description:*** Two activities are related with one synchronous operation. For a provided service, a synchronous operation is supported by two activities. For an invoked service, two activities invoke a synchronous operation asynchronously.

- ***In Formal:*** For an SBP, $r_O = \langle x, y \rangle$ ($r_O \in R_O$, x is an activity, y is an operation) belongs to a 2A-to-1syncO Pattern, iff :

i. $oType(y) = Request - response \ or \ oType(y) = Solicit - response$,

ii. $\exists r'_O = \langle x', y' \rangle \ (r'_O \in R_O) \ s.t. \ x' \neq x \ and \ y' = y$,

iii. $\nexists r''_O = \langle x'', y'' \rangle \ (r''_O \in R_O) \ s.t. \ x'' = x \ or \ x', \ or \ y'' = y.$

- ***Example:*** As shown in Figure 3.1, the activities "*Receive Inquiry*" and "*Send Acknowledgment*" of the internal business process are in a 2A-to-1syncO relation with the operation "*Start Inquiry*" of the provided service "***Flight Inquiry & Booking***". These two activities carry out the input and output of the operation separately.

4.2.4 1A-to-2asyncO Pattern

- ***Description:*** One activity is related with two asynchronous operations. For a provided service, two synchronous operations are supported by one activity. For an invoked service, an activity invokes two asynchronous operations synchronously.

- ***In Formal:*** For an SBP, $r_O = \langle x, y \rangle$ ($r_O \in R_O$, x is an activity, y is an operation) belongs to a 1A-to-2asyncO Pattern, iff :

  i. $oType(y) = One - way \ or \ oType(y) = Notification$,

  ii. $\exists r'_O = \langle x', y' \rangle \ (r'_O \in R_O) \ s.t. \ x' = x \ and \ y' \neq y$,

  iii. $\nexists r''_O = \langle x'', y'' \rangle \ (r''_O \in R_O) \ s.t. \ x'' = x, \ or \ y'' = y \ or \ y'$.

- ***Example:*** As shown in Figure 3.1, the activity "*Check Available Flights*" of the internal business process is in a 1A-to-2asyncO relation with the operations "*Receive Request*" and "*Send Flights Information*" of the invoked service "***Flight Service***". The activity calls the operation "*Receive Request*" firstly and then waits until the response comes back from the operation "*Send Flights Information*".

4.3 Discussion

In this chapter, seven control flow patterns for designing the control flow of the internal business process of an SBP are identified. These patterns provide enabling tools for SBP designers to build up the control flow of internal business processes with complicated structures. Four process-service relation patterns are identified as well for designers to deal with

the dependencies between the internal business process and services involved in an SBP. These patterns are described with both informal language and formal language of SBP model definitions in Chapter 3. All of these patterns are demonstrated by using the example in Figure 3.1.

A unified and reusable basis is provided by this patterns for managing SBPs. The unified and reusable basis makes it possible to develop general methods aiming at modeling, analyzing, and maintaining SBPs without paying consideration to some ad-hoc requirements. Further utilization and discussion are introduced in the following chapters.

5

Correctness Verification of Service-based Business Processes

To keep up with the trend of globalization and informatization, an increasing number of enterprises decide to run their business process in a service-based manner with the help of

Web Service technology. To manage such service-based business processes (SBP), it is vital that the elements of SBP and the dependencies among this elements are correctly developed and maintained. It is error-prone in the development phase of an SBP, as the dependencies among different types of elements involved in the SBP are complex which is discussed in previous chapters. Because the business regulations and customer requirements may need to change frequently, SBP is dynamic by nature. As a result, both internal business processes and involved services of SBPs may need to change accordingly. Because of the complex dependencies among the elements of an SBP, the changes occurring in one element can affect the others and propagate in the SBP with a cascading manner [36, 37] that can easily cause anomalies. Therefore, it is error-prone in the maintenance phase of an SBP as well.

It is necessary to identify the correctness properties of SBP and develop the method to verify these properties correspondingly. In this chapter, a set of correctness properties of SBP is identified that includes: the control flow soundness, the data flow soundness, the consistency between a provided service and the internal business process, and the compatibility between an invoked service and the internal business process. Corresponding solutions for verifying the specified properties are developed that includes:

- The control flow soundness of an SBP is verified by checking the liveness and boundedness of a Petri net that specifies the control flow of the SBP.
- Eleven types of data flow anomalies are identified. The data flow soundness of an SBP is verified by an algorithm to detect anomalies in it.
- An algebra-based method is proposed to verify the consistency between a provided service and the internal business process of an SBP.

- An Petri-net-based method is proposed to verify the compatibility between an invoked service and the internal business process of an SBP.

The rest of this chapter is organized as follows. In Section 5.1, the control flow soundness is specified, and corresponding verification method is proposed. In Section 5.2, the data flow soundness is specified, and corresponding verification method is proposed. In Section 5.3, the consistency between a provided service and the internal business process of an SBP is specified, and corresponding verification method is proposed. In Section 5.4, the compatibility between an invoked service and the internal business process of an SBP is specified, and corresponding verification method is proposed. Further discussion about this chapter is in Section 5.5.

5.1 Control Flow Soundness

Seven control flow patterns are identified in Chapter 4 for building up the complicated control flow structures of the internal business process of an SBP. It is error-prone to develop such an internal business process with complicated control flow structures. The errors may result in that the improper termination of the internal business process, e.g., the execution of the process is obstructed by some deadlocks, some activities have never been executed when the process terminates, and some activities are still running when the process terminates. It is vital to define the correctness property for the control flow of the internal business process and develop a corresponding method to verify the identified correctness property.

5.1.1 Definition of Control Flow Soundness

There are three types of errors that must be guaranteed to avoid in the control flow of an internal business process of an SBP. The first one is the existence of deadlocks that may stick the execution of the process. Figure 5.1 shows an example of a deadlock in a C-net. In this example, because each activity cannot be enabled until the completion of the other one, the execution of the control flow is obstructed.

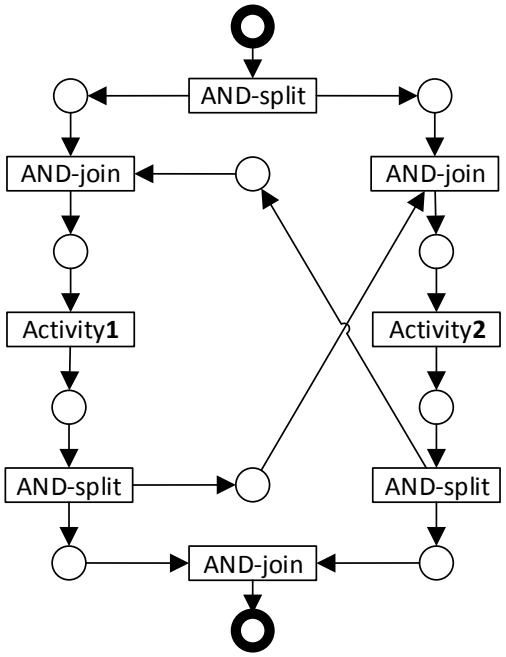

FIGURE 5.1: Deadlock Example in a C-net

The second one is the existence of dead activities that have never been executed when the process terminates. Figure 5.2 shows an example of a dead activity in a C-net. In this example, the activity *Activity7* will never be enabled when the C-net completes in any case.

The third one is the existence of dangling activities that are still running when the process terminates. Figure 5.3 shows an example of a dangling activity in a C-net. In this example,

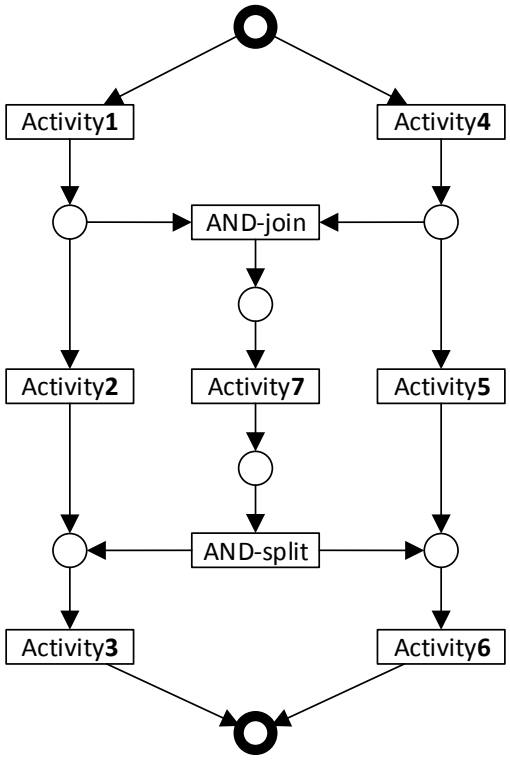

Figure 5.2: Dead Activity Example in a C-net

the activity *Activity3* may still be running when the process terminates.

For a sound C-net, the absence of all three types of errors must be guaranteed. Therefore, in the following definition of CF soundness, three conditions are identified to hold for the absence of respective error types.

Definition 7. ***(CF Soundness)*** *Let $CN = (P^C, T^C, F^C, L^C, M_0^C)$ be a C-net of an SBP. CN is CF sound if and only if all the following three conditions hold,*

i. ***Option to complete:*** *For each reachable state M^C from the initial state i^C, there is an existing firing sequence which leads from state M^C to the end state o^C. In formal:*

$\forall_{M^C}(i^C \xrightarrow{*} M^C) \Rightarrow (M^C \xrightarrow{*} o^C)$

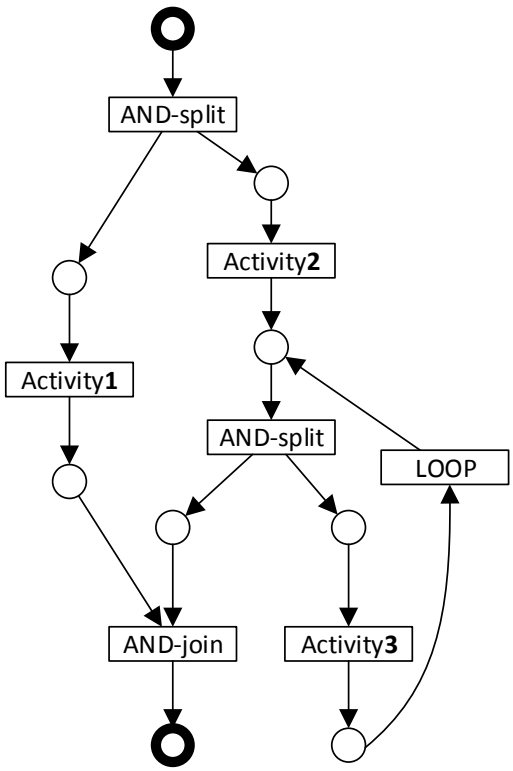

FIGURE 5.3: Dangling Activity Example in a C-net

ii. ***No dead transitions:*** *There is no dead transition in CN. In formal:*

$\forall_{t^C \in T^C}, \exists_{M^C, M'^C}\ i^C \xrightarrow{*} M^C \xrightarrow{t^C} M'^C$

iii. ***Proper completion:*** *o^C is the only state reachable from state i^C with one token in o^C. In formal:*

$\forall_{M^C}(i^C \xrightarrow{*} M^C \wedge M^C \geq o^C) \Rightarrow (M^C = o^C)$

5.1.2 Verification of Control Flow Soundness

For determining whether a given CN is CF sound, the liveness and boundedness properties of Petri nets are introduced in Definition 8 and Definition 9. In Theorem 1, a condition for a

C-net to be sound is stated. Theorem 1 transfers the problem from verifying CF soundness to verifying liveness and boundedness. In Theorem 1, an extended net $\overline{CN}$ is defined by adding a transition t^* to CN which connects the end place o^C and the initial place i^C. As an example, the extended net $\overline{CN}$ of the C-net belonging to the travel agency's SBP (Figure 3.2) are shown in Figure 5.4. The corresponding proof of Theorem 1 is proposed.

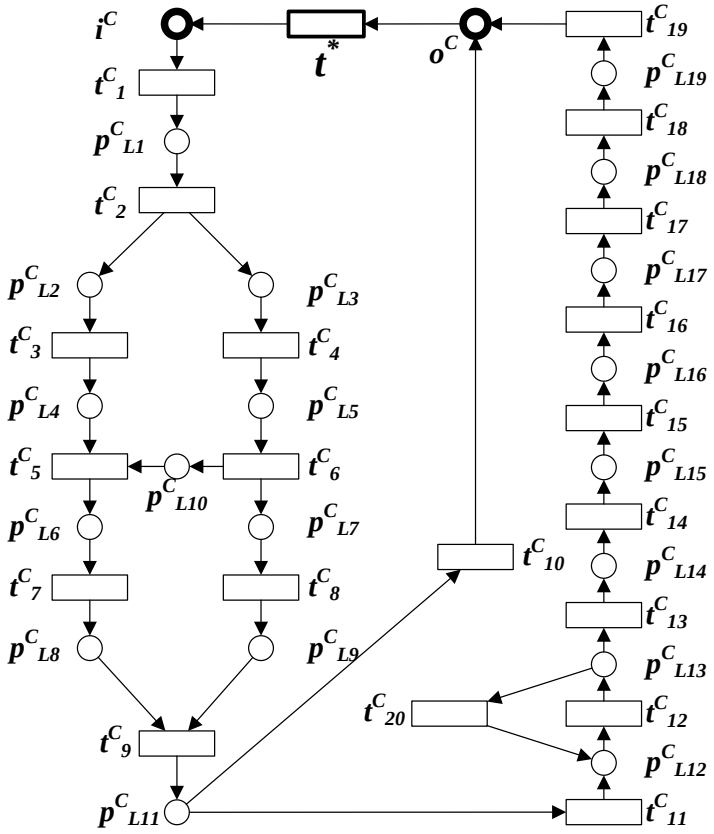

Figure 5.4: The Extended Net $\overline{CN}$ of a C-net

Definition 8. ***(Live)*** *A Petri net (PN, M_0) is live iff, for every reachable state M from M_0 and every transition t, there is a state M' which is reachable from M and enables t, formally: $\forall_M M_0 \xrightarrow{*} M$, $\forall_{t \in T}$, $\exists_{M',M''}$ $M \xrightarrow{*} M' \xrightarrow{t} M''$.*

Definition 9. ***(Bounded)*** *A Petri net (PN, M_0) is bounded iff, for every place p in every reachable state, the number of tokens in p is bounded.*

Theorem 1. *If $(\overline{CN}, i^C)$ is live and bounded, CN is CF sound.*

Proof. Since $(\overline{CN}, i^C)$ is live, i.e., $\forall_M i^C \xrightarrow{*} M$, $\forall_{t^C \in T^C}$, $\exists_{M',M''}$ $M \xrightarrow{*} M' \xrightarrow{t^C} M''$. Hence we can deduce that

$$\forall_M i^C \xrightarrow{*} M, \exists_{M',M''} M \xrightarrow{*} M' \xrightarrow{t^*} M'' \tag{1}$$

.

Since t^* has only one input place o^C and only one output place i^C. Hence M' must be with at least one token in place o^C and M'' must be with at least one token in place i^C , i.e., $M' = M_x + o^C$, $M'' = M_x + i^C$. Since $(\overline{N}, i^C)$ is bounded, M_x equals to the empty state, i.e.,

$$M' = o^C, M'' = i^C \tag{2}$$

.

By combining result (1) and result (2) we can deduce that $\forall_M (i^C \xrightarrow{*} M) \Rightarrow (M \xrightarrow{*} o^C)$ and M' is the only state which is equal to o^C. Hence, requirements (i) and (ii) of Definition 7 hold.

Since $(\overline{N}, i^C)$ is live, i.e., $\forall_M i^C \xrightarrow{*} M$, $\forall_{t^C \in T^C}$, $\exists_{M',M''}$ $M \xrightarrow{*} M' \xrightarrow{t^C} M''$. It is easy to deduce that $\forall_{t^C \in T^C}$, $\exists_{M',M''}$ $i^C \xrightarrow{*} M' \xrightarrow{t^C} M''$. Hence, requirement (iii) of Definition 7 holds.

Since all of the three requirements (i), (ii) and (iii) of Definition 7 hold, CN is CF sound. □

An existing work named CPN Tools [92] can be used to check the liveness and boundedness properties. A $\overline{CN}$ can be analyzed by CPN Tools. The Petri-net-based flow model can be analyzed by CPN Tools, and a report is generated as a result. For the input $\overline{CN}$, if each place p^C in the output report holds $M(p^C) \leq n$ (n is a non-negative integer), then $\overline{CN}$ is bounded. If each t^C belonging to $\overline{CN}$ is in the set of "live transition instances" listed by the report, then $\overline{CN}$ is live. The "dead markings" set and the "dead transitions" set are reported by CPN Tools. Therefore, the reasons for why a C-net is not CF sound are provided.

5.2 Data Flow Soundness

Modeling and verifying the data flow of a business process is as vital as the control flow [65, 83]. In this section, eleven data flow anomalies are identified and formally described by the SBP model defined in Chapter 3. These data anomalies are classified into three types which are data initialization anomalies, data redundancy anomalies, and data version anomalies. A data anomalies detection algorithm is developed.

In [83], the authors provide a method for detecting a set of data flow anomalies. Although there is a formal definition of data flow in this work, the data flow anomalies are only described by scenarios and examples. There is no formal specification for data anomalies, and a well-defined detection algorithm becomes impossible. Both data flow and data anomalies are defined formally in [65]. The authors propose a Petri-net-based approach for formulating the data flow modeling and verification. The detection algorithms for data flow anomalies in [65, 83] can only give the coarse granularity result of a detected anomaly. These algorithms

give the result such as "there is a data initialization anomaly" but can not tell the difference between "delayed initialization anomaly" and "missing input data anomaly".

The existence of the "XOR-split" pattern and "Loop" pattern leads to different practical execution orders of a set of activities due to different selections. There are some "possible execution orders" in the practical execution of these activities. As loop conditions can only affect the repetitions of a set of activities but not the existence of them, the possible execution orders of activities are only generated by the XOR selections. The set of all possible execution orders is denoted as $\Pi = \{\pi_1, \pi_2, ..., \pi_n\}$ which is used for describing some types of data flow anomalies in the following subsections.

5.2.1 Definition of Data Flow Soundness

Type-I Data Initialization Anomalies

1 **Missing Target Data:** When the process terminates, there is a final output target data that has not been initialized.

In Formal: For an IP, $\forall p^D \in P^D$, p^D is a missing target data iff: $t^{tar} \in p^D\bullet$ and $\bullet p^D \subseteq T^I$.

2 **Missing Input Data:** An input data of an activity has never been initialized when the process terminates.

In Formal: For an IP, $\forall p^D \in P^D$, p^D is a missing input data of activity α, iff: $\exists t^I_\alpha \in p^D\bullet$ and $\bullet p^D \subseteq T^I$.

3 **Delayed Initialization:** An input data of an activity has not been initialized when

the activity is enabled, but the input data will be initialized by a subsequent activity before the process terminates.

In Formal: For an IP, $\forall p^D \in P^D$, p^D is a delayed initialization data of activity α, iff: $\exists t^I_\alpha \in p^D\bullet$, $t^{leg} \notin \bullet p^D$, $\bullet p^D \subsetneq T^I$, and $\forall t^O_\beta \in \bullet p^D$, the transition labeled with β is enabled after the transition labeled with α.

4 **Uncertain Initialization:** The availability of an input data of an activity is uncertain when the activity is enabled.

In Formal: For an IP, $\forall p^D \in P^D$, p^D is an uncertain initialization data of activity α, iff: $\exists t^I_\alpha \in p^D\bullet$, $t^{leg} \notin \bullet p^D$, $\bullet p^D \subsetneq T^I$, and $\forall t^O_\beta \in \bullet p^D$, the transition labeled with β is in parallel with the transition labeled with α.

5 **Conditionally Initialized Target Data:** A target data of the process can only be initialized under specific routing condition(s), under other condition(s) it can never be initialized.

In Formal: For an IP, $\forall p^D \in P^D$, p^D is a conditionally initialized target data, iff: $t^{tar} \in p^D\bullet$, $\exists \pi_i \in \Pi$, $\bullet p^D \subseteq T^I$; and $\exists \pi_j \in \Pi$ $(i \neq j)$, $\bullet p^D \subsetneq T^I$.

6 **Conditionally Initialized Input Data:** An input data of an activity can only be initialized under specific routing condition(s), under other condition(s) it can never be initialized.

In Formal: For an IP, $\forall p^D \in P^D$, p^D is a conditionally initialized input data of activity α, iff: $t^I_\alpha \in p^D\bullet$, $\exists \pi_i \in \Pi$, $\bullet p^D \subseteq T^I$, and $\exists \pi_j \in \Pi$ $(i \neq j)$, $\bullet p^D \subsetneq T^I$.

Type-II Data Redundancy Anomalies

7 **Redundant Legacy Data:** A legacy data of the process is neither a target data nor an input data for any activities.

In Formal: For an IP, $\forall p^D \in P^D$, p^D is a redundant legacy data, iff: $t^{leg} \in \bullet p^D$ and $p^D \bullet \subseteq T^I$.

8 **Redundant Output Data:** An output data of an activity is neither a target data nor an input data for any activities.

In Formal: For an IP, $\forall p^D \in P^D$, p^D is a redundant output data of an activity α, iff: $t^O_\alpha \in \bullet p^D$ and $p^D \bullet \subseteq T^I$.

9 **Conditionally Redundant Legacy Data:** A legacy data of the process is an input data of an activity only under certain routing conditions.

In Formal: For an IP, $\forall p^D \in P^D$, p^D is a conditionally redundant legacy data, iff: $t^{leg} \in \bullet p^D$, $\exists \pi_i \in \Pi$, $p^D \bullet \subseteq T^I$, and $\exists \pi_j \in \Pi$ $(i \neq j)$, $p^D \bullet \subsetneq T^I$.

10 **Conditionally Redundant Output Data:** An output data of an activity is an input data of another activity only under certain routing conditions.

In Formal: For an IP, $\forall p^D \in P^D$, p^D is a redundant output data of an activity α, iff: $t^O_\alpha \in \bullet p^D$, $\exists \pi_i \in \Pi$, $p^D \bullet \subseteq T^I$, and $\exists \pi_j \in \Pi$ $(i \neq j)$, $p^D \bullet \subsetneq T^I$.

Type.3 Data Version Anomalies

11 **Multiple Initialized Data:** A data is initialized more than once.

In Formal: For an IP, $\forall p^D \in P^D$, p^D is a multiple initialized data iff: $\exists \pi \in \Pi$, $|\bullet p^D - T^I| > 1$.

Because of the allowance of some types of data anomalies in some practical cases, two degrees of data flow correctness of SBP are defined as follows. DF Soundness (Definition 10) is a more restrictive correctness property which does not allow the existence of any data anomalies in an SBP. Weak DF Soundness (Definition 11) only demands the absence of some specific types of data anomalies. For an SBP that is weak DF sound, the redundant data and multiple versions of a data element are allowed to exist, but any data initialization anomaly is not allowed.

Definition 10. ***(DF Soundness)*** *An SBP is DF sound iff: no data flow anomaly exists.*

Definition 11. ***(Weak DF Soundness)*** *An SBP is weak DF sound iff: no data initialization anomaly exists.*

5.2.2 Verification of Data Flow Soundness

An algorithm (Algorithm 1) for detecting data flow anomalies is developed in this subsection. The correctness verification for the data flow of an SBP can be realized by this algorithm to detect those data anomalies. The input of Algorithm 1 is the internal business process IP of the SBP, and the output is a set of data flow anomalies Λ detected by the algorithm.

Function 1 is proposed for each data element d of the internal business process of the SBP to collect all the activities which utilize the data d as an element of their input parameters into a set In_d, and to collect all the activities who utilize the data d as an element of their

Algorithm 1 Data Flow Anomalies Detection

INPUT: IP // IP is the internal business process of the concerned SBP

OUTPUT: Λ // Λ is the data flow anomaly set

$\Lambda \leftarrow \phi$

for each data element $d \in Data$ of IP **do**

Call **Function 1**

if $Out_d = \phi$ && $In_d \neq \phi$ **then**

Call **Function 2**

else if $In_d = \phi$ && $Out_d \neq \phi$ **then**

Call **Function 3**

else $\{In_d \neq \phi$ && $Out_d \neq \phi\}$

for each $y \in In_d$ **do**

Call **Function 4**

Call **Function 5**

end for

Call **Function 6**

end if

Call **Function 7**

end for

output parameters into a set Out_d. The input of Function 1 is the internal business process IP of the SBP, and the output is two sets of activities In_d and Out_d.

Function 2 of the algorithm is utilized to detect the "***1. Missing Target Data***" anomalies and the "***2. Missing Input Data***" anomalies in the case where a data d is in

Function 1
INPUT: IP

OUTPUT: In_d, Out_d

$In_d \leftarrow \phi$

$Out_d \leftarrow \phi$

for each activity $a \in Act$ of IP **do**

if d is an input data of a **then**

$In_d \leftarrow In_d \cup \{a\}$ //a set of activities who read d

else if d is an output data of a **then**

$Out_d \leftarrow Out_d \cup \{a\}$ //a set of activities who write d

end if

end for

if d is a target data of IP **then**

$In_d \leftarrow In_d \cup \{tar\}$ //tar represents a virtual activity

else if d is a legacy data of IP **then**

$Out_d \leftarrow Out_d \cup \{leg\}$ //leg represents a virtual activity

end if

the input parameters of some activities ($In_d \neq \phi$) but not in the output parameters of any activities ($Out_d = \phi$).

Function 3 of the algorithm is utilized to detect the "***7. Redundant Legacy Data***" anomalies and the "***8. Redundant Output Data***" anomalies in the case where a data d is in the output parameters of some activities ($Out_d \neq \phi$) but not in the input parameters of any activities ($In_d = \phi$).

Function 2
INPUT: IP

OUTPUT: Λ

for each $x \in In_d$ **do**

if $x = tar$ **then**

$\Lambda \leftarrow$ "d is a missing target data"

else

$\Lambda \leftarrow$ "d is a missing input data of x"

end if

end for

Function 3
INPUT: IP

OUTPUT: Λ

for each $x \in Out_d$ **do**

if $x = leg$ **then**

$\Lambda \leftarrow$ "d is a redundant legacy data"

else

$\Lambda \leftarrow$ "d is a redundant onput data of x"

end if

end for

Function 4 of the algorithm is for each activity y who utilizes the data d as an element of its input parameter to collect all the activities which are executed prior to, after, and concurrently in to respective sets $PRE(y)$, $POST(y)$, and $CON(y)$.

