User Interface Derivation based on a Role-enriched Business Process Model

By

Lei Han

A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy

Department of Computing
Faculty of Science and Engineering
Macquarie University
Supervisor: Prof. Jian Yang

February 2017

Declaration

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.

Lei Han

Acknowledgements

First and foremost, I would like to express my appreciation and thanks to my supervisor, Professor Jian Yang, for supervising my research work over my four-year Ph.D. journey. Her passion towards work and critical thinking always inspire me and help me keep progressing on my research. Other than that, her philosophies on life also influence me and make me get better every day. I also want to give my sincere gratitude to Dr. Weiliang Zhao for his precious support and guidance. His solid background knowledge and active thinking enable me to make achievements in my research efficiently.

I am also truly grateful to my dear colleagues Robertus Nugroho, Yan Mei, Pengbo Xiu and Zizhu Zhang from the Department of Computing at Macquarie University. Discussion with them offers me great inspiration on my research. Working with them over the last four years is truly a wonderful and memorable experience in my life. Here I would also like to give my special thanks to Pengbo Xiu and Zizhu Zhang for reviewing my thesis and correcting typo errors.

Many thanks are given to the administration team of Department of Computing at Macquarie University. Kind support and wonderful environment they provide are important for me to complete my thesis.

I also owe huge thanks to my friends who have accompanied me during my research journey. Jian Zhao in China, Yaguang Sun in Germany, Xiao Ma in China, Wenhai Pan in China, Xi Zhang in Australia, Xinxin Shang and Kai Zhang in Australia. It is a great joy to have them in my life.

I am much obliged to my beloved parents, Jinping Han and Junying Wang. Your love and care from China are great encouragement and support to not only my research, but also my life. I feel so happy to be with them.

Last but not least, I would like to deliver my deep appreciation to my beloved wife Yixiao Liu. I cannot complete this thesis without her patience, understanding, encouragement, sacrifice and support. Her perpetual love is always a lighthouse in my life.

Abstract

In recent decades, the boom of information and communication technology has brought countless business process changes in a wide range of organizations and enterprises. A business process (BP) describes a collection of linked tasks to produce a particular product or service. Each task is a logic unit of work performed by human users or applications. Human users participate in a business process through user interfaces (UIs). In a business process, the UI accepts input and provides output for the process users.

The implementation of a business process often involves a lot of hard coding work. In particular, the development of the UI of process often constitutes 70% to 80% of the manually written code for the BP implementation. The hard coding for UI development can cause many problems. The realization and maintenance of the UI of a process are often not only costly and effort-consuming but also error-prone due to the nature of hard coding. Moreover, the hard coding leads to a tight coupling between the BPs and their UIs. Any changes of existing UIs/BPs cannot be easily adapted without recoding.

To overcome the problems mentioned above, it is highly desirable to develop a UI derivation method based on a business process model. The business process model should support the UI logic with the following features: (1) each participating user role includes a UI logic; (2) each UI logic consists of a set of containers and the execution constraints of these containers; (3) each container includes a set of data items specified with access types (*read*, *write*).

In this thesis, we propose a UI derivation method based on a role-enriched business process model to derive complex UI logics. The proposed UI derivation method has the following features:

- A role-enriched business process model is proposed with the capabilities to specify (1) the control flow relations between tasks; (2) the relationships between the participating user roles and individual tasks; (3) the data operation flow inside each task. In the process model, we identify a set of control flow patterns and data operation patterns to build up the rules for UI derivation.
- The business process is abstracted and aggregated for each user role based on the role-enrich BP model. A set of elementary operations are developed according to the control flow patterns to reserve or abstract tasks for each user role. With the abstracted and aggregated business process (AABP), a customized UI logic is derived for each participating user role.
- Data relationships are extracted from the AABP for each user role. A set of elementary operations are developed according to the data operations inside individual tasks and the identified control flow patterns in the AABP. The extracted data relationships are the foundation to analyze and derive the UI logic.
- A set of mandatory and recommended rules are specified. The UI logics are derived from the extracted data relationships based on these specified rules.
- A UI Derivation Tool (UIDrvTool) is developed as the implementation of our proposed UI derivation approach.

In summary, this research sheds new light on the state-of-art of the UI development for business processes.

List of Publications

[1] Lei Han, Weiliang Zhao, and Jian Yang. An approach towards user interface derivation from business process model. In *International Workshop on Process-Aware Systems*, pages 19–28. Springer, 2015.

[2] Lei Han, Weiliang Zhao, and Jian Yang. User interface derivation based on role-enriched business process model. In *International Conference on Service-Oriented Computing (ICSOC)*, pages 450–464. Springer, 2016.

[3] Lei Han, Weiliang Zhao, and Jian Yang. An approach towards task abstraction and aggregation in business processes. In *Web Services (ICWS), 2017 IEEE International Conference on*. IEEE, 2017.

[4] Lei Han, Weiliang Zhao, and Jian Yang. An approach towards user interface derivation based on role-enriched business process model. *IEEE Transactions on Knowledge and Data Engineering Journal*, 2017 (in preparation).

Contents

List of Figures

LIST OF FIGURES

List of Tables

LIST OF TABLES

Chapter 1

Introduction

This chapter describes the background, the motivation and the contributions of the research work. It also provides an outline of the thesis. Section 1.1 provides an overview of the research background, in which business process and user interface are introduced. Section 1.2 discusses the motivation of our work and related key issues. Section 1.3 summarizes the contributions of the research work. Section 1.4 presents an outline of the thesis.

1.1 Background

1.1.1 Business Process

Over the last decades, the information and communication technology has gained remarkable development, focusing organizations to adjust their business strategies when accelerating their growth and delivering their business values to customers. Business strategies are specified and operationalised through business processes (BPs). A business process is a collection of ordered tasks followed by an organization to produce a particular product or service. With the advancement of information systems, the organizations around the world have set the business process automation and improvement as their major goals in order to reduce costs, increase productivity, and improve product quality. Product managers are casting about for employing cutting-edge technology, especially information technology (IT), in their business processes. IT staff

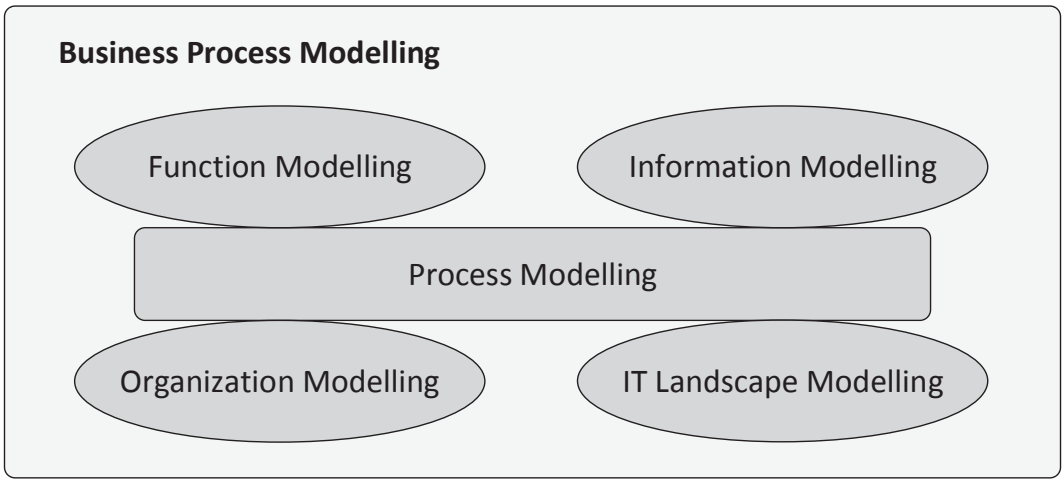


Figure 1.1: Domains of Business Process Modelling

have been keen to exploit data collected from their business processes supported by IT, and employ these critical data in a way that helps to meet the organizational business goals [1, 2, 3, 4, 5].

In a business process, different tasks coordinate with each other to realize a particular business goal. Theses tasks can be categorized as *system tasks* that are automatically performed by applications (e.g. calculating the account payable for a customer), user-interaction tasks that require both the support of information systems and human knowledge (e.g. inputting customer information into a customer management system), and manual tasks that is totally not supported by information system (e.g. sending a parcel to a business partner) [6, 7]. In order to represent a BP, a business process model can offer an essential tool to describe a BP in an intelligible manner (e.g. graphical notation). With a business process model, process participants are able to better understand the process, share the understanding with other related people, identify potential issues and improve the process. BP models can support system developers and engineers in implementing processes in software environment.

Generally speaking, business process modelling comprises five modelling domains (see Figure 1.1): *process modelling*, *function modelling*, *information modelling*, *organization modelling* and *IT landscape modelling*. The *process modelling* is the key one

in these five domains. It not only specifies the details of a BP, but also integrates the modelling results of the other four domains. The *functional modelling* explores the units of work that are enacted in the context of business processes. The *information modelling* specifies the data operated in BPs. The data are critical in a process, as decision-making in BPs depends on the particular data values. Besides, data dependencies between tasks must be considered in process design to avoid situations where a function requires certain data that is not available at that time. The *organization modelling* focuses on the representation of the organizational structure of an enterprise. The *IT landscape modelling* refers to building the landscape of operational information technology for BP implementation, including the information systems, their relationships, and their programming interfaces [8].

As a major business process modelling paradigm, the activity-centric process modelling regards process tasks and their control flow relations as first class citizens. The control flow relations are explicitly specified in an activity-centric BP model. A number of control flow patterns have been identified and summarized as languages to build up a process model. Each pattern is a category that describes a recurring scenario of a BP control flow. The data operated by a BP are modeled as data objects along with sets of object states. A particular data object at a specific state is represented as a pre-/post-condition for enabling a task, or as a main decision indicator at a specific control flow divergence point. The usage of a particular data object in different object states in combination with multiple tasks allows to derive a life-cycle of the object, which represents the manipulations performed on the data object [9, 10]. The representative languages and standards of the paradigm include Business Process Model and Notation [11], Event-driven Process Chains [12], Business Process Execution Language [13], and Yet Another Workflow Language [14].

1.1.2 User Interface

With the extensive application of information technology, there is an increasing concern on the development of the user interface (UI) in the computer-based information

systems. In an information system, the UI is the component through which a user interacts with the system. Here the term "user" may refer to either a human being or an application. In the application-system interaction, a UI accepts input for performing particular functionalities and provides results. Examples of this type of interactions are making calls to web services, and storing/retriving data into/from databases. In the human-machine interaction, a UI determines how commands are given to the system and how information is displayed on the screen. The goal of this human-machine interactions is to realize effective operations and controls on the system from human beings. On the other hand, the system gives the feedback information to human beings. There exist three main alternatives of human-machine UIs as (1) command language: the user must know the machine and program-specific instructions or codes; (2) menus: the user chooses the commands from lists displayed on the screen. (3) graphical user interface (GUI): user gives commands by selecting and clicking on icons displayed on the screen [15, 16].

The user interface design has always been a hot spot in the development of information systems. The earliest UI design standards can be dated back to the 1980s when the usability of software products is defined. One major guiding principle in the UI design is the separation of concerns proposed by Edsger Dijkstra [17], which is one of the key principles in dealing with the complexity of computer systems. With this principle, functionalities are identified and packaged in a subsystem with explicitly specified responsibilities and interfaces. The principle brings in benefits as follows:

- it enables the enormous and complex systems to be engineered;
- it allows the implemented functionalities to be reused by different applications;
- it offers a response mechanism to changes in systems and individual subsystems can be modified without changing other parts of the overall system.

With the guidance of the separation of concerns, UIs will be developed in a separate component package due to that UI developers do not want the user interfaces to be tightly coupled with other parts of the information system. This philosophy greatly

facilitates the development of graphical user interface (GUI), which is a visual way to realize human interaction with a system using items such as windows, icons, and menus etc. Comparing to the conventional command-line user interface, the GUI greatly eases the interaction between humans and systems due to that users do not need to learn the complicated command languages when using GUI. Normally, four basic dimensions are considered when structuring the graphical user interface: (1) the input/output dimension (the look), (2) the dialogue dimension (the feel), (3) the technical or functional dimension (the access to tools and services), and (4) the organizational dimension (the communication and co-operation support) [18].

1.2 Motivation and Key Issues

A business process is a collection of tasks and their control flow relations to realize a particular business goal. Each task is a unit of work performed by human users or applications. The human users participate in a process through user interfaces. In a business process, the UI requires input from the process participants to drive the process execution, whilst provides feedback information to the process participants to support their decision-making.

UIs support the interactions between/among the BP, database systems/application/web services, and the process participants (see Figure 1.2). The development of the BP UIs normally requires a lot of hard coding efforts. During the UI development, the UI engineers need to firstly analyze the process to obtain the UI design logic, which includes identifying the tasks of the process that requires UIs for human interactions, identifying the data items required within each task-specific UI, and identifying the flows between the task-specific UIs. Then, the UI engineers can develop the graphical user interfaces according to the obtained UI design logic [19].

The realization and maintenance of the BP UIs are not only costly and time-consuming, but also error-prone, due to the nature of mannual coding. These drawbacks impede the quick adaptations of business process realization. Moreover, due to the tight

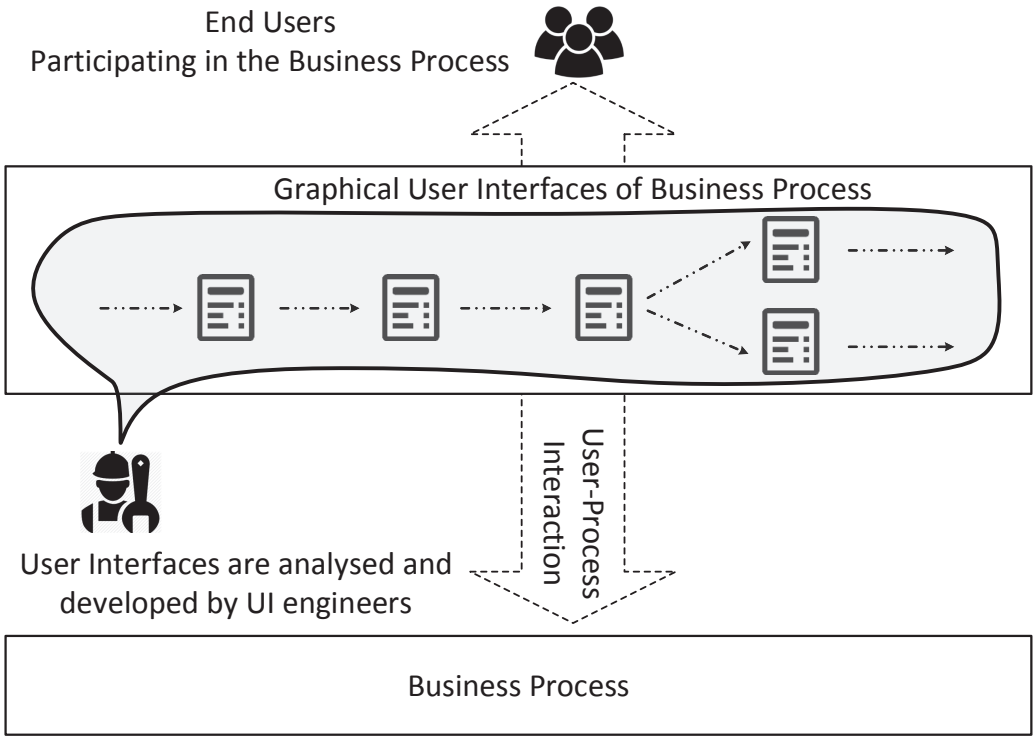


Figure 1.2: Problems in the BP UI Development

coupling between the BPs and their UIs, the changes required by the existing UIs/BPs cannot be easily adapted to the BPs/existing UIs without recoding.

To overcome the above mentioned drawbacks, a UI derivation approach based on the business process model is highly desirable (see the red parts in Figure 1.3). As the foundation of this approach, the business process model must contain all the process details required to derive the process UIs. To the best of our knowledge, the existing BP models cannot specify enough process details to support the UI derivation, and therefore must be enriched with additional elements. Based on the enriched BP model, a UI derivation approach can be developed. The input of this approach is a business process specified using the enriched BP model, and the output is the UI logic of the input business process. The UI logic supports UI developers to develop graphical user interfaces of the BP without analyzing the process.

Here we use a scenario example to demonstrate what the UI logic of a BP is.

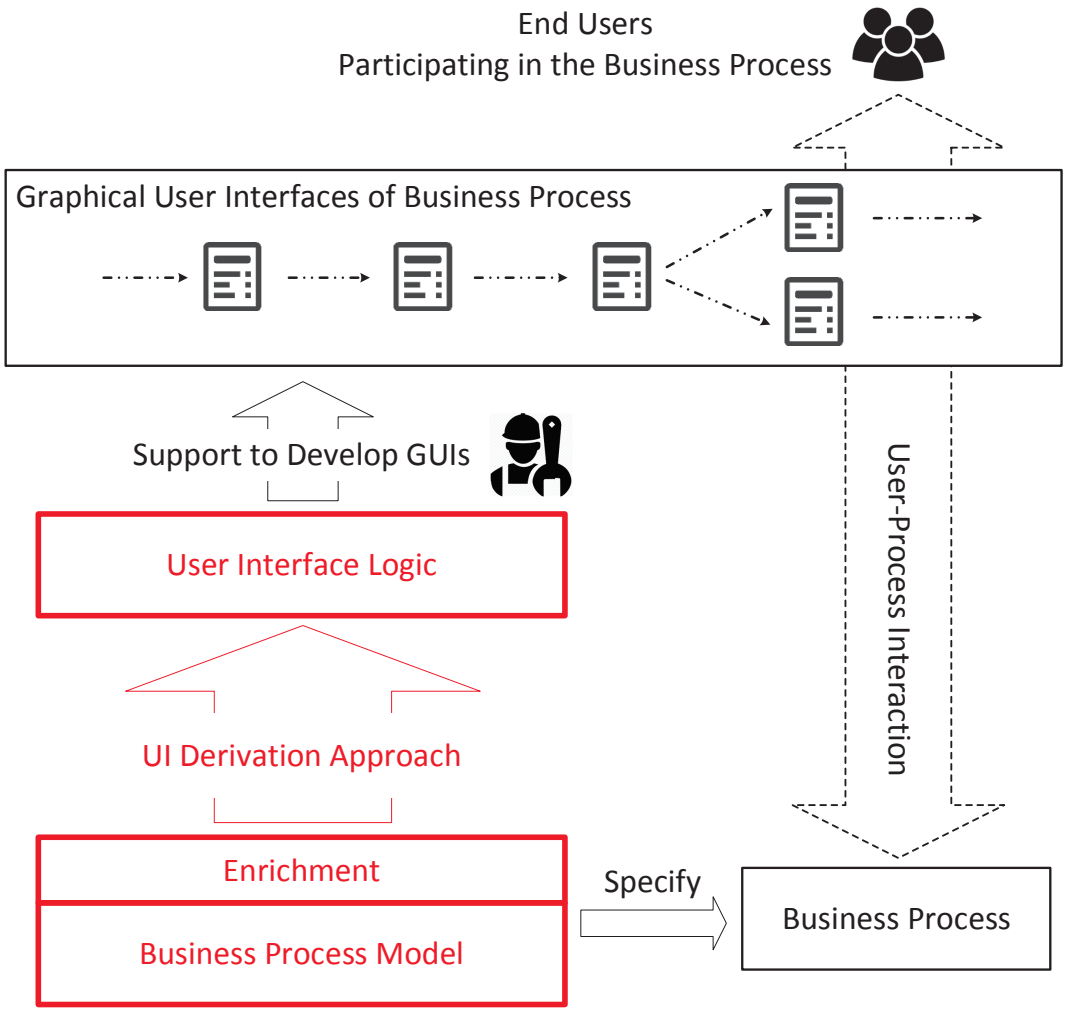


Figure 1.3: Solution to the BP UI Development

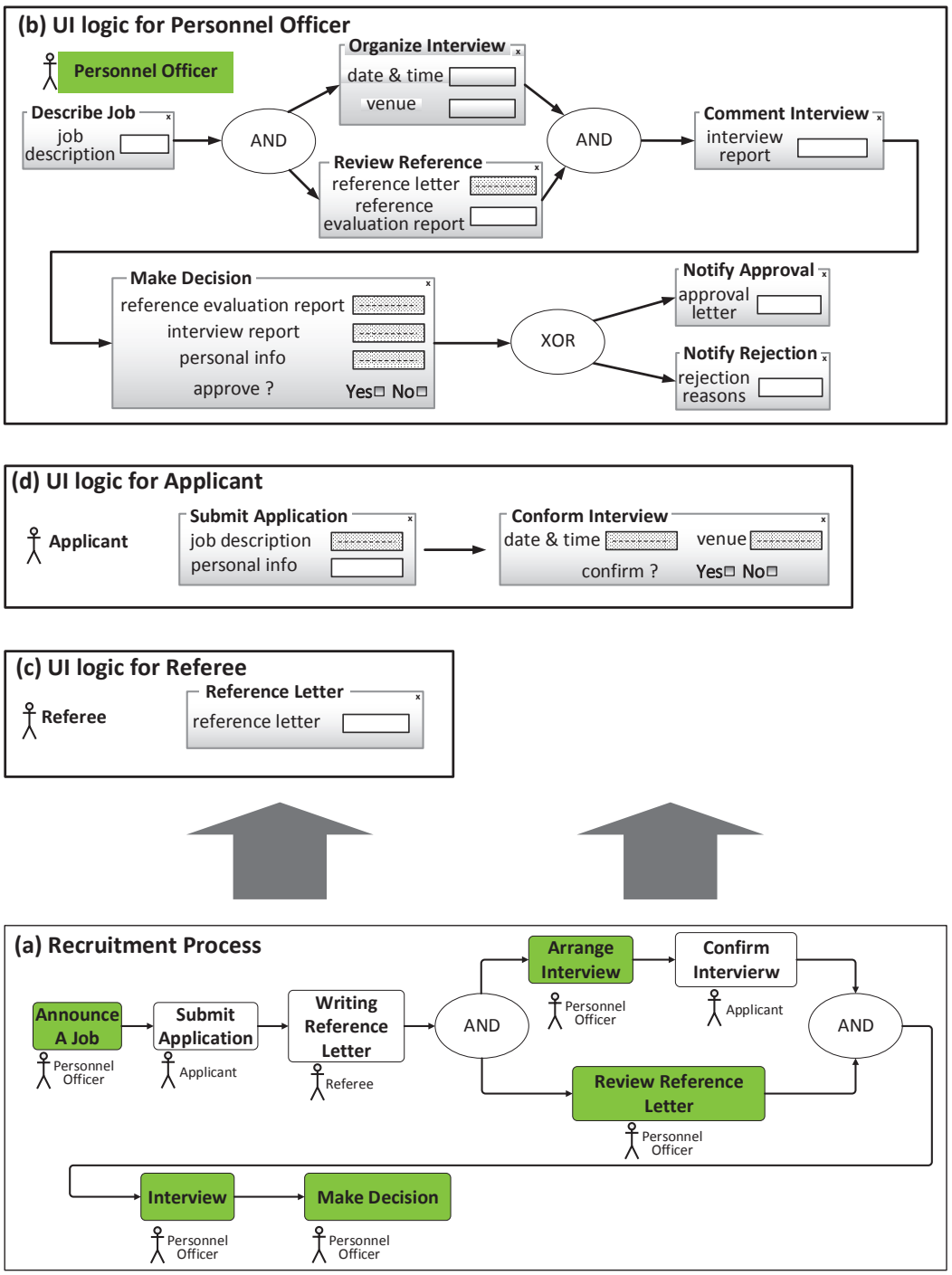


Figure 1.4: Deriving UI Logics from a Business Process

(a) Figure 1.4 shows a recruitment process at the human resource department of a company. There exist three user roles involved in the business process as `personnel officer`, `applicant`, and `referee`. In the recruitment process, there are tasks as: (1) the `personnel officer` announces a job vacancy; (2) an `applicant` lodges his application; (3) a `referee` writes a reference letter to support the application; (4) the `personnel officer` arranges an interview for the `applicant`; (5) the `applicant` confirms the interview; (6) the `personnel officer` reviews the reference letter; (7) the `personnel officer` conducts the interview; (8) the `personnel officer` makes the decision according to the evaluation of the reference letter and the interview report. Task 4, 5 are in parallel with task 6. Task 4 and task 5 are executed sequentially. Task 1, task 2, task 3, task 4, 5, 6, task 7, task 8 are executed sequentially.