Function 4
INPUT: IP

OUTPUT: $PRE(y)$, $POST(y)$, $CON(y)$

for each $z \in Out_d$ **do**

if z is enabled before y **then**

$PRE(y) \leftarrow z$ // a pre-executed activities set of y

else if z is executed after y **then**

$POST(y) \leftarrow z$ // a post-executed activities set of y

else if z is enabled in parallel with y **then**

$CON(y) \leftarrow z$ // a concurrently-executed activities set of y

end if

end for

Function 5 of the algorithm is utilized to detect the "***3. Delayed Initialization***" anomalies, the "***4. Uncertain Initialization***" anomalies, the "***5. Conditionally Initialized Target Data***" anomalies, and the "***6. Conditionally Initialized Input Data***" anomalies in the case where a data d is either in the input parameters of some activities ($In_d \neq \phi$) or in the output parameters of any activities ($Out_d = \phi$).

Function 6 of the algorithm is utilized to detect the "***9. Conditionally Redundant Legacy Data***" anomalies and the "***10. Conditionally Redundant Output Data***" anomalies in the case where a data d is either in the input parameters of some activities ($In_d \neq \phi$) or the output parameters of any activities ($Out_d = \phi$).

Function 7 of the algorithm is utilized to detect the "***11. Multiple Initialized Data***" anomalies in the special case where a data d is in the output parameters of two or more

Function 5
INPUT: IP

OUTPUT: Λ

if $|POST(y)| = |Out_d|$ **then**

$\Lambda \leftarrow$ "d is a delayed initialized input data of y"

else if $|CON(y)| > 0$ && $|PRE(y)| = 0$ && $\forall \pi \in \Pi, \exists z \in CON(y), z \in Act^{\pi}$ **then**

$\Lambda \leftarrow$ "d is an uncertain initialized input data of y"

else if $|CON(y)| > 0$ && $|PRE(y)| = 0$ && $\exists \pi \in \Pi, \forall z \in CON(y), z \notin Act^{\pi}$ **then**

$\Lambda \leftarrow$ "d is an uncertain initialized input data of y"

$\Lambda \leftarrow$ "d is a conditionally initialized input data of y"

else if $|CON(y)| > 0$ && $|PRE(y)| > 0$ && $\exists \pi \in \Pi, \forall z \in PRE(y), z \notin Act^{\pi}$ && $\exists \pi \in \Pi, \forall z \in CON(y), z \notin Act^{\pi}$ **then**

$\Lambda \leftarrow$ "d is an uncertain initialized input data of y"

$\Lambda \leftarrow$ "d is a conditionally initialized input data of y"

else if $|CON(y)| = 0$ && $|PRE(y)| > 0$ && $\exists \pi \in \Pi, \forall z \in PRE(y), z \notin Act^{\pi}$ && $y = tar$ **then**

$\Lambda \leftarrow$ "d is a conditionally initialized target data"

else if $|CON(y)| = 0$ && $|PRE(y)| > 0$ && $\exists \pi \in \Pi, \forall z \in PRE(y), z \notin Act^{\pi}$ && $y \neq tar$ **then**

$\Lambda \leftarrow$ "d is a conditionally initialized input data of y"

end if

activities ($|Out_d| > 1$) in one execution instance ($\exists \pi \in \Pi, |Act^{\pi} \cap Out_d| > 1$).

Function 6
INPUT: IP

OUTPUT: Λ

for each $x \in Out_d$ **do**

if $\exists \pi \in \Pi, \forall y \in In_d, y \notin Act^{\pi}$ && $x = leg$ **then**

$\Lambda \leftarrow$ "d is a conditionally redundant legacy data"

else if $\exists \pi \in \Pi, \forall y \in In_d, y \notin Act^{\pi}$ && $x \neq leg$ **then**

$\Lambda \leftarrow$ "d is a conditionally redundant output data of x"

end if

end for

Function 7
INPUT: IP

OUTPUT: Λ

if $|Out_d| > 1$ && $\exists \pi \in \Pi, |Act^{\pi} \cap Out_d| > 1$ **then**

$\Lambda \leftarrow$ "d is a multiple initialized data by Out_d"

end if

5.3 Consistency of Provided Service

Business can be dynamic due to regulation and customer requirements change. As a result, both internal business process and the involved services may need to change accordingly. Because of the complex dependencies between the internal business process and involved services of an SBP, the changes occurring in one element can affect the others and propagate in the SBP with a cascading manner [36, 37] that can easily cause the inconsistency between a provided service and the internal business process. It is demanding to ensure that the

internal business process is *consistent* with the provided services of an SBP. For verifying the consistency, an algebra-based method is proposed in this section which focuses on the control flow behaviors of the internal business process and provided services of an SBP.

5.3.1 Definition of Consistency

From the perspective of the service provider, the consistency of a provided service requires that the flow relation of input and output messages performed by the service interface is same with the order of the respective messages delivered by the internal business process. As an example shown in Figure 5.5, the internal business process 1 and its provided service 1 have the same flow relation for dealing with the respective messages that are to input the message *Inquiry 1* and then to input the message *Inquiry 2*, so they are consistent with each other. As the changed service 1, service $1'$ requires its operations in a parallel control flow relation which is not sequential anymore so that the message *Inquiry 1* and the message *Inquiry 2* can be handled without fixed order. So the internal business process 1 is not consistent with the provided service $1'$.

Due to the different process-service relation patterns can be contained in an SBP as discussed in Section 4.2, the order of input and output messages cannot be shown completely by the original SBP model proposed in Chapter 3. Therefore, in order to define and verify the consistency between a provided service and the internal business process of an SBP, a set of corresponding refinement rules are proposed firstly for refining the C-net of the internal business process and the S-net of the provided service based on the process-service relation patterns. The refined C-net is named as rC-net and the refined S-net is named as rS-net.

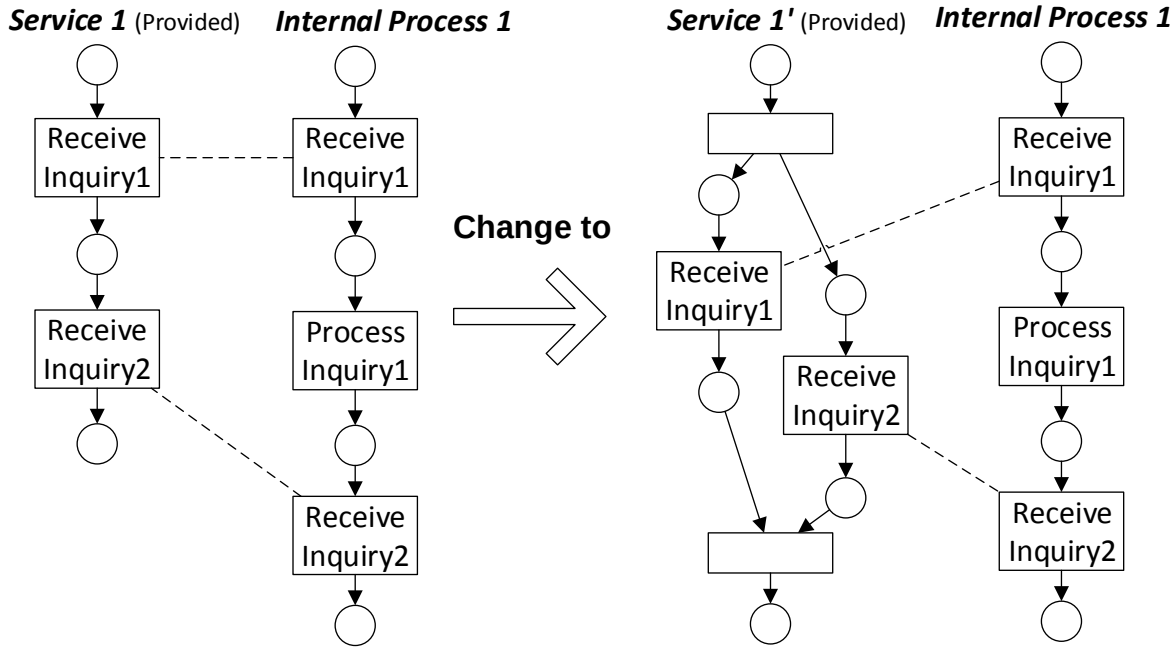

Figure 5.5: Consistency Between a Provided Service and The Internal Business Process of an SBP

***Note that the same refinement rules will be used in the next section for verifying the compatibility between an invoked service and the internal business process of an SBP.**

For each activity or operation in one of the four relation patterns, the corresponding refinement rules are shown as follows:

- **Rule 1** (For 1A-to-1syncO Pattern): Replace the activity transition by two new sequential ordered activity transitions representing the acts of "send" and "receive" separately. The order depends on the message transmission pattern of associated operation. Implement the same way on the operation.

  As shown in Figure 3.1, the activity "*Make Payment*" of the internal business process and the operation "*Make Payment*" of the provided service "***Flight Inquiry &***

Booking" are in a 1A-to-1syncO pattern. Based on Rule 1, respective fragments of the C-net of the internal business process and the S-net of the invoked service are refined as shown in Figure 5.6. The replaced parts are in gray.

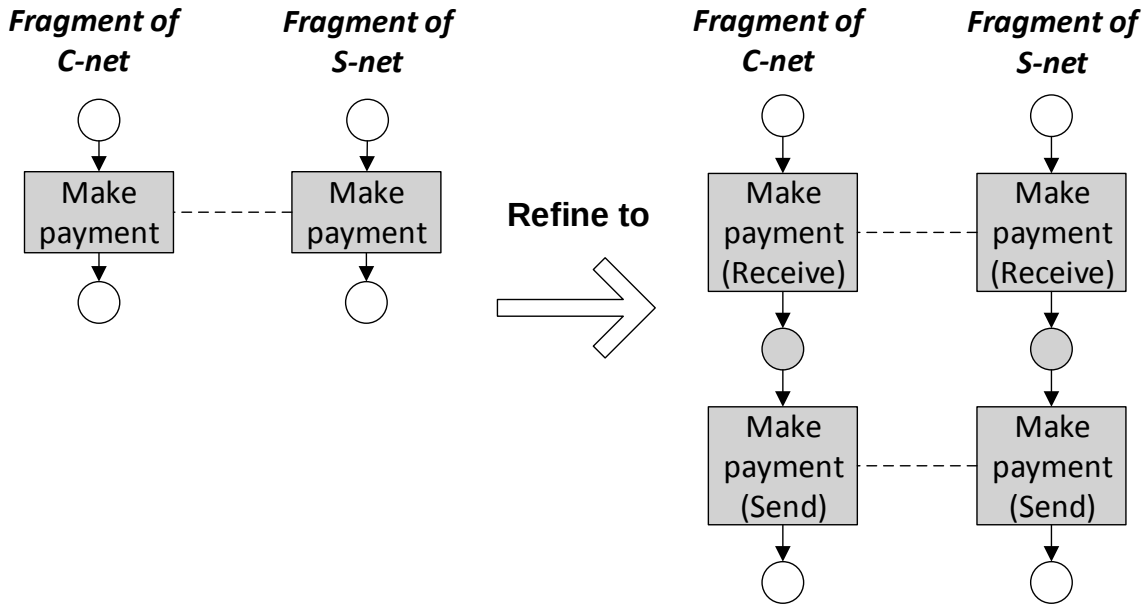

FIGURE 5.6: Refinement Rule 1 for 1A-to-1syncO Pattern

- **Rule 2** (For 1A-to-1asyncO Pattern): No action.

- **Rule 3** (For 2A-to-1syncO Pattern): Replace the operation transition by two new sequential ordered operation transitions representing the acts of "send" and "receive" separately. The order depends on the message transmission pattern of the operation.

  As shown in Figure 3.1, the activities "*Receive Inquiry*" and "*Send Acknowledgment*" of the internal business process are in a 2A-to-1syncO relation with the operation "*Start Inquiry*" of the provided service "***Flight Inquiry & Booking***". Based on Rule 3, respective fragments of the C-net of the internal business process and the S-net of the

invoked service are refined as shown in Figure 5.7. The replaced parts are in gray.

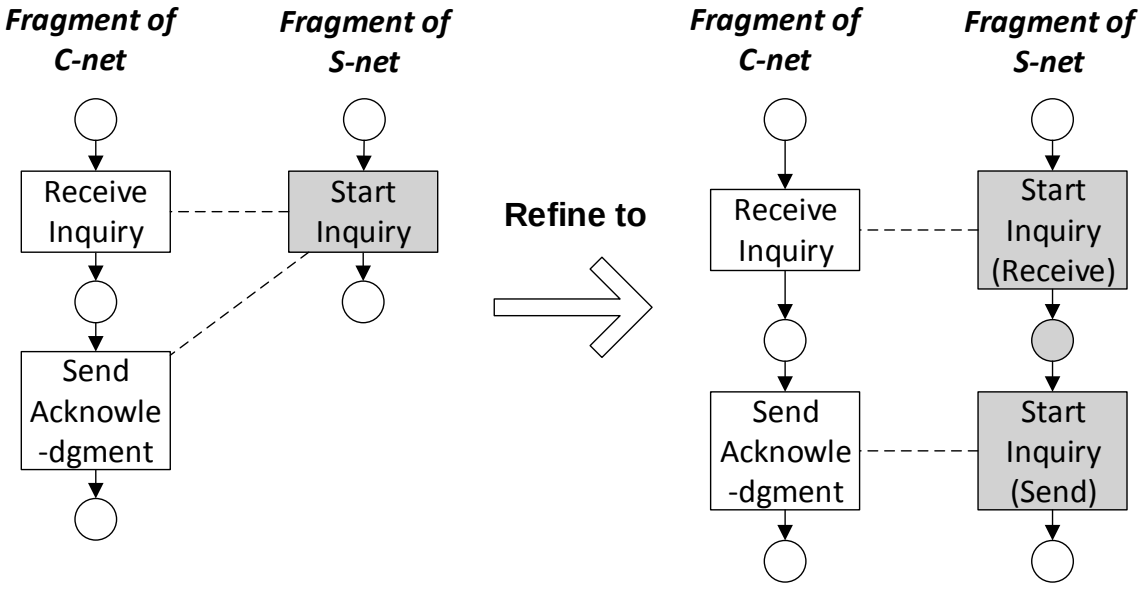

Figure 5.7: Refinement Rule 3 for 2A-to-1syncO Pattern

- **Rule 4** (For 1A-to-2asyncO Pattern): Replace the activity transition by two new sequential ordered activity transitions representing the acts of "send" and "receive" separately. The order depends on the message transmission pattern of each associated operation.

  As shown in Figure 3.1, the activity "*Check Available Flights*" of the internal business process is in a 1A-to-2asyncO relation with the operations "*Receive Request*" and "*Send Flights Information*" of the invoked service "***Flight Service***". Based on Rule 4, respective fragments of the C-net of the internal business process and the S-net of the invoked service are refined as shown in Figure 5.8. The replaced parts are in gray.

According to the refined C-net and S-net, the consistency between a provided service and

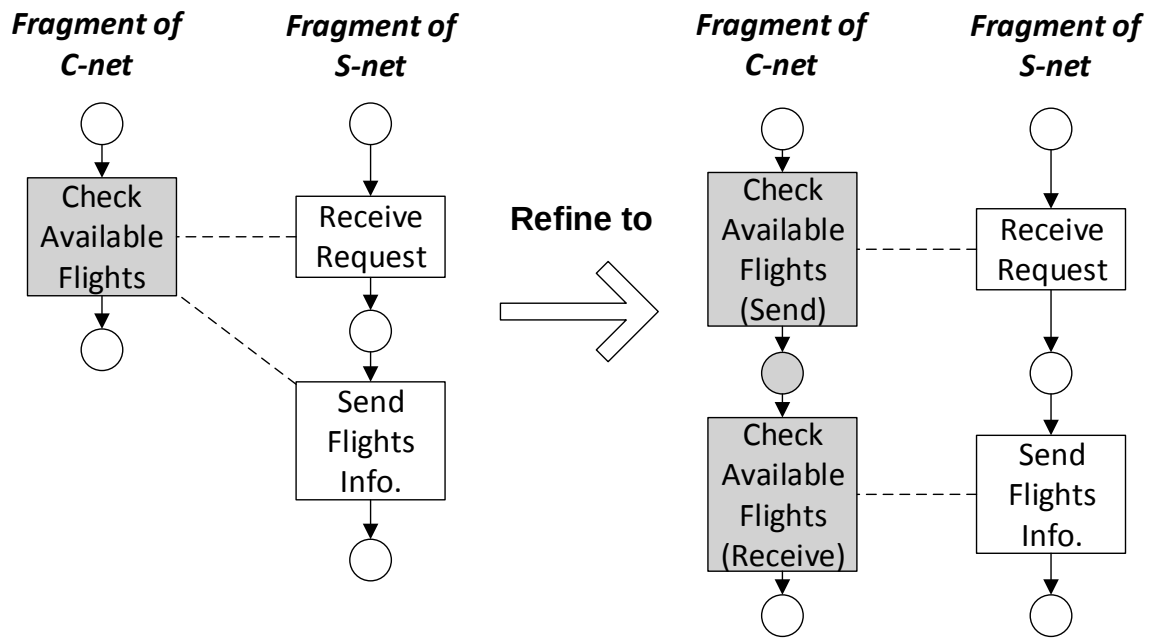

FIGURE 5.8: Refinement Rule 4 for 1A-to-2asyncO Pattern

the internal business process of an SBP are defined in Definition 12.

Definition 12. ***(Consistency)*** *For an SBP, a provided service S and the internal business process IP are consistent with each other iff, the operations of S and respective activities related with these operations of IP have the same control flow relations.*

5.3.2 Verification of Consistency

As there is an arising need for modeling a broad range of business processes on a recurring basis and describing them in an imperative way [34], many types of control flow structures can exist in an SBP. Especially for verifying the consistency between a provided service and the internal business process of an SBP, existing works [89–91] are lack of the support to some specific control flow structures (e.g., network structure). For verifying the consistency

between a provided service and the internal business process of an SBP, a model reduction method need to be applied at first on the control flow of the internal business process which enables further comparison between the reduced control flow and a corresponding provided service. Traditional model reduction methods in existing works [89–91] can only deal with the symmetrical control structures in an internal business process.

As a concept from structured programming, a symmetrical control structure can only contain symmetrical blocks with well-defined start and end nodes, e.g., sequence blocks, parallel blocks, choice blocks, and loop blocks [85]. The reduction methods in existing works work well with the processes that only contain symmetrical control flow structures, because of the feature of symmetrical structures that can be nested. However, these traditional reduction methods cannot handle network structures. Therefore, it could be challenging to deal with complex control flow structures of an SBP, in particular, network structures.

To deal with this issue, an algebra-based method is developed in this work. An algebra expression is generated as the "travel log" of the journey of tokens through a rC-net which departs from i^C and finally arrives at o^C. Because the possible existence of the choice and loop branches, some of such branches may not be executed or be executed multiple times in one execution of the process. In order to make sure that the token initialized in i^C can finally reach o^C and all transitions can be executed only once, some "normalization" is carried out on the places which have more than one input or output transitions.

Normalization on rC-net

As shown in Figure 5.9, one type of places which have more than one output transitions is the XOR-split place. The normalization on this type of places is to transfer them into

corresponding XOR-split transitions. The transferred parts are in gray.

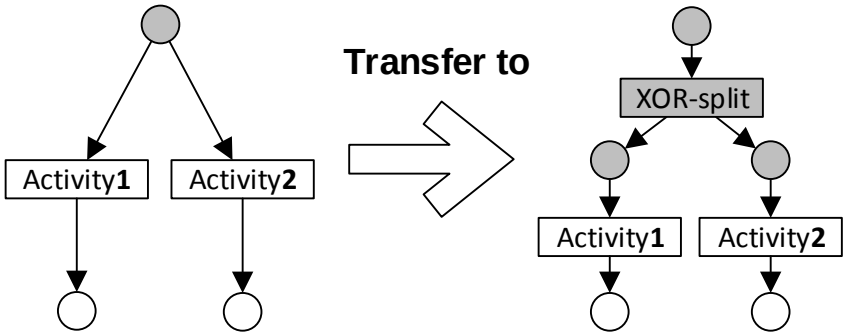

FIGURE 5.9: Normalization on XOR-split Place

As shown in Figure 5.10, one type of places which have more than one input transitions is the XOR-join place. The normalization on this type of places is to transfer them into corresponding XOR-join transitions. The transferred parts are in gray.

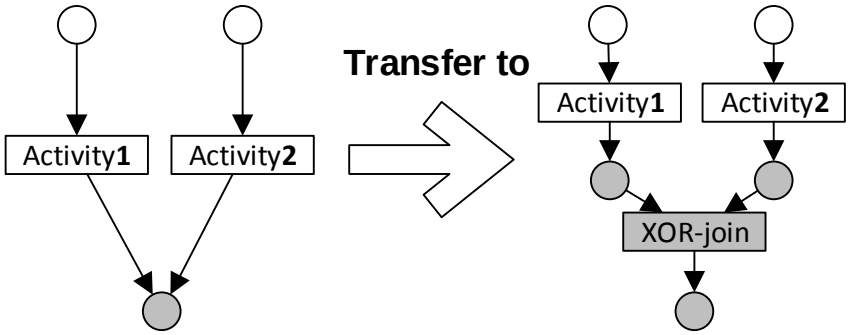

FIGURE 5.10: Normalization on XOR-join Place

As shown in Figure 5.11, one type of places which have more than one input transitions is the Loop-in place. The normalization on this type of places is to transfer them into corresponding Loop-in transitions. One type of places which have more than one output

transitions is the Loop-out place. The normalization on this type of places is to transfer them into corresponding Loop-out transitions. The transferred parts are in gray, and the directions of the arcs connecting the Loop-transition and Loop-in place are changed.

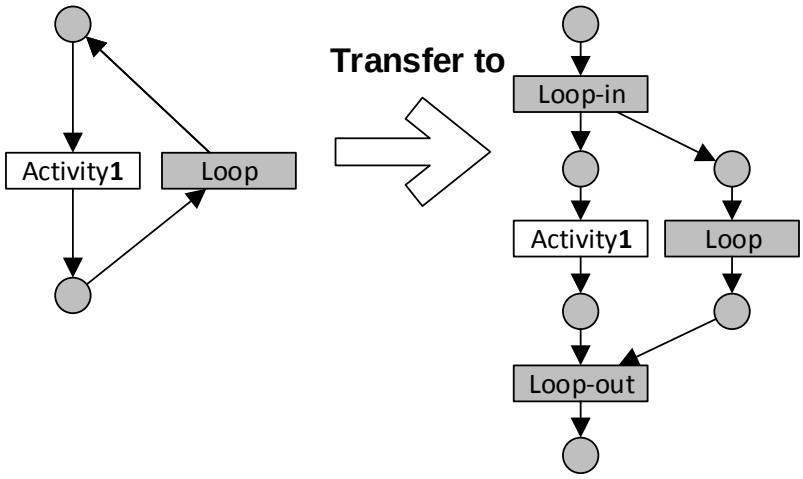

Figure 5.11: Normalization on Loop-in and Loop-out Places

Six Steps of The Postmarking Method

Because there is no place with more than one input arcs in transferred rC-net, it is easy to prove that in every marking of rC-net, the number of tokens in each place is 0 or 1. A postmark PM^C can be "stamped" on a token after it "passed through" a specific transition of the transferred rC-net. The postmarking steps and corresponding algebra are presented as follows:

1. The token in the initial marking i^C of the rC-net is with an initial postmark I. An example is shown in Figure 5.12.

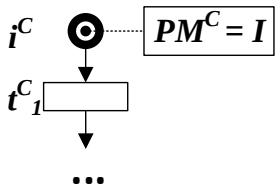

FIGURE 5.12: An Example of Postmarking Step 1

2. If a transition is labeled by an activity a_j which is associated with the concerned provided service S (i.e., $\langle a_j, S\rangle \in R_S$), and there is a token postmarked with x in the input place of this transition, the token in the output place of this transition will be postmarked with $x \blacktriangleright a_j$. An example is shown in Figure 5.13.

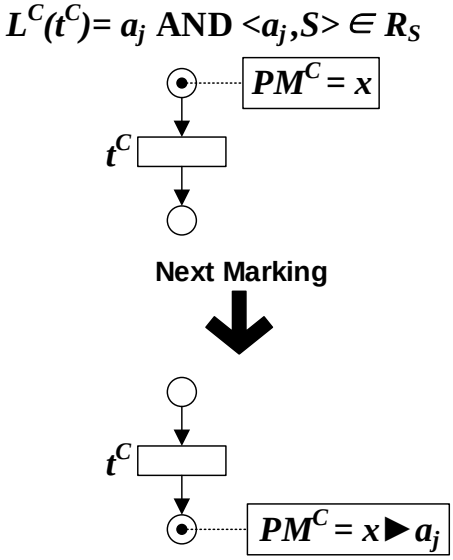

FIGURE 5.13: An Example of Postmarking Step 2

3. If a transition has $n(n > 1)$ input places with tokens postmarked with $x_1, x_2, ..., x_n$, the token in the output place of this transition will be postmarked with $x_1 \blacklozenge x_2 \blacklozenge x_3$

♦...♦x_n. An example is shown in Figure 5.14.

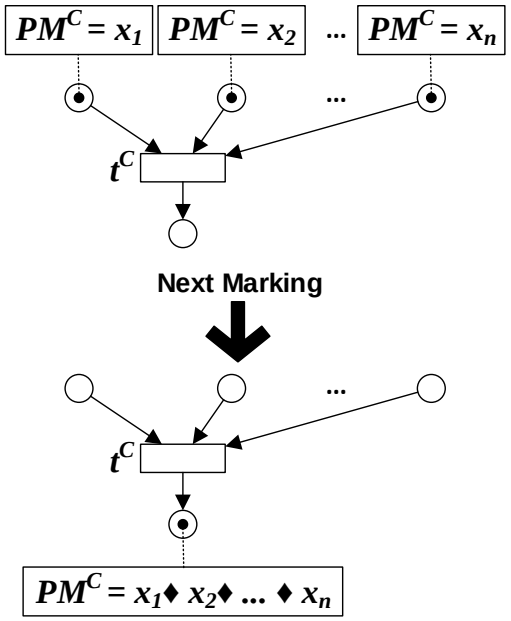

Figure 5.14: An Example of Postmarking Step 3

4. If a transition does not belong to the types of transitions which are discussed in step 2 and step 3, the token in its output place is postmarked the same as the token in its input place.