Users provide inputs to and receive information from a BP through UIs. The operation flow of input/output data for a specific user role in a BP is referred to as the UI logic of this user role. Input/output data will be grouped and put in different UI containers. For instance, (d) Figure 1.4 shows the UI logic for the user role `applicant`. There are two UI containers (`Submit Application` and `Confirm Interview`) and they will turn up in a sequential order. With `Submit Application`, an `applicant` can read the job information and provide the details of his application. With `Confirm Interview`), the `applicant` can check the interview date, time, location and confirm his attendance.

We believe the UI logic of a BP should have the following features as:

1. each participating user role should have a UI logic;
2. each UI logic consists of a set of containers and the execution constraints of these containers;
3. each container includes a set of data items specified with access types (*read*, *write*).

To specify the UI logic with the above features, we define a UI flow for each participating user role. A **UI flow** consists of a set of UI containers (containers) as well as

the operation flows between them. Each container comprises a set of data items. Each data item needs to be specified with an access type. The access type indicates if a data item is to be read or edited.

In order to derive the UI flow for each involved user role, we propose a UI derivation approach based on role-enriched business process model as illustrated in Figure 1.5. As the starting point of UI derivation, a **role-enriched business process model** ((a) Figure 1.5) is built to contain all details required to derive the UI flow. This process model specifies: (1) how user roles are involved in tasks; (2) how complex control flow patterns affect data relationships; (3) how data are operated in individual tasks. Then, a series of control flow patterns and data operation patterns are identified as the basis to build up elementary operations and rules for UI derivation. A control flow pattern describes a scenario of a control flow relation between tasks in a BP; a data flow pattern describes a scenario of a control flow relation between data items within a task of a BP.

As the **first step** (Step 1 in Figure 1.5), the tasks in the role-enriched business process are abstracted and aggregated for each involved user role. In this process, the tasks related to a particular user role are kept, and the tasks not associated with this user role are hidden as abstracted nodes. As the result of this step, a unique Abstracted and Aggregated BP (AABP) is generated for each user role participating in the role-enriched BP (see (b), (c), (d) in Figure 1.5). And therefore an AABP is dedicated to derive UI logics related to an involved user role.

Other than the goal of UI derivation, the technique of task abstraction and aggregation also has the following significance: (1) firstly, the details of BP tasks must be hidden and abstracted from certain users due to information security requirements such as privacy, confidentiality, and conflict of interest; (2) secondly, task abstraction and aggregation are a foundation for deriving customized descriptions of a BP for participating users according to the users' requirements and intentions [20]. The customized BP descriptions may play an important role in the modelling of BP collaboration, BP visualization, and authority control; (3) thirdly, AABPs highlight the requirements associated with a specific user role and preserve some information of other user roles for

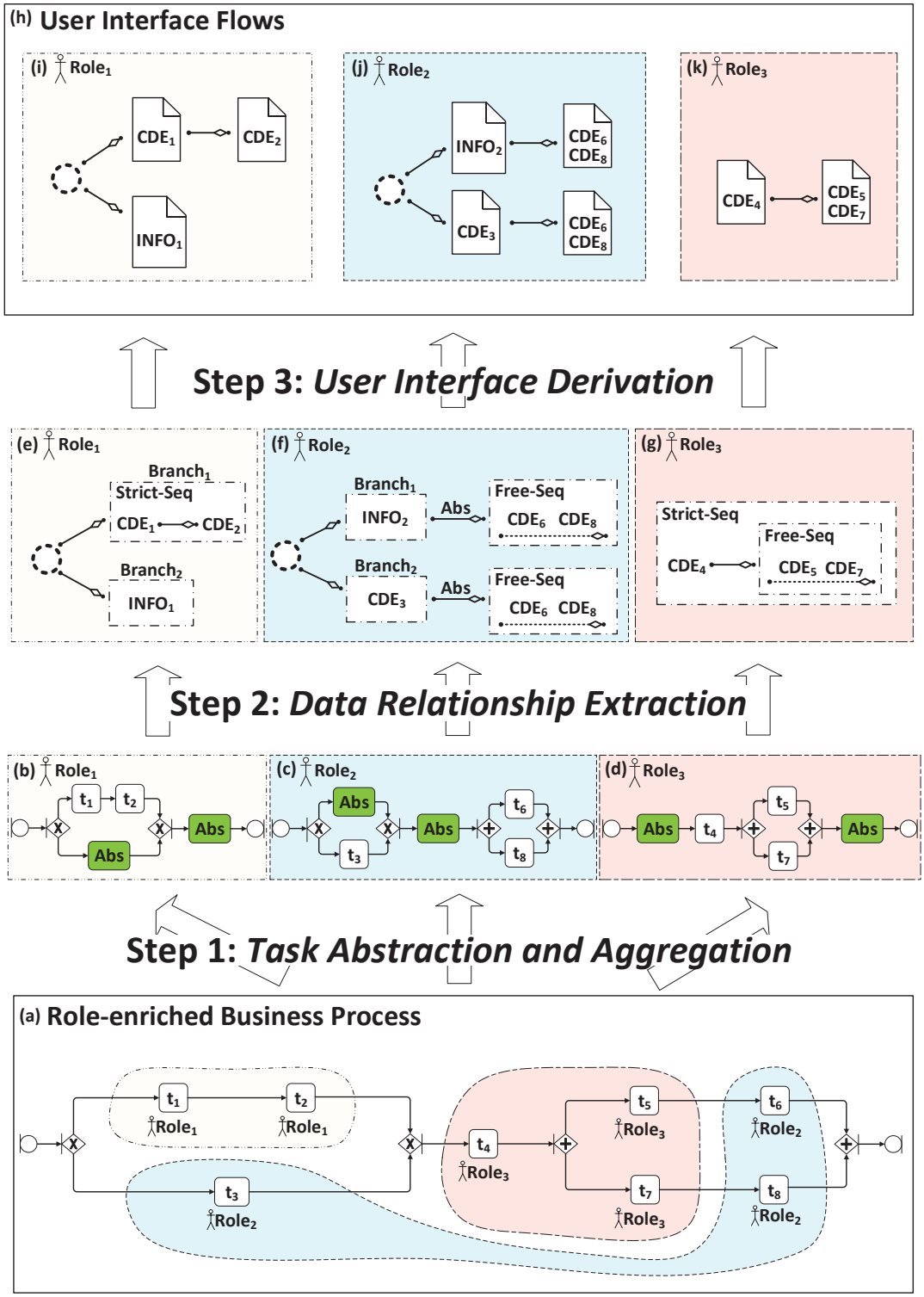


Figure 1.5: Overall Framework of UI Derivation Approach

the effective control flow in a BP. AABPs can be used to enable the development and updating of software components such as UIs related to different user roles.

As the **second step** (Step 2 in Figure 1.5), data relationships (see (e), (f), (g) in Figure 1.5) are extracted from the AABP of a specific user role according to data operations and control flow patterns in the AABP. The data relationships describe the temporal relationships between the data entities operated by the tasks related to one user role, and the dependencies between the data entities and the abstracted nodes in an AABP. According to the extracted data related to one user role, the UI derivation rules can be analyzed and built up.

As the **third step** (Step 3 in Figure 1.5), the UI logic is derived from the obtained data relationships at the second step. The UI derivation algorithm is developed based on a set of UI derivation rules. These UI derivation rules are classified as constraints and recommendations. The derived UI logic will support the generation and updating of the graphical user interfaces.

1.3 Research Contributions

In this work, we develop an approach for user interface derivation based on the proposed role-enriched BP model to automatically derive the UI logic of a business process for each participating user role. This approach can help application developers to derive UIs for BPs. Business analysts can use this approach to trace the changes happening in either business processes or related user interfaces so as to adapt changes. The UI logics derived by this approach are able to support UI designers to develop graphical UIs of business processes. In this approach, a role-enriched BP model is built up as the foundation, and there are three derivation steps as task abstraction and aggregation, data relationship extraction, and UI derivation. The contributions of the proposed approach are summarized as follows.

1.3.1 Role-enriched Business Process Model

We propose a role-enriched business process model as the foundation of the UI derivation approach. In this BP model, four aspects are specified as (1) the tasks in BP and control flow relations between these tasks; (2) the relationships between participating user roles and individual tasks; (3) the data operation flow in each task that represents a set of data items and the operation flow relations between these data items; (4) the access type (*read* or *write*) of each data item. The BPMN modelling language is extended to specify the role-enriched BP model. A set of formalized rules are specified to regulate a well-formed role-enriched business process. In the BP model, we identify a set of control flow patterns and data operation patterns to build the elementary operations and rules for UI derivation. With the role-enriched BP model, process modelers are able to specify a BP that has complex control flow patterns and capture the details of data operations inside individual tasks of a BP.

1.3.2 Task Abstraction and Aggregation

We propose a method for task abstraction and aggregation of a BP based on the role-enriched BP model. As a result of the method, an AABP is produced that provides customized descriptions of a BP for different user roles and this description is used to derive UIs of a BP related to the participating user roles. In this method, tasks of a BP are abstracted and aggregated for each user role according to the identified control flow patterns. A set of elementary operations for task abstraction and aggregation in BPs are specified. The algorithm for deriving the AABP is developed with the elementary operations as cornerstones. The structural consistency between the BP and the AABP has been analyzed. The derived AABPs with the proposed method can be used to derive business process views for process participants, and support analyzing, developing, and updating software components such as user interfaces related to different user roles.

1.3.3 Data Relationship Extraction

We propose a method for extracting the data relationships from the AABP for each user role. We use a tree graph to represent the data relationships extracted from an AABP. The extracted data relationships are the foundation to analyze and derive the UI logic. In this method, a set of elementary operations are developed according to the data operations inside individual tasks and the identified control flow patterns in the AABP. The algorithm for data relationship extraction is developed with the elementary operations as cornerstones.

1.3.4 User Interface Derivation

We propose a method for deriving the UI logic from the the extracted data relationships. The derived UI logic is the final output of the proposed UI derivation approach in this thesis. A set of UI derivation rules are specified. These UI derivation rules can be classified into two categories as **Constraints** and **Recommendations**. The **Constraints** include rules that must be followed by the UI designers. The **Recommendations** include rules that are recommended to be followed by the UI designers. The UI logic is derived by using the UI derivation algorithm with these rules as cornerstones.

1.3.5 UI Derivation Tool Development

We develop a UI Derivation Tool that implements our proposed UI derivation approach. As the input of this tool, the role-enriched BP is specified using JavaScript Object Notation. The algorithms for all the UI derivation steps, including task abstraction and aggregation, data relationship extraction, and UI derivation, are implemented as separate functional modules. The elementary operations and rules used by each derivation step is implemented as individual functions. We also develop a GUI Generator is developed to visualize the derived UI logics. A set of sequenced Windows Forms are generated for each user role as the output of the tool. This tool is capable of dealing with the business processes with complex control flow relationships and with tasks containing complex data operation flows.

1.4 Thesis Outline

This section presents the overall structure of the thesis.

In Chapter 2, we introduce the basic concepts related to business process and user interface, and overview the state-of-the-art of the research and standardization in the related areas. More specifically, business process modelling, workflow patterns, business process view generation, and UIs of business processes and web applications are discussed.

In Chapter 3, we propose the role-enriched business process model. The formal syntax and well-formness of this process model are introduced. The BPMN is extended to specify the process model. A set of control flow patterns and data operation patterns are identified. A scenario example is introduced to illustrate how to specify a process using the role-enriched BP model.

In Chapter 4, we present the method of task abstraction and aggregation for different user roles involved in a BP. A set of elementary operations are developed according to the identified control flow patterns in the role-enriched BP. The algorithm for task abstraction and aggregation is developed with the elementary operations as cornerstones. The BP structural properties are discussed, and the structural consistency between the AABP and the original BP is analyzed.

In Chapter 5, we provide the method of data relationship extraction from the AABP for each user role. A set of elementary operations are developed according to the data operations inside individual tasks and the identified control flow patterns in the AABP. The algorithm for data relationship extraction is developed with the elementary operations as cornerstones.

In Chapter 6, we present the method of UI logic derivation from the extracted data relationships. A set of UI derivation rules are specified. The UI derivation algorithm is developed with these rules as cornerstones.

In Chapter 7, we describe a tool developed as prototype that implements our proposed UI derivation approach introduced in Chapter 3, 4, 5 and 6.

In Chapter 8, we conclude this thesis by re-emphasising the contributions and discussing the directions for future research.

Chapter 2

Literature Review

In this chapter, we introduce the background related to the remainder chapters of this thesis and review related works. This is to give readers a better understanding of the work in this thesis. Section 2.1 presents the languages for business process modelling. Section 2.2 summarizes the commonly-used workflow patterns. Section 2.3 discusses the concept of business process view and its derivation from business processes. Section 2.4 introduces the principles of web application UIs design and discusses the studies on the UI derivation from business process models. Section 2.5 provides a summary and discussion on this chapter.

2.1 Business Process Modelling

M. Dumas et al [21] argued that a business process model was characterized by three properties: mapping, abstraction, and fit for purpose. (1) A BP model implies a mapping of a collection of real-world BPs which have similar characteristics. (2) A BP model only documents required aspects of the collection of BPs; irrelevant details of the processes are abstracted from the BP model. (3) A BP model exists for a particular purpose that determines what aspects of the BPs are documented when creating a process model.

These purposes of modelling BPs fall into two classes: business-oriented modelling and IT-oriented modelling. (1) The business-oriented process models contain high-level

information regarding the process description, and are built up by process analysts. This type of process models is mainly used for understanding the process and communicating the understanding with other BP stakeholders. Through analyzing the BP models, problems hidden in BPs are also able to be identified and resolved. (2) The IT-oriented process models are created by software developers and engineers and used for the development of BP applications. Thus, implementation details must be specified in the process models to be utilized as blueprints for application development, or deployed to a BP management system [6, 21].

According to the different BP modelling purposes, the BP modelling languages are classified as three categories: conceptual languages, formal languages, and execution languages [22, 23, 24].

2.1.1 Conceptual Languages

Conceptual BP modelling languages enable process modelers to express control flow between/among tasks, operated data, and participated user roles in a graphical way. Graphical standards are high-level specifications of business process, and therefore are easy to understand without prior technical training. However, this type of standards does not allow for formal analysis and are not able to be executed due to the lack of well-defined semantics. In the following, two popular conceptual languages are introduced as Business Process Model and Notation, and Unified Modelling Language - Activity Diagrams.

2.1.1.1 Business Process Model and Notation

Business process model and notation (BPMN) is currently a widely accepted standard to graphically represent business processes. It was firstly released by Business Process Management Initiative (BPMI) in 2004. Since 2005 the Object Management Group (OMG) has been in charge of maintaining BPMN, when the OMG and the BPMI merged [25]. In 2011, the version 2.0 of BPMN was released, in which the execution semantics for the diagramming and notational constructs of BPMN [26] were introduced

.

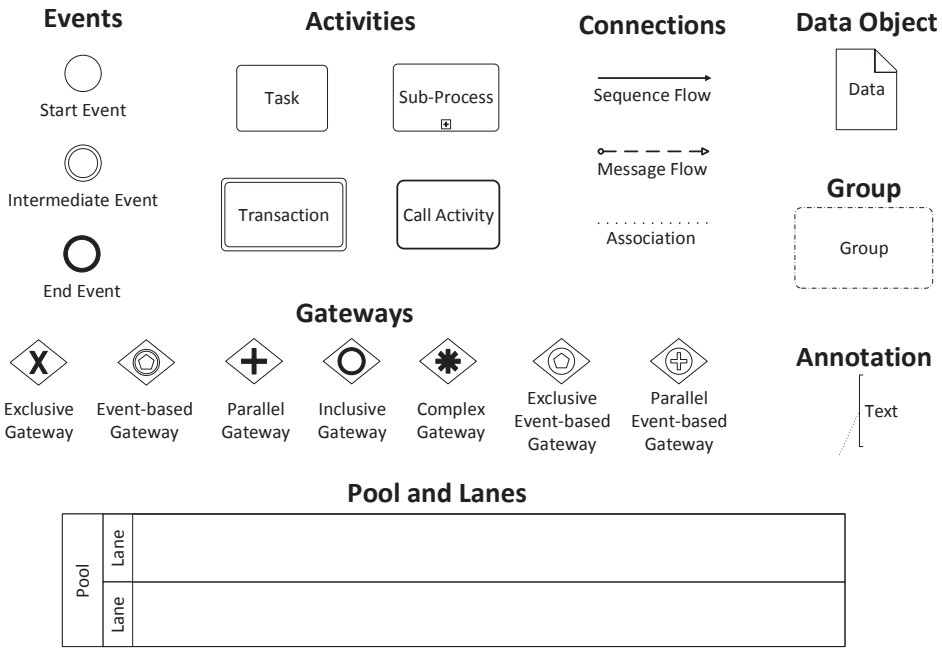


Figure 2.1: Basic BPMN Elements

BPMN not only supports business users with graphical notations, but is also capable of specifying complex semantics for technical users. The versatility of BPMN bridges the gap between business analysts and software engineers by providing a rich set of understandable and readily symbols [27, 28]. Initially, business analysts model the business processes using the provided BPMN annotations according the analyzed business requirements. Then these BPMN processes are implemented by software engineers for actual execution, which allows end users to participate in the processes and to monitor the process executions.

BPMN 2.0 contains over 100 symbols, which can be grouped into four categories: flow objects, connecting objects, swim lanes, and artifacts. In the following, we only introduce the basic elements for each category (see Figure 2.1) [11, 29].

Flow objects comprise events, activities, and gateways.

An event is denoted as a circle, which includes three common types as start event, intermediate event, and end event. A start event represents a trigger of a process. An intermediate event represents something that happens in the process. An end event

represents the termination of a process.

An activity, denoted as a rounded-corner rectangle, represents the work to be done, and comprises task, sub-process, transaction, and call activity. A task represents an atomic unit of work to be completed. A sub-process represents a compound activity included in a process, and is able to be expanded and collapsed. A transaction is a type of sub-process, inside which all the activities are treated as a whole. That means all the activities inside a transaction must be completed; if at least one of the inside activities fails to be done, the transaction will roll back to its initial status. A call activity denotes a point in a process, where a predefined process/ task is reused.

A gateway is denoted as a diamond, which represents the divergence and convergence of the control flow in a process. There are seven common types of gateways as exclusive gateway, event-based gateway, parallel gateway, inclusive gateway, complex gateway, exclusive event-based gateway, and parallel event-based gateway. An exclusive gateway provides alternative flows in a process. An event-based gateway is similar to an exclusive gateway, but the choice of the alternative flows is determined by the event. A parallel gateway provides simultaneous flows in a process. An inclusive gateway provides alternative flows each of which is evaluated. A complex gateway is to specify the behavior of complex synchronization. An exclusive event-based gateway/parallel event-based gateway instantiates process when each/all of the subsequent events occur(s).

Connecting objects, used to join the flow object, include sequence flow, message flow, and association. Swim lanes comprise pool and lane, which are used to represent different levels of the user roles participating in a process. Artifacts enable process modelers to add more information in a BPMN process so that the process becomes more readable. There exist three types of artifacts as data object, group, and annotation.

Extensions of BPMN have been proposed for various purposes. A.Meyer et al [30, 31] extended two BPMN constructs - *data objects* and *lanes* to allow the specification of complex data dependencies (e.g. m:n relationships) in BPMN. N.Lohmann et al [32] proposed the extended BPMN aiming to express artifact-centric BPs, which models processes from the perspective of data objects (a.k.a. artifacts) by specifying

the artifacts operated in a process and the life cycles of these artifacts [33, 34]. Due to the informal nature of BPMN, some works on BPMN formalization have been done. A hot debate is the semantic formalization of *Inclusive Gateways*. H.Volzer [35] used graphs to analyze the formal semantics of the BPMN 2.0 *Inclusive Converging Gateways* (a.k.a. *Or-joins*). D.Christiansen et al [36] proposed the semantic formalization of BPMN 2.0 *Inclusive Gateways* under the scenario of a minimal subset of BPMN constructs including start/stop events, and inclusive/exclusive gateways. R.Dijkman et al [37] proposed a mapping tool from BPMN to a formal language Petri net [38] in order to support static analysis of semantics in BPMN processes. P.Wong et al [39, 40] utilized Z notation [41] to express the states of a process. Then a formal language called Communicating Sequential Processes (CSP) [42] was utilized to formally specify the behaviors of the hierarchically refined BPMN processes.

2.1.1.2 Unified Modelling Language - Activity Diagrams

The Unified Modelling Language (UML) [43] is a widely-used graphical standard for object-oriented software analysis and design. UML contains 13 distinctive modelling notations ranging from the use case diagrams, which are high-level notations used to specify the interactions between the users and the system functions, to the object diagrams, which are low-level notations used to describe individual instances and the relationships between these instances. Among the modelling notations, Activity Diagrams (ADs) aim to specify both organizational and computational processes. UML-AD are flowchart-like notations to graphically model business processes; but unlike flowchart, UML ADs support parallel control flows. The fundamental shapes of UML-AD are listed as follows:

- A rounded-corner rectangle denotes an action which is a logic unit of work to be done.
- A diamond denotes a decision which diverges the process control flow according to the provided decision.

- A bar denotes the divergence/convergence of parallel actions.
- A black circle denotes the start of a process;
- An encircled black circle denotes the end of a process.

All the above constructs are connected using arrows to represent the control flow of a process [44, 45, 46].

UML-ADs provide full support for common control flow patterns like sequential, parallel, and conditional; also activity decomposition is able to be well specified in UML-ADs. Nevertheless, the data and resources associated to BPs cannot be well expressed in UML ADs; in addition, UML-ADs are difficult to get started due to their complication, and require business analysts to have prior technical backgrounds. Comparing to UML-AD, BPMN is more simple and intuitive for non-IT users to pick up, which is also the reason why UML AD is falling from favor as a BP modelling language [47, 48, 49].

2.1.2 Formal Languages

Formal BP modelling standards provide process modelers with rigorous and clear semantics to specify BPs in a theoretical way. The explicit representations of processes allow for qualitative and quantitative analysis of BP properties including soundness, well-formness and so on [24]. In the following, three major formal languages are introduced as finite state machine, Petri net, and linear temporal logic.

2.1.2.1 Finite State Machine

Finite-state machine (FSM) is a formalized model that is widely used in the modelling of systems in computer science. According the definition by J. Hopcroft [50], a FSM is defined as follows:

Definition 1: Finite State Machine. A FSM is denoted as a tuple $M = (Q, \Sigma, \Delta, \sigma, q_0)$, where:

- Q is a finite set of symbols representing states;

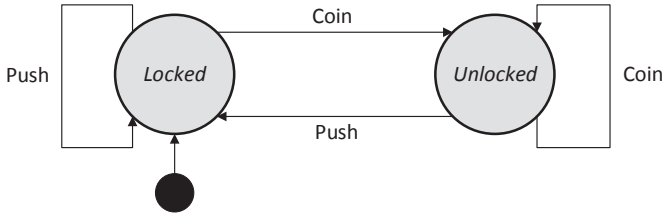


Figure 2.2: Finite State Machine Diagram for Turnstile

- Σ is a finite set of symbols representing the possible inputs;
- Δ is a finite set of symbols representing the possible outputs;
- σ is a transition function that maps $Q \times \Sigma$ to $Q \times \Delta$;
- $q_0 \in Q$ denotes initial state.

In a FSM, there exist a finite number of possible states. The machine must be in only state at a given time, and this state is called the current state of the FSM. The state of the machine can change from one to another when triggered by an event or meeting a condition, and the shift of state is called transition. A FSM is capable of analyzing problems due to the mathematical nature, and providing the solutions in a formal way [22, 51].

Figure 2.2 illustrates a classic example of a coin-operated turnstile modelled in FSM. A turnstile, having three arms, is usually set at the gate of a subway station for access control. The turnstile has two states: the arms are locked and unlocked, and the initial state is locked. There exist two inputs that influence the turnstile's state: inserting a coin in the slot and pushing the arm. In the locked state, pushing the arm does not affect the turnstile and the state will not change. When inserting a coin, the state will change from locked to unlocked. In the unlocked state, inserting additional coins will not change the existing state. When the arm is pushed by a customer, the state will change from unlocked back to locked [52].

2.1.2.2 Petri Nets

Petri nets are an established formal language for BP modelling due to their solid mathematical foundation along with intuitive graphical description, and the widespread support by software tools. The mathematical nature of Petri nets make them offer explicit semantics, and allow for the formal specification of process behaviors as well as BP analysis. The graphical expression of Petri nets enables this language to be self-describing and a effective process modelling tool, which facilitates the communication among the users involved in the BP design [53].

A Petri net is a directed network composed of four constructs: *places*, *transitions*, *directed arcs*, and *tokens*. *Places* are shown as circles; *transitions* are represented as rectangles or bars. A *directed arc* connects a *place*/*transition* and a *transition*/*place* to indicate the flow relation. It is not possible to link two *places* or two *transitions*. *Tokens*, denoted as black dots, are held in *places*. The distribution of *tokens* amongst *places* indicates the state of a Petri net [54]. A widely-accepted formal definition of Petri nets is proposed by T.Murata [55], and shown as follows:

Definition 2: Petri net. A Petri net is denoted as $N = (P, T, F, M_0)$, where:

- $P = \{p_i : i = 1, 2, ..., |P|\}$ is a finite set of places;
- $T = \{t_j : j = 1, 2, ..., |T|\}$ is finite set of transitions;
- $F \subseteq \{P \times T\} \cup \{T \times P\}$ is a set of flow relations between places and transitions;
- M_0: $P \rightarrow \{0, 1, 2, ...\}$ is the initial marking.

$M = \{M(p_1), M(p_2), ..., M(p_{|P|})\}$ expresses a marking, which specifies a state of Petri net. Starting from the initial marking M_0, a Petri net is able to reach a series of states through the firing of transitions. For a particular transition t, we use $\bullet t = \{p \in P : (p, t) \in F\}$ to represent the set of input places of t, and $t\bullet = \{p \in P : (t, p) \in F\}$ to denote the set of output places of t. A transition t is *enabled* under M denoted as $M[t >$, if $\forall p \in \bullet t$, $M_p \geq 1$. An enabled transition may fire written as $M[t > M'$, which will change the current Petri net marking M to a new marking M'. The firing of a

transition t consumes one token from each of the input places of t, and produces one token in each of the output places of t.

Based on the above classic formalism of Petri net, a lot of studies have been conducted on the extension of Petri net with additional properties to allow for modelling and analyzing specific scenarios. One of the famous extensions is colored Petri net (CPN) [56, 57], in which each token is assigned with data values, and therefore is called a colored token. Due to the extension with data values, the tokens in CPN become distinguishable between each other, which makes CPN compact and especially advantageous in modelling interactive systems such as collaborative BPs [58]. Another important Petri net extension is hierarchical Petri net (HPN), in which the views of Petri net at different abstraction and refinement levels are able to be specified and analyzed.

Apart from the above two well-known extensions, there also exist other studies on extending Petri nets for various specific uses. W.M.P.Aalst et al conducted a series of studies and investigations on the application of Petri nets in the context of workflow management [59, 60]. A BP redesign framework was proposed in [38, 61], in which the High-level Petri nets are utilized to model and analyze business processes. [62] proposed a Business Procedure net (BP-net) to model a business procedure, which allows for validating and verifying the soundness property. As an extension to the BP-net, a Workflow net (WF-net) was developed in [63, 64, 65] as a sound model to specify workflows.

2.1.2.3 Linear Temporal Logic

Linear Temporal Logic (LTL) is a modal logic with time-related modalities to reason about dynamic scenarios. LTL lets modelers develop formulas as an infinite sequence of states in which each point in time has a unique successor. Examples of LTL formulae are a condition will be true until another fact becomes true, a condition will eventually be true, etc. [66, 67, 68]. The formal syntax of LTL formulas are defined as follows:

- **if** p $\in$ propositional variables **then** p is an LTL formula;

- **if** ψ and φ are LTL formulas **then** $\neg\psi$, $\varphi \vee \psi$, $\mathbf{X}\ \psi$, and $\varphi\mathbf{U}\psi$ are LTL formulas ($\mathbf{X}$ denotes "next", $\mathbf{U}$ denotes "until").

LTL was first proposed for the formal verification of computer programs by Amir Pnueli in 1977 [69] and is also widely used for modelling business processes, typically for modelling declarative BPs. Unlike procedural BPs where execution sequences and alternatives must be explicitly represented, declarative BPs are specified as a set of execution constraints between/among tasks, that is anything is possible unless explicitly forbidden [24, 70]. In order to model the declarative BPs, LTL offers various temporal operators such as *eventually* ($\diamond$), *always* ($\Box$), and *until* ($\sqcup$) to build up LTL formulas, which are used to specify execution constraints of declarative BPs. One of the LTL applications is the Declarative Service Flow Language (DecSerFlow) proposed in [71]. Three sets of formula templates are developed as (1) *Existence Formulas* which specifies the possible number of task executions, (2) *Relation Formulas* which represents the dependencies between two tasks, (3) *Negation Formulas* which are the negated version of relation formulas. The DecSerFlow enables service designers not only to specify, enact, and monitor service flows, but also to enforce/check the conformance of service flows.

2.1.3 Execution Languages

Unlike conceptual and formal languages allowing for understanding and analyzing processes, the execution languages, providing technical syntax and semantics, aim to enable the deployment of the modelled BPs and the execution of the BP instances. In the following, two execution languages are introduced as extensible markup language and web service business process execution language.

2.1.3.1 Extensible Markup Language

The eXtensible Markup Language (XML) is a widely used markup language in computer science, developed by the World Wide Web Consortium (W3C) in 1996. A set of regulations are defined in XML for encoding documents that are understandable for

both humans and machines. XML was originally designed to meet the challenges of large-scale electronic publishing, and today is playing an increasingly significant role in the exchange of various types of data on the Web. The aims of XML are to ensure that the specified data must be simple, general, and usable across the Internet [72].

XML is a generalization of Hyper Text Markup language (HTML). The design goal of HTML is to display the data, while XML is designed to structure, store, and transport data. Another major difference is that HTML provides a predefined set of tags that represent the structuring and rendering facilities of modern Web browsers, while XML does not contain any pre-defined document tags. Instead, XML allows applications to define their own sets of tags according to the syntactical rules of XML [73].

2.1.3.2 Web Service Business Process Execution Language

The Web Service Business Process Execution Language (WS-BPEL or simply BEPL), emerging as a de-facto standard for BP modelling in web service environment, is widely adopted on multiple prominent platforms such as Oracle BPEL, SAP Netweaver, Microsoft BizTalk, and IBM WebSphere. The first version of this language called BPEL4WS 1.1 was released in 2003. It was the combination of two languages as Web Services Flow Language and Xlang. In 2007, the second version named WS-BPEL was released as an OASIS standard. BPEL is an XML based programming language and capable of specifying how a BP is built up based on the invocation of existing web services and the interaction with process participants. To support web service based BPs, BPEL not only concentrates on the message exchange and interaction between BPs and web services, but also support complex exception handling, BP installation, and long running transactions for BP enactment [23, 74, 75, 76].

As introduced by T. Andrews et al who created BPEL [13], a business process can be modelled in two ways:

- an executable business process, which specifies actual behaviors and details of the interactions between the participants and the process;

- an abstract business process, which focuses on specifying the the mutually visible messages exchanged between the participants involved in the process. Concrete operational details and behaviors of the participants are not revealed in the process. Therefore, the abstract business process is not able to be execution.

This definition indicates that the implementation details of a BP are able to be specified only through executable business processes, and that is also the initial deign goal of BPEL.

Due to the execution feature and widespread use of BPEL, there exist a number of studies on the translation between BPEL and other graphical/formal languages to bridge the gap between BP implementation and understanding analysis (e.g. [77, 78, 79, 80, 81]). So far, these translation studies merely concentrate on the translations of process control flows, other BP aspects such as resource scheduling and process semantics are not covered [82, 83].

2.2 Workflow Patterns

A pattern "is an abstraction from a concrete form which keeps recurring in specific non-arbitrary context". This description, proposed by D.Riehle et al [84], is one of the famous definitions on the concept of pattern in software development. The design patterns are both independent from the implementation technology and independent from the essential domain requirements that are to be solved [85].

Workflow patterns provide the solutions for business requirements in an imperative workflow expression, and are the guidance for BP modelling. On the one hand, workflow patterns are independent from specific workflow languages. For this reason, the expressive powers of different BP modelling languages and notations are able to examined and compared through the workflow patterns. On the other hand, workflow patterns are independent from implementation solutions, and potential mappings need to be developed between workflow patterns and implementation solutions [86, 87]. Following the way of describing the patterns for object-oriented software design [88], workflow patterns are normally specified through four aspects: *conditions* that state

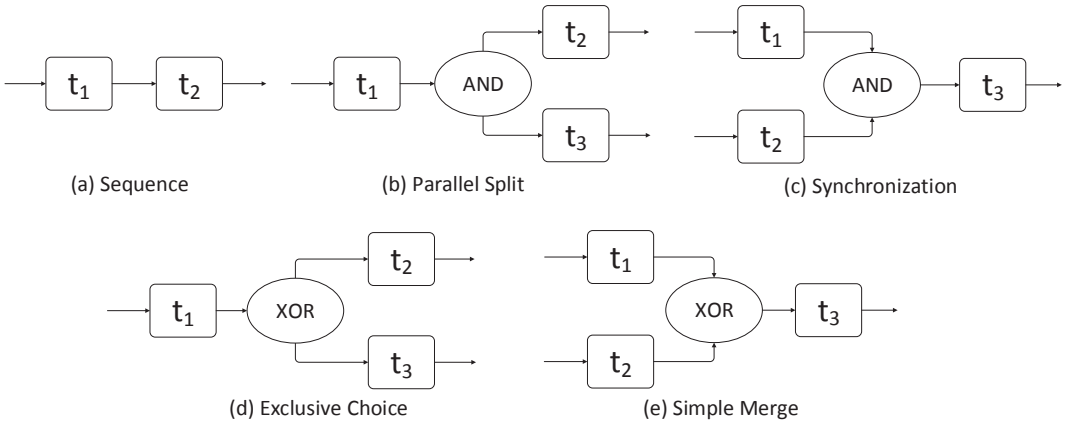


Figure 2.3: Basic Control Flow Patterns

where the pattern is applicable; *examples* that demonstrate the applicable business situations; *problems* (especially semantic problems) about how to express the patterns in existing workflow languages; and *implementation solutions* of the patterns.

A series of research on workflow patterns have been done by the team named Workflow Patterns Initiative [86, 89, 90, 91, 92], which is led by Professor Wil van der Aalst and Professor Arthur ter Hofstede. Their research provides a relatively comprehensive examination of various perspectives of workflow patterns, including control flow, data, resource, and exception handling. In our UI derivation approach proposed in the thesis, the control flow relations in a business process, and the data operation flow in each individual task of the business process refer to the most fundamental and commonly-used patterns in their work. In the following, the basic and frequently-used workflow patterns are classified as six categories as *Basic Control Flow Patterns*, *Advanced Branching and Synchronization Patterns*, *Iteration Patterns*, *Multiple Instance Patterns*, *State-based Patterns*, and *Cancellation Patterns*.

2.2.1 Basic Control Flow Patterns

This group comprises five patterns (see Figure 2.3): *Sequence*, *Parallel Split*, *Synchronization*, *Exclusive Choice*, and *Simple Merge*. These patterns are the most fundamental aspects of the process control, and closely match the elementary control flow concepts initially proposed by the Workflow Management Coalition (WfMC) [93].

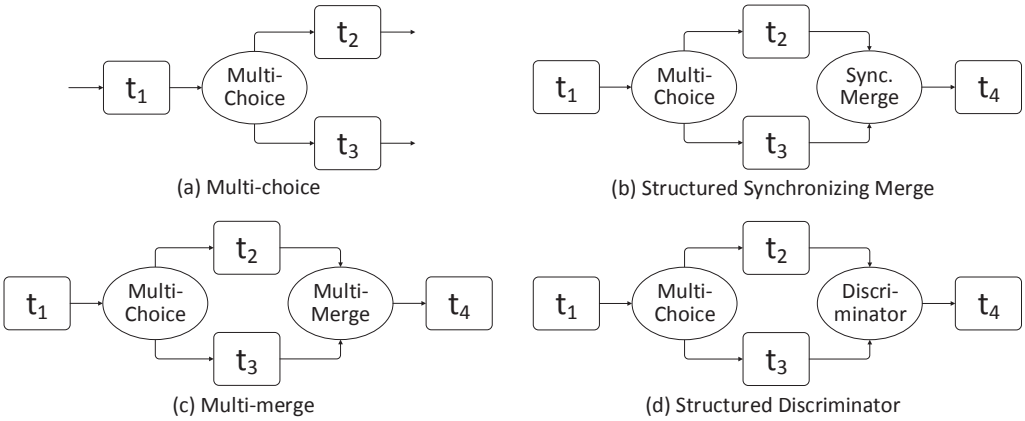


Figure 2.4: Advanced Branching and Synchronization Patterns

Sequence ((a) in Figure 2.3) captures that a task (t_2) is enabled after the preceding task (t_1) is completed.

Parallel Split ((b) in Figure 2.3) captures that after a task (t_1) is completed, all the subsequent parallel branches (t_2 and t_3) execute concurrently.

Synchronization ((c) in Figure 2.3) captures that only after the executions of all the branches (t_1 and t_2) are completed, the task (t_3) following these branches executes.

Exclusive Choice ((d) in Figure 2.3) captures that after a task (t_1) is completed, one and only one subsequent branch (t_2 or t_3) is chosen to execute according to a mechanism that can select one of the succeeding branches.

Simple Merge ((e) in Figure 2.3) captures that after the executions of any one of the branches (t_1 and t_2) is completed, the task (t_3) following these branches executes.

2.2.2 Advanced Branching and Synchronization Patterns

This group comprises four patterns (see Figure 2.4): *Multi-choice*, *Structured Synchronizing Merge*, *Multi-merge*, and *Structured Discriminator*.

Multi-choice ((a) in Figure 2.4) captures that one incoming branch (t_1) is diverged into two or more outgoing branches (t_2 and t_3) by the multi-choice gateway. And when the incoming branch is enabled, the control thread is passed through one or multiple outgoing branches according to a mechanism that choose the outgoing branches to be enabled.

Structured Synchronizing Merge ((b) in Figure 2.4) captures that two or more incoming branches (t_2 and t_3) are converged into one single outgoing branch (t_4) by the structured synchronizing merge gateway. And the control thread is only able to be passed through the structured synchronizing merge gateway when all the enabled incoming branches (t_2, or t_3, or both t_2 and t_3) are completed. The structured synchronizing merge gateway usually exists after one single multi-choice gateway in a business process, and merges all the paths emanating from the multi-choice gateway. There exist no paths splits or joins in any of these paths.

Multi-merge ((c) in Figure 2.4) captures that two or more incoming branches (t_2 and t_3) are converged into one single outgoing branch (t_4) by the multi-merge gateway. Each enabled incoming branch cause that its control thread is passed to the outgoing branch. There is no synchronization of the control threads in the multi-merge gateway.

Structured Discriminator ((d) in Figure 2.4) also known as *1-out-of-M Join*, captures that two or more incoming branches (t_2 and t_3) are converged into one single outgoing branch (t_4) by the structured discriminator gateway. When a particular incoming branch (say t_2) is enabled and completed firstly, the control thread is passed to the outgoing branch. The enabling and completion of other incoming branches (t_3) will not result in the control thread being passed to the outgoing branch (t_4). The structured discriminator gateway is reset only when all the incoming branches are enabled.

2.2.3 Iteration Patterns

This group comprises three patterns (see Figure 2.5): *Structured Loop with Post-evaluation*, *Structured Loop with Pre-evaluation*, and *Unstructured loop*.

Structured Loop with Post-evaluation ((a) in Figure 2.5) captures a loop structure in a business process which has a single entry and exit point. The task or subprocess inside the loop (t_2 and t_3) is executed repeatedly until the "jumping-out condition" at the repeat gateway is met.