5. After a token is postmarked with the form in step 3, an operation similar to factorization of polynomials is performed immediately. In this operation, the symbol ► is treated as a multiplication relation (i.e., "×") but without the commutative law and the symbol ♦ is treated as an addition relation (i.e., "+"). The usage of specific types of brackets here is not based on the traditional rules in polynomial decomposition but depends on the type of the "passed through" transition which has more than one input places. The usage rules for brackets are (examples are shown in Figure 5.15):

i. "( )" is for the AND-join transitions.

ii. "[ ]" is for the XOR-join transitions.

iii. "{ }" is for the Loop-out transitions.

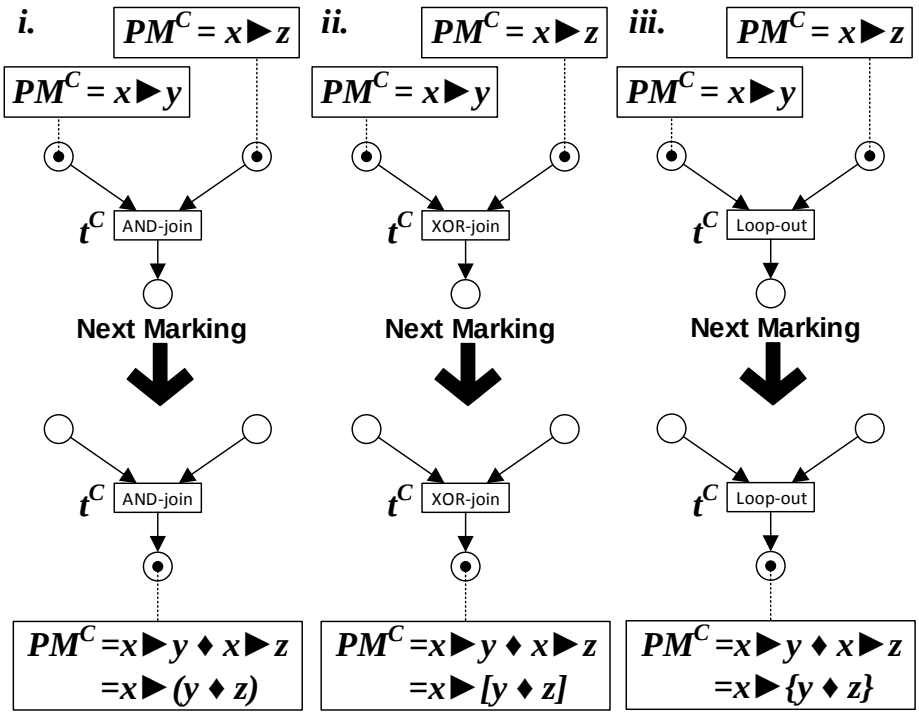

FIGURE 5.15: Examples of Using Brackets in Step 5

Note that if the same type of brackets are nested, only the outside bracket set will be kept. For example:

$$PM^C = I \blacktriangleright x \blacklozenge I \blacktriangleright (y \blacklozenge z) = I \blacktriangleright (x \blacklozenge (y \blacklozenge z)) = I \blacktriangleright (x \blacklozenge y \blacklozenge z)$$

.

6. After step 5, the function A on the postmark PM^C has the output as (here $y, z, z_1, ...z_m$ are postmark expressions):

$$A(PM^C) = \begin{cases} & \text{IF } PM^C \text{ IS IN FORM AS:} \\ y & y \blacktriangleright (n) \text{ or } y \blacktriangleright [n] \text{ or } y \blacktriangleright \{n\}\ (n \in \mathbb{N}) \\ y \blacktriangleright z & y \blacktriangleright (z \blacklozenge n),\ (n \in \mathbb{N}) \\ y \blacktriangleright (z_1 \blacklozenge z_2 \blacklozenge ... \blacklozenge z_m) & y \blacktriangleright (z_1 \blacklozenge z_2 \blacklozenge ... z_m \blacklozenge n),\ (m > 1, n \in \mathbb{N}) \\ y \blacktriangleright [z_1 \blacklozenge z_2 \blacklozenge ... \blacklozenge z_m \blacklozenge 1] & y \blacktriangleright [z_1 \blacklozenge z_2 \blacklozenge ... z_m \blacklozenge n],\ (m > 0, n \in \mathbb{N}) \\ y \blacktriangleright \{z \blacklozenge 1\} & y \blacktriangleright \{z \blacklozenge n\},\ (n \in \mathbb{N}) \\ PM^C & \text{OTHERWISE} \end{cases}$$

An Example to Demonstrate The Postmarking Method

Figure 5.16 shows an example to demonstrate the proposed postmarking method. The activities which are related to the concerned provided service are marked with gray color. The activity "***a3***" of the internal business process is related with the operation "***o1***" of the provided service. The activity "***a2***" of the internal business process is related with the operation "***o2***" of the provided service. According to the refinement conditions and normalization conditions identified in this section, there is no need for any refinement and normalization on the C-net of this internal business process.

As shown in Figure 5.17 and Figure 5.18, the C-net of the internal business process has seven markings. The "consumed" tokens by the fired transitions are shown with dashed line in each marking.

- $\boldsymbol{M_0}$ - In the first marking M_0, one token is in the initial place i^C and the transition $AND - split1$ is enabled.

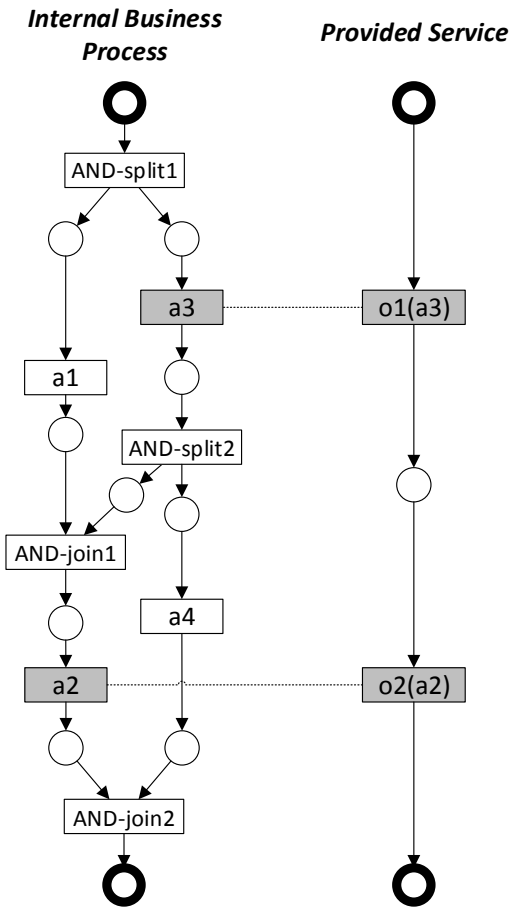

FIGURE 5.16: An Example to Demonstrate The Postmarking Method

According to the step 1 of the postmarking method, the only one token in the initial place i^C of the C-net is postmarked with I, i.e.,

$$PM^C(token1) = I.$$

- $\boldsymbol{M_1}$ - In the second marking M_1, after the transition $AND-split1$ fired, the transitions $a1$ and $a3$ are enabled.

  According to the step 4 of the postmarking method, because the transition $AND-split1$ is not in the same type with the transitions discussed in step 2 and step 3

of the postmarking method, each token in the two output places of the transition is postmarked the same as the token in its input place, i.e.,

$$PM^C(token1) = I,$$

$$PM^C(token2) = I.$$

- $\boldsymbol{M_2}$ - In the third marking M_2, after the transitions $a1$ and $a3$ fired, the transition $AND - split2$ is enabled.

  According to the step 4 of the postmarking method, because the transition $a1$ is not in the same type with the transitions discussed in step 2 and step 3 of the postmarking method, the token 1 in the output place of the transition is postmarked the same as the token in its input place.

  According to the step 2 of the postmarking method, because the activity $a3$ is related with the provided service, the token 2 in the output place of the transition $a3$ is postmarked with $I \blacktriangleright a3$, i.e.,

$$PM^C(token1) = I,$$

$$PM^C(token2) = I \blacktriangleright a3.$$

- $\boldsymbol{M_3}$ - In the fourth marking M_3, after the transition $AND-split2$ fired, the transitions $AND - join1$ and $a4$ are enabled.

  According to the step 4 of the postmarking method, because the transition $AND - split2$ is not in the same type with the transitions discussed in step 2 and step 3

of the postmarking method, each token in the two output places of the transition is postmarked the same as the token in its input place, i.e.,

$$PM^C(token1) = I,$$

$$PM^C(token2) = I \blacktriangleright a3,$$

$$PM^C(token2) = I \blacktriangleright a3.$$

- $\boldsymbol{M_4}$ - In the fifth marking M_4, after the transitions $AND - join1$ and $a4$ fired, the transition $a2$ is enabled.

  According to the step 3 of the postmarking method, because the transition $AND - join1$ has two input places, the token 1 in the output place of the transition $AND - join1$ is postmarked with $I \blacklozenge I \blacktriangleright a3$.

  According to the step 5 of the postmarking method, by performing the factorization of polynomials on the postmark of the token 1 in the output place of the transition $AND - join1$, the postmark of token 1 is transferred to $I \blacktriangleright (1 \blacklozenge a3)$.

  According to the step 6 of the postmarking method, because the postmark of the token 1 is in the form as $y \blacktriangleright (z \blacklozenge n), (n \in \mathbb{N})$, by performing the function A, the postmark of the token 1 is finally transferred to $I \blacktriangleright a3$.

  According to the step 4 of the postmarking method, because the transition $a4$ is not in the same type with the transitions discussed in step 2 and step 3 of the postmarking method, the token 2 in the output place of the transition is postmarked the same as the token in its input place, i.e.,

$$PM^C(token1) = I \blacktriangleright a3,$$

$$PM^C(token2) = I \blacktriangleright a3.$$

- $\boldsymbol{M_5}$ - In the sixth marking M_5, after the transition $a2$ fired, the transition $AND-join2$ is enabled.

  According to the step 2 of the postmarking method, because the activity $a2$ is related with the provided service, the token 1 in the output place of the transition $a2$ is postmarked with $I \blacktriangleright a3 \blacktriangleright a2$, i.e.,

$$PM^C(token1) = I \blacktriangleright a3 \blacktriangleright a2,$$

$$PM^C(token2) = I \blacktriangleright a3.$$

- $\boldsymbol{M_6}$ - In the seventh marking M_6, after the transition $AND-join2$ fired, the execution of the C-net terminates.

  According to the step 3 of the postmarking method, because the transition $AND-join2$ has two input places, the token 1 in the output place of the transition $AND-join2$ is postmarked with $I \blacktriangleright a3 \blacktriangleright a2 \blacklozenge I \blacktriangleright a3$.

  According to the step 5 of the postmarking method, by performing the factorization of polynomials on the postmark of the token 1 in the output place of the transition $AND-join2$, the postmark of token 1 is transferred to $I \blacktriangleright a3 \blacktriangleright (a2 \blacklozenge 1)$.

  According to the step 6 of the postmarking method, because the postmark of the token 1 is in the form as $y \blacktriangleright (z \blacklozenge n), (n \in \mathbb{N})$, by performing the function A, the postmark of the token 1 is finally transfered to $I \blacktriangleright a3 \blacktriangleright a2$, i.e.,

$$PM^C(token1) = I \blacktriangleright a3 \blacktriangleright a2.$$

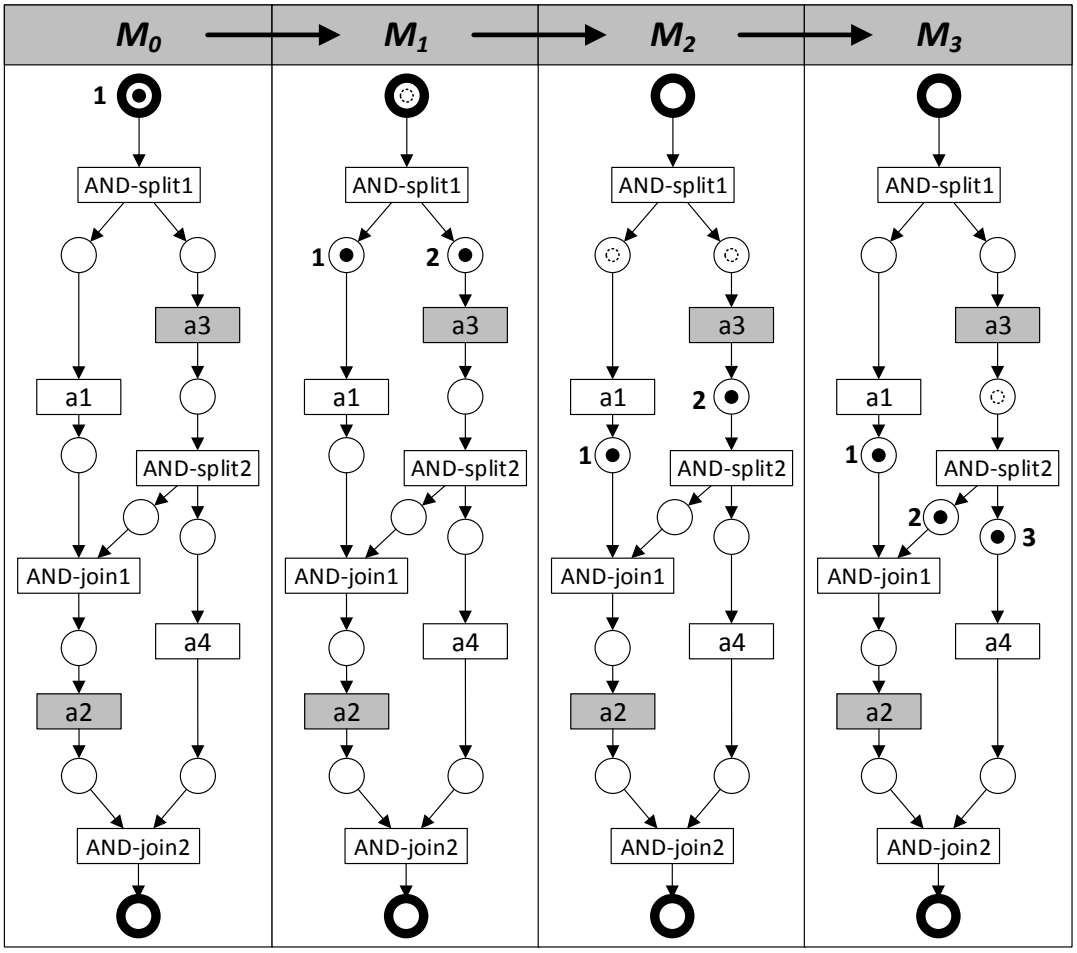

FIGURE 5.17: Marking M_0 to M_3 of the C-net Example

Verifying The Consistency

The similar postmarking method can be performed on the S-net of the concerned provided service. The only difference with the method on the C-net of the internal business process is in step 2, if a transition is labeled by an operation op_i and there is a token postmarked with x in the input place of this transition, the token in the output place of this transition will be postmarked with $x \blacktriangleright a_j$, where a_j is the activity related with the operation op_i (i.e., $\langle a_j, op_i \rangle \in R_O$).

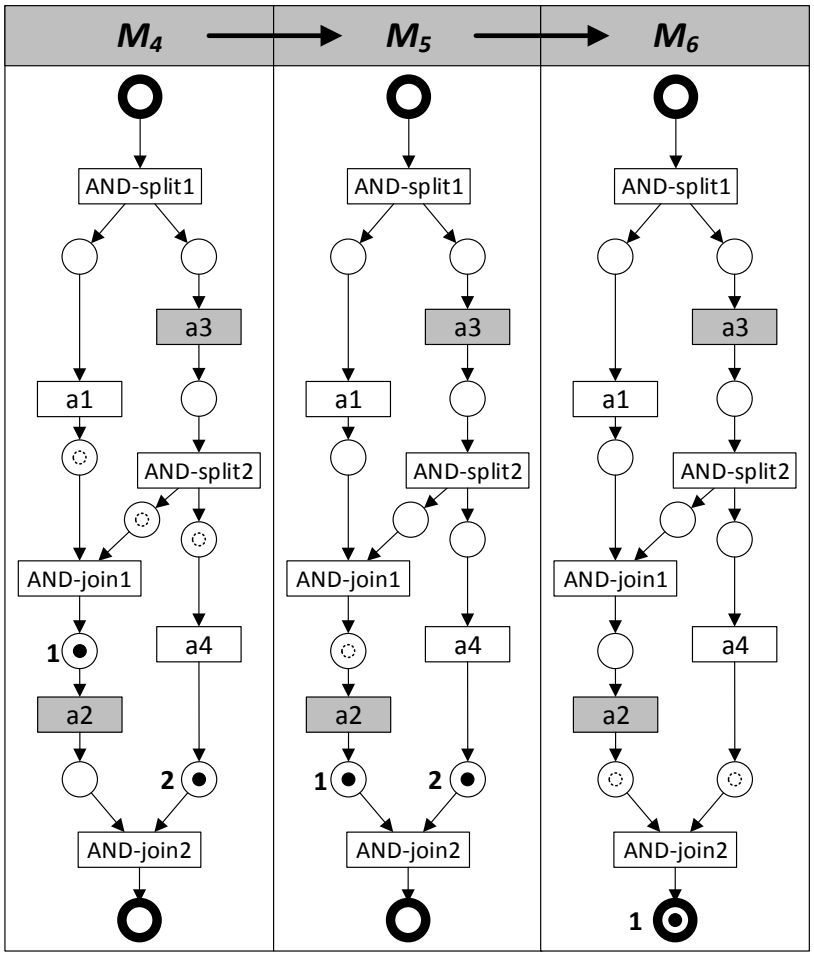

Figure 5.18: Marking M_4 to M_6 of the C-net Example

Then, a comparison can be carried out on the postmark PM^C of the token in the ending place o^C of the C-net and the postmark PM^S of the token in the ending place o^S of the S-net. It is easy to prove that if the two postmarks are with the equivalent algebraic expression, the operations of the provided service and respective activities related with these operations of the internal business process have the same control flow relations. Therefore, according to Definition 12, Theorem 2 is established.

Theorem 2. *For an SBP, if the postmark PM^C of the token in the ending place o^C of*

the C-net and the postmark PM^S of the token in the ending place o^S of the S-net of a provided service meet $PM^C = PM^S$, the internal business process and the provided service are consistent with each other.

5.4 Compatibility of Invoked Service

Business can be dynamic due to regulation and customer requirements change. As a result, both internal business process and the involved services may need to change accordingly. Because of the complex dependencies between the internal business process and involved services of an SBP, the changes occurring in one element can affect the others and propagate in the SBP with a cascading manner [36, 37] that can easily cause the incompatibility between an invoked service and the internal business process. It is demanding to ensure that the internal business process is *compatible* with the invoked services of an SBP. For verifying the compatibility, a Petri-net-based method is proposed in this section which focuses on analyzing the interactions between the internal business process and invoked services of an SBP.

5.4.1 Definition of Compatibility

From the perspective of the service requester, the compatibility of an invoked service requires that there is no conflict during the interactions with the invoked service. As an example shown in Figure 5.19, the internal business process 1 initiates the interaction by sending an inquiry to the invoked service 1 firstly. Then the internal business process waits for the feedback from the invoked service 1. Finally, after the internal business process receives the

feedback, a rating of service is given to the invoked service. On the service side, the rating can be received at any time which is not forced to be received after sending the feedback to its requester. It is easy to see that there is no any conflict of interaction between the internal business process and the invoked service, so they are compatible with each other. As the changed service 1, service $1'$ requires that the rating must be received before the feedback is sent to the requester. After the invoked service $1'$ receives the inquiry from the requester, there is a conflict that both the requester's internal business process and the invoked service are waiting for messages correspondingly generated by the subsequent operation or activity. So the internal business process 1 is not compatible with the provided service $1'$.

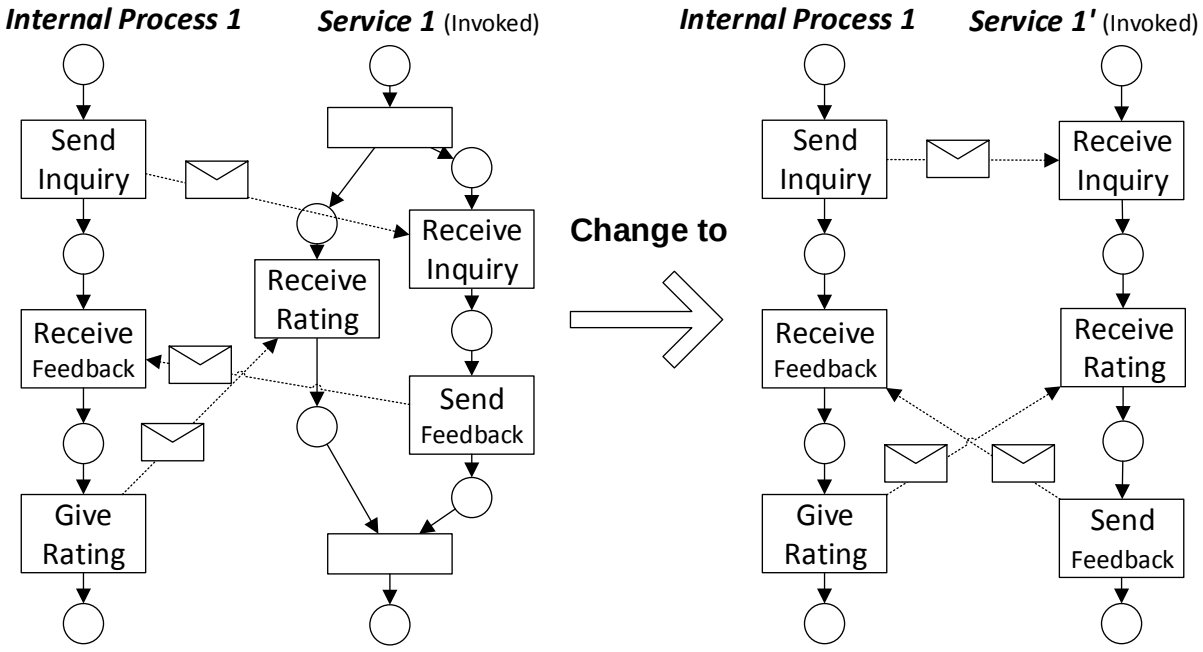

Figure 5.19: Compatibility Between an Invoked Service and the Internal Business Process of an SBP

Due to the different process-service relation patterns can be contained in an SBP as discussed in Section 4.2, the order of input and output messages cannot be shown completely

by the original SBP model proposed in Chapter 3. Therefore, in order to define and verify the compatibility between an invoked service and the internal business process of an SBP, a set of corresponding refinement rules are proposed in sub Section 5.3.1 for refining the C-net of the internal business process and the S-net of the invoked service based on the process-service relation patterns. The refined C-net is named as rC-net and the refined S-net is named as rS-net.

According to the refined C-net and S-net, the compatibility between an invoked service and the internal business process of an SBP are defined in Definition 13.

Definition 13. ***(Compatibility)*** *For an SBP, an invoked service S and the internal business process IP are compatible with each other iff, there is no any conflict during the interaction between them.*

5.4.2 Verification of Compatibility

According to Definition 13, for a compatible invoked service of an SBP, conflicts in the interaction between the internal business process and the invoked service must be avoided. To verify the compatibility between the internal business process and an invoked service, some "normalization" is carried out on the rC-net of the internal business process and the rS-net of an invoked service of a being verified SBP. The rC-net and the rS-net are combined by connecting each operation transition of the rS-net and its corresponding related activity transition of the rC-net with a place named as interface place, and directions of the arcs represent message flow depending on the message transmission types of the operations (Definition 5). An overall initial place i that connects i^C of the rC-net and i^S of the rS-net

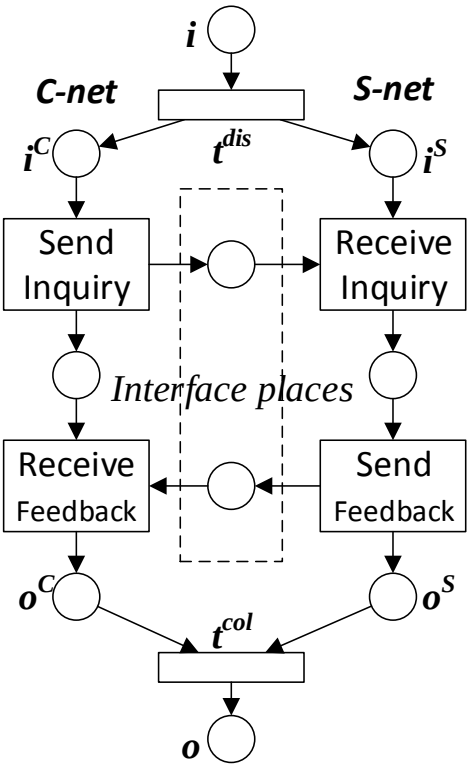

Figure 5.20: An Example of the Normalization on rC-net and rS-net

by a distributor transition t^{dis}, an overall ending place o that connects o^C of the rC-net and o^S of the rS-net by a collector transition t^{col}. An example of the normalization is shown in Figure 5.20.

As indicated in Definition 13, for verifying the compatibility between an invoked service and the internal business process of an SBP, there is a need for a method which is able to determine the existence of conflicts during the interactions between the service and the process. It is easy to prove that if the combined net (containing the rC-net of the internal business process and the rS-net of the invoked service) is CF sound (Definition 7), the interaction between the service and the process is conflict-free so that he Theorem 3 holds.

Therefore, the compatibility can be verified by utilizing the same verification method for the CF soundness in Section 5.1.2.

Theorem 3. *For an invoked service and the internal business process of an SBP, if the combined net of their respective rS-net and rC-net is CF sound, they are compatible with each other.*

5.5 Discussion

In this chapter, for verifying the correctness of an SBP, a set of correctness properties have been identified firstly which are CF soundness of control flow, DF soundness and weak DF soundness of data flow, consistency of provided services, and compatibility of invoked services. Then, the verification methods of these correctness properties are developed correspondingly based on the Petri-net-based SBP model proposed in Chapter 3.

For verifying the CF soundness, the issue is transformed into verifying the liveness and boundedness properties by the assistance of CPN tools and respective proof of the equivalent transformation is given. Approaches at the early time for modeling and analyzing business processes by Petri net are in the workflow area. One of the most famous approaches is Workflow nets [61, 77, 78]. These early works mainly focus on the process structure but neglect the data flow inside.

For verifying the DF soundness and the weak DF soundness, a set of data flow anomalies are identified, and an anomaly detection algorithm is developed correspondingly. The proposed data anomaly detection method can provide more details of an anomaly rather than only a simple type in other works [65, 83].

Because provided services and invoked services have different features, two correctness properties, consistency of provided services and compatibility of invoked services, are identified accordingly. For verifying the consistency of provided services, an algebra-based method is developed. An algebraic expression is generated by the method as the "postmark" on a token of a C-net or an S-net. The proposed method can support both the symmetrical and network structures in the control flow of an internal business process. The latter cannot be handled in the existing works [89–91].

For verifying the compatibility of invoked services, a Petri-net-based method is developed. Some refinement is implied at first on the C-net of the internal business process and the S-net of an invoked service which combines these two nets into one combined net. The verification of the compatibility is transformed into verifying the CF soundness of the combined net.

In the following chapters, the change management of SBPs will be discussed in details by the assistance of the SBP model and design patterns in Chapter 3&4 and the correctness verification of SBP in this chapter.

6

Change Patterns of Service-based Business Processes

Due to the distributed and dynamic nature, changes of the internal business process and involved services of an SBP can happen often. There are dependencies between the internal

business process and involved services [35], therefore, the changes occurring in one side may affect the other one in certain degrees and can propagate in the whole SBP like the cascading effect which makes the change management in SBP a really challenging problem [36, 37].

A change of an activity belonging to a process may affect other activities and related services of the process. A service change may require further changes of one or multiple activities of the process. Change propagation refers to that a single change of an activity or a service causes a series of changes associated with activities and/or services. In order to deal with complicated situations of SBP changes, it is necessary to have a mechanism of change management. The foundation of the change management of SBP is to identify the common change types or change patterns. In this chapter, a set of change patterns of SBP are identified based on the common changes frequently occurring in the real world cases. These change types or patterns can be utilized as enabling tools for SBP designers to transform the ad-hoc change requirements into general changes and decompose complex changes into several change primitives. Each change pattern introduced in this chapter is described in both informal and formal specification. The formal specification of a change pattern is based on the SBP model defined in Chapter 3. A change of an SBP can be represented by a set of operations performed on the SBP model.

The rest of this chapter is organized as follows. In Section 6.1, the change patterns of the internal business process of an SBP are identified. In Section 6.2, the change patterns of the services involved in an SBP are identified. Further discussion about this chapter is in Section 6.3.

6.1 Change Patterns of Internal Business Processes

6.1.1 Activity Existence Changes

This type of changes refers to like the changes of the existence of activities. Only the "delete" and "insert" operation can change the existence of an activity. The "replace" operation is also considered in this work as a combination which "deletes" the targeting activity at first and then "inserts" another activity to the position of the deleted activity.

PC-1 ***Delete an activity***

Description: The *delete an activity* change refers to that an activity is removed from the internal business process of an SBP.

In Formal: Assume that the deleted activity is a.

For the set of activities Act' of the changed internal business process IP',

$$Act' = Act - \{a\}.$$

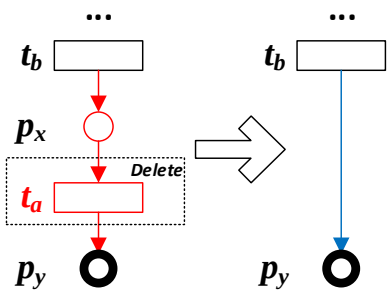

Figure 6.1: An Example of C-net Changes in PC-1

For calculating the changed control flow net CN' of the changed internal business

process IP', there are three different types of operations for this change in respective cases. The differences between these cases depend on the different possible positions of the transition t_a that is labeled with the activity a (i.e.,$L^C(t_a) = a$) in the original control flow net CN (Figure 6.1 shows an example of case ii.):

i. In the case of: in CN, the input place of t_a is p_x, the output place of t_a is p_y which has only one input transition t_a and one output transition t_b. For CN',

$$T'^C = T^C - \{t_a\},$$

$$P'^C = P^C - \{p_y\},$$

$$F'^C = F^C \cup \{\langle p_x, t_b\rangle\} - \{\langle p_x, t_a\rangle, \langle t_a, p_y\rangle, \langle p_y, t_b\rangle\}.$$

ii. In the case of: in CN, the input place of t_a is p_x which has only one input transition t_b and one output transition t_a, the output place of t_a is p_y which is not a "1-in-1-out" type of place. For CN',

$$T'^C = T^C - \{t_a\},$$

$$P'^C = P^C - \{p_x\},$$

$$F'^C = F^C \cup \{\langle t_b, p_y\rangle\} - \{\langle t_b, p_x\rangle, \langle p_x, t_a\rangle, \langle t_a, p_y\rangle\}.$$

iii. In the other cases, in CN, the input place of t_a is p_x, the output place of t_a is p_y. For CN',

$$T'^C = T^C \cup \{t_c\} - \{t_a\} \quad (L^C(t_c) = \tau),$$

$$P'^C = P^C,$$

$$F'^C = F^C \cup \{\langle p_x, t_c\rangle, \langle t_c, p_y\rangle\} - \{\langle p_x, t_a\rangle, \langle t_a, p_y\rangle\}.$$

For the set of data elements $Data'$ of the changed internal business process IP',

$$Data' = Data.$$

For calculating the changed combined control flow and data flow net CDN' of the changed internal business process IP', the same operations for CN' will be implied on the control flow part of the original CDN. For the data flow part, the respective I/O parameter transitions and places of the deleted activity a will be removed, and the flow relations associated with these transitions and places will be removed as well.

PC-2 ***Add an activity sequentially***

Description: The *add an activity sequentially* change refers to that an activity is inserted before/after a specific node of the control flow of the internal business process of an SBP. The specific node can be the start/end event, any gateway, or another activity.

In Formal: Assume that the added activity is a.

For the set of activities Act' of the changed internal business process IP',

$$Act' = Act \cup \{a\}.$$

For calculating the changed control flow net CN' of the changed internal business process IP', there are four different types of operations for this change in respective cases. The differences between these cases depend on the different change requirements of where to insert the transition t_a that is labeled with the activity a (i.e.,$L^C(t_a) = a$) in the original control flow net CN (Figure 6.2 shows an example of case iii.):

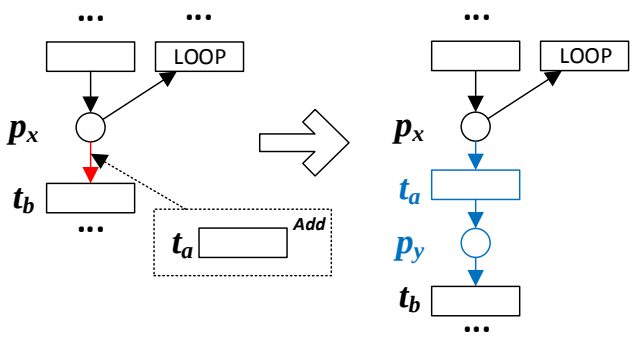

FIGURE 6.2: An Example of C-net Changes in PC-2

i. In the case of: in CN, the transition t_a will be inserted sequentially **after** a specific transition t_b that has only one output place p_x. The respective change requirement is to add an activity a sequentially after another activity or an AND-join gateway. For CN',

$$T'^C = T^C \cup \{t_a\},$$

$$P'^C = P^C \cup \{p_y\},$$

$$F'^C = F^C \cup \{\langle t_b, p_y\rangle, \langle p_y, t_a\rangle, \langle t_a, p_x\rangle\} - \{\langle t_b, p_x\rangle\}.$$

ii. In the case of: in CN, the transition t_a will be inserted sequentially **before** a specific transition t_b that has only one input place p_x. The respective change requirement is to add an activity a sequentially before another activity or an AND-split gateway. For CN',

$$T'^C = T^C \cup \{t_a\},$$

$$P'^C = P^C \cup \{p_y\},$$

$$F'^C = F^C \cup \{\langle p_x, t_a\rangle, \langle t_a, p_y\rangle, \langle p_y, t_b\rangle\} - \{\langle p_x, t_b\rangle\}.$$

iii. In the case of: in CN, the transition t_a will be inserted sequentially **after** a specific place p_x that has only one output transition t_b or has only two output transitions t_b and a LOOP transition. The respective change requirement is to add an activity a sequentially after the start event, an XOR-join gateway, or a LOOP-out gateway. For CN',

$$T'^C = T^C \cup \{t_a\},$$

$$P'^C = P^C \cup \{p_y\},$$

$$F'^C = F^C \cup \{\langle p_x, t_a\rangle, \langle t_a, p_y\rangle, \langle p_y, t_b\rangle\} - \{\langle p_x, t_b\rangle\}.$$

iv. In the case of: in CN, the transition t_a will be inserted sequentially **before** a specific place p_x that has only one input transition t_b or has only two input transitions t_b and a LOOP transition. The respective change requirement is to add an activity a sequentially before the end event, an XOR-split gateway, or a LOOP-in gateway. For CN',

$$T'^C = T^C \cup \{t_a\},$$

$$P'^C = P^C \cup \{p_y\},$$

$$F'^C = F^C \cup \{\langle t_b, p_y\rangle, \langle p_y, t_a\rangle, \langle t_a, p_x\rangle\} - \{\langle t_b, p_x\rangle\}.$$

For the set of data elements $Data'$ of the changed internal business process IP',

$$Data' = Data \cup InPara(a) \cup OutPara(a).$$

For calculating the changed combined control flow and data flow net CDN' of the changed internal business process IP', the same operations for CN' will be implied on

the control flow part of the original CDN. For the data flow part, the respective I/O parameter transitions and places of the added activity a and the possible new data element places will be added. The flow relations associated with these transitions and places will be added as well.

PC-3 ***Add an activity in parallel to existing activities***

Description: The *add an activity in parallel to existing activities* change refers to that an activity is inserted as a parallel branch to one or more existing activities of the control flow of the internal business process of an SBP.

In Formal: Assume that the added activity is a.

For the set of activities Act' of the changed internal business process IP',

$$Act' = Act \cup \{a\}.$$

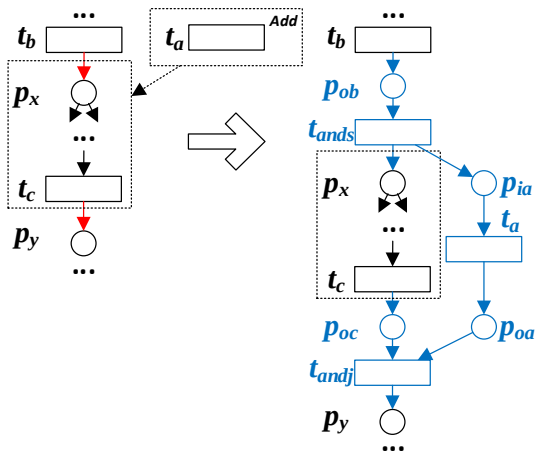

FIGURE 6.3: An Example of C-net Changes in PC-3

For calculating the changed control flow net CN' of the changed internal business

process IP', there are five different types of operations for this change in respective cases. The differences between these cases depend on the different change requirements of where to insert the transition t_a that is labeled with the activity a (i.e.,$L^C(t_a) = a$) in the original control flow net CN (Figure 6.3 shows an example of case iv.):

i. In the case of: in CN, the transition t_a will be inserted in parallel to a continuous fragment starting from the transition t_b and ending to the transition t_c. The transition t_b must have only one input place (assuming it is p_x) and the transition t_c must have only one output place (assuming it is p_y). The fragment contains one or more activity transitions. The respective change requirement is to add an activity a in parallel to a fragment starting from an activity or an AND-split gateway, ending to another activity or an AND-join gateway. For CN',

$$T'^C = T^C \cup \{t_a, t_{ands}, t_{andj}\}, \quad (L^C(t_{ands}) = \tau,\ L^C(t_{andj}) = \tau)$$

$$P'^C = P^C \cup \{p_{ib}, p_{oc}, p_{ia}, p_{oa}\},$$

$$F'^C = F^C \cup \{\langle p_x, t_{ands}\rangle, \langle t_{ands}, p_{ib}\rangle, \langle t_{ands}, p_{ia}\rangle, \langle p_{ib}, t_b\rangle, \langle p_{ia}, t_a\rangle,$$
$$\langle t_c, p_{oc}\rangle, \langle t_a, p_{oa}\rangle, \langle p_{oc}, t_{andj}\rangle, \langle p_{oa}, t_{andj}\rangle, \langle t_{andj}, p_y\rangle\} - \{\langle p_x, t_b\rangle, \langle t_c, p_y\rangle\}.$$

ii. In the case of: in CN, the transition t_a will be inserted in parallel to a continuous fragment starting from the place p_x and ending to the place p_y. The place p_x must have only one "non-LOOP-transition" input transition (assuming it is t_b) and the place p_y must have only one "non-LOOP-transition" output transition (assuming it is t_c). The fragment contains one or more activity transitions. The respective

change requirement is to add an activity a in parallel to the fragment starting from an XOR-split or LOOP-in gateway, ending to an XOR-join or LOOP-out gateway. For CN',

$$T'^C = T^C \cup \{t_a, t_{ands}, t_{andj}\}, \quad (L^C(t_{ands}) = \tau,\ L^C(t_{andj}) = \tau)$$

$$P'^C = P^C \cup \{p_{ob}, p_{ic}, p_{ia}, p_{oa}\},$$

$$F'^C = F^C \cup \{\langle t_b, p_{ob}\rangle, \langle p_{ob}, t_{ands}\rangle, \langle t_{ands}, p_x\rangle, \langle t_{ands}, p_{ia}\rangle, \langle p_{ia}, t_a\rangle,$$
$$\langle t_a, p_{oa}\rangle, \langle p_y, t_{andj}\rangle, \langle p_{oa}, t_{andj}\rangle, \langle t_{andj}, p_{ic}\rangle, \langle p_{ic}, t_c\rangle\} - \{\langle t_b, p_x\rangle, \langle p_y, t_c\rangle\}.$$

iii. In the case of: in CN, the transition t_a will be inserted in parallel to a continuous fragment starting from the transition t_b and ending to the place p_y. The transition t_b must have only one input place (assuming it is p_x) and the place p_y must have only one "non-LOOP-transition" output transition (assuming it is t_c). The fragment contains one or more activity transitions. The respective change requirement is to add an activity a in parallel to the fragment starting from an activity or an AND-split gateway, ending to an XOR-join or LOOP-out gateway. For CN',

$$T'^C = T^C \cup \{t_a, t_{ands}, t_{andj}\}, \quad (L^C(t_{ands}) = \tau,\ L^C(t_{andj}) = \tau)$$

$$P'^C = P^C \cup \{p_{ib}, p_{ic}, p_{ia}, p_{oa}\},$$

$$F'^C = F^C \cup \{\langle p_x, t_{ands}\rangle, \langle t_{ands}, p_{ib}\rangle, \langle t_{ands}, p_{ia}\rangle, \langle p_{ib}, t_b\rangle, \langle p_{ia}, t_a\rangle,$$
$$\langle t_a, p_{oa}\rangle, \langle p_y, t_{andj}\rangle, \langle p_{oa}, t_{andj}\rangle, \langle t_{andj}, p_{ic}\rangle, \langle p_{ic}, t_c\rangle\} - \{\langle p_x, t_b\rangle, \langle p_y, t_c\rangle\}.$$

iv. In the case of: in CN, the transition t_a will be inserted in parallel to a continuous fragment starting from the place p_x and ending to the transition t_c. The place p_x must have only one "non-LOOP-transition" input transition (assuming it is t_b) and the transition t_c must have only one output place (assuming it is p_y). The fragment contains one or more activity transitions. The respective change requirement is to add an activity a in parallel to the fragment starting from an XOR-split or LOOP-in gateway, ending to an activity or an AND-join gateway. For CN',

$$T'^C = T^C \cup \{t_a, t_{ands}, t_{andj}\}, \quad (L^C(t_{ands}) = \tau,\ L^C(t_{andj}) = \tau)$$

$$P'^C = P^C \cup \{p_{ob}, p_{oc}, p_{ia}, p_{oa}\},$$

$$F'^C = F^C \cup \{\langle t_b, p_{ob}\rangle, \langle p_{ob}, t_{ands}\rangle, \langle t_{ands}, p_x\rangle, \langle t_{ands}, p_{ia}\rangle, \langle p_{ia}, t_a\rangle,$$
$$\langle t_c, p_{oc}\rangle, \langle t_a, p_{oa}\rangle, \langle p_{oc}, t_{andj}\rangle, \langle p_{oa}, t_{andj}\rangle, \langle t_{andj}, p_y\rangle\} - \{\langle t_b, p_x\rangle, \langle t_c, p_y\rangle\}.$$

v. In the case of: in CN, the transition t_a will be inserted as a new branch of a parallel fragment starting from the transition t_{ands} and ending to the transition t_{andj}. The respective change requirement is to add an activity a in a parallel fragment as another branch. For CN',

$$T'^C = T^C \cup \{t_a\},$$

$$P'^C = P^C \cup \{p_{ia}, p_{oa}\},$$

$$F'^C = F^C \cup \{\langle t_{ands}, p_{ia}\rangle, \langle p_{ia}, t_a\rangle, \langle t_a, p_{oa}\rangle, \langle p_{oa}, t_{andj}\rangle\}.$$

For the set of data elements $Data'$ of the changed internal business process IP',

$$Data' = Data \cup InPara(a) \cup OutPara(a).$$

For calculating the changed combined control flow and data flow net CDN' of the changed internal business process IP', the same operations for CN' will be implied on the control flow part of the original CDN. For the data flow part, the respective I/O parameter transitions and places of the added activity a and the possible new data element places will be added. The flow relations associated with these transitions and places will be added as well.

PC-4 ***Add an activity conditionally to existing activities***

Description: The *add an activity conditionally to existing activities* change refers to that an activity is inserted as an XOR branch to one or more existing activities of the control flow of the internal business process of an SBP.

In Formal: Assume that the added activity is a.

For the set of activities Act' of the changed internal business process IP',

$$Act' = Act \cup \{a\}.$$

For calculating the changed control flow net CN' of the changed internal business process IP', there are five different types of operations for this change in respective cases. The differences between these cases depend on the different change requirements of where to insert the transition t_a that is labeled with the activity a (i.e.,$L^C(t_a) = a$) in the original control flow net CN (Figure 6.4 shows an example of case iii.):

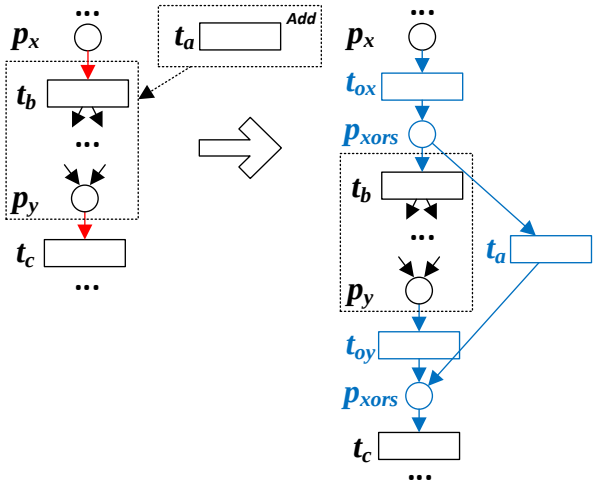

Figure 6.4: An Example of C-net Changes in PC-4

i. In the case of: in CN, the transition t_a will be inserted conditionally to a continuous fragment starting from the transition t_b and ending to the transition t_c. The transition t_b must have only one input place (assuming it is p_x) and the transition t_c must have only one output place (assuming it is p_y). The fragment contains one or more activity transitions. The respective change requirement is to add an activity a conditionally to the fragment starting from an activity or an AND-split gateway, ending to another activity or an AND-join gateway. For CN',

$$T'^C = T^C \cup \{t_a, t_{ox}, t_{iy}\}, \quad (L^C(t_{ox}) = \tau,\ L^C(t_{iy}) = \tau)$$

$$P'^C = P^C \cup \{p_{xors}, p_{xorj}\},$$

$$F'^C = F^C \cup \{\langle p_x, t_{ox}\rangle, \langle t_{ox}, p_{xors}\rangle, \langle p_{xors}, t_b\rangle,$$

$$\langle p_{xors}, t_a\rangle, \langle t_c, p_{xorj}\rangle, \langle t_a, p_{xorj}\rangle, \langle p_{xorj}, t_{iy}\rangle, \langle t_{iy}, p_y\rangle\} - \{\langle p_x, t_b\rangle, \langle t_c, p_y\rangle\}.$$

ii. In the case of: in CN, the transition t_a will be inserted conditionally to a continuous fragment starting from the place p_x and ending to the place p_y. The place p_x must have only one "non-LOOP-transition" input transition (assuming it is t_b) and the place p_y must have only one "non-LOOP-transition" output transition (assuming it is t_c). The fragment contains one or more activity transitions. The respective change requirement is to add an activity a conditionally to the fragment starting from an XOR-split or LOOP-in gateway, ending to an XOR-join or LOOP-out gateway. For CN',

$$T'^C = T^C \cup \{t_a, t_{ix}, t_{oy}\}, \quad (L^C(t_{ix}) = \tau,\ L^C(t_{oy}) = \tau)$$

$$P'^C = P^C \cup \{p_{xors}, p_{xorj}\},$$

$$F'^C = F^C \cup \{\langle t_b, p_{xors}\rangle, \langle p_{xors}, t_{ix}\rangle, \langle t_{ix}, p_x\rangle,$$

$$\langle p_{xors}, t_a\rangle, \langle p_y, t_{oy}\rangle, \langle t_{oy}, p_{xorj}\rangle, \langle t_a, p_{xorj}\rangle, \langle p_{xorj}, t_c\rangle\} - \{\langle t_b, p_x\rangle, \langle p_y, t_c\rangle\}.$$

iii. In the case of: in CN, the transition t_a will be inserted conditionally to a continuous fragment starting from the transition t_b and ending to the place p_y. The transition t_b must have only one input place (assuming it is p_x) and the place p_y must have only one "non-LOOP-transition" output transition (assuming it is t_c). The fragment contains one or more activity transitions. The respective change requirement is to add an activity a conditionally to the fragment starting from an activity or an AND-split gateway, ending to an XOR-join or LOOP-out gateway. For CN',

$$T'^C = T^C \cup \{t_a, t_{ox}, t_{oy}\}, \quad (L^C(t_{ox}) = \tau,\ L^C(t_{oy}) = \tau)$$

$$P'^C = P^C \cup \{p_{xors}, p_{xorj}\},$$

$$F'^C = F^C \cup \{\langle p_x, t_{ox}\rangle, \langle t_{ox}, p_{xors}\rangle, \langle p_{xors}, t_b\rangle,$$
$$\langle p_{xors}, t_a\rangle, \langle p_y, t_{oy}\rangle, \langle t_{oy}, p_{xorj}\rangle, \langle t_a, p_{xorj}\rangle, \langle p_{xorj}, t_c\rangle\} - \{\langle p_x, t_b\rangle, \langle p_y, t_c\rangle\}.$$

iv. In the case of: in CN, the transition t_a will be inserted conditionally to a continuous fragment starting from the place p_x and ending to the transition t_c. The place p_x must have only one "non-LOOP-transition" input transition (assuming it is t_b) and the transition t_c must have only one output place (assuming it is p_y). The fragment contains one or more activity transitions. The respective change requirement is to add an activity a conditionally to the fragment starting from an XOR-split or LOOP-in gateway, ending to an activity or an AND-join gateway. For CN',

$$T'^C = T^C \cup \{t_a, t_{ix}, t_{iy}\}, \quad (L^C(t_{ix}) = \tau,\ L^C(t_{iy}) = \tau)$$

$$P'^C = P^C \cup \{p_{xors}, p_{xorj}\},$$

$$F'^C = F^C \cup \{\langle t_b, p_{xors}\rangle, \langle p_{xors}, t_{ix}\rangle, \langle t_{ix}, p_x\rangle,$$
$$\langle p_{xors}, t_a\rangle, \langle t_c, p_{xorj}\rangle, \langle t_a, p_{xorj}\rangle, \langle p_{xorj}, t_{iy}\rangle, \langle t_{iy}, p_y\rangle\} - \{\langle t_b, p_x\rangle, \langle t_c, p_y\rangle\}.$$

v. In the case of: in CN, the transition t_a will be inserted as a new branch of a conditional fragment starting from the place p_{xors} and ending to the place p_{xorj}. The respective change requirement is to add an activity a in a conditional fragment as another branch. For CN',

$$T'^C = T^C \cup \{t_a\},$$

$$P'^{C} = P^{C},$$

$$F'^{C} = F^{C} \cup \{\langle p_{xors}, t_a \rangle, \langle t_a, p_{xorj} \rangle\}.$$

For the set of data elements $Data'$ of the changed internal business process IP',

$$Data' = Data \cup InPara(a) \cup OutPara(a).$$

For calculating the changed combined control flow and data flow net CDN' of the changed internal business process IP', the same operations for CN' will be implied on the control flow part of the original CDN. For the data flow part, the respective I/O parameter transitions and places of the added activity a and the possible new data element places will be added. The flow relations associated with these transitions and places will be added as well.

PC-5 ***Replace an activity by another one***

Description: The *replace an activity by another one* change refers to that an activity is replaced by another activity in the internal business process of an SBP.

In Formal: Assume that the replaced activity is a and the alternative activity is b.