Structured Loop with Pre-evaluation ((b) in Figure 2.5) captures a loop structure in a business process which has a single entry and exit point. The "jumping-in condition" at the while gateway is evaluated before the execution of the loop. If the evaluation

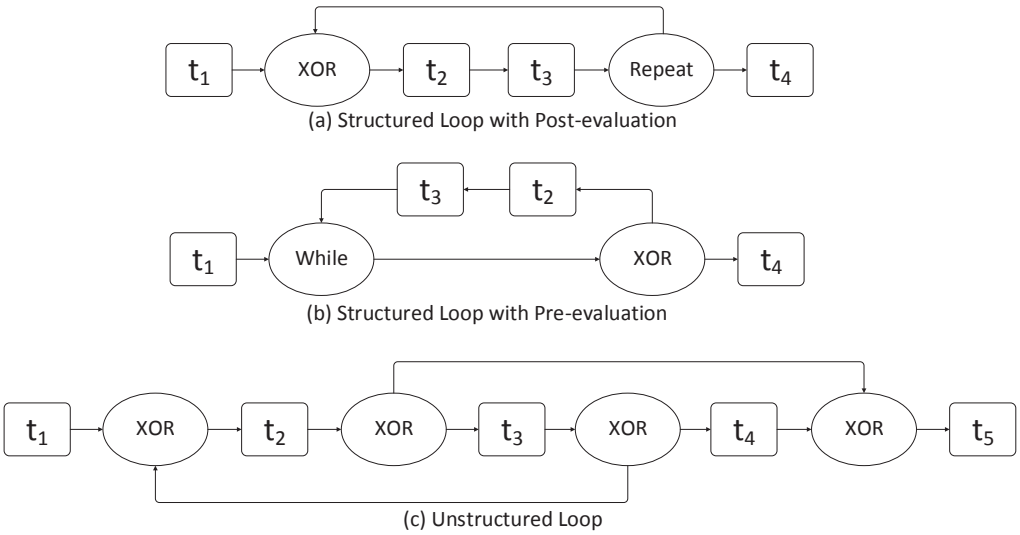


Figure 2.5: Iteration Patterns

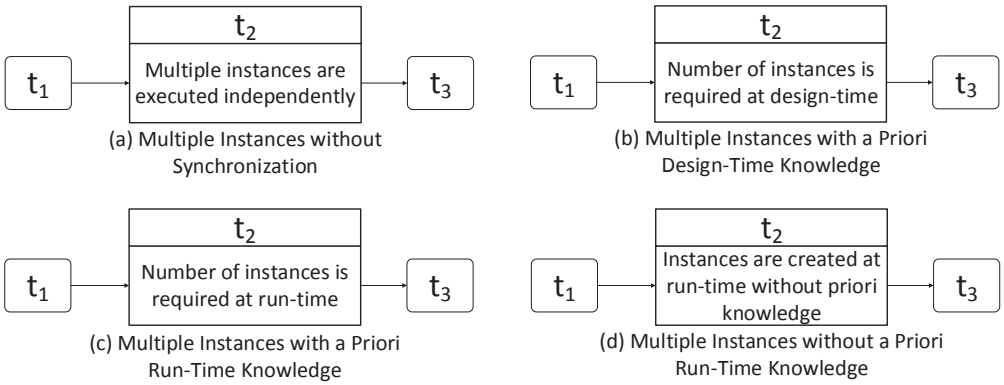


Figure 2.6: Multiple Instance Patterns

is false, the loop (t_2 and t_3) is skipped (and t_4 is executed); if the evaluation is true, the task or subprocess inside the loop is executed repeatedly until the "jumping-in condition" at the while gateway is met.

Unstructured Loop ((c) in Figure 2.5) captures a loop structure in a business process which has more than one entry and exit point. Each of the entry and exit points is connected with distinct branches.

2.2.4 Multiple Instance Patterns

This group comprises four patterns (see Figure 2.6): *Multiple Instances without Synchronization*, *Multiple Instances with a Priori Design-Time Knowledge*, *Multiple Instances with a Priori Run-Time Knowledge*, and *Multiple Instances without a Priori Run-Time Knowledge*.

Multiple Instances without Synchronization ((a) in Figure 2.6) captures that under a given process instance, a multi-instance task (t_2) is able to create multiple instances. The executions of these task instances are in parallel and independent of each other, and must be under the context of that given process instance. There exists no requirements to enforce the synchronization on the completion of the task instances.

Multiple Instances with a Priori Design-Time Knowledge ((b) in Figure 2.6) captures that under a given process instance, a multi-instance task (t_2) is able to create multiple instances, and the number of instances is set at design-time. The executions of these task instances are in parallel and independent of each other. The synchronization is required to be enforced on the completion of the task instances before any subsequent tasks are enabled.

Multiple Instances with a Priori Run-Time Knowledge ((c) in Figure 2.6) captures that under a given process instance, a multi-instance task (t_2) is able to create multiple instances, and the number of instances is set at rum-time but before the task instances are created. The executions of these task instances are in parallel and independent of each other. The synchronization is required to be enforced on the completion of the task instances before any subsequent tasks are enabled.

Multiple Instances without a Priori Run-Time Knowledge ((d) in Figure 2.6) captures that under a given process instance, a multi-instance task (t_2) is able to create multiple instances. The number of instances is decided by a number of runtime factors, such as resource availability, state data and inter-process communications, and is not known until the final instance has completed. The executions of these task instances are in parallel and independent of each other. At any time, while instances are running, it is available to initiate additional instances. The synchronization is required to be

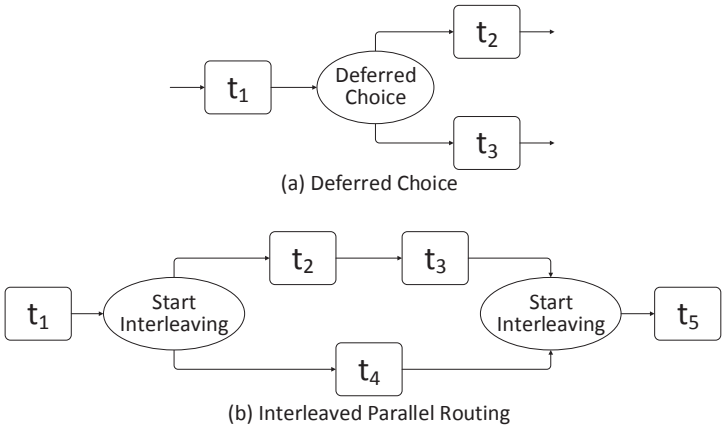


Figure 2.7: State-based Patterns

enforced on the completion of all the task instances before any subsequent tasks are enabled.

2.2.5 State-based Patterns

This group comprises two patterns (see Figure 2.7): *Deferred Choice* and *Interleaved Parallel Routing*.

Deferred Choice ((a) in Figure 2.7) captures that one incoming branch (t_1) is diverged into multiple outgoing branches (t_2 and t_3) by the deferred choice gateway. One of the outgoing branches is chosen to be executed based on interaction with the operating environment. Before the decision is made, each of the out-going branches represents possible future courses of execution. The decision is made by initiating the first task in one of the out-going branches i.e. there is no explicit choice but rather a race between the different out-going branches. After the decision is made, the alternative out-going branches other than the chosen one are cancelled.

Interleaved Parallel Routing ((b) in Figure 2.7) captures that in a parallel structure of a BP, a set of tasks (t_2, t_3, t_4) has a partial execution ordering which defines additional requirements. Each task in the set must be executed once and they can be completed in any order that in accordance with the partial ordering. Nevertheless, no two tasks can be executed at the same time (i.e. no two tasks can be enabled under the same process instance at the same time).

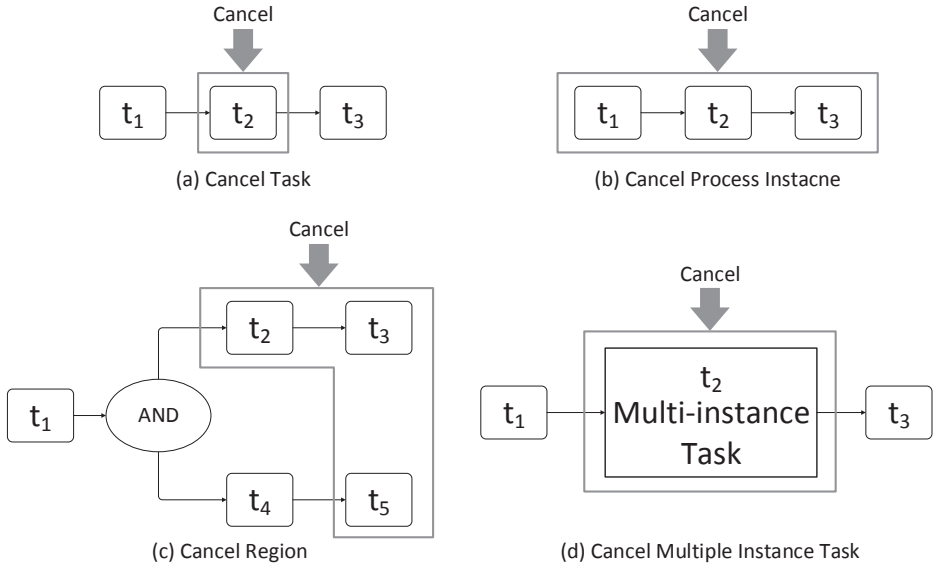


Figure 2.8: Cancellation Patterns

2.2.6 Cancellation Patterns

This group comprises four patterns (see Figure 2.8): *Cancel Task*, *Cancel Process Instance*, *Cancel Region*, and *Cancel Multiple Instance Task*.

Cancel Task ((a) in Figure 2.8) captures that an enabled task (t_2) is removed before the execution is started. If the execution of the task has already started, the currently running task instance is halted and removed.

Cancel Process Instance ((b) in Figure 2.8) captures that an entire process instance (t_1, t_2, t_3) is removed. If the process instance contains currently executing tasks and the tasks that may execute at future time, the process instance is halted and recorded as having completed unsuccessfully.

Cancel Region ((c) in Figure 2.8) captures that under a given process instance, a set of tasks (t_2, t_3, t_5) are disabled. If any task in the task set is currently executing or enabled, it is withdrawn. Note that the tasks in the task set are not required to be a connected subset in the overall business process.

Cancel Multiple Instance Task ((d) in Figure 2.8) captures that under a given process instance, multiple task instances are created by a multi-instance task (t_2). The number of to-be-created task instances is require at design-time. These task instances

are independent of each other and execute in parallel. At any time, the multi-instance task can be withdrawn, and any task instances that have not completed are cancelled. Task instances that have already completed are not influenced.

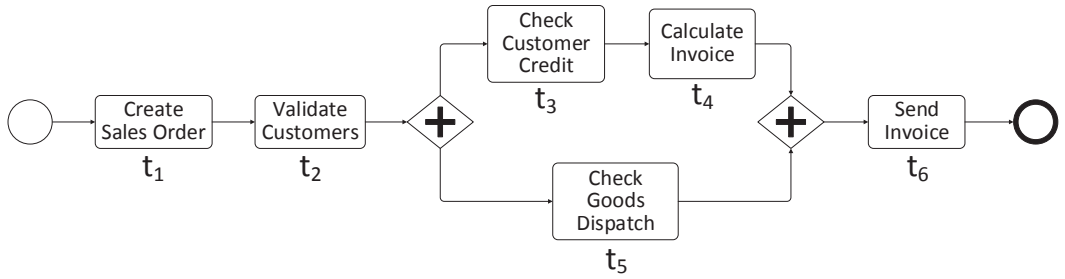
2.3 Business Process View Generation

A business process is a collection of linked tasks that provides services. Each task is a unit of work performed by human users or applications. It is often necessary to generate process views for each user role participating in the BP according to the their relationships with the process, observation intentions, and so on. A process view captures a partial representation from the actual business process, and separates the process representation from the executable BP [94, 95].

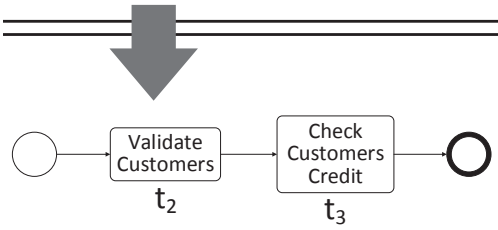
The significance of the process view generation is summarized as follows: (1) Firstly, the details of BP tasks must be hidden and abstracted from certain users due to information security requirements such as privacy, confidentiality, and conflict of interest. (2) Secondly, the process view generation lays a solid foundation for deriving customized descriptions of an BP for participating users, according to the users' requirements and intentions [20]. The customized BP descriptions may play an important role in the modelling of BP collaboration, BP visualization, and authority control. (3) Thirdly, process views highlight the requirements associated with a specific user role and preserve some information of other user roles for the effective control flow in a BP. Process views can be used to enable the development and update of software components such as user interfaces (UIs) related to different user roles.

As a scenario example, Figure 2.9 shows account receivable (AR) process and its related process views for the involved user roles [96]. Both the process and related views are represented in BPMN.

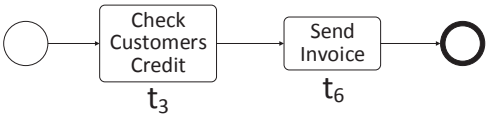
(a) in Figure 2.9 represents all the details of the AR process, which involves three user roles: `Clerk-A`, `Clerk-B` and `AR Officer`. Due to the consideration of fraud connection, the policy of duty separation applies to the process that does not allow one person to both validate customers (t_2) and calculate invoice (t_4). Therefore, `Clerk-A`



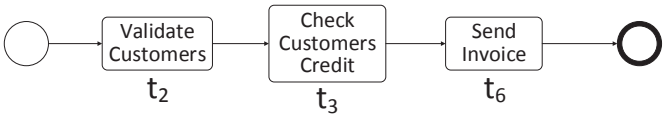
(a) Account Receivable (AR) Process



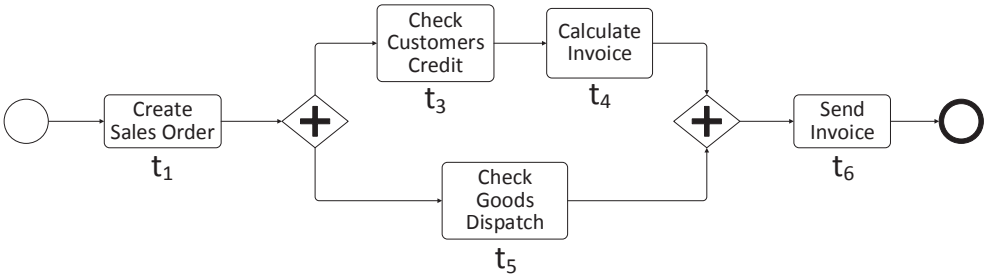
(b) Process View for **Clerk-A**



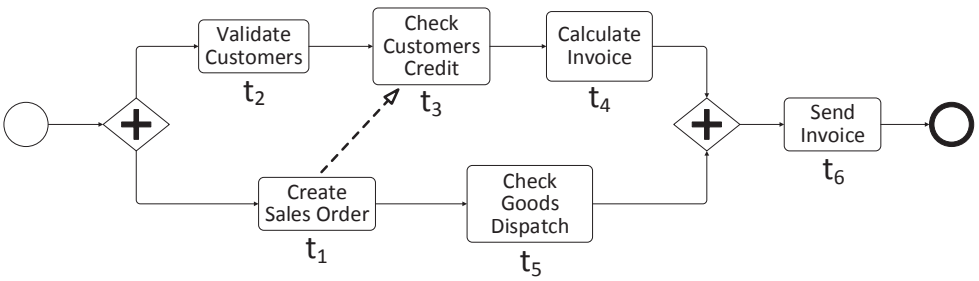
(c) Process View for **Clerk-B**



(d) Process View for **Clerk-C**



(e) Process View for **AR Officer**



(f) Combined Process View through merging (d) and (e)

Figure 2.9: Examples of a BP and Related Process Views

is responsible for validating customers (t_2) and checking customer credit (t_3). And `Clerk-B` also has the authority to perform t_3, and while he/she is in change of sending invoice (t_6). The `AR Officer` is exclusively authorized to initiate the instance of the AR process, create sales order (t_1), calculate invoice (t_4), and check goods dispatch (t_5). As the `AR Officer` plays a management role, he/she is authorized to supervise and perform all the tasks except t_2 to ensure duty separation.

To realize the diverse representations of the AR process as above, two requirements must be met as: (1) each user role participating the process must have a process view; (2) each of the generated process view must be consistent with the AR process in terms of process structure and execution order. In doing so, the user roles are able to participating in the process execution while the privacy and confidentiality can be ensured for each user role. According to these requirements, (b), (c) and (e) in Figure 2.9 show the generated process views for `Clerk-A`, `Clerk-B` and `AR Officer`, respectively. (b) keeps the tasks (t_2 and t_3) participated by `Clerk-A`, which have sequential execution order. (c) keeps the tasks (t_3 and t_6) participated by `Clerk-B`, which have sequential execution order. (e) removes the only task t_2, which is not authorized to the `AR Officer`.

If a new clerk (say `Clerk-C`) is recruited in the AR example as a backup of `Clerk-A` and `Clerk-B`, a dynamic view generation must be realized to support the analysis of potential violation against the polices of privacy and confidentiality. In this case, `Clerk-C` needs to obtain the perspectives of `Clerk-A` and `Clerk-B` as shown in (c) by merging (b) and (c). However if `Clerk-C` and the `AR Officer` have married recently, their combined view must be analyzed according to the polices of privacy and confidentiality. The process view (f) is obtained by merging (d) and (e) to describe the shared information of `Clerk-C` and the `AR Officer`. All the shared information of (d) and (e) are kept in (f). In (f), the dashed arrow represents a synchronization mechanism between the tasks t_1 and t_3, which is t_3 can only start after the completion of t_1. t_1 and t_2 are relocated to two different parallel branches due to that the execution order between t_1 and t_2 is not modelled in either (d) or (e).

This example demonstrates what the process view is for a user role involved in a process, and how the process view evolves to adapt to the change of a user role's perspective.

Process view techniques have been recognized as a significant tool for better granularity control of the business process representation. There exist a number of works focusing process view derivation.

A.Polyvyanyy et al [97, 98] proposed a methodology for task abstraction of BPs in order to generalize process models. In this work, based on the identification of insignificant tasks and their execution flow relationships, four abstraction rules are proposed aiming to cover four general situations of control flow patterns: sequential, block, loop, and dead end. Each of the abstraction rules is forced to meet two requirements as: (1) the constraints on task execution orders in a business process must be kept; (2) the absolute effort to be paid for a process execution should be preserved. Based on the four rules, an abstraction strategy is introduced which organize the rules to realize stepwise process abstraction. In the abstraction approach, the ordering of task execution in the BP is managed to be preserved in the abstracted process.

C.Gnther et al [99] proposed a task abstraction approach that used a set of metrics to evaluate the significance of task nodes and edge of BPs. The judgement of task significance is realized through evaluating three nonfunctional properties as unary significance, binary significance and binary correlation. This approach offers a feasible solution to identify what process information should be abstracted.

R.Eshuis et al [100, 101] proposed a two-step approach to build up process views, in which both process privacy and preferences of customers are ensured. Firstly, the tasks related to the user privacy are abstracted based on five rules. Then from the perspective of customization, relevant tasks are selected to be abstracted according to personalized requirements and tastes. In order to ensure consistency of the sequence of tasks, the abstraction rules enforce a group of tasks to be abstracted together, which may cause the loss of useful information. This approach focuses on two types of control flow structure as sequential and parallel.

J.Kolb et al [20, 102, 103, 104] proposed a semi-automatic view construction approach, which allows individual users to choose elementary operations and a set of parameters in the realization of hiding and aggregation of tasks in a BP. The elementary operations are divided into two groups: reduction operations to hide tasks from users, and aggregation operations to aggregate tasks as abstracted nodes. The parameter set includes order-preserving, strict order-preserving, state consistent, strict state consistent, dependency preserving, dependency erasing and dependency generating. These parameters are used to measure the degrees of execution state consistency and information loss during the view construction. As a semi-automatic approach, users are involved in the view construction to customize their choices of meeting different parameters. The view derivation approach is capable of effectively deal with complex and long-running business processes. Similarly, R.Bobrik et al [105] proposed a parameterized view derivation approach that is able to be used on both model level and instance level. Two groups of rules are developed based on graph reduction and graph aggregation. The parameter set are the same as the one developed by J.Kolb et al.

X.Zhao et al [96] proposed a view derivation approach based on user role hierarchy. The dependencies between involved roles are specified in their BP model. Their view derivation rules can cover a set of basic symmetric process structures as split/join structure, AND-split/join structure, XOR-split/join structure and so on. Two algorithms are proposed to enable two operations as view filtering and view merging. View filtering contains removing specified tasks, adjusting links/synchronization links, and checking split/join structures; view merging comprises matching XOR-split/join structures, combining views and removing redundant links, adding AND-split/join gateways, and checking AND-split/join structures. This work provides a deep analysis on how the evolution of the perspectives of user roles affects process views.

S.Yongchareon et al [106] developed a view construction for the non-well-structured BPMN processes where the control flow patterns of BP cannot be identified. Instead, events and exceptions in a process structure are analysed for view derivation. Four rules to ensure process view consistency are proposed as order preservation, branch preservation, event-attached task preservation, and message flow preservation. Another

work [107] by S.Yongchareon et al introduced a view construction approach for artifact-centric BPs. The state evolutions of data objects are explicitly represented and a set of construction rules are built up. Their view construction approach comprises two steps: (1) state composition to nesting specified state in a composite state, and (2) state hiding to conceal a specified set of tasks.

D.Liu and M.Shen [108] proposed an order-preserving approach for BP view derivation, in which the structural consistency between the business process and the derived view is ensured. In their approach, a process view is derived by aggregating multiple tasks as virtual activities (VAs) and virtual dependencies between these VAs. Three derivation rules (Membership Rule, Atomicity Rule, and Order Preservation Rule) are proposed in the view derivation approach. As an extension of this approach, a role-relevant view derivation mechanism was proposed in [109]. In this mechanism, the relationships between activities and participating user roles are considered when evaluating the degrees of relevance between user roles and activities.

2.4 User Interfaces

2.4.1 User Interfaces of Business Processes

In a business process, user interfaces are used to provide output data and require input data to/from the process users. The "user" here comprises (1) human beings participating in the business process, and (2) applications/web services invoked by the business process.

Figure 2.10 illustrates the architecture of workflow management system (WfMS) to explain the relationships between a business process and its related UIs. The WfMS architecture organizes different modules that are involved in the modelling and execution of workflows. The *Workflow Modelling* module allows BP modelers to design a business process from the perspective of implementation. For each task in a BP model, the details of execution environment are required to be specified. The *Workflow Model Repository* module is used to store the workflow models of an organization. The *Workflow Engine* module is the core component of the architecture, and is responsible for

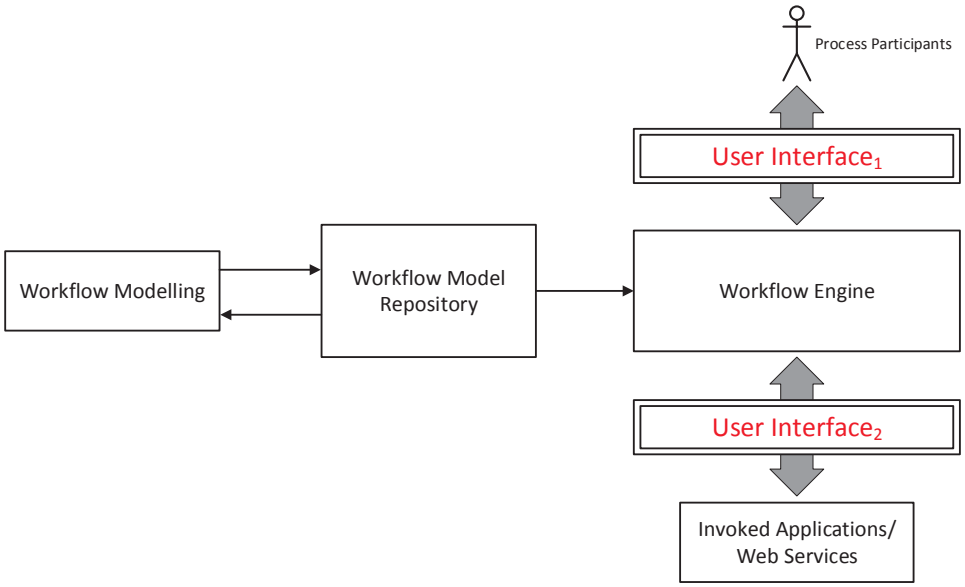


Figure 2.10: User Interfaces of Business Processes

workflow execution. Workflow instances are created by the engine according to the defined workflow models. When the engine executes a process that requires the interactions with human being participants, graphical user interfaces ($UserInterface_1$ in Figure 2.10) must exist for the communication between the BP and the participants. Through the $UserInterface_1$, the process engine is able to provide process participants with output data and ask for input data from the process participants. When the engine executes a process that needs to call application/web services, $UserInterface_2$ must exist as a bridge to transfer the data required by both the engine and the application/web services [6].

A number of works have been done to derive the UIs of business processes.

J. Kolb et al [19, 104] proposed a five-step method to generate the UI logic of a BPMN process. Firstly, the role-specific views are derived, which contains the tasks and their control flow relations related to one particular user role. Secondly, a User Interface Dialog (UID) is generated for each process view. The UID is a top-level container that holds all UI elements used to deal with the tasks within a process view. Thirdly, a set of complex transformation patterns are utilized to transform the fragment of a process

view into UI elements. Each complex transformation pattern produces a tab container element (TCE) that originates from a Single-Entry-Single-Exit (SESE) block in the process view. Fourthly, individual tasks in a process view are transformed into form group elements (FGEs). The control flow relations between tasks in the process view are represented in corresponding TCEs. Lastly, the data operated by the tasks in the process view are transformed into field elements (FEs) located inside FGEs. In their BP model, a series of elementary and complex patterns are identified to support the derivation of the UI logic from the role-specific views. Four basic control flow patterns (sequential, parallel, exclusive, and loop) have been specified and the execution flows between data items inside a task of BP have not been covered. Two types of changes (local changes and global changes) happening in either the process or related UIs are able to be adapted through this approach.

V.Kunzle et al in [110, 111] proposed an object-aware approach for BP modelling, in which the evolutions of data objects and constraints between data objects are specified. Using the BP modelling approach, the related UIs are able to be derived based on a authorization table. This table specifies that different users may own different access authorities on object attributes in a given micro process state. The authorization table can help to determine which user role is able to read/write which object attributes at a particular state of a micro process instance. According to the table, a UI form is generated from the object attributes accessed by a specific user role at a particular state of a micro process instance.

K.Sousa et al [112] developed a model-driven approach to derive UIs from a business process as Figure 2.11 according to the UsiXML models [113, 114, 115, 116]. In this approach, the UI derivation can be described as three phases:

- *Conception Phase* models business process which is the staring point of UI derivation.
- *Management Phase* builds up task model and domain model, based on which the abstract UI are analysed and modeled. Then the concrete UI is developed according to the abstract UI.

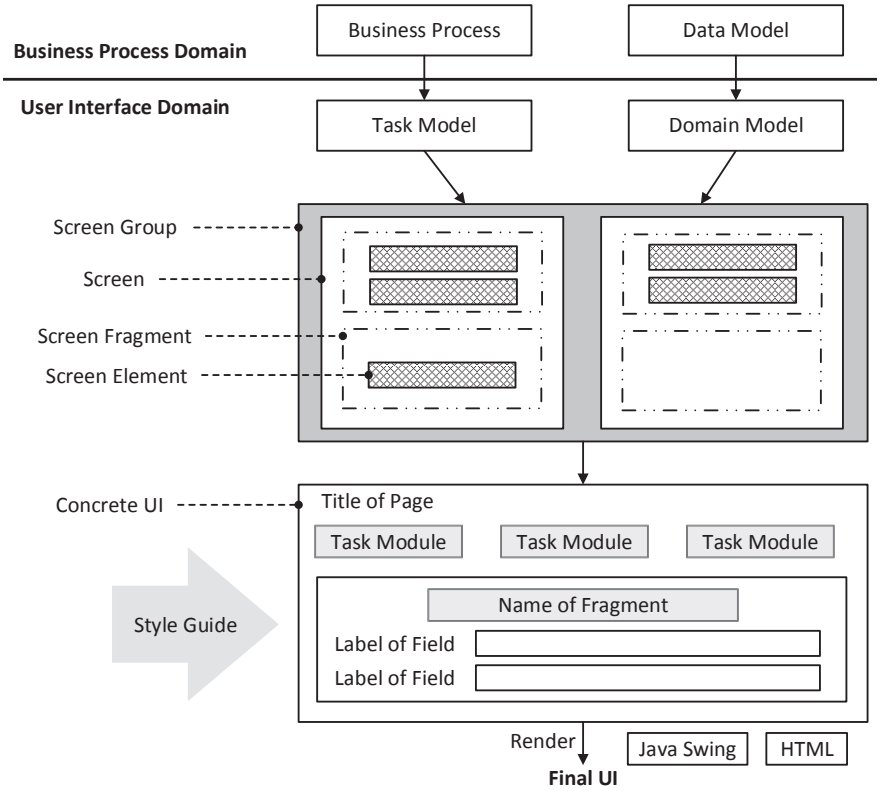


Figure 2.11: Model-driven Approach for UI Derivation from a BP

- *Application Phase* to generate the final UI of the BP.

In this approach, the consistency between the BP and the UI are preserved. The traceability is also realized to support the change management.

Artifact-centric approach is another paradigm to model BPs. It focuses on the evolutions of artifacts and associated constraints [33, 117]. A BP is specified with artifacts and data dependencies are explicitly described accordingly. S. Yongchareon et al [118] developed a framework for UI derivation based on artifact techniques. In this work, a UI flow model is proposed as the UI derivation result, which is built up based on behavioral analysis on the artifacts. An IBM team [119, 120] developed the Siena and its successor Barcelona for supporting UI derivation from artifact-centric process models. These works are able to generate the UI flow types originated from the basic BP control flow patterns such as sequential, parallel, conditional and loop.

N.Sukaviriya et al [121, 122] proposed an approach to transform a process model into a human interaction perspective. This approach is very limited in providing details of UI layouts and UI flows based on the specified data elements, user roles, tasks.

2.4.2 User Interfaces of Web Applications

The rapid development of the Internet has caused people's huge dependency on it. The Internet has turned from a virtual entity into a living world for the people. As a result of this phenomenon, radical changes are taking place in the web applications and influencing the way businesses are operated. To adapt to the changes, high-quality web applications are required to be developed along with the simple, responsive and easy-to-use UIs. In order to design such high-quality user interfaces, the fundamental principles must be followed by the UI designers. According to the usage-centered design proposed by Larry Constantine and Lucy Lockwood [123, 124, 125, 126, 127], the principles of UI design can be summarized as follows:

- *The structure principle: Design should organize the user interface purposefully, in meaningful and useful ways based on clear, consistent models that are apparent and recognizable to users, putting related things together and separating unrelated things, differentiating dissimilar things and making similar things resemble one another. The structure principle is concerned with overall user interface architecture.*

- *The simplicity principle: The design should make simple, common tasks easy, communicating clearly and simply in the user's own language, and providing good shortcuts that are meaningfully related to longer procedures.*

- *The visibility principle: The design should make all needed options and materials for a given task visible without distracting the user with extraneous or redundant information. Good designs do not overwhelm users with alternatives or confuse with unneeded information.*

- *The feedback principle: The design should keep users informed of actions or interpretations, changes of state or condition, and errors or exceptions that are relevant and of interest to the user through clear, concise, and unambiguous language familiar to users.*

- *The tolerance principle: The design should be flexible and tolerant, reducing the cost of mistakes and misuse by allowing undoing and redoing, while also preventing errors wherever possible by tolerating varied inputs and sequences and by interpreting all reasonable actions.*

- *The reuse principle: The design should reuse internal and external components and behaviors, maintaining consistency with purpose rather than merely arbitrary consistency, thus reducing the need for users to rethink and remember.*

2.5 Summary and Discussion

This chapter has discussed the related works on the UI derivation from business process models in details. Through reviewing the literature, we can see the state-of-the-art of four areas as business process modelling, workflow patterns, business process views and BP UI derivation.

The aims of BP modelling languages are related to three aspects: process description, formalization and execution. In each aspect, there exist widely-used and ripe languages, which have been reviewed and compared to support the development of BP model in our thesis.

The workflow patterns are a powerful tool to analyze and model business processes. The team of Workflow Patterns Initiative has already analyzed the existing process-aware information systems, and summarized over one hundred patterns covering the control flows, data and resources of BPs. The patterns provide a solid foundation for business process modelling. In our UI derivation approach proposed in the thesis, two aspects refer to the fundamental and commonly-used patterns from the Workflow

Patterns Initiative as (1) the control flow relations in a business process, and (2) the data operation flow in each individual task of the business process.

A process view describes a partial representation of the actual business process. The significance of the process view generation is threefold: (1) Firstly, the details of BP tasks must be hidden and abstracted from certain users due to information security requirements such as privacy, confidentiality, and conflict of interest. (2) Secondly, the process view generation lays a solid foundation for deriving customized descriptions of an BP for participating users, according to the users' requirements and intentions. The customized BP descriptions may play an important role in the modelling of BP collaboration, BP visualization, and authority control. (3) Thirdly, process views highlight the requirements associated with a specific user role and preserve some information of other user roles for the effective control flow in a BP. Process views can be used to enable the development and update of software components such as user interfaces (UIs) related to different user roles.

We have reviewed the existing studies on the approaches of the UI derivation from BP models, summarized the drawbacks of these approaches as follows:

- A.Deutsch et al [128, 129] study the data-driven web applications that interact with human users and other applications through web sites. As the interaction progresses, a web application can require data from human users and retrieve data from underlying database. In a web page, the contents, their structures and task actions are determined by the current state and input of this page. The sequence of events, including states, inputs and actions caused by the interactions are specified with Linear Temporal Logics for the verification purpose. Although this work does not consider the UI derivation from the context of business processes, their technique enables to guide the specification of BP models from the user-application perspective.

- There are a number of semi-automatic UI derivation approaches, in which the data relationships are extracted from the business processes to help further analyze and derive the UI logics. N.Sukaviriya et al [121, 122, 130] propose an UI derivation

approach to transform a process model into a human interaction perspective. This perspective specifies the relationships between the required UI logic and business data/rules. This approach is very limited in providing details related to UI layouts and UI flows. J.Fons et al [131, 132] propose a UI derivation approach based on the navigational model of Object-Oriented Web Solutions (OOWS). When a business process is built, the workflow constraints specified in the BP are mapped onto (1) navigational constraints between/among the pages of activities, and (2) data input of forms and queries on the workflow data for checking the BP status. This mapping can help to ensure the consistency between the UI logic and the BP logic.

- Q.Limbourg et al [113] develop a User Interface eXtensible Markup Language (UsiXML) to support the UI development and change management for enterprise information systems. This language enables UIs to be described at different abstraction levels while the mapping between these levels can be maintained. As their successive work, J.Vanderdonckt [114, 115, 116] proposes UI engineering method, in which the UIs are specified with UsiXML language, and then the model-to-model transformations (abstraction, reification and translation) between UIs are realized in a unified manner. K.Sousa et al [112] develop a model-driven approach to derive UIs based on a the UsiXML business process model [113]. This proposed approach has four steps as process modelling, task derivation, task refinement, and UI model derivation. In this approach, the consistency between the BP and the UI are preserved. The traceability is also realized to support the change management. However, these UI derivation methods have no capability to differentiate between constraints and recommendations.

- J.Kolb et al [19, 104] propose a five-step method to generate the UI logic of a BPMN process. The role-specific views are derived for each involved user role, which contains tasks and their control flow relations. A set of elementary and complex transformation patterns are identified to support the derivation of the UI logic from the role-specific views. There generated UI is represented as a User

Interface Dialog, which holds structured elements, e.g. tab container element, form group element and field element. In their BP model, only four basic control flow patterns (sequential, parallel, exclusive, and loop) have been specified and the execution flows between data items inside a task of BP have not been covered.

- V.Kunzle et al in [110, 111] propose an object-aware approach for BP modelling, in which the evolutions of data objects and constraints between data objects are specified. Through this modelling approach, the related UIs are able to be derived based on an authorization table. This table specifies that different users may own different access authorities on object attributes in a given micro process state. According to the table, a UI form is generated from the object attributes accessed by a specific user role at a particular state of a micro process instance. Their derived UI logic can only cover limited UI flow types (sequential, parallel, and conditional) due to that their BP model only includes the sequential, parallel and conditional data execution flow types.

- Artifact-centric approach is a declarative BP modelling paradigm, which treats BP data as first-class citizens. It focuses on the evolutions of key business entities and associated constraints [33, 117, 133, 134]. An artifact-centric BP is specified with the key business entities and their life-cycles. S.Yongchareon et al [118] develop a framework for UI derivation based on the artifact-centric BP model. Firstly, a UI flow model is derived by (1) extracting the dependencies between different artifact life cycles and (2) the data stored in each artifact. Then a role-specific UI flow model is derived from the UI flow model. Other than this work, an IBM team [119, 120] develops the Siena and its successor Barcelona for supporting UI derivation from artifact-centric process models. These works cannot generate the UI flow types originated from complex BP control flow patterns such as the Advanced Branching and Synchronization Patterns introduced in Section 2.2.2.

To sum up, the existing UI derivation approaches based on BP models are still in infancy stages. In all the above mentioned approaches, the roles of process participants

are not specified. The process models in these works only handle major basic control flow patterns, including sequential, parallel, conditional and loop. The complicated control flow patterns are not considered in their works. Besides, their UI derivation rules are not able to differentiate between mandatory and recommended ones. In order to solve these drawbacks, we aim to develop an approach for UI derivation to effectively support to derive UIs for each user role involved in a business process. The main contributions of the approach are listed as follows:

In this thesis, we propose a UI derivation method based on role-enriched business process model to derive the above complex UI logics. The proposed UI derivation method has the following features:

- A role-enriched business process model is proposed with the capabilities to specify (1) the control flow relations between tasks; (2) the relationships between the participating user roles and individual tasks; (3) the data operation flow inside each task. In the process model, we identify a set of control flow patterns and data operation patterns to build up the rules for UI derivation. These control flow and data operation patterns specify both basic and complicated situations.

- The business process is abstracted and aggregated for each user role based on the role-enrich BP model. A set of elementary operations are developed according to the control flow patterns to reserve or abstract tasks for each user role. With the AABP, a customized UI logic is derived for each participating user role.

- Data relationships are extracted from the AABP for each user role. A set of elementary operations are developed according to the data operations inside individual tasks and the identified control flow patterns in the AABP. The extracted data relationships are the foundation to analyze and derive the UI logic.

- A set of mandatory and recommended rules are specified. The UI logics are derived from the extracted data relationships based on these specified rules.

Chapter 3

Role-enriched Business Process Model

In this chapter, we discuss the role-enriched business process model, which is utilized as the foundation to build up our UI derivation approach as shown in Figure 1.5. Section 3.1 provides an introduction of this chapter. Section 3.2 introduces the formal syntax of the role-enriched BP model. Section 3.3 discusses the well-formness of the role-enriched BP model. Section 3.4 presents the extensions of BPMN. Section 3.5 summarizes the identified control flow patterns of a role-enriched BP. Section 3.6 summarizes the identified data operation patterns inside individual tasks of a role-enriched BP. Section 3.7 introduces a scenario example. Section 3.8 provides a summary and discussion on this chapter.

3.1 Introduction

The role-enrich business process model follows the activity-centric BP modelling paradigm [135], which focuses on tasks and their control flow relations. This modelling paradigm employs tasks and flow-controlling structures (gateways) as first class modeling constructs and considers the data objects in specific data states as pre-/post-conditions for task enabling or as main decision indicator at exclusive gateways. The usage of one data object in different data states in combination with multiple tasks allows to derive a life cycle for the object, which describes the manipulations performed on a

data object [9, 136]. The major representative standard in industry is the Business Process Model and Notation [27].

These features make the activity-centric paradigm powerful for expressing control flow relations between tasks, while it is disadvantageous for describing the data operated by a process, especially when specifying the data operation flow within a single task. Beyond existing features, our proposed role-enriched BP model has the capabilities to specify: (1) how user roles are involved in tasks; (2) how data are operated in individual tasks; (3) how complex control flow patterns affect data relationships. With these extensions, our process model allows for deriving the desired UI logic. In this chapter, we firstly choose a formal way to describe our role-enriched business process model so that all the process elements and their relationships can be explicitly expressed. Then we analyze the standards of a well-formed role-enriched BP model. After that, we conduct a series of necessary extensions on BPMN to specify the role-enriched BP. Besides, a set of control flow patterns and data operation patterns are identified and summarized as the basis of building up the elementary operations for task abstraction and aggregation. Lastly, we introduce a scenario example to illustrate how to specify a process using the proposed role-enriched BP model.

3.2 Formal Syntax

For the purpose of deriving the UI of a business process for each participating user role, we propose a role-enriched business process model (Re-BP model) to specify a BP with a set of tasks and control flow relations between these tasks. The participating user roles are labeled in each individual task. The data operation flow in each task specifies a set of data items and the operation flow relations between these data items. Each data item has an access type as *read* or *write*. The formal syntax is introduced as follows to specify the role-enriched business process model.

Definition 1: User Role. A user role is denoted as r, which represents a group of users or an organizational unit participating in a business process. Each user role is responsible for one or multiple tasks of the business process.

Definition 2: Data Operation Flow. A data operation flow is denoted as a tuple $df = (N_d, type_d, type_A, SF_d^{fix}, SF_d^{free})$, where:

- $N_d = \{e_d^s, e_d^e\} \cup G_d \cup A$ where e_d^s and e_d^e are the start event and end event respectively. $G_d = G_d^{in} \cup G_d^{out}$, G_d^{in} is a finite set of entry gateways and G_d^{out} is a finite set of exit gateways. The entry and exit gateways are used to control that the execution thread enters and leaves a structural block of the data operation flow. A is a finite set of data items operated by user roles.

- $type_d$: $G_d \rightarrow$ {*Sequential*, *Conditional*, *Loop*} is a mapping function to give each gateway a type. These gateways are the elements that control how the data flow diverges or converges.

- $type_A : A \rightarrow$ {*read*, *write*} is a mapping function to specify the access type of each data item $a \in A$. There are two kinds of access types as *read* and *write*. The *read* represents the data item provides particular information to a user role, and the *write* denotes the data item requires input information from a user role.

- SF_d^{fix} and SF_d^{free} represent fixed-order sequence flow and free-order sequence flow respectively. The fixed-order sequence flow means that the sequence of involved data items must be in a fixed order; the free-order sequence flow means that the order in the sequence of involved data entities is free. We use $\langle a, b \rangle$ to represent the fixed-order sequence flow from data item a to data item b; we use $[a, b]$ to represent the free-order sequence flow relation from data item a to data item b.

Based on the formal definitions of user role and data operation flow model, the role-enriched business process model is formally defined as follows.

Definition 3: Role-enriched Business Process Model. A role-enriched BP model is denoted as a tuple $rm = (N_t, type_t, SF_t^{fix}, SF_t^{free}, refine, R, \rho)$, where:

- $N_t = \{e_t^s, e_t^e\} \cup G_t \cup T$ where e_t^s and e_t^s are start event and end event respectively. $G_t = G_t^{in} \cup G_t^{out}$, G_t^{in} is a finite set of entry gateways and G_t^{out} is a finite set of exit gateways. The entry and exit gateways are used to control that the execution

thread enters and leaves a structural block of the control flow in a business process. T is a finite set of tasks, each of which indicates a logic unit of work.

- $type_t$: $G_t \rightarrow$ {*Sequential*, *Parallel-A*, *Parallel-B*, *Parallel-C*, *Conditional*, *Loop*} is a mapping function to give each gateway a type. These gateways are elements that control how the task flow diverges or converges.