For the set of activities Act' of the changed internal business process IP',

$$Act' = Act \cup \{b\} - \{a\}.$$

For calculating the changed control flow net CN' of the changed internal business process IP', assume that the input and output places of the transition t_a labeled with the activity a (i.e.,$L^C(t_a) = a$) are p_x and p_y respectively in the original control flow net CN. For the changed control flow net CN' (Figure 6.5 shows an example of PC-5.),

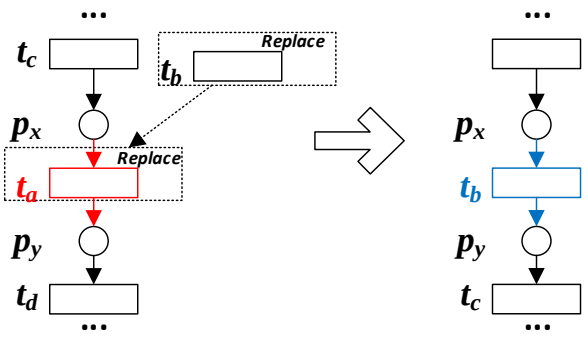

Figure 6.5: An Example of C-net Changes in PC-5

$$T'^C = T^C \cup \{t_b\} - \{t_a\},$$

$$P'^C = P^C,$$

$$F'^C = F^C \cup \{\langle p_x, t_b\rangle, \langle t_b, p_y\rangle\} - \{\langle p_x, t_a\rangle, \langle t_a, p_y\rangle\}.$$

For the set of data elements $Data'$ of the changed internal business process IP',

$$Data' = Data \cup InPara(b) \cup OutPara(b).$$

For calculating the changed combined control flow and data flow net CDN' of the changed internal business process IP', the same operations for CN' will be implied on the control flow part of the original CDN. For the data flow part, the respective I/O parameter transitions and places of the replaced activity a will be removed, and the flow relations associated with these transitions and places will be removed. The respective I/O parameter transitions and places of the alternative activity b and the possible new data element places will be added, and the flow relations associated with these transitions and places will be added.

6.1.2 Activity Order Changes

This type of changes refers to as the change of execution order of activities. By this type of changes, the existence of activities is not changed.

PC-6 ***Add a conditional shortcut***

Description: The *add a conditional shortcut* change refers to that a "conditional shortcut" is added to make a fragment be executed conditionally in the internal business process of an SBP.

In Formal: For the set of activities Act' of the changed internal business process IP',

$$Act' = Act.$$

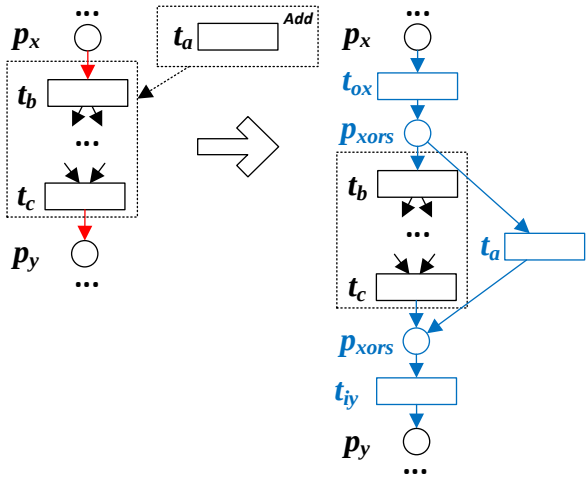

FIGURE 6.6: An Example of C-net Changes in PC-6

For calculating the changed control flow net CN' of the changed internal business process IP', there are five different types of operations for this change in respective

cases. The differences between these cases depend on the different change requirements of where to insert the conditional shortcut transition t_a that is not labeled with any activity (i.e.,$L^C(t_a) = \tau$) in the original control flow net CN (Figure 6.6 shows an example of case i.):

i. In the case of: in CN, the transition t_a will be inserted conditionally to a continuous fragment starting from the transition t_b and ending to the transition t_c. The transition t_b must have only one input place (assuming it is p_x) and the transition t_c must have only one output place (assuming it is p_y). The fragment contains one or more activity transitions. The respective change requirement is to add a conditional shortcut to the fragment starting from an activity or an AND-split gateway, ending to another activity or an AND-join gateway. For CN',

$$T'^C = T^C \cup \{t_a, t_{ox}, t_{iy}\}, \quad (L^C(t_{ox}) = \tau,\ L^C(t_{iy}) = \tau)$$

$$P'^C = P^C \cup \{p_{xors}, p_{xorj}\},$$

$$F'^C = F^C \cup \{\langle p_x, t_{ox}\rangle, \langle t_{ox}, p_{xors}\rangle, \langle p_{xors}, t_b\rangle,$$

$$\langle p_{xors}, t_a\rangle, \langle t_c, p_{xorj}\rangle, \langle t_a, p_{xorj}\rangle, \langle p_{xorj}, t_{iy}\rangle, \langle t_{iy}, p_y\rangle\} - \{\langle p_x, t_b\rangle, \langle t_c, p_y\rangle\}.$$

ii. In the case of: in CN, the transition t_a will be inserted conditionally to a continuous fragment starting from the place p_x and ending to the place p_y. The place p_x must have only one "non-LOOP-transition" input transition (assuming it is t_b) and the place p_y must have only one "non-LOOP-transition" output transition (assuming it is t_c). The fragment contains one or more activity transitions.

The respective change requirement is to add a conditional shortcut to the fragment starting from an XOR-split or LOOP-in gateway, ending to an XOR-join or LOOP-out gateway. For CN',

$$T'^C = T^C \cup \{t_a, t_{ix}, t_{oy}\}, \quad (L^C(t_{ix}) = \tau,\ L^C(t_{oy}) = \tau)$$

$$P'^C = P^C \cup \{p_{xors}, p_{xorj}\},$$

$$F'^C = F^C \cup \{\langle t_b, p_{xors}\rangle, \langle p_{xors}, t_{ix}\rangle, \langle t_{ix}, p_x\rangle,$$

$$\langle p_{xors}, t_a\rangle, \langle p_y, t_{oy}\rangle, \langle t_{oy}, p_{xorj}\rangle, \langle t_a, p_{xorj}\rangle, \langle p_{xorj}, t_c\rangle\} - \{\langle t_b, p_x\rangle, \langle p_y, t_c\rangle\}.$$

iii. In the case of: in CN, the transition t_a will be inserted conditionally to a continuous fragment starting from the transition t_b and ending to the place p_y. The transition t_b must have only one input place (assuming it is p_x) and the place p_y must have only one "non-LOOP-transition" output transition (assuming it is t_c). The fragment contains one or more activity transitions. The respective change requirement is to add a conditional shortcut to the fragment starting from an activity or an AND-split gateway, ending to an XOR-join or LOOP-out gateway. For CN',

$$T'^C = T^C \cup \{t_a, t_{ox}, t_{oy}\}, \quad (L^C(t_{ox}) = \tau,\ L^C(t_{oy}) = \tau)$$

$$P'^C = P^C \cup \{p_{xors}, p_{xorj}\},$$

$$F'^C = F^C \cup \{\langle p_x, t_{ox}\rangle, \langle t_{ox}, p_{xors}\rangle, \langle p_{xors}, t_b\rangle,$$

$$\langle p_{xors}, t_a\rangle, \langle p_y, t_{oy}\rangle, \langle t_{oy}, p_{xorj}\rangle, \langle t_a, p_{xorj}\rangle, \langle p_{xorj}, t_c\rangle\} - \{\langle p_x, t_b\rangle, \langle p_y, t_c\rangle\}.$$

iv. In the case of: in CN, the transition t_a will be inserted conditionally to a continuous fragment starting from the place p_x and ending to the transition t_c. The place p_x must have only one "non-LOOP-transition" input transition (assuming it is t_b) and the transition t_c must have only one output place (assuming it is p_y). The fragment contains one or more activity transitions. The respective change requirement is to add a conditional shortcut to the fragment starting from an XOR-split or LOOP-in gateway, ending to an activity or an AND-join gateway. For CN',

$$T'^C = T^C \cup \{t_a, t_{ix}, t_{iy}\}, \quad (L^C(t_{ix}) = \tau,\ L^C(t_{iy}) = \tau)$$

$$P'^C = P^C \cup \{p_{xors}, p_{xorj}\},$$

$$F'^C = F^C \cup \{\langle t_b, p_{xors}\rangle, \langle p_{xors}, t_{ix}\rangle, \langle t_{ix}, p_x\rangle,$$

$$\langle p_{xors}, t_a\rangle, \langle t_c, p_{xorj}\rangle, \langle t_a, p_{xorj}\rangle, \langle p_{xorj}, t_{iy}\rangle, \langle t_{iy}, p_y\rangle\} - \{\langle t_b, p_x\rangle, \langle t_c, p_y\rangle\}.$$

v. In the case of: in CN, the transition t_a will be inserted as a new branch of a conditional fragment starting from the place p_{xors} and ending to the place p_{xorj}. The respective change requirement is to add a conditional shortcut in a conditional fragment as another branch. For CN',

$$T'^C = T^C \cup \{t_a\},$$

$$P'^C = P^C,$$

$$F'^C = F^C \cup \{\langle p_{xors}, t_a\rangle, \langle t_a, p_{xorj}\rangle\}.$$

For the set of data elements $Data'$ of the changed internal business process IP',

$$Data' = Data.$$

For calculating the changed combined control flow and data flow net CDN' of the changed internal business process IP', the same operations for CN' will be implied on the control flow part of the original CDN. The data flow part of CDN' keeps the same with the original CDN.

PC-7 ***Delete a conditional shortcut***

Description: The *delete a conditional shortcut* change refers to that a "conditional shortcut" is removed from the internal business process of an SBP.

In Formal: For the set of activities Act' of the changed internal business process IP',

$$Act' = Act.$$

For calculating the changed control flow net CN' of the changed internal business process IP', the conditional shortcut transition t_a that is not labeled with any activity (i.e.,$L^C(t_a) = \tau$) and relative flow relations will be removed from the original control flow net CN (Figure 6.7 shows an example of case ii.):

$$T'^C = T^C - \{t_a\},$$

$$P'^C = P^C,$$

$$F'^C = F^C - \{\langle p_{xors}, t_a\rangle, \langle t_a, p_{xorj}\rangle\}.$$

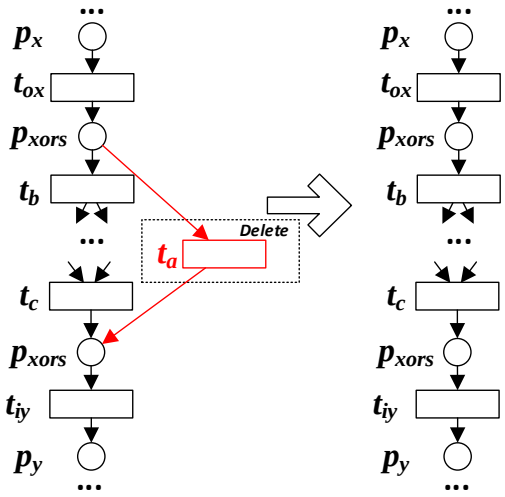

Figure 6.7: An Example of C-net Changes in PC-7

For the set of data elements $Data'$ of the changed internal business process IP',

$$Data' = Data.$$

For calculating the changed combined control flow and data flow net CDN' of the changed internal business process IP', the same operations for CN' will be implied on the control flow part of the original CDN. The data flow part of CDN' keeps the same with the original CDN.

PC-8 ***Add a loop***

Description: The *add a loop* change refers to that a "loop" is added to make a fragment be repeatedly executed in the internal business process of an SBP.

In Formal: For the set of activities Act' of the changed internal business process IP',

$$Act' = Act.$$

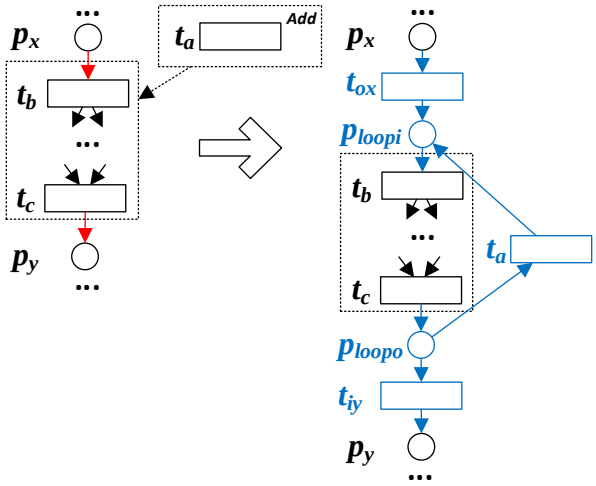

FIGURE 6.8: An Example of C-net Changes in PC-8

For calculating the changed control flow net CN' of the changed internal business process IP', there are four different types of operations for this change in respective cases. The differences between these cases depend on the different change requirements of where to insert the loop transition t_a that is not labeled with any activity (i.e.,$L^C(t_a) = \tau$) in the original control flow net CN (Figure 6.8 shows an example of case i.):

i. In the case of: in CN, the loop transition t_a will be inserted to a continuous fragment starting from the transition t_b and ending to the transition t_c. The transition t_b must have only one input place (assuming it is p_x) and the transition t_c must have only one output place (assuming it is p_y). The fragment contains one or more activity transitions. The respective change requirement is to add a loop to the fragment starting from an activity or an AND-split gateway, ending

to another activity or an AND-join gateway. For CN',

$$T'^C = T^C \cup \{t_a, t_{ox}, t_{iy}\}, \quad (L^C(t_{ox}) = \tau,\ L^C(t_{iy}) = \tau)$$

$$P'^C = P^C \cup \{p_{loopi}, p_{loopo}\},$$

$$F'^C = F^C \cup \{\langle p_x, t_{ox}\rangle, \langle t_{ox}, p_{loopi}\rangle, \langle p_{loopi}, t_b\rangle,$$

$$\langle t_c, p_{loopo}\rangle, \langle p_{loopo}, t_a\rangle, \langle t_a, p_{loopi}\rangle, \langle p_{loopo}, t_{iy}\rangle, \langle t_{iy}, p_y\rangle\} - \{\langle p_x, t_b\rangle, \langle t_c, p_y\rangle\}.$$

ii. In the case of: in CN, the loop transition t_a will be inserted to a continuous fragment starting from the place p_x and ending to the place p_y. The place p_x must have only one "non-LOOP-transition" input transition (assuming it is t_b) and the place p_y must have only one "non-LOOP-transition" output transition (assuming it is t_c). The fragment contains one or more activity transitions. The respective change requirement is to add a loop to the fragment starting from an XOR-split or LOOP-in gateway, ending to an XOR-join or LOOP-out gateway. For CN',

$$T'^C = T^C \cup \{t_a, t_{ix}, t_{oy}\}, \quad (L^C(t_{ix}) = \tau,\ L^C(t_{oy}) = \tau)$$

$$P'^C = P^C \cup \{p_{loopi}, p_{loopo}\},$$

$$F'^C = F^C \cup \{\langle t_b, p_{loopi}\rangle, \langle p_{loopi}, t_{ix}\rangle, \langle t_{ix}, p_x\rangle,$$

$$\langle p_y, t_{oy}\rangle, \langle t_{oy}, p_{loopo}\rangle, \langle p_{loopo}, t_a\rangle, \langle t_a, p_{loopi}\rangle, \langle p_{loopo}, t_c\rangle\} - \{\langle t_b, p_x\rangle, \langle p_y, t_c\rangle\}.$$

iii. In the case of: in CN, the loop transition t_a will be inserted to a continuous fragment starting from the transition t_b and ending to the place p_y. The transition t_b

must have only one input place (assuming it is p_x) and the place p_y must have only one "non-LOOP-transition" output transition (assuming it is t_c). The fragment contains one or more activity transitions. The respective change requirement is to add a loop to the fragment starting from an activity or an AND-split gateway, ending to an XOR-join or LOOP-out gateway. For CN',

$$T'^C = T^C \cup \{t_a, t_{ox}, t_{oy}\}, \quad (L^C(t_{ox}) = \tau,\ L^C(t_{oy}) = \tau)$$

$$P'^C = P^C \cup \{p_{loopi}, p_{loopo}\},$$

$$F'^C = F^C \cup \{\langle p_x, t_{ox}\rangle, \langle t_{ox}, p_{loopi}\rangle, \langle p_{loopi}, t_b\rangle,$$

$$\langle p_y, t_{oy}\rangle, \langle t_{oy}, p_{loopo}\rangle, \langle p_{loopo}, t_a\rangle, \langle t_a, p_{loopi}\rangle, \langle p_{loopo}, t_c\rangle\} - \{\langle p_x, t_b\rangle, \langle p_y, t_c\rangle\}.$$

iv. In the case of: in CN, the loop transition t_a will be inserted to a continuous fragment starting from the place p_x and ending to the transition t_c. The place p_x must have only one "non-LOOP-transition" input transition (assuming it is t_b) and the transition t_c must have only one output place (assuming it is p_y). The fragment contains one or more activity transitions. The respective change requirement is to add a loop to the fragment starting from an XOR-split or LOOP-in gateway, ending to an activity or an AND-join gateway. For CN',

$$T'^C = T^C \cup \{t_a, t_{ix}, t_{iy}\}, \quad (L^C(t_{ix}) = \tau,\ L^C(t_{iy}) = \tau)$$

$$P'^C = P^C \cup \{p_{loopi}, p_{loopo}\},$$

$$F'^C = F^C \cup \{\langle t_b, p_{loopi}\rangle, \langle p_{loopi}, t_{ix}\rangle, \langle t_{ix}, p_x\rangle,$$

$$\langle t_c, p_{loopo}\rangle, \langle p_{loopo}, t_a\rangle, \langle t_a, p_{loopi}\rangle, \langle p_{loopo}, t_{iy}\rangle, \langle t_{iy}, p_y\rangle\} - \{\langle t_b, p_x\rangle, \langle t_c, p_y\rangle\}.$$

For the set of data elements $Data'$ of the changed internal business process IP',

$$Data' = Data.$$

For calculating the changed combined control flow and data flow net CDN' of the changed internal business process IP', the same operations for CN' will be implied on the control flow part of the original CDN. The data flow part of CDN' keeps the same with the original CDN.

PC-9 ***Delete a loop***

Description: The *delete a loop* change refers to that a "loop" is removed from the internal business process of an SBP.

In Formal: For the set of activities Act' of the changed internal business process IP',

$$Act' = Act.$$

For calculating the changed control flow net CN' of the changed internal business process IP', the loop transition t_a that is not labeled with any activity (i.e.,$L^C(t_a) = \tau$) and relative flow relations will be removed from the original control flow net CN (Figure 6.9 shows an example of PC-9.):

$$T'^C = T^C - \{t_a\},$$

$$P'^C = P^C,$$

$$F'^C = F^C - \{\langle p_{loopo}, t_a\rangle, \langle t_a, p_{loopi}\rangle\}.$$

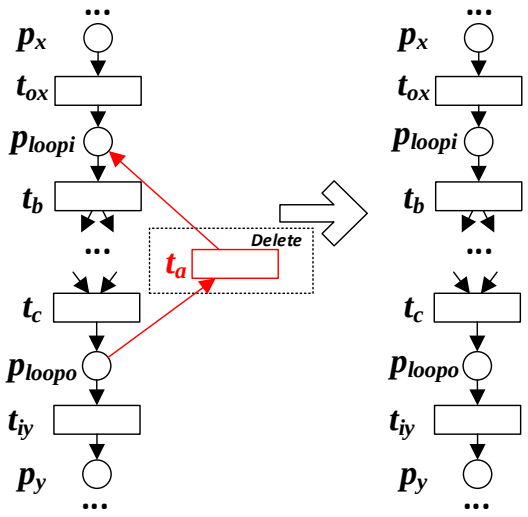

FIGURE 6.9: An Example of C-net Changes in PC-9

For the set of data elements $Data'$ of the changed internal business process IP',

$$Data' = Data.$$

For calculating the changed combined control flow and data flow net CDN' of the changed internal business process IP', the same operations for CN' will be implied on the control flow part of the original CDN. The data flow part of CDN' keeps the same with the original CDN.

PC-10 ***Add a control dependency***

Description: The *add a control dependency* change refers to that a control dependency is inserted between two parallel branches of the control flow of the internal business process of an SBP.

In Formal: For the set of activities Act' of the changed internal business process

IP',

$$Act' = Act.$$

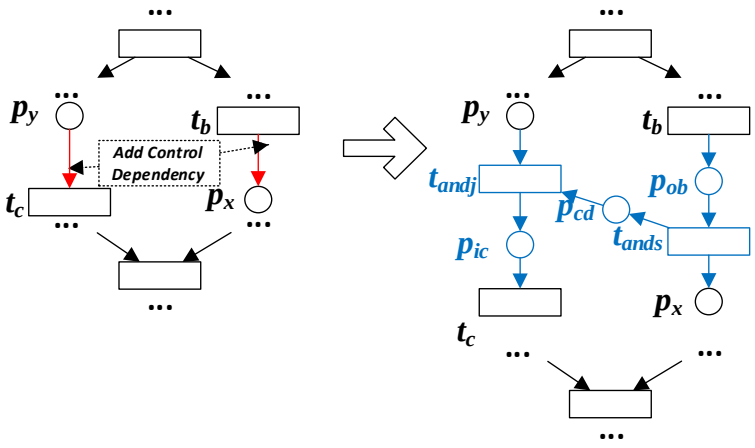

Figure 6.10: An Example of C-net Changes in PC-10

For calculating the changed control flow net CN' of the changed internal business process IP', there are four different types of operations for this change in respective cases. The differences between these cases depend on the different change requirements of where to insert the control dependency in the original control flow net CN (Figure 6.10 shows an example of case i.):

i. In the case of: in CN, the control dependency will be inserted after the execution of transition t_b and before the execution of transition t_c, which t_b and t_c are in two parallel branches. The transition t_b must have only one output place (assuming it is p_x) and the transition t_c must have only one input place (assuming it is p_y). The respective change requirement is to add a control dependency between two transitions in parallel branches (the first transition can be an activity transition

or an AND-join transition, the second transition can be an activity transition or an AND-split transition). For CN',

$$T'^C = T^C \cup \{t_{ands}, t_{andj}\}, \quad (L^C(t_{ands}) = \tau,\ L^C(t_{andj}) = \tau)$$

$$P'^C = P^C \cup \{p_{ob}, p_{ic}, p_{cd}\},$$

$$F'^C = F^C \cup \{\langle t_b, p_{ob}\rangle, \langle p_{ob}, t_{ands}\rangle, \langle t_{ands}, p_x\rangle,$$

$$\langle t_{ands}, p_{cd}\rangle, \langle p_{cd}, t_{andj}\rangle, \langle p_y, t_{andj}\rangle, \langle t_{andj}, p_{ic}\rangle, \langle p_{ic}, t_c\rangle\} - \{\langle t_b, p_x\rangle, \langle p_y, t_c\rangle\}.$$

ii. In the case of: in CN, the control dependency will be inserted after the place p_x and before the place p_y, which p_x and p_y are in two parallel branches. The place p_x must have only one "non-LOOP-transition" output transition (assuming it is t_b) and the place p_y must have only one "non-LOOP-transition" input transition (assuming it is t_c). The respective change requirement is to add a control dependency between two places in parallel branches (the first place can be an XOR-join place or a LOOP-out place, the second place can be an XOR-split place or a LOOP-in place). For CN',

$$T'^C = T^C \cup \{t_{ands}, t_{andj}\}, \quad (L^C(t_{ands}) = \tau,\ L^C(t_{andj}) = \tau)$$

$$P'^C = P^C \cup \{p_{ib}, p_{oc}, p_{cd}\},$$

$$F'^C = F^C \cup \{\langle p_x, t_{ands}\rangle, \langle t_{ands}, p_{ib}\rangle, \langle p_{ib}, t_b\rangle,$$

$$\langle t_{ands}, p_{cd}\rangle, \langle p_{cd}, t_{andj}\rangle, \langle t_c, p_{oc}\rangle, \langle p_{oc}, t_{andj}\rangle, \langle t_{andj}, p_y\rangle\} - \{\langle p_x, t_b\rangle, \langle t_c, p_y\rangle\}.$$

iii. In the case of: in CN, the control dependency will be inserted after the execution of transition t_b and before the place p_y, which t_b and p_y are in two parallel branches. The transition t_b must have only one output place (assuming it is p_x) and the place p_y must have only one "non-LOOP-transition" input transition (assuming it is t_c). The respective change requirement is to add a control dependency between a transition and a place in two parallel branches (the transition can be an activity transition or an AND-join transition, the place can be an XOR-split place or a LOOP-in place). For CN',

$$T'^C = T^C \cup \{t_{ands}, t_{andj}\}, \quad (L^C(t_{ands}) = \tau,\ L^C(t_{andj}) = \tau)$$

$$P'^C = P^C \cup \{p_{ob}, p_{oc}, p_{cd}\},$$

$$F'^C = F^C \cup \{\langle t_b, p_{ob}\rangle, \langle p_{ob}, t_{ands}\rangle, \langle t_{ands}, p_x\rangle,$$

$$\langle t_{ands}, p_{cd}\rangle, \langle p_{cd}, t_{andj}\rangle, \langle t_c, p_{oc}\rangle, \langle p_{oc}, t_{andj}\rangle, \langle t_{andj}, p_y\rangle\} - \{\langle t_b, p_x\rangle, \langle t_c, p_y\rangle\}.$$

iv. In the case of: in CN, the control dependency will be inserted after the place p_x and before the execution of transition t_c, which p_x and t_c are in two parallel branches. The place p_x must have only one "non-LOOP-transition" output transition (assuming it is t_b) and the transition t_c must have only one input place (assuming it is p_y). The respective change requirement is to add a control dependency between a place and a transition in two parallel branches (the place can be an XOR-join place or a LOOP-out place, the transition can be an activity transition or an AND-split transition). For CN',

$$T'^C = T^C \cup \{t_{ands}, t_{andj}\}, \quad (L^C(t_{ands}) = \tau,\ L^C(t_{andj}) = \tau)$$

$$P'^C = P^C \cup \{p_{ib}, p_{ic}, p_{cd}\},$$

$$F'^C = F^C \cup \{\langle p_x, t_{ands}\rangle, \langle t_{ands}, p_{ib}\rangle, \langle p_{ib}, t_b\rangle,$$

$$\langle t_{ands}, p_{cd}\rangle, \langle p_{cd}, t_{andj}\rangle, \langle p_y, t_{andj}\rangle, \langle t_{andj}, p_{ic}\rangle, \langle p_{ic}, t_c\rangle\} - \{\langle p_x, t_b\rangle, \langle p_y, t_c\rangle\}.$$

For the set of data elements $Data'$ of the changed internal business process IP',

$$Data' = Data.$$

For calculating the changed combined control flow and data flow net CDN' of the changed internal business process IP', the same operations for CN' will be implied on the control flow part of the original CDN. The data flow part of CDN' keeps the same with the original CDN.

PC-11 ***Delete a control dependency***

Description: The *delete a control dependency* change refers to that a control dependency between two parallel branches is removed from the internal business process of an SBP.

In Formal: For the set of activities Act' of the changed internal business process IP',

$$Act' = Act.$$

For calculating the changed control flow net CN' of the changed internal business process IP', the place p_{cd}, which connects the transition t_{ands} ($L^C(t_{ands}) = \tau$) and the transition t_{andj} ($L^C(t_{andj}) = \tau$) of the control dependency, and relative flow relations

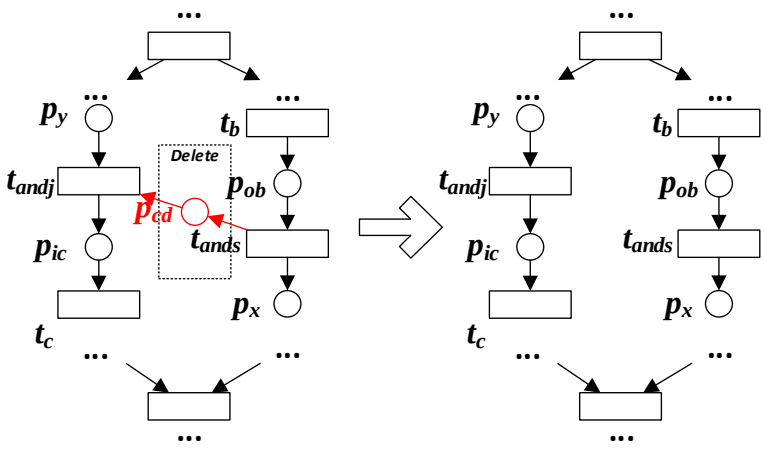

Figure 6.11: An Example of C-net Changes in PC-11

will be removed from the original control flow net CN (Figure 6.11 shows an example of PC-11.):

$$T'^C = T^C,$$

$$P'^C = P^C - \{p_{cd}\},$$

$$F'^C = F^C - \{\langle t_{ands}, p_{cd}\rangle, \langle p_{cd}, t_{andj}\rangle\}.$$

For the set of data elements $Data'$ of the changed internal business process IP',

$$Data' = Data.$$

For calculating the changed combined control flow and data flow net CDN' of the changed internal business process IP', the same operations for CN' will be implied on the control flow part of the original CDN. The data flow part of CDN' keeps the same with the original CDN.

6.1.3 Process Data Changes

This type of changes refers to as the directly manipulating on the data elements of the internal business process. Three change patterns are introduced in this subsection: *update the legacy data*, *update the target data*, and *update an activity*. For updating the process data set $Data$ correspondingly, the only change in the data set is to add new data elements which means that the original data elements cannot be removed.

PC-12 ***Update the legacy data***

Description: The *update the legacy data* change refers to that the set of legacy data elements of the internal business process of an SBP is changed.

In Formal: For the set of activities Act' of the changed internal business process IP',

$$Act' = Act.$$

Figure shows an example

For the control flow net CN' of the changed internal business process IP',

$$T'^C = T^C,$$

$$P'^C = P^C,$$

$$F'^C = F^C.$$

For the set of data elements $Data'$ of the changed internal business process IP', assume that the original legacy data set is $LegData$ and the changed legacy data set is $LegData'$,

$$Data' = Data \cup LegData'.$$

For calculating the changed combined control flow and data flow net CDN' of the changed internal business process IP', the same operations for CN' will be implied on the control flow part of the original CDN. For the data flow part, the data places of the possible new added data elements that do not belong to $Data$ will be added correspondingly. The relative data flow connections connecting the legacy data transition t^{leg} and the new added legacy data places will be added, and the relative data flow connections connecting the legacy data transition t^{leg} and the data places that are no longer legacy data elements will be removed.

PC-13 ***Update the target data***

Description: The *update the target data* change refers to that the set of target data elements of the internal business process of an SBP is changed.

In Formal: For the set of activities Act' of the changed internal business process IP',

$$Act' = Act.$$

Figure shows an example

For the control flow net CN' of the changed internal business process IP',

$$T'^C = T^C,$$

$$P'^C = P^C,$$

$$F'^C = F^C.$$

For the set of data elements $Data'$ of the changed internal business process IP', assume that the original target data set is $TarData$ and the changed target data set is

$TarData'$,

$$Data' = Data \cup TarData'.$$

For calculating the changed combined control flow and data flow net CDN' of the changed internal business process IP', the same operations for CN' will be implied on the control flow part of the original CDN. For the data flow part, the data places of the possible new added data elements that do not belong to $Data$ will be added correspondingly. The relative data flow connections connecting the new added target data places and the target data transition t^{tar} will be added, and the relevant data flow connections connecting the data places that are no longer target data elements and the target data transition t^{tar} will be removed.

PC-14 ***Update an activity***

Description: The *update an activity* change refers to that the set of data elements belonging to the input and output parameters of an activity from the internal business process of an SBP is changed.

In Formal: For the set of activities Act' of the changed internal business process IP',

$$Act' = Act.$$

Figure shows an example

For the control flow net CN' of the changed internal business process IP',

$$T'^C = T^C,$$

$$P'^C = P^C,$$

$$F'^{C} = F^{C}.$$

For the set of data elements $Data'$ of the changed internal business process IP', assume that the original data set $Data(a)$ consists of the input and output parameters of the activity a, and the changed data set $Data(a)'$ consists of the input and output parameters of the updated activity a, i.e., $Data(a) = InPara(a) \cup OutPara(a)$ and $Data(a)' = InPara(a)' \cup OutPara(a)'$,

$$Data' = Data \cup Data(a)'.$$

For calculating the changed combined control flow and data flow net CDN' of the changed internal business process IP', the same operations for CN' will be implied on the control flow part of the original CDN. For the data flow part, the data places of the possible new added data elements that do not belong to $Data$ will be added correspondingly. The relative data flow connections connecting the new added input data places and the input parameter transition t_a^I of the updated activity a will be added, and the relative data flow connections connecting the data places that are no longer input data of a and the input parameter transition t_a^I of the updated activity a will be removed. The relative data flow connections connecting the output parameter transition t_a^O of the updated activity a and the new added output data places will be added, and the relative data flow connections connecting the output parameter transition t_a^O of the updated activity a and the data places that are no longer output data of a will be removed.

6.2 Change Patterns of Services

Service change patterns are introduced from the perspectives of two of the four elements of S: Op and SN. As defined in Section 3, four types of elements consist in a service, i.e., $S = (sId, Op, SN, sType)$. Because the change types about changing a service's identification or changing the type of a service (e.g., from a provided service to an invoked service) are not considered in this work, only the elements of operation set Op and the control flow of the operations SN will be affected by the service change patterns discussed in this section.

6.2.1 Operation Existence Changes

This type of changes refers to as the changes of the existence of operations. An operation existence change occurs due to adding or removing operations in a service. The "replace" operation is also considered in this work as a combination which "removing" the targeting operation at first and then "adding" another operation to the position of the deleted one.

SC-1 ***Delete an operation***

Description: The *delete an operation* change refers to that an operation is removed from a service involved in an SBP.

In Formal: Assume that the removed operation is o, for the set of operations Op' of the changed service S',

$$Op' = Op - \{o\}.$$

For the service net SN' of the changed service S', the change operations on the original SN are the same as the change pattern *PC-1* in the last section.

SC-2 ***Add an operation sequentially***

Description: The *add an operation sequentially* change refers to that an operation is inserted before/after a specific node of the control flow of a service involved in an SBP. The specific node can be the start/end event, any gateway, or another operation.

In Formal: Assume that the added operation is o, for the set of operations Op' of the changed service S',

$$Op' = Op \cup \{o\}.$$

For the service net SN' of the changed service S', the change operations on the original SN are the same as the change pattern *PC-2* in the last section.

SC-3 ***Add an operation in parallel to existing operations***

Description: The *add an operation in parallel to existing operations* change refers to that an operation is inserted as a parallel branch to one or more existing operations of the control flow of a service involved in an SBP.

In Formal: Assume that the added operation is o, for the set of operations Op' of the changed service S',

$$Op' = Op \cup \{o\}.$$

For the service net SN' of the changed service S', the change operations on the original SN are the same as the change pattern *PC-3* in the last section.

SC-4 ***Add an operation conditionally to existing operations***

Description: The *add an operation conditionally to existing operations* change refers

to that an operation is inserted as an XOR branch to one or more existing operations of the control flow of a service involved in an SBP.

In Formal: Assume that the added operation is o, for the set of operations Op' of the changed service S',

$$Op' = Op \cup \{o\}.$$

For the service net SN' of the changed service S', the change operations on the original SN are the same as the change pattern *PC-4* in the last section.

SC-5 ***Replace an operation by another one***

Description: The *replace an operation by another one* change refers to that an operation is replaced by another one in a service involved in an SBP.

In Formal: Assume that the replaced operation is o and the alternative operation is u. , for the set of operations Op' of the changed service S',

$$Op' = Op \cup \{u\} - \{o\}.$$

For the service net SN' of the changed service S', the change operations on the original SN are the same as the change pattern *PC-5* in the last section.

6.2.2 Operation Order Changes

This type of changes refers to as the change of invocation order of operations of a service involved in an SBP. By this type of changes, the existence of operations is not changed.

SC-6 ***Add a conditional shortcut***

Description: The *add a conditional shortcut* change refers to that a "conditional shortcut" is added to make a fragment be invoked conditionally in a service involved in an SBP.

In Formal: For the set of operations Op' of the changed service S',

$$Op' = Op.$$

For the service net SN' of the changed service S', the change operations on the original SN are the same as the change pattern *PC-6* in the last section.

SC-7 ***Delete a conditional shortcut***

Description: The *delete a conditional shortcut* change refers to that a "conditional shortcut" is removed from a service involved in an SBP.

In Formal: For the set of operations Op' of the changed service S',

$$Op' = Op.$$

For the service net SN' of the changed service S', the change operations on the original SN are the same as the change pattern *PC-7* in the last section.

SC-8 ***Add a loop***

Description: The *add a loop* change refers to that a "loop" is added to make a fragment be repeatedly invoked in a service involved in an SBP.

In Formal: For the set of operations Op' of the changed service S',

$$Op' = Op.$$

For the service net SN' of the changed service S', the change operations on the original SN are the same as the change pattern *PC-8* in the last section.

SC-9 ***Delete a loop***

Description: The *delete a loop* change refers to that a "loop" is removed from a service involved in an SBP.

In Formal: For the set of operations Op' of the changed service S',

$$Op' = Op.$$

For the service net SN' of the changed service S', the change operations on the original SN are the same as the change pattern *PC-9* in the last section.

SC-10 ***Add a control dependency***

Description: The *add a control dependency* change refers to that a control dependency is inserted between two parallel branches of the control flow of a service involved in an SBP.

In Formal: For the set of operations Op' of the changed service S',

$$Op' = Op.$$

For the service net SN' of the changed service S', the change operations on the original SN are the same as the change pattern *PC-10* in the last section.

SC-11 ***Delete a control dependency***

Description: The *delete a control dependency* change refers to that a control dependency is removed from a service involved in an SBP.

In Formal: For the set of operations Op' of the changed service S',

$$Op' = Op.$$

For the service net SN' of the changed service S', the change operations on the original SN are the same as the change pattern *PC-11* in the last section.

6.3 Discussion

In this chapter, a set of change patterns for the changes occurring in the internal business process and the services involved in an SBP have been classified and identified. The internal process changes are classified into three major types as: *activity existence changes*, *activity order changes*, and *process data changes*. The involved service changes are classified into two major types as *operation existence changes* and *operation order changes*. Fourteen internal process change patterns and eleven involved service change patterns are identified with both the informal and formal description based on the SBP model defined in Chapter 3. Different cases for each change pattern are considered in this work as discussed in the formal description of these change patterns. These change patterns can be utilized as change primitives for handling complex change requirements. The change patterns proposed in this chapter provide a solid foundation the change management in SBP. In order to manage change, the first step is to "know" the change which means to analyze what the change it is, how to implement the change on the original SBP, and is there any impact along with the change? For analyzing a change on a formal basis, it is necessary to have a standardized and reusable method to assist the designer to disassemble the change into a set of change

primitives and then imply them on the original SBP model. After implying a change on an SBP by the assistance of respective change patterns, the change impact analysis and how to deal with the impact will be discussed in the following chapter.

7

Change Management Framework For Service-based Business Processes

Due to the distributed and dynamic nature, changes of the internal business process and involved services of an SBP can happen from time to time. There are dependencies between

the internal business process and involved services [35]; therefore, the changes occurring in one side may affect the other one in certain degrees and can propagate in the whole SBP like the cascading effect which makes the change management in SBP a challenging problem [36, 37].

Figure 7.1 shows an example of a change occurring on the travel agency's SBP from Figure 3.1. Some new regulations request that the financial company, the provider of the payment service, must send both the electronic payment confirmation and paper payment confirmation to the payers. Therefore, to get the postal address of each payer, the service designer decides to add an operation "*Request Mailing Address*" sequentially between the operations "*Receive Request*" and "*Request to Sign Agreement*" (Change 1 in Figure 7.1). Due to the dependencies between the invoked payment service and the internal business process of the travel agency, the activity "*Send Mailing Address*" need to be inserted correspondingly which is associated with the new added operation "*Request Mailing Address*" sequentially between the activities "*Invoke Payment Service*" and "*Sign Agreement*" (Change 2 in Figure 7.1). Because there is no any address data of customers which has ever been collected by the internal business process, the process designer decides to add an activity "*Request Mailing Address*" in parallel with the activities "*Prepare Bill*" and "*Send Bill*" which can collect the data of mailing address from each customer (Change 3 in Figure 7.1). Due to the dependencies between the internal business process of the travel agency and the flight inquiry and booking service provided by the travel agency, the operation "*Request Mailing Address*" need to be added correspondingly in parallel with the operation "*Send Bill*" which is associated with the new added activity "*Request Mailing Address*" in the internal business

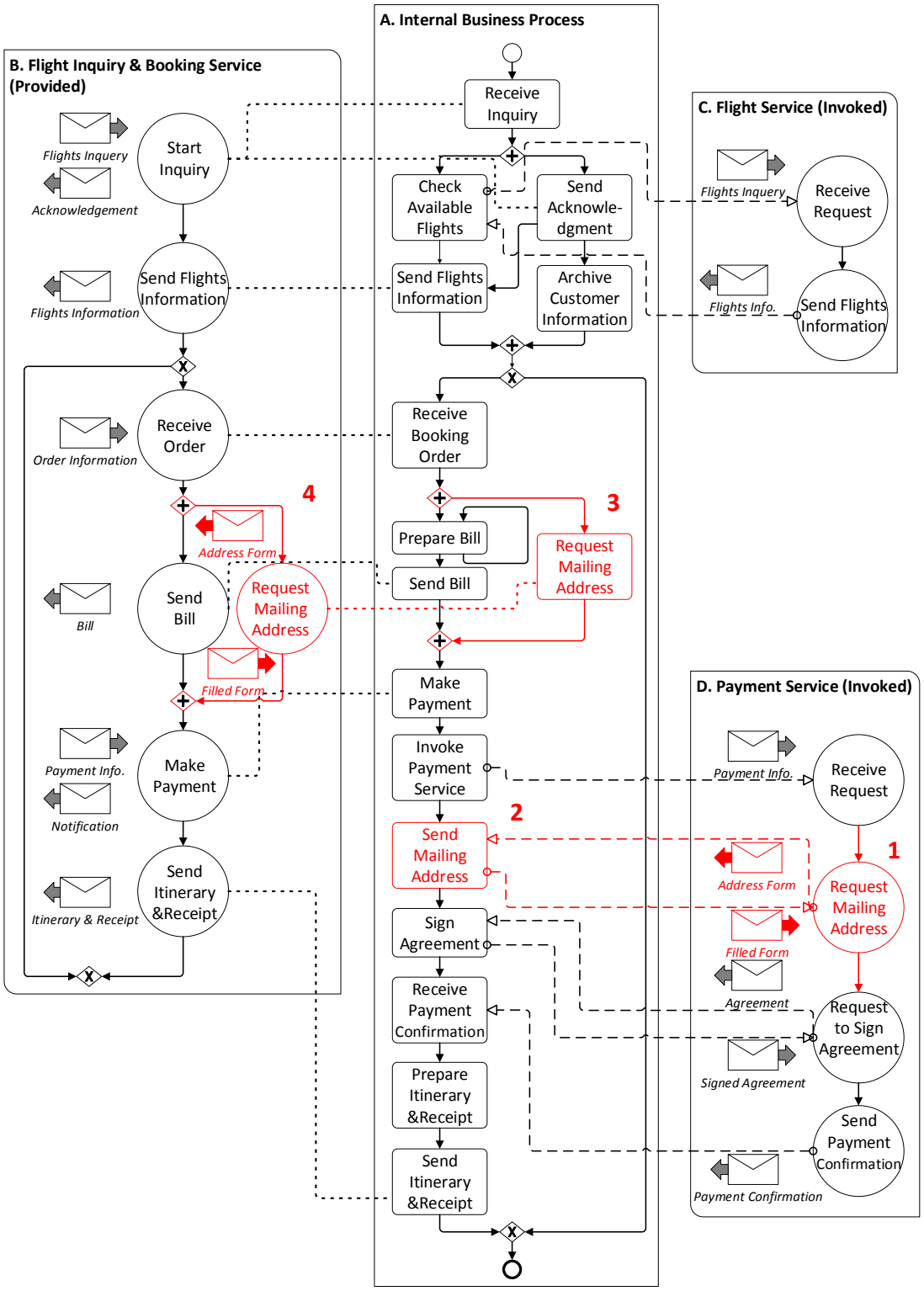

FIGURE 7.1: An Example of SBP Change

process (Change 4 in Figure 7.1).

According to this example, it is easy to see that in the context of SBP, a service change may require further changes of one or multiple activities of the internal business process. A change of an activity belonging to the internal business process may affect other activities and related services of the process. Change propagation refers to that a single change of an activity or a service causes a series of changes associated with activities and/or services like the cascading effect. Therefore, change management in the context of SBP is a challenging issue due to the complex dependencies between internal business processes and services involved. To deal with complicated situations of SBP changes, it is necessary to have a framework to help the change management, in particular, to make the change propagation under control. In this chapter, based on the proposed SBP model and SBP change patterns, a change management framework is proposed for managing SBP changes. This framework can provide guidelines and an enabling tool for SBP designers to deal with change management problems in the real world.

7.1 Management of Internal Business Process Changes

7.1.1 Identifying the Primary Process Change

In this step, when a change request raises on the internal business process of an SBP, the change request will be marked as a **primary process change**. Based on the change patterns identified in Chapter 6, the primary process change will be analyzed and decomposed into several primitive changes.

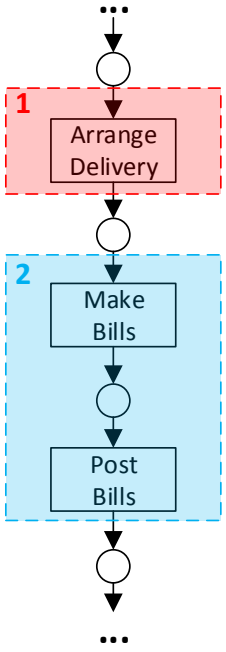

Figure 7.2: An Example of Primary Change

An example will demonstrate how this step works as shown in Figure 7.2. Figure 7.2 shows a C-net fragment of the internal business process of an SBP. A change request raises which is to make part 1 and part 2 be executed in parallel. This change is marked as the primary process change. As the result of the analysis of this primary process change, based on the change patterns identified in Chapter 6, the primary process change is decomposed into four primitive changes which are:

- firstly, delete the activity "*Make Bills*" (PC-1 in Chapter 6),
- secondly, delete the activity "*Post Bills*" (PC-1 in Chapter 6),
- thirdly, add the activity "*Make Bills*" in parallel to the activity "*Arrange Delivery*" (PC-3 in Chapter 6),

- finally, add the activity "*Post Bills*" sequentially after the activity "*Make Bills*" (PC-2 in Chapter 6).

7.1.2 Implementing the Primary Process Change

According to the formal descriptions in Chapter 6 of the change primitives decomposed from the primary process change, the primary process change will be implemented on the SBP model.

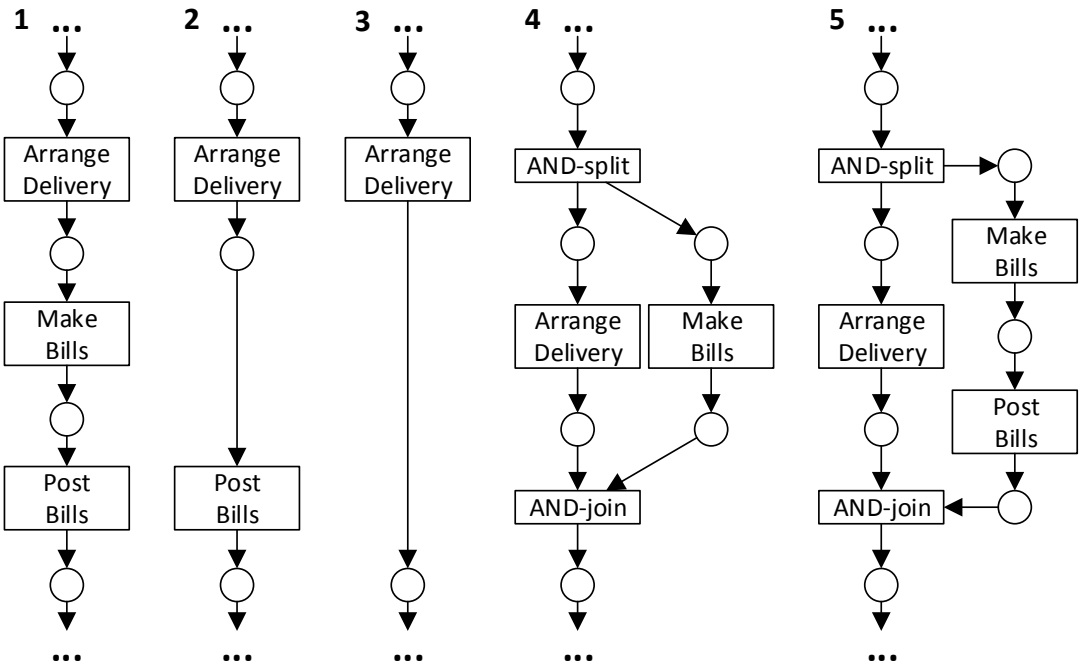

Figure 7.3: An Example Showing How to Implement a Change

To implement the primary process change of the example in Figure 7.2, the four primitive changes in step 1 will be implemented on the original SBP model one after another which follows the order in step 1. As shown in Figure 7.3, Figure 7.3-1 is the original C-net of

the SBP. Because either the input or output place of the transition representing the activity "*Make Bills*" is an "1-in-1-out" place, the implementation of the first primitive change of deleting the activity "*Make Bills*" on the C-net will follow the change operations in PC-1, case i (Chapter 6). Figure 7.3-2 is the result of implementing the first primitive change. Because either the input or output place of the transition representing the activity "*Post Bills*" is an "1-in-1-out" place, the implementation of the second primitive change of deleting the activity "*Post Bills*" on the C-net will follow the change operations in PC-1, case i (Chapter 6). Figure 7.3-3 is the result of implementing the second primitive change. Because the transition representing the activity "*Arrange Delivery*" has only one input place and one output place, the implementation of the third primitive change of adding the activity "*Make Bills*" in parallel to the activity "*Arrange Delivery*" on the C-net will follow the change operations in PC-3, case i (Chapter 6). Figure 7.3-4 is the result of implementing the third primitive change. Because the transition representing the activity "*Make Bills*" has only one output place, the implementation of the fourth primitive change of adding the activity "*Post Bills*" sequentially after the activity "*Make Bills*" on the C-net will follow the change operations in PC-2, case i (Chapter 6). Figure 7.3-5 is the result of implementing the fourth primitive change. After implementing the primitive changes in the control flow, the data flow of the original SBP will be changed correspondingly.

7.1.3 Analyzing and Handling the Change Impact on the Process Control Flow

The correctness of the changed C-net from step 2 will be checked by using the verification method of CF soundness proposed in Chapter 5. If the verification result is sound, the following part of this step will be skipped.

If the verification result is not sound and several control flow error(s) are detected, for fixing the error(s), further changes on the control flow of the internal business process will be designed and implemented as solutions by the designers. The corresponding solutions depend on the specific cases of detected errors.

7.1.4 Analyzing and Handling the Change Impact on the Process Data Flow

The correctness of the changed CD-net (combined control and data flow net) from step 2 will be checked, and data flow anomalies will be detected by using the verification method of DF (data flow) soundness proposed in Chapter 5. If no data flow anomaly is detected, the following part of this step will be skipped.

If any data flow anomaly is detected, for handling the detected anomaly(s), further changes in the control flow of the internal business process will be designed and implemented as solutions by the designers. For each specific type of data flow anomaly, corresponding solution guidelines are proposed as follows.

1 **Missing Target Data:** If a *missing target data* anomaly is detected from the changed

SBP. It means that the process is unable to achieve its business goal. The designers must make a solution to fix the anomaly. The potential solutions for handling this data flow anomaly are:

i. to remove the missed target data from the target data set by a further change of updating the target data (PC-13).

ii. to enable an existing activity which is not in any conditionally executed branch to produce the missed target data by a further change of updating an activity (PC-14).

iii. to implement a further change of replacing an existing activity which is not in any conditionally executed branch by a new one which can produce the missed target data (PC-5).

iv. to implement a further change of adding a new activity which can produce the missed target data, the added activity must not be in any conditionally executed branch (PC-2 or PC-3).