- SF_t^{fix} and SF_t^{free} represent fixed-order sequence flow and free-order sequence flow respectively. The fixed-order sequence flow means that the sequence of involved tasks must be in a fixed order; the free-order sequence flow means that the order in the sequence of involved tasks is free. We use $\langle a, b \rangle$ to represent the fixed-order sequence flow from task a to task b; we use $[a, b]$ to represent the free-order sequence flow relation from task a to task b.

- $refine : T \rightarrow DF$ is a refinement function on tasks. $DF = \{df_1, df_2, ..., df_n\}$ stands for a finite set of data operation flows. The refinement function links tasks and their corresponding data operation flows.

- R is a finite set of user roles defined as above.

- $\rho = T \times R$ specifies relationships between user roles and tasks. This relationship captures that by which user roles a particular task is performed. Note that each task can be participated by one or many user roles.

3.3 Well-formed Role-enriched Business Process Model

In this section, we introduce a set of rules which specify the characteristics of a well-formed Re-BP model. These rules allow for correctly deriving UIs of a BP for each involved user role. In a well-formed role-enriched business process, the control flow patterns and data flow patterns are able to be correctly identified, which are the basis to build up the rules of UI derivation.

The "well-formness" means that the role-enriched business process must be structured correctly. It contains two aspects: on the one hand, the data operation flows

between data items within each individual task must be well-formed. A data operation flow within a task must have one start event and one end event, and this is to explicitly indicate the starting and ending of the data operation flow; all the data flow objects, including events, data items and gateways, must follow their own connecting rules when connecting with each other to ensure the proper execution in the data operation flow; the data operation flows between data items must be modelled according to the data operation patterns, which will be introduced in the latter Section 3.6, and this is to make sure the UI derivation rules are built up properly. On the other hand, the control flow relations between tasks within the Re-BP model must be well-formed. A Re-BP model must have one start event and one end event, and this is to explicitly indicate the starting and ending of the business process; all the flow objects, including events, tasks and gateways, must follow their own connecting rules when connecting with each other to ensure the proper execution in the business process; the control flow relations between tasks must be modelled according to the control flow patterns, which will be introduced in the latter Section 3.5, and this is to make sure the UI derivation rules are built up properly. In the following, the regulations of a well-formed Re-BP model are formally specified.

Definition 4: Well-formness of Role-enriched Business Process Model.
Given a role-enriched business process model $rm = (N_t, type_t, SF_t^{fix}, SF_t^{free}, refine, R, \rho)$, with related set of data operation flows $DF = \{df_i = (N_{i,d}, type_{i,d}, type_{i,A}, SF_{i,d}^{fix}, SF_{i,d}^{free}) | i = 0, 1, 2, ..., n, n \geq 0\}$, rm is said to be well-formed *iff* the following statements hold:

1. A data operation flow must contain one and only one start event and one and only one end event.

   $\forall df_i$, *where* $i = 0, 1, 2, ..., n, n \geq 0$, $\exists! e_{i,d}^s \in E_{i,d}^s, \exists! e_{i,d}^e \in E_{i,d}^e$: $e_{i,d}^s \in N_{i,d} \wedge e_{i,d}^e \in N_{i,d}$.

2. In a data operation flow, a start event must connect to an entry gateway with one and only one outgoing fixed-order sequence flow, and with no incoming fixed-order or free-order sequence flows.

$\forall e^s_{i,d} \in E^s_{i,d}$, *where* $i = 0, 1, 2, ..., n, n \geq 0, E^s_{i,d} \subsetneqq df_i$, $\exists! g_{i,d} = (g^{in}_{i,d}, g^{out}_{i,d}) \in G_{i,d}$: $\langle e^s_{i,d}, g^{in}_{i,d} \rangle \in SF^{fix}_{i,d}$;

$\forall e^s_{i,d} \in E^s_{i,d}$, *where* $i = 0, 1, 2, ..., n, n \geq 0, E^s_{i,d} \subsetneqq df_i$, $\nexists n_{i,d} \in N_{i,d} \setminus \{e^s_{i,d}\}$: $\langle n_{i,d}, e^s_{i,d} \rangle \in SF^{fix}_{i,d} \wedge [n_{i,d}, e^s_{i,d}] \in SF^{free}_{i,d}$.

3. In a data operation flow, an end event must connect to an exit gateway with one and only one incoming fixed-order sequence flow, and with no outgoing fixed-order or free-order sequence flows.

$\forall e^e_{i,d} \in E^e_{i,d}$, *where* $i = 0, 1, 2, ..., n, n \geq 0, E^e_{i,d} \subsetneqq df_i$, $\exists! g_{i,d} = (g^{in}_{i,d}, g^{out}_{i,d}) \in G_{i,d}$: $\langle g^{out}_{i,d}, e^e_{i,d} \rangle \in SF^{fix}_{i,d}$;

$\forall e^e_d \in E^e_d, where E^e_d \subsetneqq df$, $\nexists n_d \in N_d \setminus \{e^e_d\}$: $\langle e^e_d, n_d \rangle \in SF^{fix}_d \wedge [e^e_d, n_d] \in SF^{free}_d$.

4. In a data operation flow, an data item must have one and only one incoming sequence flow and one and only one outgoing sequence flow.

$\forall a_i \in A_i$, *where* $i = 0, 1, 2, ..., n, n \geq 0, A_i \subsetneqq df_i$, $\exists! d^x_{i,d}, d^y_{i,d} \in G^{in}_{i,d} \cup G^{out}_{i,d} \cup A_i \setminus \{a_i\}$, *where* $d^x_{i,d} \neq d^y_{i,d}$: $\langle d^x_{i,d}, a_i \rangle \in SF^{fix}_{i,d} \wedge [d^x_{i,d}, a] \in SF^{free}_{i,d} \wedge \langle a_i, d^y_{i,d} \rangle \in SF^{fix}_{i,d} \wedge [a_i, d^y_{i,d}] \in SF^{free}_{i,d}$.

5. In a data operation flow, a *sequential* entry gateway or a *sequential* exit gateway must have one and only one incoming sequence flow, and one and only one outgoing sequence flow.

$\forall g_{i,d} = (g^{in}_{i,d}, g^{out}_{i,d}) \in \{G_{i,d} | type^G_{i,d}(G_{i,d}) = Sequential\}$, *where* $i = 0, 1, 2, ..., n, n \geq 0, \{G_{i,d} | type^G_t(G_{i,d}) = Sequential\} \in df_{i,d}$, $\exists! n^w_{i,d}, n^x_{i,d} \in N_{i,d} \setminus \{g^{in}_{i,d}, g^{out}_{i,d}\}$: $\langle n^w_{i,d}, g^{in}_{i,d} \rangle \in SF^{fix}_{i,d} \wedge [n^w_{i,d}, g^{in}_{i,d}] \in SF^{free}_{i,d} \wedge \langle g^{in}_{i,d}, n^x_{i,d} \rangle \in SF^{fix}_{i,d} \wedge [g^{in}_{i,d}, n^x_{i,d}] \in SF^{free}_{i,d}$;

$\forall g_{i,d} = (g^{in}_{i,d}, g^{out}_{i,d}) \in \{G_{i,d} | type^G_d(G_{i,d}) = Sequential\}$, *where* $i = 0, 1, 2, ..., n, n \geq 0, \{G_{i,d} | type^G_d(G_{i,d}) = Sequential\} \in df_{i,d}$, $\exists! n^y_{i,d}, n^z_{i,d} \in N_{i,d} \setminus \{g^{in}_{i,d}, g^{out}_{i,d}\}$: $\langle n^y_{i,d}, g^{out}_{i,d} \rangle \in SF^{fix}_{i,d} \wedge [n^y_{i,d}, g^{out}_{i,d}] \in SF^{free}_{i,d} \wedge \langle g^{out}_{i,d}, n^z_{i,d} \rangle \in SF^{fix}_{i,d} \wedge [g^{out}_{i,d}, n^z_{i,d}] \in SF^{free}_{i,d}$.

6. In a data operation flow, an entry gateway of *Parallel-A*, *Parallel-B*, *Parallel-C*, or *Conditional* must have one and only one incoming sequence flow and more than one outgoing fixed-order sequence flows. An exit gateway of *Parallel-A*, *Parallel-B*, *Parallel-C*, or *Conditional* must have one and only one outgoing sequence flow and more than one incoming fixed-order sequence flows.

   $\forall g_{i,d} = (g^{in}_{i,d}, g^{out}_{i,d}) \in \{G_{i,d} | type^G_{i,d}(G_{i,d}) \neq Sequential, Loop\}$, *where* $i = 0, 1, 2, ..., n, n \geq 0, \{G_{i,d} | type^G_{i,d}(G_{i,d}) \neq Sequential, Loop\} \in df_{i,d}$, $\exists! n^x_{j,d} \in N_{i,d} \setminus \{g^{in}_{i,d}\}$, $\exists! n^i_{i,d} \in N_{i,d} \setminus \{n^x_{i,d}, g^{in}_{i,d}\}$, *where* $j \in \{1, 2, ..., m\} \wedge m \geq 2$: $\langle n^x_{i,d}, g^{in}_{i,d} \rangle \in SF^{fix}_{i,d} \wedge [n^x_{i,d}, g^{in}_{i,d}] \in SF^{free}_{i,d} \wedge \langle g^{in}_{i,d}, n^j_{i,d} \rangle \in SF^{fix}_{i,d}$;

   $\forall g_{i,d} = (g^{in}_{i,d}, g^{out}_{i,d}) \in \{G_{i,d} | type^G_{i,d}(G_{i,d}) \neq Sequential, Loop\}$, *where* $i = 0, 1, 2, ..., n, n \geq 0, \{G_d | type^G_{i,d}(G_{i,d}) \neq Sequential, Loop\} \in df_{i,d}$, $\exists! n^y_{i,d} \in N \setminus \{g^{out}_{i,d}\}$, $n^k_{i,d} \in N_{i,d} \setminus \{n^y_{i,d}, g^{out}_{i,d}\}$, *where* $k \in \{1, 2, ..., p\} \wedge p \geq 2$: $\langle g^{out}_{i,d}, n^y_{i,d} \rangle \in SF^{fix}_{i,d} \wedge [g^{out}_{i,d}, n^y_{i,d}] \in SF^{free}_{i,d} \wedge \langle n^k_{i,d}, g^{out}_{i,d} \rangle \in SF^{fix}_{i,d}$.

7. In a data operation flow, for each gateway type, the entry gateway and exit gateway must not be connected directly.

   $\forall g_{i,d} = (g^{in}_{i,d}, g^{out}_{i,d}) \in \{G_{i,d} | type^G_{i,d}(G_{i,d}) \neq Loop\}$, *where* $i = 0, 1, 2, ..., n, n \geq 0, \{G_{i,d} | type^G_{i,d}(G_{i,d}) \neq Loop\} \in df_{i,d}$: $\langle g^{in}_{i,d}, g^{out}_{i,d} \rangle \notin SF^{fix}_{i,d} \wedge [g^{in}_{i,d}, g^{out}_{i,d}] \notin SF^{fix}_{i,d}$.

8. A role-enriched business process must contain one and only one start event, and one and only one end event.

   $\forall rm$, $\exists! e^s_t \in E^s_t, \exists! e^e_t \in E^e_t$: $e^s_t \in N_t \wedge e^e_t \in N_t$.

9. In a role-enriched business process, a start event must connect to an entry gateway. This entry gateway has one and only one outgoing fixed-order sequence flow, and no incoming fixed-order or free-order sequence flows.

$\forall e_t^s \in E_t^s$, *where* $E_t^s \subsetneqq rm$, $\exists! g_t = (g_t^{in}, g_t^{out}) \in G_t$: $\langle e_t^s, g_t^{in} \rangle \in SF_t^{fix}$;

$\forall e_t^s \in E_t^s$, *where* $E_t^s \subsetneqq rm$, $\nexists n_t \in N_t \backslash \{e_t^s\}$: $\langle n_t, e_t^s \rangle \in SF_t^{fix} \wedge [n_t, e_t^s] \in SF_t^{free}$.

10. In a role-enriched business process, an end event must connect to an exit gateway with one and only one incoming fixed-order sequence flow, and no outgoing fixed-order or free-order sequence flows.

$\forall e_t^e \in E_t^e$, *where* $E_t^e \subsetneqq rm$, $\exists! g_t = (g_t^{in}, g_t^{out}) \in G_t$: $\langle g_t^{out}, e_t^e \rangle \in SF_t^{fix}$;

$\forall e_t^e \in E_t^e$, *where* $E_t^e \subsetneqq rm$, $\nexists n_t \in N_t \backslash \{e_t^e\}$: $\langle e_t^e, n_t \rangle \in SF_t^{fix} \wedge [e_t^e, n_t] \in SF_t^{free}$.

11. In a role-enriched business process, a task must have one and only one incoming sequence flow, and only one outgoing sequence flow.

$\forall t \in T, where T \subsetneqq rm$, $\exists! g_t^x, g_t^y \in G_t^{in} \cup G_t^{out} \cup T \backslash \{t\}$, *where* $g_t^x \neq g_t^y$: $\langle g_t^x, t \rangle \in SF_t^{fix} \wedge [g_t^x, t] \in SF_t^{free} \wedge \langle t, g_t^y \rangle \in SF_t^{fix} \wedge [t, g_t^y] \in SF_t^{free}$.

12. In a role-enriched business process, any *sequential* entry gateway or *sequential* exit gateway must have one and only one incoming sequence flow, and one and only one outgoing sequence flow.

$\forall g_t = (g_t^{in}, g_t^{out}) \in \{G_t | type_t^G(G_t) = Sequential\}$, *where* $\{G_t | type_t^G(G_t) = Sequential\} \in rm$, $\exists! n_t^w, n_t^x \in N_t \backslash \{g_t^{in}, g_t^{out}\}$: $\langle n_t^w, g_t^{in} \rangle \in SF_t^{fix} \wedge [n_t^w, g_t^{in}] \in SF_t^{free} \wedge \langle g_t^{in}, n_t^x \rangle \in SF_t^{fix} \wedge [g_t^{in}, n_t^x] \in SF_t^{free}$;

$\forall g_t = (g_t^{in}, g_t^{out}) \in \{G_t | type_t^G(G_t) = Sequential\}$, *where* $\{G_t | type_t^G(G_t) = Sequential\} \in rm$, $\exists! n_t^y, n_t^z \in N_t \backslash \{g_t^{in}, g_t^{out}\}$: $\langle n_t^y, g_t^{out} \rangle \in SF_t^{fix} \wedge [n_t^y, g_t^{out}] \in SF_t^{free} \wedge \langle g_t^{out}, n_t^z \rangle \in SF_t^{fix} \wedge [g_t^{out}, n_t^z] \in SF_t^{free}$.

13. In a role-enriched business process, any entry gateway of *Parallel-A*, *Parallel-B*, *Parallel-C*, or *Conditional* must have one and only one incoming sequence flow and more than one outgoing fixed-order sequence flow; any exit gateway of

Parallel-A, *Parallel-B*, *Parallel-C*, or *Conditional* must have one and only one outgoing sequence flow and more than one incoming fixed-order sequence flow.

$\forall g_t = (g_t^{in}, g_t^{out}) \in \{G_t | type_t^G(G_t) \neq Sequential, Loop\}$, *where* $\{G_t | type_t^G(G_t) \neq Sequential, Loop\} \in rm$, $\exists! n_t^x \in N_t \setminus \{g_t^{in}\}$, $\exists! n_t^i \in N_t \setminus \{n_t^x, g_t^{in}\}$, *where* $i \in \{1, 2, ..., m\} \wedge m \geq 2$: $\langle n_t^x, g_t^{in} \rangle \in SF_t^{fix} \wedge [n_t^x, g_t^{in}] \in SF_t^{free} \wedge \langle g_t^{in}, n_t^i \rangle \in SF_t^{fix}$;

$\forall g_t = (g_t^{in}, g_t^{out}) \in \{G_t | type_t^G(G_t) \neq Sequential, Loop\}$, *where* $\{G_t | type_t^G(G_t) \neq Sequential, Loop\} \in rm$, $\exists! n_t^y \in N \setminus \{g_t^{out}\}$, $n_t^j \in N_t \setminus \{n_t^y, g_t^{out}\}$, *where* $j \in \{1, 2, ..., k\} \wedge k \geq 2$: $\langle g_t^{out}, n_t^y \rangle \in SF_t^{fix} \wedge [g_t^{out}, n_t^y] \in SF_t^{free} \wedge \langle n_t^j, g_t^{out} \rangle \in SF_t^{fix}$.

14. In a role-enriched business process, any *Loop* entry gateway must have one and only one incoming sequence flow to indicate entering of the *Loop* block, any *Loop* exit gateway must have one and only one outgoing sequence flow to indicate exit of the *Loop* block; there exists one and only one fixed-order sequence flow directly pointing from the *Loop* exit gateway to the *Loop* entry gateway to indicate iteration of the *Loop* block.

$\forall g_t = (g_t^{in}, g_t^{out}) \in \{G_t | type_t^G(G_t) = Loop\}$, *where* $\{G_t | type_t^G(G_t) = Loop\} \in rm$, $\exists! n_t^x \in N_t \setminus \{g_t^{in}, g_t^{out}\}$: $\langle n_t^x, g_t^{in} \rangle \in SF_t^{fix} \wedge [n_t^x, g_t^{in}] \in SF_t^{free}$;

$\forall g_t = (g_t^{in}, g_t^{out}) \in \{G_t | type_t^G(G_t) = Loop\}$, *where* $\{G_t | type_t^G(G_t) = Loop\} \in rm$, $\exists! n_t^y \in N_t \setminus \{g_t^{in}, g_t^{out}\}$: $\langle g_t^{out}, n_t^y \rangle \in SF_t^{fix} \wedge [g_t^{out}, n_t^y] \in SF_t^{free}$;

$\forall g = (g_t^{in}, g_t^{out}) \in \{G_t | type_t^G(G_t) = Loop\}$, *where* $\{G_t | type_t^G(G_t) = Loop\} \in rm$: $\langle g_t^{in}, g_t^{out} \rangle \in SF_t^{fix}$.

15. In a role-enriched business process, for each gateway type except *Loop*, the entry gateway and exit gateway must not be connected directly.

$\forall g_t = (g_t^{in}, g_t^{out}) \in \{G_t | type_t^G(G_t) \neq Loop\}$, *where* $\{G_t | type_t^G(G_t) \neq Loop\} \in rm$: $\langle g_t^{in}, g_t^{out} \rangle \notin SF_t^{fix} \wedge [g_t^{in}, g_t^{out}] \notin SF_t^{free}$.

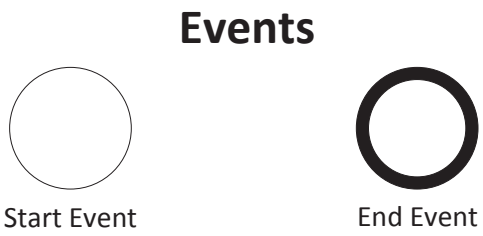


Figure 3.1: BPMN Constructs - Events

3.4 Extension of BPMN

In order to specify the role-enriched business process model in the proposed derivation framework, we choose the BPMN that is a widely-used standard modelling language for business processes, especially at the level of domain analysis and high-level system design. The current version is BPMN 2.0 [27], which has already provided a rich set of constructs to specify tasks, control flows, and gateways. However, in order to specify the role-enriched business process model, the current BPMN does not have enough expressive power in terms of three aspects: (1) the relations between tasks and user roles, (2) the operated data inside each task, and (3) the complicated control flow relations between tasks and data operation flows between data items. To bridge these gaps, we provide the extended BPMN as follows.

Figure 3.1 illustrates the start event and end event in BPMN. A **start event** is drawn as a circle with an open center, to indicate the staring point of either a role-enriched business process or a data operation flow within a task of the role-enriched business process. When the trigger for a start event occurs, a new business process/data operation flow is instantiated and a token is generated for each outgoing sequence flow from the starting event. An **end event** is drawn as a bold circle with an open center, to indicate the ending point of either a role-enriched business process or a data operation flow within a task of the role-enriched business process. When the trigger for an end event occurs, the instance of the business process/data operation flow is completed and the token from each incoming sequence flow is gone.