2 **Missing Input Data:** If a *missing input data* anomaly is detected from the changed SBP, it means that an input data of an activity of the SBP's internal business process has never been initialized when the process terminates. The potential solutions for handling this data flow anomaly are:

i. to remove the missed input data from the input parameter of the anomaly activity by a further change of updating the activity (PC-14).

ii. to enable an existing activity which is undoubtedly executed before the anomaly activity to produce the missed input data for the anomaly activity by a further change of updating the existing activity (PC-14).

iii. to implement a further change of replacing an existing activity which is undoubtedly executed before the anomaly activity by a new one which can produce the missed input data for the anomaly activity (PC-5).

iv. to implement a further change of adding a new activity which is certainly executed before the anomaly activity to produce the missed input data for the anomaly activity (PC-2 or PC-3).

3 **Delayed Initialization:** If a *delayed initialization* anomaly is detected from the changed SBP, it means that an input data of an activity has not been initialized when the activity is enabled, but the input data will be initialized by a subsequent activity before the process terminates. The potential solutions for handling this data flow anomaly are:

i. to remove the delayed initialized data from the input parameter of the anomaly activity by a further change of updating the activity (PC-14).

ii. to enable an existing activity which is certainly executed before the anomaly activity to produce the delayed initialized for the anomaly activity by a further change of updating the existing activity (PC-14).

iii. to implement a further change of replacing an existing activity which is certainly executed before the anomaly activity by a new one which can produce the delayed

initialized for the anomaly activity (PC-5).

iv. to implement a further change of adding a new activity which is undoubtedly executed before the anomaly activity in order to produce the delayed initialized for the anomaly activity (PC-2 or PC-3).

v. to implement a further change of deleting an existing activity which produces the delayed initialized data (PC-1) and then adding the deleted activity which is certainly executed before the anomaly activity to produce the delayed initialized data for the anomaly activity (PC-2 or PC-3).

vi. to implement a further change of deleting the anomaly activity (PC-1) and then adding the deleted activity which is certainly executed after the producer activity of the delayed initialized data (PC-2 or PC-3).

4 **Uncertain Initialization:** If an *uncertain initialization* anomaly is detected from the changed SBP. It means that the two activities, the data producer, and the data consumer, are in two parallel branches. The potential solutions for handling this data flow anomaly are same as the solutions for handling the *delayed initialization* anomaly.

5 **Conditionally Initialized Target Data:** If a *conditionally initialized target data* anomaly is detected from the changed SBP, it means that the producer activity of the conditionally initialized target data is in a conditionally executed branch. The potential solutions for handling this data flow anomaly are:

i. to remove the conditionally initialized target data from the target data set by a further change of updating the target data (PC-13).

ii. to enable an existing activity which is not in any conditionally executed branch to produce the conditionally initialized target data by a further change of updating an activity (PC-14).

iii. to implement a further change of replacing an existing activity which is not in any conditionally executed branch by a new one which can produce the conditionally initialized target data (PC-5).

iv. to implement a further change of adding a new activity which can produce the conditionally initialized target data, the added activity must not be in any conditionally executed branch (PC-2 or PC-3).

v. to implement a further change of deleting the activity in condition branch which produces the conditionally initialized target data (PC-1) and then adding the deleted activity which is certainly executed before the termination of the process (PC-2 or PC-3).

6 **Conditionally Initialized Input Data:** If a *conditionally initialized input data* anomaly is detected from the changed SBP, it means that an input data of an activity can only be initialized under specific routing condition(s), under other condition(s) it can never be initialized. The potential solutions for handling this data flow anomaly are same as the solutions for handling the *delayed initialization* anomaly.

7 **Redundant Legacy Data:** If a *redundant legacy data* anomaly is detected from the changed SBP. It means that a legacy data of the process is neither a target data nor an input data for any activities. In most cases, the *redundant legacy data* anomaly

is ignored by the designer due to it will not cause any severe impact. If a designer still decides to fix this type of anomaly, the only potential solution is to remove the redundant data from the legacy data set of the SBP by a further change of updating the legacy data (PC-12).

8 **Redundant Output Data:** If a *redundant output data* anomaly is detected from the changed SBP, it means that an output data of an activity is neither a target data nor an input data for any activities. In most cases, the *redundant output data* anomaly is ignored by the designer due to it will not cause any severe impact. If a designer still decides to fix this type of anomaly, the only potential solution is to remove the redundant data from the output parameter of the anomaly activity by a further change of updating an activity (PC-14).

9 **Conditionally Redundant Legacy Data:** If a *conditionally redundant legacy data* anomaly is detected from the changed SBP. It means that a legacy data of the process is an input data of an activity only under certain routing conditions. In most cases, the *redundant legacy data* anomaly is ignored by the designer due to it will not cause any severe impact.

10 **Conditionally Redundant Output Data:** If a *conditionally redundant output data* anomaly is detected from the changed SBP. It means that an output data of an activity is an input data of another activity only under certain routing conditions. In most cases, the *redundant output data* anomaly is ignored by the designer due to it will not cause any severe impact.

11 **Multiple Initialized Data:** If a *multiple initialized data* anomaly is detected from the changed SBP, it means that a data element is initialized more than once. If there is a data version control system for the process, the *multiple initialized data* anomaly will be ignored by the designer. If not, a mechanism for manage the data version must be developed to ensure that the multiple versions of the data will not cause a conflict during the execution of the process.

7.1.5 Analyzing and Handling the Change Impact on Services Involved

The last four steps deal with the change propagation within the internal business process element of an SBP. The change propagation across different elements (between the internal business process and services involved in an SBP) will be discussed in this step.

Because of the complex dependencies between the internal business process and involved services of an SBP as discussed in Chapter 3, the changes on the internal business process may not only propagate within the process but also propagate to the services involved in the SBP, which may cause *inconsistency* and *incompatibility* as discussed in Chapter 5. For analyzing the impact by an internal process change on services involved in an SBP, the *consistency* between each provided service and the changed internal business process of the SBP and the *compatibility* between each invoked service and the changed internal business process of the SBP will be checked by the verification methods in Chapter 5.

If the verification result shows that each service involved in the SBP is consistent or compatible with the changed internal business process, there is no need for further changes

on services involved in the SBP. If not, for the provided service which is inconsistent with the changed internal business process, the service will be changed in order to ensure that the postmark PM^S of the token in the ending place o^S of the S-net of the provided service and the postmark PM^C of the token in the ending place o^C of the C-net of the changed internal business process meet $PM^S = PM^C$ (Theorem 2).

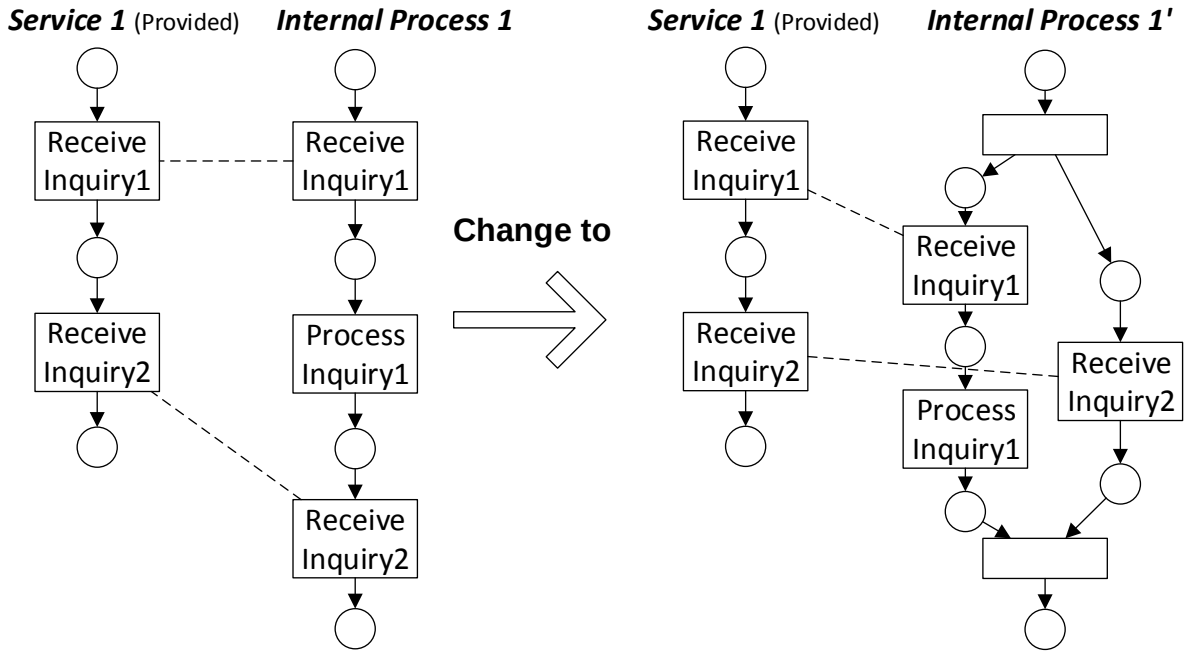

Figure 7.4: An Example of the Inconsistency Caused by Process Change

An example is shown in Figure 7.4. Firstly, the internal business process 1 is updated to the internal business process $1'$ based on a change requirement of moving the activity *ReceiveInquiry2* in parallel with the other two activities. Secondly, the postmark of the token in the final markings of the service 1 and the changed internal business process $1'$ are calculated by the method in Chapter 5 as: $PM^S = I \blacktriangleright ReceiveInquiry1 \blacktriangleright ReceiveInquiry2$ and $PM'^C = I \blacktriangleright (ReceiveInquiry1 \blacklozenge ReceiveInquiry2)$. The difference between PM^S and

PM'^{C} indicates that the provided service is inconsistent with the internal business process. Thirdly, it is necessary to have a further change on the provided service 1 to put the operations *ReceiveInquiry1* and *ReceiveInquiry2* into two parallel branches.

For the invoked service which is incompatible with the changed internal business process, the service will be changed in order to ensure that the combined net of their respective rS-net and rC-net is CF sound (Theorem 3). The potential solutions for handling the incompatibility are:

i. The organization who runs this SBP has to decide on whether to invoke a new service which is compatible with the changed internal business process and stop interacting with the current one.

ii. If the organization decides to continue invoking the current one, they must communicate with the service provider to ask whether the invoked service can be changed correspondingly.

7.2 Management of Service Change

7.2.1 Identifying the Primary Service Change

In this step, when a change request raises on a service involved in an SBP, the change request will be marked as a **primary service change**. When the primary service change occurs on an invoked service, it is necessary for the designer of the SBP to make a decision on whether to keep interacting with the service or replace it with another one. If the designer of the SBP decides to keep interacting with the changed invoked service or if the primary service change

occurs on a provided service, based on the service change patterns identified in Chapter 6, the primary service change will be analyzed and decomposed into several primitive changes same as the method of identifying the primary process changes proposed in the last section.

7.2.2 Implementing the Primary Service Change

According to the formal descriptions in Chapter 6 of the change primitives decomposed from the primary service change, the primary service change will be implemented on the SBP model same as the method of identifying the primary process changes proposed in the last section.

7.2.3 Analyzing and Handling the Change Impact on the Service Control Flow

The correctness of the changed S-net from step 2 will be checked by using the verification method of CF soundness proposed in Chapter 5. If the verification result is sound, the following part of this step will be skipped. Under normal circumstances, the control flow of a service is much simpler than the control flow of the internal business process of an SBP. Therefore, the changes of services rarely cause errors on the control flows of these services.

If the verification result is not sound and several control flow error(s) are detected, for fixing the error(s), further changes on the control flow of the service will be designed and implemented as solutions by the designers. The corresponding solutions depend on the specific cases of detected errors.

7.2.4 Analyzing and Handling the Change Impact on the Internal Business Process

The last three steps deal with the change propagation within the service element of an SBP. The change propagation across different elements (between the service and the internal business process of the SBP) will be discussed in this step.

Because of the complex dependencies between the internal business process and involved services of an SBP as discussed in Chapter 3, the changes on a service involved may not only propagate within the service but also propagate to the internal business process of the SBP, which may cause *inconsistency* and *incompatibility* as discussed in Chapter 5. For analyzing the impact by a service change on the internal business process of an SBP, if the changed service is a provided service, the *consistency* between the service and the internal business process of the SBP will be checked; if the changed service is an invoked service, the *compatibility* between the service and the internal business process of the SBP will be checked.

If the verification result shows that the changed service involved in the SBP is consistent or compatible with the internal business process, there is no need for further changes on the internal business process of the SBP. If not, for the changed provided service which is inconsistent with the internal business process, the process will be changed in order to ensure that the postmark PM^C of the token in the ending place o^C of the C-net of the internal business process and the postmark PM^S of the token in the ending place o^S of the S-net of the changed provided service meet $PM^C = PM^S$ (Theorem 2).

An example is shown in Figure 7.5. Firstly, the provided service, service 1, is updated to

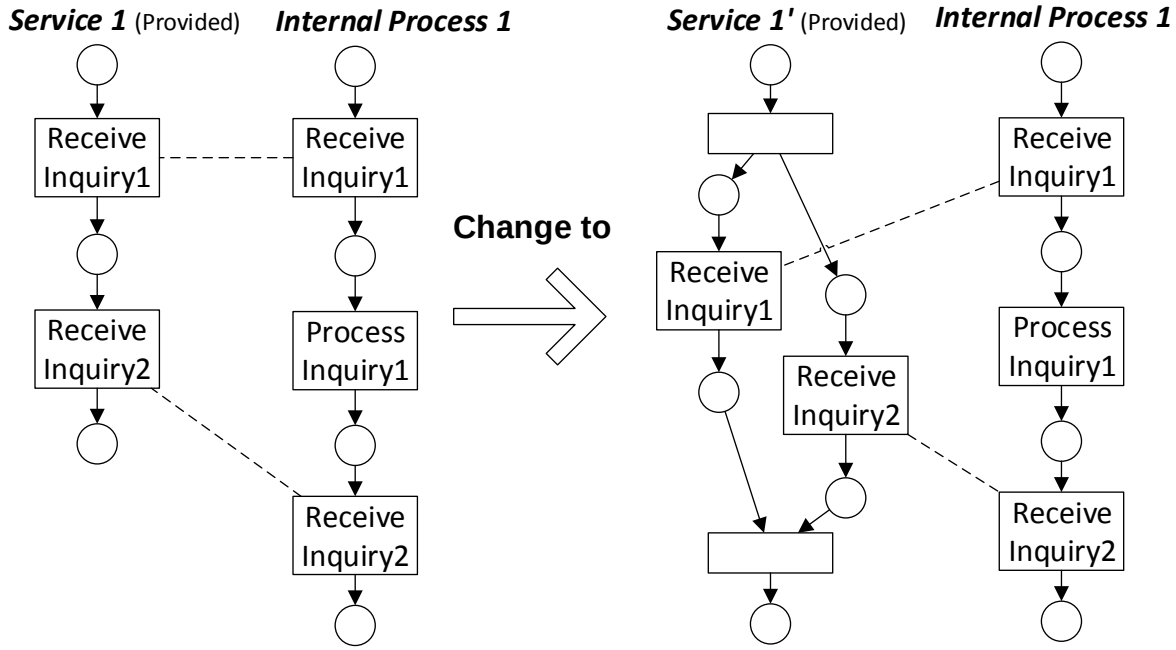

Figure 7.5: An Example of the Inconsistency Caused by Provided Service Change

service $1'$ based on a change requirement. Secondly, the labels of the token in the final markings of the internal business process and service $1'$ are calculated by this method as $PM^C = I \blacktriangleright ReceiveInquiry1 \blacktriangleright ReceiveInquiry2$ and $PM^S = I \blacktriangleright (ReceiveInquiry1 \blacklozenge ReceiveInquiry2)$. The difference between l_P and $l_{S1'}$ indicates that the changed provided service is inconsistent with the internal business process. Thirdly, it is necessary to have a further change in the internal business process to put activities *ReceiveInquiry1* and *ReceiveInquiry2* into two parallel branches. Different with the change propagation from the internal business process to a provided service of an SBP, the change propagation from a provided service to the internal business process of an SBP has multiple potential solutions which can meet $PM^C = PM^S$. For the example shown in Figure 7.5, the potential solutions of the change propagation from the changed provided service $1'$ to the internal business process are proposed in Figure 7.6.

The designers of the SBP must select one solution of a process change from the candidate solutions.

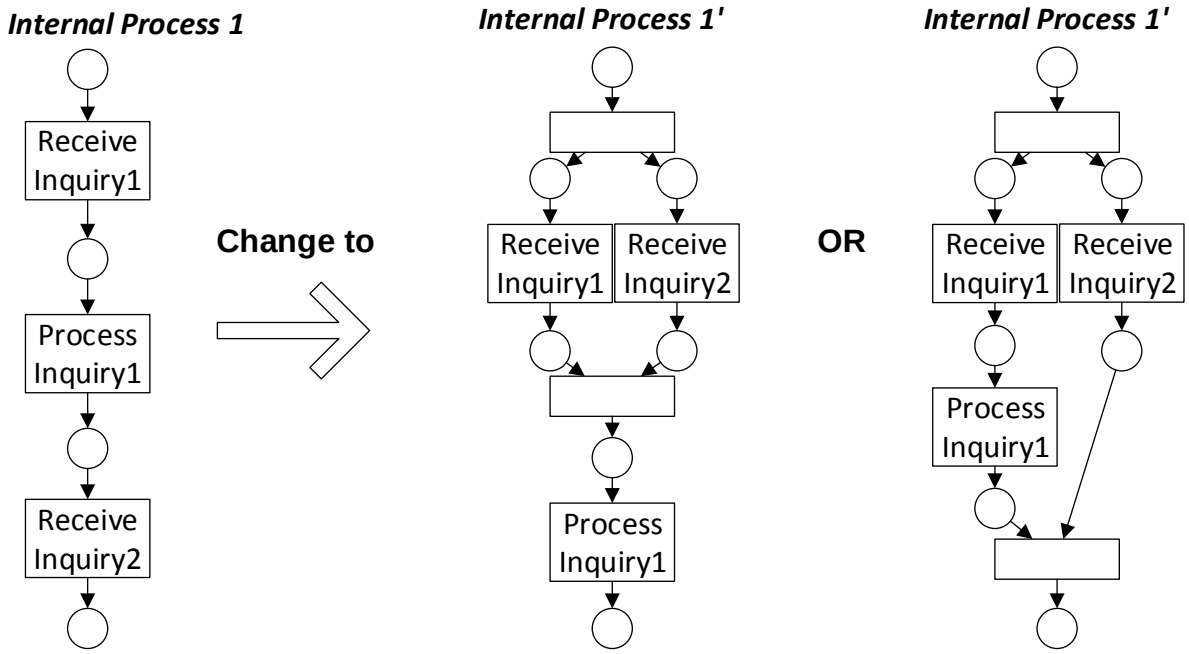

Figure 7.6: The Potential Solutions for the Change Example in Figure 7.5

7.2.5 Managing the Internal Business Process Change

After a decision is made, the selected candidate solution of a change in the internal business process will become a **primary process change**. For managing this change, the same method is used as the method of managing internal business process changes in the last section.

7.3 Discussion

Based on the model definitions and correctness verification introduced in the former chapters, in this chapter, a change management framework for managing the changes in SBPs is proposed. For handling a process change or a service change, the proposed framework can be utilized as guidelines and an enabling tool to manage changes of SBPs. This change management framework provides semi-automatic supports for software developers, and manual efforts are still necessary for the complete solution of the change management in SBPs.

8

Conclusion and Future Work

Service-Oriented Computing (SOC) paradigm and web service technologies are important enablers for cooperation and collaboration of organizations. The business processes of such organizations are referred to as service-based business processes (SBP). There are two significant differences between the SBP management and the traditional business process management (BPM). The first difference is that there are more types of components contained in

SBPs and the dependencies between the SBP components are more complex. It is vital that the rich types of components and the complex dependencies among them are correctly developed and maintained. Therefore, it is necessary to develop a practical and robust method to model and verify the correctness of SBP. The second difference is that the SBP management requires more consideration of changes due to the dynamic nature of the service-based components in SBPs. SBPs subject to changes arising from both the internal and external requirements of the involved organizations. The changes can affect the correctness of SBPs which may lead to failures. Therefore, it is necessary to have a method to manage the changes in SBPs. The contribution of this work is summarized as follows:

- An SBP model has been defined for capturing the characteristics of different types of components and the dependencies among them. Petri net language is employed for building up the SBP model due to its formal semantic, graphical representation, state-based features, and the support by a rich set of existing analysis tools. SBP representing an SBP is defined on the top of the SBP model which is a 4-tuple contains four elements: the internal business process (IP), the involved service set (Σ), and the relations between the process and the services (R_S and R_O). An IP consists of the activity set (Act), the data set ($Data$), the control flow (CN), and the data flow (CDN). A service $S(S \in \Sigma)$ consists of the identification of the service (sId), the type of the service ($sType$), the operation set of the service (Op), and the control flow of these operations (SN). The relations between the internal process and the services involved in the SBP are represented by R_S and R_O. An $r_S \in R_S$ describes the relationship between an activity and a service which means that the activity interacts

with the invoked service or the activity supports the provided service. An $r_O \in R_O$ describes the relations between an activity and an operation of a service which means that the activity calls the operation of the invoked service or the activity carries out the operation of the provided service. The proposed model can be utilized as a foundation for further analysis and management work.

- Two classes of SBP design patterns have been identified which are control flow patterns and process-service relation patterns supported by the SBP model in this work. Seven control flow patterns have been identified with both formal and informal descriptions which are *sequence*, *AND-split*, *AND-join*, *XOR-split*, *XOR-join*, *Loop*, and *control dependency*. A consolidated and recurring basis is provided by these control flow patterns for modeling the complex control flow structures of SBPs. Due to the support of the *control dependency pattern*, network structures may exist in the control flow of an SBP which may lead to some challenging issues in the verification phase of this SBP. Due to different granularity levels of activities from the internal business process and their related operations from services involved in an SBP, four process-service relation patterns have been identified with both formal and informal descriptions which are *1A-to-1syncO* (one activity is related with one synchronous operation), *1A-to-1asyncO* (one activity is related with one asynchronous operation), *2A-to-1syncO* (two activities are related with one synchronous operation), and *1A-to-2asyncO* (one activity is related with two asynchronous operations). These SBP design patterns provide reusable solutions for SBP designers to deal with recurrent problems when building up SBP models.

- A set of correctness properties of SBP have been specified including the *CF soundness* (control flow soundness), the *DF soundness* (data flow soundness), the *consistency* between a provided service and the internal business process, and the *compatibility* between an invoked service and the internal business process. For verifying the *CF soundness*, a traditional verification method of Petri net has been employed which transfers the soundness verification problem into the *liveness* and *boundedness* verification problem because a rich set of existing tools can support the verification of these two properties of a Petri net. For verifying the *DF soundness*, a set of data flow anomalies have been classified and specified, and an algorithm for detecting these data flow anomalies has been developed correspondingly. For verifying the *consistency* between a provided service and the internal business process, an algebra-based method has been proposed. Different from other existing works, this method provides support for handling the network structures. For verifying the *compatibility* between an invoked service and the internal business process, a Petri net based method has been proposed which is similar to the method for verifying the *CF soundness*. These identified correctness properties and corresponding verification methods can be utilized as enablers by SBP developers for guaranteeing the absence of errors during the design and maintenance phases of SBPs.

- For managing changes in SBPs, at first, fourteen internal business process change patterns and eleven service change patterns have been identified on the basis of the SBP model. The internal process change patterns have been classified into three major types as: *activity existence changes*, *activity order changes*, and *process data changes*.

The service change patterns have been classified into two major types as *operation existence changes* and *operation order changes*. For handling complex SBP changes, these SBP change patterns can be used to decompose complex changes into primitive changes. These SBP change patterns can help the designers to analyze and implement complex change requirements on the original SBP model which is the first and the most important step for analyzing the changes impact and managing the changes.

Based on the SBP model, design patterns, correctness verification methods, and change patterns, a change management framework for handling SBP changes has been proposed in this work. In this framework, there are fives steps for handling the changes occurring in the internal business process of an SBP, which are: 1.*identify the primary process change*, 2. *implementing the primary process change*, 3. *analyzing and handling the change impact on the process control flow*, 4. *analyzing and handling the change impact on the process data flow*, 5. *analyzing and handling the change impact on the services involved*. There are fives steps for handling the changes occurring in a service involved in an SBP, which are: 1.*identify the primary service change*, 2. *implementing the primary service change*, 3. *analyzing and handling the change impact on the service control flow*, 4. *analyzing and handling the change impact on the internal business process*, 5. *managing the internal business process change*. This change management framework provides guidelines for SBP designers to dealing with change requirements arising in SBPs.

Change management is a traditional topic in software engineering area. Due to the rapid development of web technologies in recent two decades, service-based business processes have

been applied by an increasing number of organizations. A huge amount of existing researches have discussed the change management in the fields of business processes and web services, but few of them choose the service based business processes as the research setting with the consideration on the dependencies between the business processes and services involved. In this work, a conceptual model has been defined for modeling SBPs, and the correctness properties with corresponding verification methods have been developed based on the SBP model. With the assistance of the SBP model and verification, a framework for managing changes in SBP has been proposed.

The change management framework is still far away from the mature application which can solve the practical problems. The future research directions in SBP"change management are discussed as follows:

- The SBP model will be upgraded for modeling more features of SBPs. The resources involved in the internal business process of an SBP will be considered. Because in the practical cases, not only the control structures or data elements of a process can be affected by changes, but also the resources involved in the process can be affected. The change impact on process resources may lead to further changes in resource allocation or resource authorization. Another important feature of processes is the temporal feature of SBPs. In this work, the temporal feature has not been considered but it is very critical for the SBPs which needs precise time control. The QoS (quality of service) features of the services involved in an SBP will be contained in the SBP model for quantitatively measuring the change impact on a specific service. There are relations between the three new added features of SBP model. For example, the change on the

temporal feature of the internal business process of an SBP can affect the QoS feature of a specific service involved in the SBP and also can affect the resource allocation of the process.