Figure 3.2 illustrates the gateways in BPMN. Gateways are the mechanisms to

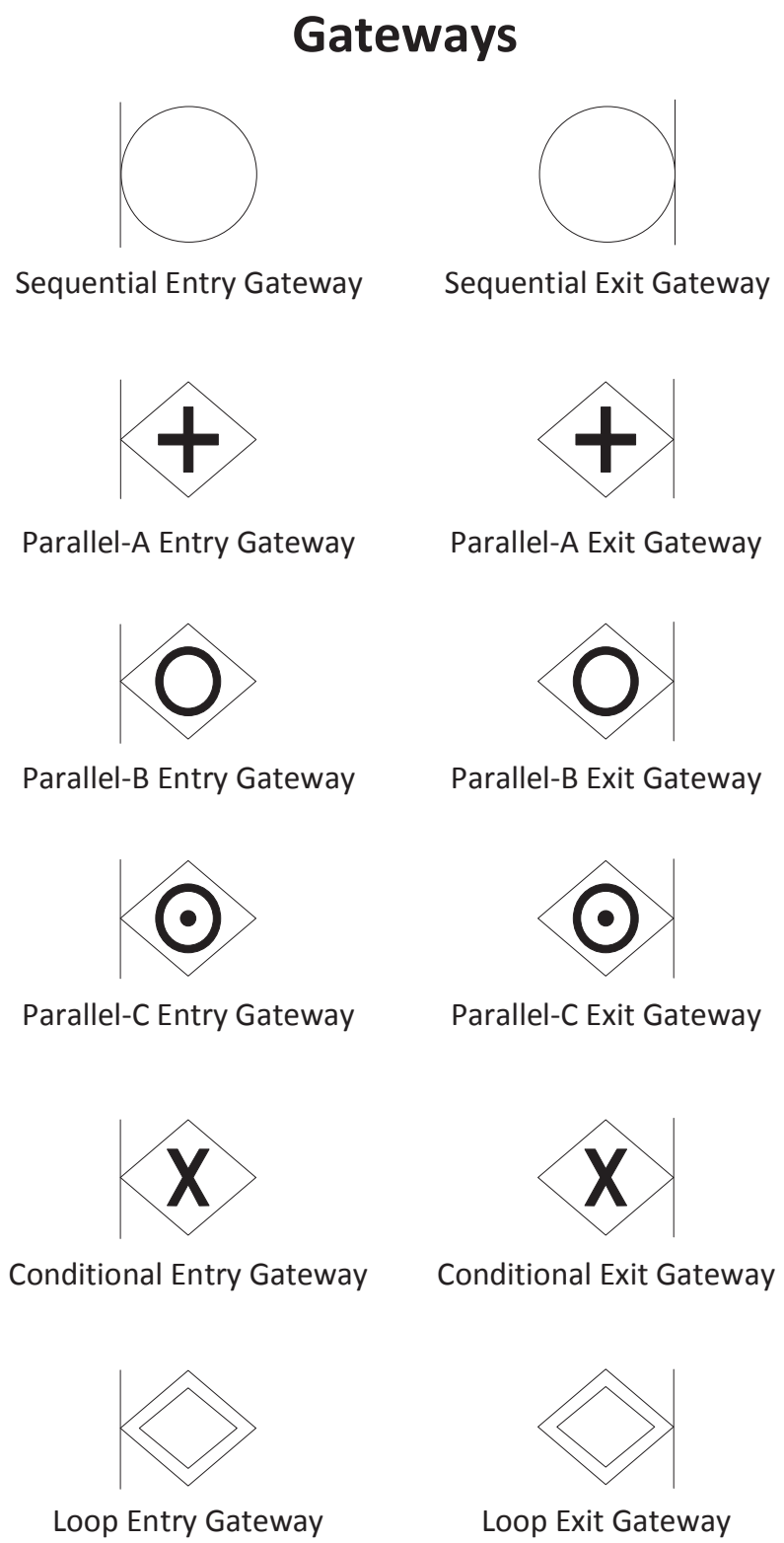


Figure 3.2: BPMN Constructs - Gateways

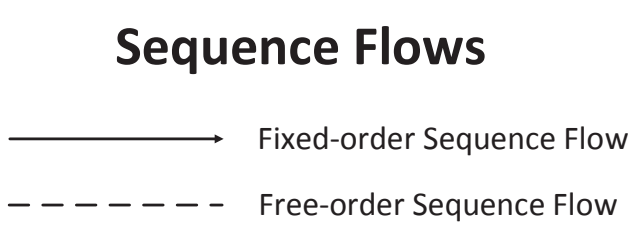


Figure 3.3: BPMN Constructs - Sequence Flows

control how the execution threads pass through sequence flows as they converge and diverge within a process or a data operation flow. We utilize six classes of symbols to annotate gateway types including **Sequential**, **Parallel-A**, **Parallel-B**, **Parallel-C**, **Conditional** and **Loop**. We also explicitly distinguish the entrance and exit of each gateway type: a gateway with a bar on its left side indicates an entrance gateway, which only allows the execution thread to enter a control flow block or a data operation block; a gateway with a bar on its right side represents an exit gateway, which only allows the execution thread to leave a control flow block or a data operation block.

A **Sequential** gateway is annotated using an empty circle; a **Parallel-A** gateway is annotated using a diamond with a bold "plus" symbol inside; a **Parallel-B** gateway is annotated using a diamond with a circle inside; a **Parallel-C** gateway is annotated using a diamond with a circle and a dot inside; a **Conditional** gateway is annotated using a diamond with a bold "X" symbol inside; a **Loop** is annotated using a diamond with a small-sized diamond inside. Note that none of the above gateways equals to the OR gateway in the original BPMN.

Figure 3.3 illustrates the sequence flows in BPMN. A sequence flow is used to show the order of flow elements in a business process or a data operation flow. Each sequence flow has only one source and only one target. There are two classes of sequence flows as **Fixed-order Sequence Flow** and **Free-order Sequence Flow**. A **Fixed-order Sequence Flow** is denoted as a solid single line with a solid arrowhead, and represents the connected two flow objects have a fixed order from sources to target; a **Free-order Sequence Flow** is denoted as a dashed line, and represents the connected two flow objects must have an order but the order is free.

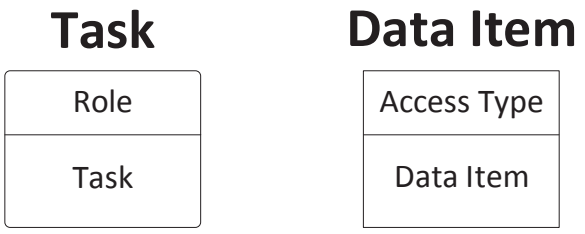


Figure 3.4: BPMN Constructs - Task and Data Item

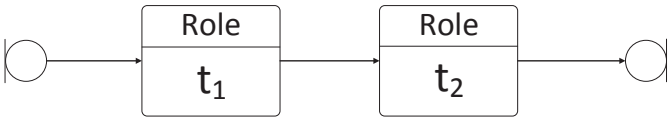


Figure 3.5: Control Flow Pattern - Strict-order Sequential

Figure 3.4 illustrates the tasks and data items in BPMN. A task, representing a logic unit of work, is denoted as a round-corner rectangle. The relation between the user roles and a task is captured by annotating the involved user roles in the upper part of the task, and the task name is annotated in the lower part of the task. A data item, representing a single piece of data operated in a task, is denoted as a rectangle. The access type on this data item is captured in the upper part of the data item, and the data item name is represented in the lower part of the data item.

3.5 Identified Control Flow Patterns

In this section, a set of control flow patterns are identified and summarized as seven classes: **Strict-order Sequential**, **Free-order Sequential**, **Parallel-A**, **Parallel-B**, **Parallel-C**, **Conditional**, and **Loop**. Each pattern describes an elementary scenario of the control flow in a role-enriched business process. The control flow patterns are the foundation to build up elementary operations of task abstraction and aggregation, which will be introduced in Chapter 4. In addition, the control flow patterns are also a language that is capable of helping process modelers to model the role-enriched business processes. In the following, each pattern is introduced in detail.

Figure 3.5 illustrates the pattern **Strict-order Sequential**, which specifies that the

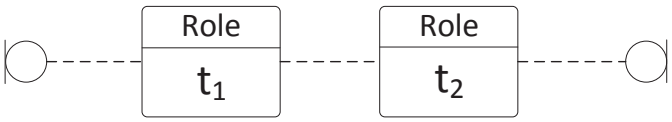


Figure 3.6: Control Flow Pattern - Free-order Sequential

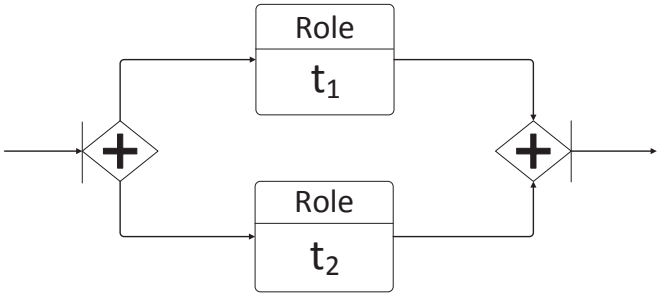


Figure 3.7: Control Flow Pattern - Parallel-A

tasks in this pattern must be executed in a strict sequential order. We use sequential entry gateway and sequential exit gateway to indicate the entering and exiting of this pattern block. And the fixed-order sequence flows are used to connect gateways and tasks. In this scenario, there exist two tasks t_1 and t_2 in the pattern. t_1 must be performed firstly, then followed by t_2. And the execution order cannot be changed.

Figure 3.6 illustrates the pattern **Free-order Sequential**, which specifies that the tasks in this pattern must be executed one after another but the order is free. We use sequential entry gateway and sequential exit gateway to indicate the entering and exiting of this pattern block, and the free-order sequence flows are used to connect gateways and tasks. In this scenario, there exist two tasks t_1 and t_2 in the pattern. t_1 and t_2 must be executed one by one, and "firstly t_1 then t_2" and "firstly t_2 then t_1" are both available.

Figure 3.7 illustrates the pattern **Parallel-A**, which specifies that all branches in this pattern must be executed concurrently and the pattern completes when all branches have completed. We use Parallel-A entry gateway and Parallel-A exit gateway to indicate the entering and exiting of this pattern block. And the fixed-order sequence flows are used to connect gateways and tasks. In this scenario, there exist two tasks t_1 and t_2 locating on two individual branches of this pattern. These two tasks are

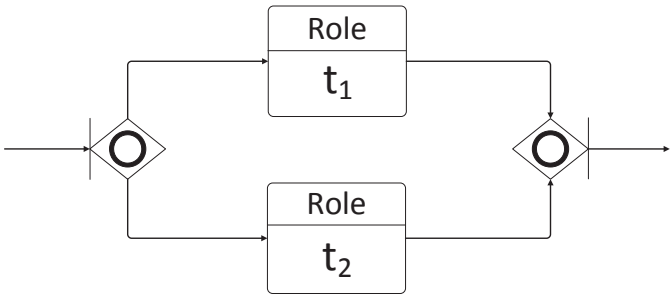


Figure 3.8: Control Flow Pattern - Parallel-B

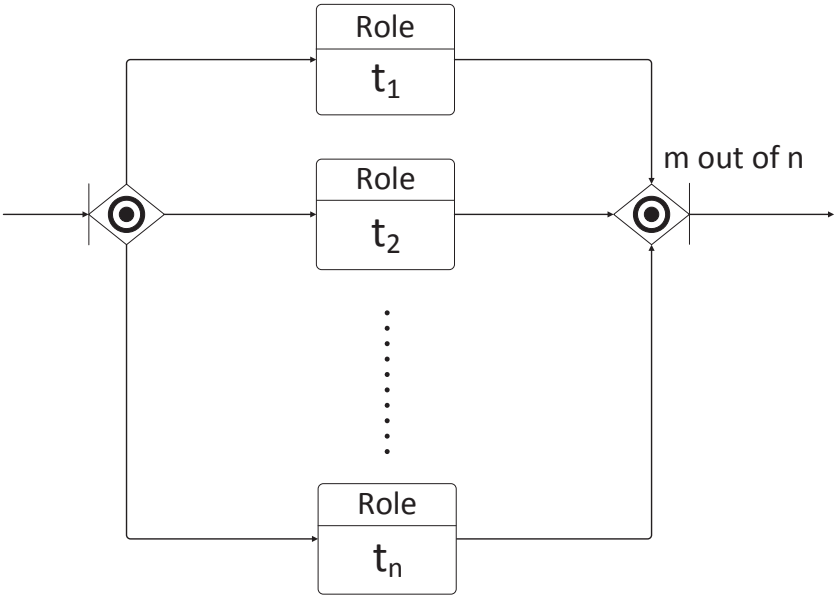


Figure 3.9: Control Flow Pattern - Parallel-C

performed in parallel. Only when t_1 and t_2 have both been completed, the pattern **Parallel-A** is completed.

Figure 3.8 illustrates the pattern **Parallel-B**, which specifies that all branches in this pattern must be executed concurrently and the pattern completes when any branch has completed. We use Parallel-B entry gateway and Parallel-B exit gateway to indicate the entering and exiting of this control flow block. And the fixed-order sequence flows are used to connect gateways and tasks. In this scenario, there exist two tasks t_1 and t_2 locating on two individual branches of this pattern. These two tasks are performed in parallel. The pattern is completed when any one of the two tasks has been completed.

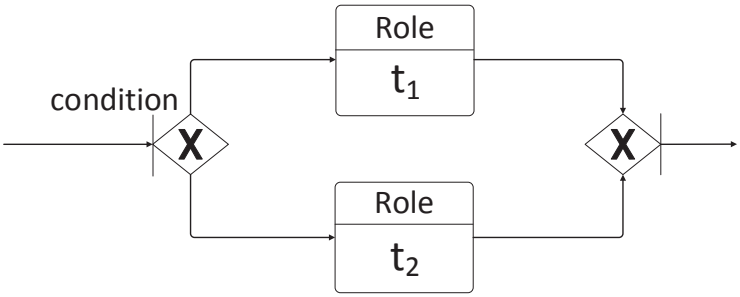


Figure 3.10: Control Flow Pattern - Conditional

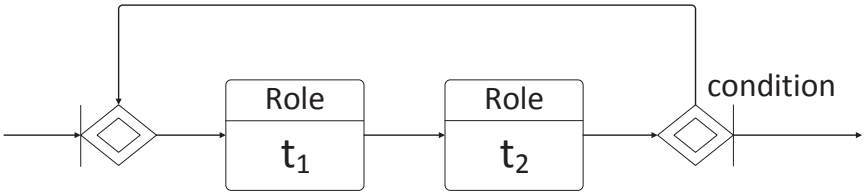


Figure 3.11: Control Flow Pattern - Loop

Figure 3.9 illustrates the pattern **Parallel-C**. It specifies that there exist n branches in this pattern, and all theses branches must be executed concurrently. The pattern completes when any m ($m \leq n$) branches from these n branches have been completed. We use Parallel-C entry gateway and Parallel-C exit gateway to indicate the entering and exiting of this control flow block. And the fixed-order sequence flows are used to connect gateways and tasks. In this scenario, there exist n tasks from t_1 to t_n locating on n branches. All the n tasks are performed in parallel. The pattern is completed when any m two tasks from these n tasks have been completed.

Figure 3.10 illustrates the pattern **Conditional**, which specifies that the executed branch in this pattern is decided according to the runtime condition. We use Conditional entry gateway and Conditional exit gateway to indicate the entering and exiting of this pattern block. The *condition* is notated at the Conditional entry gateway, and the fixed-order sequence flows are used to connect gateways and tasks. In this scenario, there exist two tasks t_1 and t_2 locating on two individual branches of this pattern. According to the *condition*, one of these two branches is selected for execution during runtime.

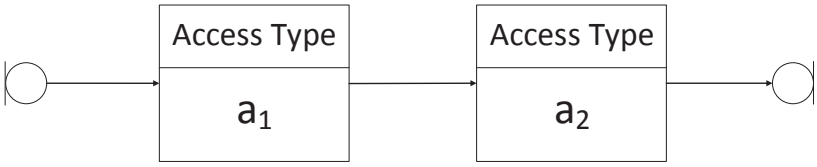


Figure 3.12: Data Operation Pattern - Strict-order Sequential

Figure 3.11 illustrates the pattern **Loop**, which specifies that all the tasks in the loop must be executed iteratively until the "jumping-out" condition is met. We use Loop entry gateway and Loop exit gateway to indicate the entering and exiting of this pattern block. The "jumping-out" *condition* is notated at the Loop Exit Gateway, and the fixed-order sequence flows are used to connect gateways and tasks. In this scenario, there exist two tasks t_1 and t_2 inside the loop. These two tasks are executed repeatedly following the order as "firstly t_1 then t_2". The execution of the pattern **Loop** is completed only when the *condition* has been met.

3.6 Identified Data Operation Patterns

In this section, a set of data operation patterns are identified and summarized as four classes: **Strict-order Sequential**, **Free-order Sequential**, **Conditional**, and **Loop**. Each pattern describes an elementary scenario of the data operations inside individual tasks of a role-enriched business process, which is different from a control flow pattern that specifies an elementary scenario of the task executions in a process. The data operation patterns are the foundation to build up elementary operations of task abstraction and aggregation, which will be introduced in Chapter 4. In addition, the data operation patterns are also a language that is capable of helping process modelers to model the data operations insides individual tasks of the role-enriched business processes. Note that in our process model, we do not consider the situation that multiple role, who perform a single task, have different authorities of the data in the task. We also do not consider the modelling of process data from different process instances.

In the following, each pattern is introduced in detail.

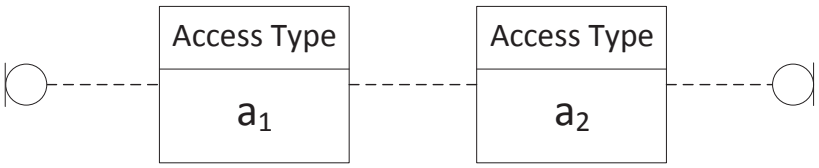


Figure 3.13: Data Operation Pattern - Free-order Sequential

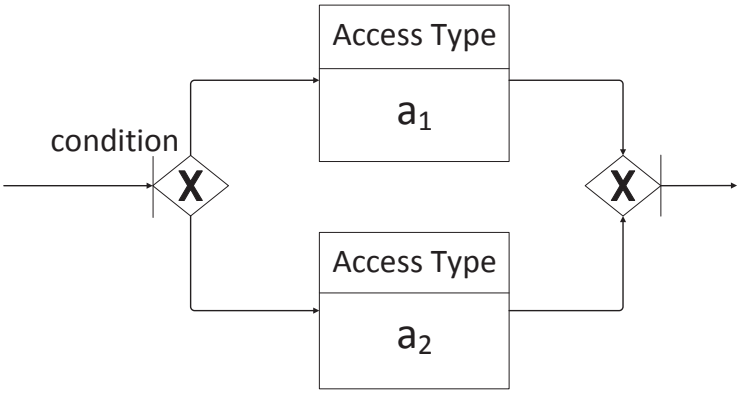


Figure 3.14: Data Operation Pattern - Conditional

Figure 3.12 illustrates the pattern **Strict-order Sequential**, which specifies that the data items in this pattern must be executed in a strict sequential order. We use sequential entry gateway and sequential exit gateway are used to indicate the entering and exiting of this pattern block. And the fixed-order sequence flows are used to connect gateways and data items. In this scenario, there exist two data ietms a_1 and a_2 in the pattern. a_1 must be performed firstly, then followed by a_2. And the execution order cannot be changed.

Figure 3.13 illustrates the pattern **Free-order Sequential**, which specifies that the data items in this pattern must be executed one after another but the order is free. We use sequential entry gateway and sequential exit gateway to indicate the entering and exiting of this pattern block, and the free-order sequence flows are used to connect gateways and data items. In this scenario, there exist two data items a_1 and a_2 in the pattern. a_1 and a_2 must be executed one by one, and "firstly a_1 then a_2" and "firstly a_2 then a_1" are both available.

Figure 3.14 illustrates the pattern **Conditional**, which specifies that the executed

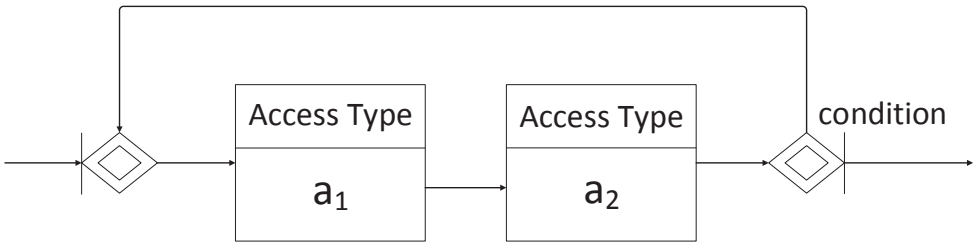
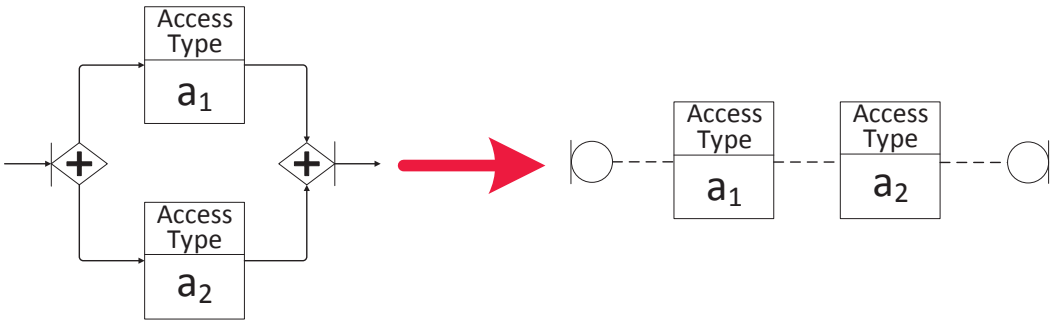


Figure 3.15: Data Operation Pattern - Loop

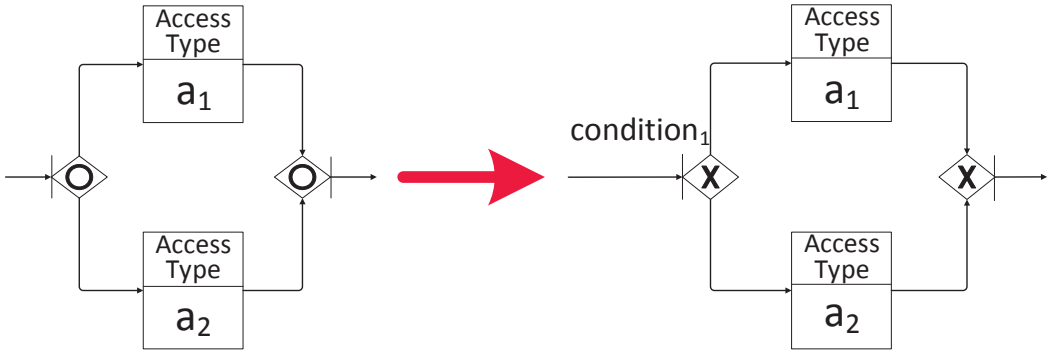
branch in this pattern is decided according to the runtime condition. We use Conditional entry gateway and Conditional exit gateway to indicate the entering and exiting of this pattern block. The *condition* is notated at the Conditional entry gateway, and the fixed-order sequence flows are used to connect gateways and data items. In this scenario, there exist two data items a_1 and a_2 locating on two individual branches of this pattern. According to the *condition*, one of these two branches is selected for execution during runtime.