- An automated change reaction mechanism will be developed. The change management framework in this work can only be utilized as guidelines but can not guarantee the optimal solution for handling SBP changes. With the automated change reaction mechanism, when a change occurs in an SBP, the optimal solutions for handling this changes which can minimize the negative impact will be generated automatically. To realize this imagination, the change impact in SBPs needs to be quantitatively measured at first. Then, by utilizing the change impact measurement, the possible solutions for handling a change will be measured and sorted. This change reaction mechanism can improve the effectiveness and efficiency of change management in the dynamic context of SBPs.

References

[1] M. P. Papazoglou. *Service-oriented computing: Concepts, characteristics and directions.* In *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on*, pp. 3–12 (IEEE, 2003).

[2] A. Martens. *Analyzing web service based business processes.* In *International Conference on Fundamental Approaches to Software Engineering*, pp. 19–33 (Springer, 2005).

[3] W. M. van der Aalst. *Making work flow: On the application of petri nets to business process management.* In *International Conference on Application and Theory of Petri Nets*, pp. 1–22 (Springer, 2002).

[4] W. M. Van Der Aalst, A. H. Ter Hofstede, and M. Weske. *Business process management: A survey.* In *International conference on business process management*, pp. 1–12 (Springer, 2003).

[5] Y. Breitbart, A. Deacon, H.-J. Schek, A. Sheth, and G. Weikum. *Merging application-centric and data-centric approaches to support transaction-oriented multi-system workflows.* ACM Sigmod Record **22**(3), 23 (1993).

[6] W. Van Der Aalst and K. M. Van Hee. *Workflow management: models, methods, and systems* (MIT press, 2004).

[7] F. Leymann and D. Roller. *Production workflow: concepts and techniques* (2000).

[8] D. C. Marinescu. *Internet-based workflow management.* Toward a Semantic Web, New York (2002).

[9] P. Lawrence. *Workflow handbook 1997* (John Wiley & Sons, Inc., 1997).

[10] M. Weske. *Business Process Management* (Springer, 2012).

[11] M. P. Papazoglou. *Whats in a service?* In *European Conference on Software Architecture*, pp. 11–28 (Springer, 2007).

[12] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. *Service-oriented computing: State of the art and research challenges.* Computer **40**(11) (2007).

[13] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. *Service-oriented computing: A research roadmap.* International Journal of Cooperative Information Systems **17**(02), 223 (2008).

[14] H. Haas and A. Brown. *Web services glossary.* W3C Working Group Note (11 February 2004) **9** (2004).

[15] *Web services description language (wsdl) 1.1.* Tech. rep., W3C (2001). Https://www.w3.org/TR/2001/NOTE-wsdl-20010315.

[16] *Uddi version 3.0.2.* Tech. rep., OASIS (2004). Http://www.uddi.org/pubs/uddi-v3.0.2-20041019.htm.

[17] *Soap version 1.2.* Tech. rep., W3C (2007). Https://www.w3.org/TR/soap12.

[18] *Web services business process execution language version 2.0.* Tech. rep., OASIS (2007). Http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[19] W. M. Van Der Aalst. *Business process management: a comprehensive survey.* ISRN Software Engineering **2013** (2013).

[20] M. M. Lehman and L. A. Belady. *Program evolution: processes of software change* (Academic Press Professional, Inc., 1985).

[21] K. H. Bennett and V. T. Rajlich. *Software maintenance and evolution: a roadmap.* In *Proceedings of the Conference on the Future of Software Engineering*, pp. 73–87 (ACM, 2000).

[22] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel, *et al. Software engineering for self-adaptive systems: A second research roadmap.* In *Software Engineering for Self-Adaptive Systems II*, pp. 1–32 (Springer, 2013).

[23] M. Poppendieck and T. Poppendieck. *Implementing lean software development: From concept to cash* (Pearson Education, 2007).

[24] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. *Workflow evolution.* Data & Knowledge Engineering **24**(3), 211 (1998).

[25] Y. Wang and Y. Wang. *A survey of change management in service-based environments.* Service Oriented Computing and Applications **7**(4), 259 (2013).

[26] *Web services architecture.* Tech. rep., W3C (2004). Https://www.w3.org/TR/ws-arch.

[27] S. Thatte. *Message exchange protocols for web services.* In *W3C Workshop on Web Services* (Microsoft Corporation, 2001).

[28] Y. Liu, A. H. Ngu, and L. Z. Zeng. *Qos computation and policing in dynamic web service selection.* In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pp. 66–73 (ACM, 2004).

[29] M. Papazoglou. *The challenges of service evolution.* In *Advanced Information Systems Engineering*, pp. 1–15 (Springer, 2008).

[30] C. Zeginis and D. Plexousakis. *Web service adaptation: State of the art and research challenges.* Self **2**, 5 (2010).

[31] J. Rao and X. Su. *A survey of automated web service composition methods.* In *International Workshop on Semantic Web Services and Web Process Composition*, pp. 43–54 (Springer, 2004).

[32] F. A. Cummins. *Bpm meets soa: a new era in business design.* In *Handbook on Business Process Management 1*, pp. 531–555 (Springer, 2015).

[33] G. M. Giaglis. *A taxonomy of business process modeling and information systems modeling techniques*. International Journal of Flexible Manufacturing Systems **13**(2), 209 (2001).

[34] W. M. van Der Aalst, A. H. Ter Hofstede, B. Kiepuszewski, and A. P. Barros. *Workflow patterns*. Distributed and Parallel Databases **14**(1), 5 (2003).

[35] F. Leymann, D. Roller, and M.-T. Schmidt. *Web services and business process management*. IBM systems Journal **41**(2), 198 (2002).

[36] Y. Wang, J. Yang, and W. Zhao. *Change impact analysis for service based business processes*. In *2010 IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, pp. 1–8 (IEEE, 2010).

[37] Y. Wang, J. Yang, W. Zhao, and J. Su. *Change impact analysis in service-based business processes*. Service Oriented Computing and Applications **6**(2), 131 (2012).

[38] P. Xiu, J. Yang, and W. Zhao. *A change management framework for service based business process*. In *Proceedings of the Australasian Computer Science Week Multiconference*, p. 36 (2017).

[39] B. Weber, M. Reichert, and S. Rinderle-Ma. *Change patterns and change support features–enhancing flexibility in process-aware information systems*. Data & knowledge engineering **66**(3), 438 (2008).

[40] E. Söderström, B. Andersson, P. Johannesson, E. Perjons, and B. Wangler. *Towards*

a framework for comparing process modelling languages. In *International Conference on Advanced Information Systems Engineering*, pp. 600–611 (Springer, 2002).

[41] M. Zur Muehlen and J. Recker. *How much language is enough? theoretical and practical use of the business process modeling notation.* In *International Conference on Advanced Information Systems Engineering*, pp. 465–479 (Springer, 2008).

[42] B. P. Model. *Notation (bpmn) version 2.0.* OMG Specification, Object Management Group (2011).

[43] J. Mendling and M. Weidlich. *Business Process Model and Notation* (Springer, 2012).

[44] W. M. Van der Aalst. *Formalization and verification of event-driven process chains.* Information and Software technology **41**(10), 639 (1999).

[45] A.-W. Scheer, O. Thomas, and O. Adam. *Process modeling using event-driven process chains.* Process-Aware Information Systems pp. 119–146 (2005).

[46] O. Thomas and M. Fellmann. *Semantic business process management: Ontology-based process modeling using event-driven process chains.* IBIS **4**, 29 (2007).

[47] M. Dumas and A. H. Ter Hofstede. *Uml activity diagrams as a workflow specification language.* In *International Conference on the Unified Modeling Language*, pp. 76–90 (Springer, 2001).

[48] R. Eshuis and R. Wieringa. *A formal semantics for uml activity diagrams-formalising workflow models* (2001).

[49] H. Eshuis. *Semantics and verification of UML activity diagrams for workflow modelling.* Ph.D. thesis (2002).

[50] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, *et al. Business process execution language for web services* (2003).

[51] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. F. Ferguson. *Web services platform architecture: SOAP, WSDL, WS-policy, WS-addressing, WS-BPEL, WS-reliable messaging and more* (Prentice Hall PTR, 2005).

[52] C. Ouyang, E. Verbeek, W. M. Van Der Aalst, S. Breutel, M. Dumas, and A. H. Ter Hofstede. *Formal semantics and analysis of control flow in ws-bpel.* Science of computer programming **67**(2-3), 162 (2007).

[53] C. A. Petri. *Kommunikation mit automaten* (1962).

[54] T. Murata. *Petri nets: Properties, analysis and applications.* Proceedings of the IEEE **77**(4), 541 (1989).

[55] K. Jensen. *Coloured Petri nets: basic concepts, analysis methods and practical use*, vol. 1 (Springer Science & Business Media, 2013).

[56] J. E. Coolahan and N. Roussopoulos. *Timing requirements for time-driven systems using augmented petri nets.* IEEE transactions on software engineering (5), 603 (1983).

[57] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with generalized stochastic Petri nets* (John Wiley & Sons, Inc., 1994).

[58] B. Walter. *Timed petri nets for modelling and analyzing protocols with real-time characteristics.* Protocol Specification, Testing and Verification III, pp. 149–159 (1983).

[59] W. van der Aalst. *Interval timed coloured petri nets and their analysis.* Application and Theory of Petri Nets 1993 pp. 453–472 (1993).

[60] W. M. Van Der Aalst. *Three good reasons for using a petri-net-based workflow management system.* In *Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC96)*, pp. 179–201 (Citeseer, 1996).

[61] W. van der Aalst. *Verification of workflow nets.* Application and Theory of Petri Nets 1997 pp. 407–426 (1997).

[62] C.-C. Dolean and R. Petrusel. *Data-flow modeling: A survey of issues and approaches.* Informatica Economica **16**(4), 117 (2012).

[63] M. Varea, B. M. Al-Hashimi, L. A. Cortés, P. Eles, and Z. Peng. *Dual flow nets: Modeling the control/data-flow relation in embedded systems.* ACM Transactions on Embedded Computing Systems (TECS) **5**(1), 54 (2006).

[64] S. Fan, W. Dou, and J. Chen. *Dual workflow nets: Mixed control/data-flow representation for workflow modeling and verification.* In *Advances in Web and Network Technologies, and Information Management*, pp. 433–444 (Springer, 2007).

[65] L. Cong, Q. ZENG, D. Hua, *et al. Formulating the data-flow modeling and verification for workflow: A petri net based approach.* International Journal of Science and Engineering Applications **3**, 107 (2014).

[66] J. Zhang, C. K. Chang, J.-Y. Chung, and S. W. Kim. *Ws-net: A petri-net based specification model for web services.* In *Web Services, 2004. Proceedings. IEEE International Conference on*, pp. 420–427 (IEEE, 2004).

[67] B. Srivastava and J. Koehler. *Web service composition-current solutions and open problems.* In *ICAPS 2003 workshop on Planning for Web Services*, vol. 35, pp. 28–35 (2003).

[68] N. Milanovic and M. Malek. *Current solutions for web service composition.* IEEE Internet Computing **8**(6), 51 (2004).

[69] R. Hamadi and B. Benatallah. *A petri net-based model for web service composition.* In *Proceedings of the 14th Australasian database conference-Volume 17*, pp. 191–200 (Australian Computer Society, Inc., 2003).

[70] M. C. Jaeger, G. Rojec-Goldmann, and G. Muhl. *Qos aggregation for web service composition using workflow patterns.* In *Enterprise Distributed Object Computing Conference, 2004. EDOC 2004. Proceedings. Eighth IEEE International*, pp. 149–159 (IEEE, 2004).

[71] X. Yi and K. J. Kochut. *A cp-nets-based design and verification framework for web services composition.* In *Web Services, 2004. Proceedings. IEEE International Conference on*, pp. 756–760 (IEEE, 2004).

[72] Y. Yang, Q. Tan, and Y. Xiao. *Verifying web services composition based on hierarchical colored petri nets.* In *Proceedings of the first international workshop on Interoperability of heterogeneous information systems*, pp. 47–54 (ACM, 2005).

[73] W.-L. Dong, H. Yu, and Y.-B. Zhang. *Testing bpel-based web service composition using high-level petri nets.* In *Enterprise Distributed Object Computing Conference, 2006. EDOC'06. 10th IEEE International*, pp. 441–444 (IEEE, 2006).

[74] D. Zhovtobryukh. *A petri net-based approach for automated goal-driven web service composition.* Simulation **83**(1), 33 (2007).

[75] W. Tan, Y. Fan, and M. Zhou. *A petri net-based method for compatibility analysis and composition of web services in business process execution language.* IEEE Transactions on Automation Science and Engineering **6**(1), 94 (2009).

[76] W. Tan, Y. Fan, M. Zhou, and Z. Tian. *Data-driven service composition in enterprise soa solutions: A petri net approach.* IEEE Transactions on Automation Science and Engineering **7**(3), 686 (2010).

[77] W. M. Van Der Aalst, K. M. van Hee, A. H. ter Hofstede, N. Sidorova, H. Verbeek, M. Voorhoeve, and M. T. Wynn. *Soundness of workflow nets: classification, decidability, and analysis.* Formal Aspects of Computing **23**(3), 333 (2011).

[78] K. Van Hee, N. Sidorova, and M. Voorhoeve. *Generalised soundness of workflow nets is decidable.* In *International Conference on Application and Theory of Petri Nets*, pp. 197–215 (Springer, 2004).

[79] A. Martens. *On compatibility of web services.* Petri Net Newsletter **65**(12-20), 100 (2003).

[80] J. Dehnert and P. Rittgen. *Relaxed soundness of business processes.* In *International*

Conference on Advanced Information Systems Engineering, pp. 157–170 (Springer, 2001).

[81] F. Puhlmann and M. Weske. *Investigations on soundness regarding lazy activities.* In *International Conference on Business Process Management*, pp. 145–160 (Springer, 2006).

[82] S. Sadiq, M. Orlowska, W. Sadiq, and C. Foulger. *Data flow and validation in workflow modelling.* In *Proceedings of the 15th Australasian database conference-Volume 27*, pp. 207–214 (Australian Computer Society, Inc., 2004).

[83] S. X. Sun, J. L. Zhao, J. F. Nunamaker, and O. R. L. Sheng. *Formulating the data-flow perspective for business process management.* Information Systems Research **17**(4), 374 (2006).

[84] N. Trčka, W. M. Van der Aalst, and N. Sidorova. *Data-flow anti-patterns: Discovering data-flow errors in workflows.* In *International Conference on Advanced Information Systems Engineering*, pp. 425–439 (Springer, 2009).

[85] M. Reichert and P. Dadam. *Adept flexsupporting dynamic changes of workflows without losing control.* Journal of Intelligent Information Systems **10**(2), 93 (1998).

[86] M. Reichert, S. Rinderle, and P. Dadam. *On the common support of workflow type and instance changes under correctness constraints.* In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pp. 407–425 (Springer, 2003).

[87] W. Song, X. Ma, S. C. Cheung, H. Hu, and J. Lü. *Preserving data flow correctness in process adaptation.* In *Services Computing (SCC), 2010 IEEE International Conference on*, pp. 9–16 (IEEE, 2010).

[88] K. Becker, A. Lopes, D. S. Milojicic, J. Pruyne, and S. Singhal. *Automatically determining compatibility of evolving services.* In *Web Services, 2008. ICWS'08. IEEE International Conference on*, pp. 161–168 (2008).

[89] A. Wombacher, P. Fankhauser, B. Mahleko, and E. Neuhold. *Matchmaking for business processes based on choreographies.* In *e-Technology, e-Commerce and e-Service, 2004. EEE'04. 2004 IEEE International Conference on*, pp. 359–368 (IEEE, 2004).

[90] A. Martens. *Consistency between executable and abstract processes.* In *e-Technology, e-Commerce and e-Service, 2005. EEE'05. Proceedings. The 2005 IEEE International Conference on*, pp. 60–67 (IEEE, 2005).

[91] W. Fdhila, C. Indiono, S. Rinderle-Ma, and M. Reichert. *Dealing with change in process choreographies: Design and implementation of propagation algorithms.* Information systems **49**, 1 (2015).

[92] K. Jensen, L. M. Kristensen, and L. Wells. *Coloured petri nets and cpn tools for modelling and validation of concurrent systems.* International Journal on Software Tools for Technology Transfer **9**(3-4), 213 (2007).

[93] V. Rajlich. *A model for change propagation based on graph rewriting.* In *Software Maintenance, 1997. Proceedings., International Conference on*, pp. 84–91 (IEEE, 1997).

[94] Y. Kataoka, D. Notkin, M. D. Ernst, and W. G. Griswold. *Automated support for program refactoring using invariants.* In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM'01)*, p. 736 (IEEE Computer Society, 2001).

[95] M. Lanza and S. Ducasse. *Understanding software evolution using a combination of software visualization and software metrics.* In *In Proceedings of LMO 2002 (Langages et Modèles à Objets* (Citeseer, 2002).

[96] T. Mens and T. Tourwé. *A survey of software refactoring.* IEEE Transactions on software engineering **30**(2), 126 (2004).

[97] A. H. Skarra and S. B. Zdonik. *The management of changing types in an object-oriented database.* In *ACM Sigplan Notices*, vol. 21, pp. 483–495 (ACM, 1986).

[98] J. Banerjee, W. Kim, H.-J. Kim, and H. F. Korth. *Semantics and implementation of schema evolution in object-oriented databases*, vol. 16 (ACM, 1987).

[99] C. Yu and L. Popa. *Semantic adaptation of schema mappings when schemas evolve.* In *Proceedings of the 31st international conference on Very large data bases*, pp. 1006–1017 (VLDB Endowment, 2005).

[100] M. Grossniklaus, S. Leone, A. De Spindler, and M. C. Norrie. *Dynamic metamodel extension modules to support adaptive data management.* In *International Conference on Advanced Information Systems Engineering*, pp. 363–377 (Springer, 2010).

[101] J. Kramer and J. Magee. *The evolving philosophers problem: Dynamic change management.* IEEE Transactions on software engineering **16**(11), 1293 (1990).

[102] H. Evans and P. Dickman. *Drastic: A run-time architecture for evolving, distributed, persistent systems.* In *European Conference on Object-Oriented Programming*, pp. 243–275 (Springer, 1997).

[103] G. Taentzer, M. Goedicke, and T. Meyer. *Dynamic change management by distributed graph transformation: Towards configurable distributed systems.* In *International Workshop on Theory and Application of Graph Transformations*, pp. 179–193 (Springer, 1998).

[104] C. Dorn and S. Dustdar. *Interaction-driven self-adaptation of service ensembles.* In *International Conference on Advanced Information Systems Engineering*, pp. 393–408 (Springer, 2010).

[105] E. Serral, P. Valderas, and V. Pelechano. *Supporting runtime system evolution to adapt to user behaviour.* In *International Conference on Advanced Information Systems Engineering*, pp. 378–392 (Springer, 2010).

[106] E. E. Schmidt and B. W. Lampson. *Software version management system* (1985). US Patent 4,558,413.

[107] W. A. Babich. *Software configuration management: coordination for team productivity* (Addison-Wesley Longman Publishing Co., Inc., 1986).

[108] R. Conradi and B. Westfechtel. *Version models for software configuration management.* ACM Computing Surveys (CSUR) **30**(2), 232 (1998).

[109] J. Estublier. *Software configuration management: a roadmap.* In *Proceedings of the Conference on the Future of Software Engineering*, pp. 279–289 (ACM, 2000).

[110] J. M. Küster, C. Gerth, A. Förster, and G. Engels. *A tool for process merging in business-driven development.* In *CAiSE Forum*, vol. 344 (Citeseer, 2008).

[111] A. Iyengar, V. Jessani, and M. Chilanti. *WebSphere business integration primer: Process server, BPEL, SCA, and SOA* (IBM Press, 2007).

[112] J. M. Küster, C. Gerth, A. Förster, and G. Engels. *Detecting and resolving process model differences in the absence of a change log.* In *International Conference on Business Process Management*, pp. 244–260 (Springer, 2008).

[113] C. Gerth, J. M. Küster, and G. Engels. *Language-independent change management of process models.* In *International Conference on Model Driven Engineering Languages and Systems*, pp. 152–166 (Springer, 2009).

[114] C. Gerth, M. Luckey, J. M. Küster, and G. Engels. *Detection of semantically equivalent fragments for business process model change management.* In *Services Computing (SCC), 2010 IEEE International Conference on*, pp. 57–64 (IEEE, 2010).

[115] C. Gerth, J. Küster, M. Luckey, and G. Engels. *Precise detection of conflicting change operations using process model terms.* Model Driven Engineering Languages and Systems pp. 93–107 (2010).

[116] C. Gerth, J. M. Küster, M. Luckey, and G. Engels. *Detection and resolution of conflicting change operations in version management of process models.* Software & Systems Modeling **12**(3), 517 (2013).

[117] S. W. Sadiq. *Handling dynamic schema change in process models.* In *Database Conference, 2000. ADC 2000. Proceedings. 11th Australasian*, pp. 120–126 (IEEE, 2000).

[118] G. Joeris and O. Herzog. *Managing evolving workflow specifications.* In *Cooperative Information Systems, 1998. Proceedings. 3rd IFCIS International Conference on*, pp. 310–319 (IEEE, 1998).

[119] G. Joeris and O. Herzog. *Managing evolving workflow specifications with schema versioning and migration rules.* Tech. rep., TZI Technical Report 15, University of Bremen (1999).

[120] M. Kradolfer and A. Geppert. *Dynamic workflow schema evolution based on workflow type versioning and workflow migration.* In *Cooperative Information Systems, 1999. CoopIS'99. Proceedings. 1999 IFCIS International Conference on*, pp. 104–114 (IEEE, 1999).

[121] D. Frank, L. Fong, and L. Lam. *A continuous long running batch orchestration model for workflow instance migration.* In *Services Computing (SCC), 2010 IEEE International Conference on*, pp. 226–233 (IEEE, 2010).

[122] S. R. Ponnekanti and A. Fox. *Interoperability among independently evolving web services.* In *Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, pp. 331–351 (Springer-Verlag New York, Inc., 2004).

[123] B. Benatallah, F. Casati, D. Grigori, H. R. M. Nezhad, and F. Toumani. *Developing adapters for web services integration.* In *CAiSE*, vol. 3520, pp. 415–429 (Springer, 2005).

[124] W. Kongdenfha, R. Saint-Paul, B. Benatallah, and F. Casati. *An aspect-oriented framework for service adaptation.* In *ICSOC*, vol. 4294, pp. 15–26 (Springer, 2006).

[125] H. R. Motahari 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*, pp. 993–1002 (ACM, 2007).

[126] 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*, pp. 731–740 (ACM, 2010).

[127] A. Martens, S. Moser, A. Gerhardt, and K. Funk. *Analyzing compatibility of bpel processes.* In *Telecommunications, 2006. AICT-ICIW'06. International Conference on Internet and Web Applications and Services/Advanced International Conference on*, pp. 147–147 (IEEE, 2006).

[128] N. Lohmann, P. Massuthe, C. Stahl, and D. Weinberg. *Analyzing interacting bpel processes.* Business Process Management pp. 17–32 (2006).

[129] N. Lohmann, P. Massuthe, C. Stahl, and D. Weinberg. *Analyzing interacting ws-bpel processes using flexible model generation.* Data & Knowledge Engineering **64**(1), 38 (2008).

[130] 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*, pp. 107–121 (IBM Press, 2003).

[131] P. Kaminski, H. Müller, and M. Litoiu. *A design for adaptive web service evolution.* In *Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems*, pp. 86–92 (ACM, 2006).

[132] R. Fang, L. Lam, L. Fong, D. Frank, C. Vignola, Y. Chen, and N. Du. *A version-aware approach for web service directory.* In *Web Services, 2007. ICWS 2007. IEEE International Conference on*, pp. 406–413 (IEEE, 2007).

[133] M. Fokaefs, R. Mikhaiel, N. Tsantalis, E. Stroulia, and A. Lau. *An empirical study on web service evolution.* In *Web Services (ICWS), 2011 IEEE International Conference on*, pp. 49–56 (IEEE, 2011).

[134] V. Andrikopoulos, S. Benbernou, and M. P. Papazoglou. *Managing the evolution of service specifications.* In *International Conference on Advanced Information Systems Engineering*, pp. 359–374 (Springer, 2008).

[135] V. Andrikopoulos, S. Benbernou, and M. P. Papazoglou. *Evolving services from a contractual perspective.* In *CAiSE*, vol. 9, pp. 290–304 (Springer, 2009).

[136] S. Rinderle, A. Wombacher, and M. Reichert. *Evolution of process choreographies in dychor.* On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE pp. 273–290 (2006).

[137] A. Wombacher. *Alignment of choreography changes in bpel processes.* In *Services Computing, 2009. SCC'09. IEEE International Conference on*, pp. 1–8 (IEEE, 2009).

[138] 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 fourth Asia-Pacific conference on Comceptual modelling-Volume 67*, pp. 49–59 (Australian Computer Society, Inc., 2007).

[139] 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*, pp. 468–477 (IEEE Computer Society, 2007).

[140] X. Liu, C. Liu, M. Rege, and A. Bouguettaya. *Semantic support for adaptive long term composed services.* In *Web Services (ICWS), 2010 IEEE International Conference on*, pp. 267–274 (IEEE, 2010).

[141] X. Liu, A. Bouguettaya, J. Wu, and L. Zhou. *Ev-lcs: A system for the evolution of long-term composed services.* IEEE Transactions on Services Computing **6**(1), 102 (2013).

[142] C. Alexander. *A pattern language: towns, buildings, construction* (Oxford University Press, 1977).

[143] E. Gamma. *Design patterns: elements of reusable object-oriented software* (Pearson Education India, 1995).

[144] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects*, vol. 2 (John Wiley & Sons, 2013).

[145] N. Russell, A. H. Ter Hofstede, W. M. Van Der Aalst, and N. Mulyar. *Workflow control-flow patterns: A revised view.* BPM Center Report BPM-06-22, BPMcenter. org pp. 06–22 (2006).

[146] N. Russell, A. H. Ter Hofstede, D. Edmond, and W. M. van der Aalst. *Workflow data patterns* (2004).

[147] N. Russell, A. H. Ter Hofstede, D. Edmond, and W. M. van der Aalst. *Workflow resource patterns.* Tech. rep., BETA Working Paper Series, WP 127, Eindhoven University of Technology, Eindhoven (2004).

[148] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web services: concepts, architectures and applications* (Springer Science & Business Media, 2013).