Figure 3.15 illustrates the pattern **Loop**, which specifies that all the data items in the loop must be executed iteratively until the "jumping-out" condition is met. We use Loop entry gateway and Loop exit gateway to indicate the entering and exiting of this pattern block. The "jumping-out" *condition* is notated at the Loop Exit Gateway, and the fixed-order sequence flows are used to connect gateways and data items. In this scenario, there exist two data items a_1 and a_2 inside the loop. These two data items are executed repeatedly following the order as "firstly a_1 then a_2". The execution of the pattern **Loop** is completed only when the *condition* has been met.

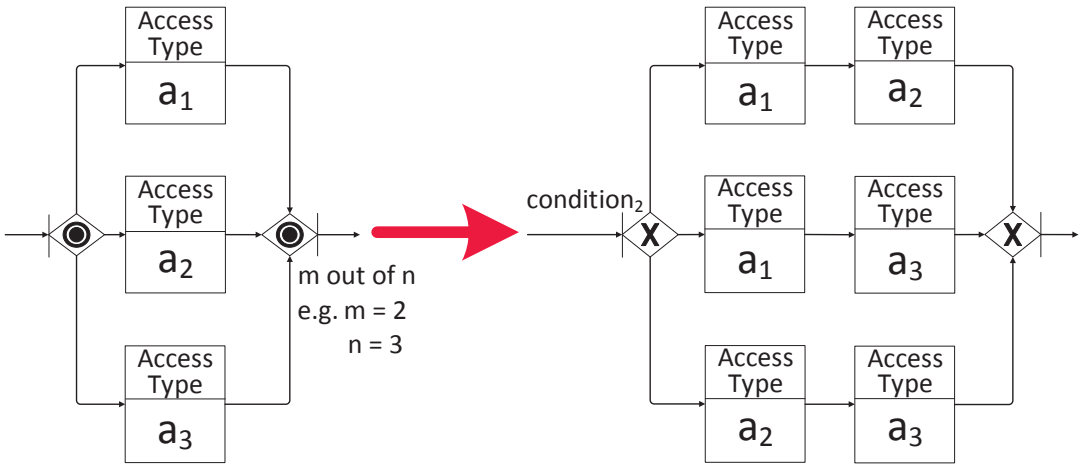
When considering the data operation pattern **Parallel-A**, each branch of this pattern must be completed by the same user role, and this user role must execute these branches one by one in a free order. (a) of Figure 3.16 illustrates this scenario. The **Parallel-A** block contains two branches, each of which has one data item as a_1 and a_2. Both a_1 and a_2 are executed by one user role (say r) due to this **Parallel-A** block locates inside one task. According to the execution rules of **Parallel-A**, r is able to execute a_1 and a_2 in a free order and one by one. This situation equals to that a_1



(a) Transformation of Parallel-A



(b) Transformation of Parallel-B



(c) Transformation of Parallel-C

Figure 3.16: Transformation of Three Data Operation Patterns: Parallel-A, Parallel-B, Parallel-C

and a_2 have the relation of **Free-order Sequential**. Therefore **Parallel-A** equals to **Free-order Sequential**.

When considering the data operation pattern **Parallel-B**, one and only one branch of this pattern must be completed by one user role. (b) of Figure 3.16 illustrates this scenario. The **Parallel-B** block contains two branches, each of which has one data item as a_1 and a_2. Both a_1 and a_2 are executed by one user role (say r) due to this **Parallel-B** block locates inside one task. According to the execution rules of **Parallel-B**, r chooses one data item from a_1 and a_2 to execute during runtime. This situation equals to that a_1 and a_2 have the relation of **Conditional**. Therefore **Parallel-B** equals to **Conditional**.

Parallel-C is similar to **Parallel-B**. In the pattern **Parallel-C**, m of n branches must be completed by one user role, and this user role must execute these m branches one by one in a free order. (c) of Figure 3.16 illustrates this scenario of **Parallel-C** containing three branches ($n = 3$) in total, each of which has one data item as a_1, a_2 and a_3. And two of these branches ($m = 2$) are required to be completed. According to the execution rules of **Parallel-C**, the user role must execute any two and only two data items from a_1, a_2 and a_3. There are three alternatives to realize this goal: (1) a_1 and a_2 are selected for execution; (2) a_1 and a_3 are selected for execution; (3) a_2 and a_3 are selected for execution. And these three alternatives become three branches of a **Conditional** pattern. Therefore, **Parallel-C** equals to **Conditional**.

To sum up, **Parallel-A**, **Parallel-B** or **Parallel-C** will never turn up in the data operation patterns.

3.7 Scenario Example

In this section, we use the scenario example introduced in Figure 1.4 of Section 1.2 to demonstrate how a business process is specified using our proposed role-enriched BP model.

Figure 3.17 shows the recruitment process represented with BPMN 2.0. Figure 3.18 illustrates the recruitment process specified with role-enriched business process

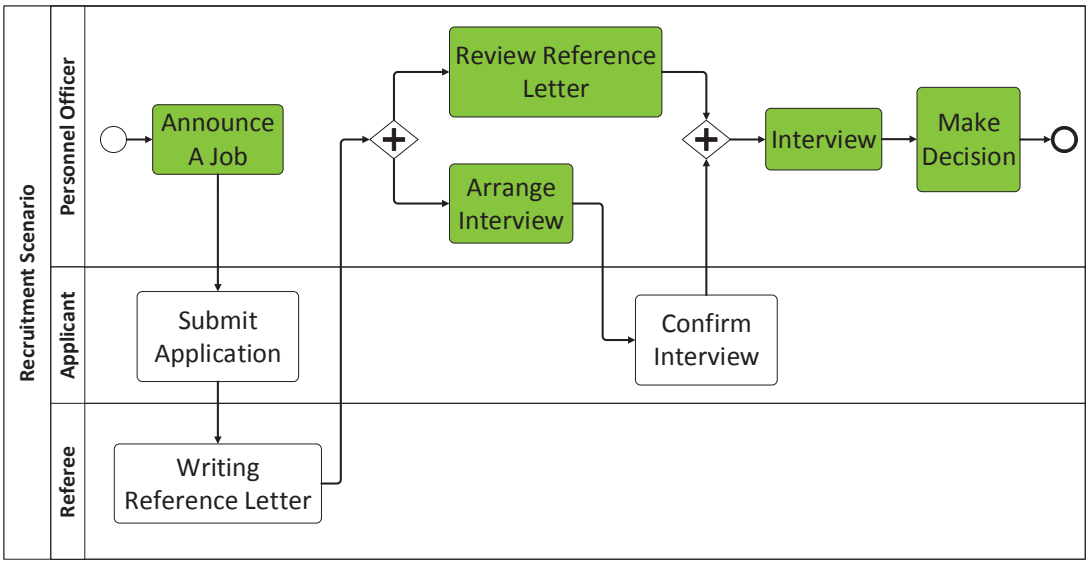


Figure 3.17: Recruitment Process Specified with Original BPMN 2.0

model. In this role-enriched business process, two aspects are explicitly expressed: the relationships between each task and its involved user role(s), the control flow relations between the tasks, and the data operations between the data items inside each task.

- In task 1, the `personnel officer` provides the data item *job description* with input.
- In task 2, the `applicant` reads information from *job description*, then offers input to *personal info* according to the information.
- In task 3, the `referee` writes an reference letter in the data item *reference letter*.
- In task 4, the `personnel officer` offers interview-related details in two data items *data&time* and *venue*. The operation order is free.
- In task 5, the `applicant` reads information of interview from *data&time* and *venue*, then confirm the his/her attendance in *conformation*.
- In task 6, the `personnel officer` firstly reads *reference letter*, then provides his/her opinions in *reference evaluation report*.

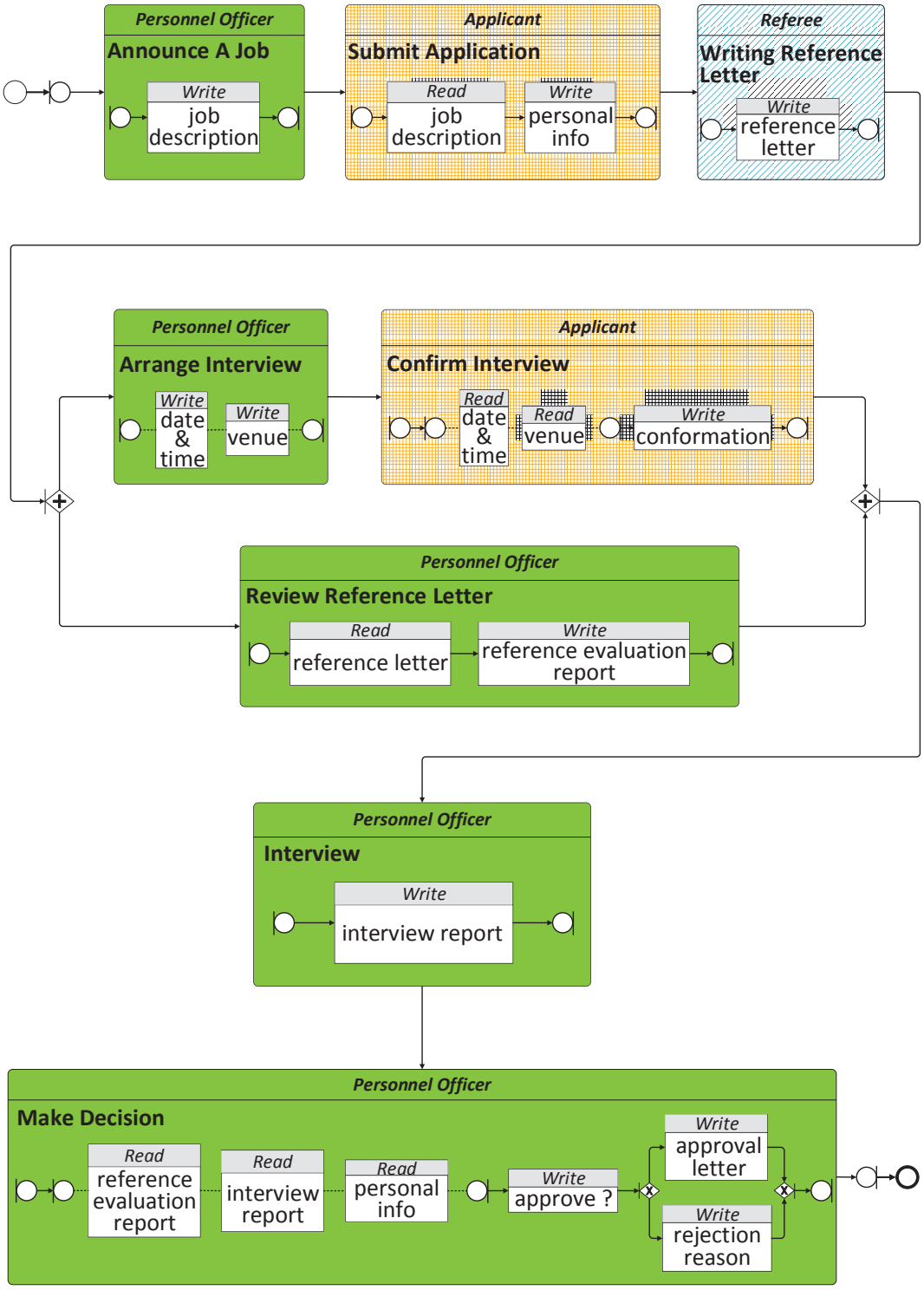


Figure 3.18: Recruitment Process Specified with Role-enriched Business Process Model

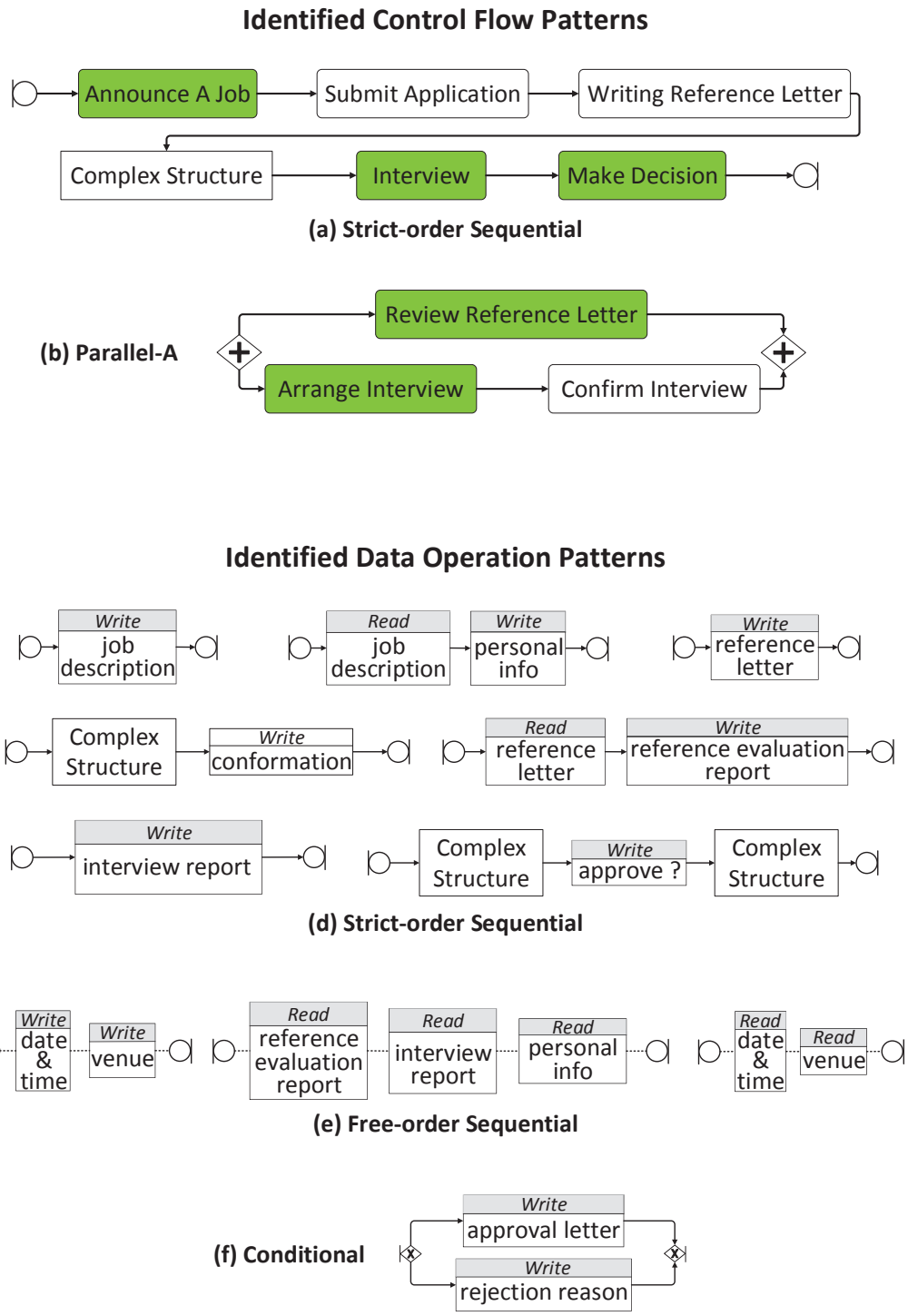


Figure 3.19: Identified Control Flow Patterns and Data Operation Patterns from Recruitment Process

- In task 7, the `personnel officer` provides interview results in *interview report*.
- In task 8, the `personnel officer` firstly evaluate three documents from *reference evaluation report*, *interview report* and *personal info*, then provides the final decision in *approve?*. If the application is approved, *approval letter* is provided with input. If the application is rejected, *rejection reason* is provided with input.

Figure 3.19 summarizes the control flow patterns and date operation patterns identified from the role-enriched BP representing the recruitment scenario. Two control flow patterns are identified as (a) **Strict-order Sequential** containing 1 block, and (b) **Parallel-A** containing 1 block; and three data operation patterns are identified as (c) **Strict-order Sequential** containing 7 blocks, (d) **Free-order Sequential** containing 3 blocks, and (e) **Conditional** containing 1 block.

3.8 Summary and Discussion

Currently, there exist two major business process modelling paradigms as artifact-centric paradigm and activity-centric paradigm. The artifact-centric paradigm [34, 70, 117, 137] treats data objects and their object life cycles as the first class modeling constructs, and the synchronization of multiple data objects is realized according to their data state changes, i.e., data state changes in different object life cycles require to be performed together. The synchronization information is stored together with the object life cycles rather than in a control unit. As a result, the ordering of tasks is not modeled explicitly. Another weakness of this paradigm is there does not exist widely-used process languages to specify this kind of process model currently. The activity-centric paradigm focuses on tasks and their execution ordering, and the control flow relations between tasks can be explicitly specified, e.g. the control flow patterns identified in [86, 89, 92]. And there exist many process modelling languages (e.g. BPMN, Petri net et al) supporting the specification of the activity-centric models. But the data operated by tasks are difficult to be specified explicitly.

After the above analysis, we have chosen activity-centric paradigm as the basis to develop our role-enriched BP model based on two reasons: this paradigm is widely supported by existing modelling languages and it is powerful in specifying process control flows. We have enriched the BP model with the relationships between tasks and involved user roles. In addition, the data operation flows within individual tasks have also been extended. A formal way has been used to define our role-enriched BP model. The well-formness of the process model is discussed. In order to specify the role-enriched BP model, we have extended the core set of constructs in the BPMN language, including task, gateways, sequence flows, and data items. We have identified seven control flow patterns and four data operation patterns as the foundation to build up the elementary operations for task abstraction and aggregation in next chapter. Lastly, we have introduced an example that demonstrates how a recruitment process is specified with our proposed role-enriched BP model.

Chapter 4

Task Abstraction and Aggregation

In this chapter, we discuss the task abstraction and aggregation, which is the first step of our UI derivation approach as shown in Figure 1.5. Section 4.1 provides an introduction of this chapter. Section 4.2 introduces the formal syntax of the abstracted and aggregated business process model. Section 4.3 discusses the approach of task abstraction and aggregation. In this section, two aspects are included as the elementary operations and algorithms. Section 4.4 conducts the structural consistency analysis in the elementary operations. Section 4.5 introduces a scenario example. Section 4.6 provides a summary and discussion on this chapter.

4.1 Introduction

A business process is a collection of linked tasks that provides services. Each task is a unit of work performed by human users or applications. It is often necessary to abstract and aggregate some details of tasks in a BP for user roles participating in the BP. The reasons of the task abstraction and aggregation fall into three categories: (1) Firstly, the details of BP tasks must be hidden and abstracted from certain users due to information security requirements such as privacy, confidentiality, and conflict of interest. (2) Secondly, task abstraction and aggregation are a foundation for deriving customized descriptions of a BP for participating users, according to the users'

requirements and intentions. The customized BP descriptions may play an important role in the modelling of BP collaboration, BP visualization, and authority control. (3) Thirdly, AABPs highlight the requirements associated with a specific user role and preserve some information of other user roles for the effective control flow in a BP. AABPs can be used to enable the development and updating of software components such as UIs related to different user roles [138, 139, 140]. As the the first step of our work of the UI derivation, the task abstraction and aggregation reserve the tasks related to a specific user role, and abstract the tasks not related to this user role.

4.2 Abstracted and Aggregated BP Model

This section provides the syntax of the AABP that is the output process after the task abstraction and aggregation. An AABP for a specific user role comprises (1) a set of tasks that are executed all by this user role, (2) a set of abstracted nodes which are generated from the tasks irrelevant to this user role, and (3) the control flow relations between the tasks and the abstracted nodes.

Definition 5: Abstracted and Aggregated BP Model. Given a user role $r \in R$, an abstracted and aggregated BP model for the user role r is denoted as $rm^r = (N^r, type^r, SF^r_{fix}, SF^r_{free}, refine^r)$, where:

- $N^r = \{e^r_s, e^r_e\} \cup G^r \cup T^r \cup ABS^r$ where e^r_s, e^r_e indicate start event, and end event respectively. $G^r = G^r_{in} \cup G^r_{out}$ is a finite set of gateways. G^{in}_d is a finite set of entry gateways and G^{out}_d is a finite set of exit gateways. The entry and exit gateways are used to control that the execution thread enters and leaves a structural block of the data operation flow. T^r is a finite set of tasks which are participated by user role r. ABS^r is a finite set of abstracted nodes, each of which is abstracted from one or multiple tasks of the role-enriched BP, and these tasks are not participated by r.

- $type^r$: $G^r \rightarrow$ {*Sequential*, *Parallel-A*, *Parallel-B*, *Parallel-C*, *Conditional*, *Loop*} is a mapping function to give each gateway a type.

- SF^r_{fix} and SF^r_{free} represent fixed-order sequence flow and free-order sequence flow respectively. The fixed-order sequence flow means that the sequence of involved tasks must be in a fixed order; the free-order sequence flow means that the order in the sequence of involved tasks is free.

- $refine^r : T^r \rightarrow DF^r$ is a refinement function on tasks. $DF^r = \{df^r_1, df^r_2, ..., df^r_n\}$ stands for a finite set of data operation flows operated by user role r.

4.3 Task Abstraction and Aggregation

This section introduces the abstraction and aggregation of tasks in the role-enriched BP. An AABP is derived for each user role involved in the role-enriched BP. To derive the AABP, a series of elementary operations are specified. The algorithm for task abstraction and aggregation is developed by utilizing these elementary operations.

4.3.1 Elementary Operations

In this section, 18 elementary operations for task abstraction and aggregation are specified. Each elementary operation abstracts and aggregates tasks and their control flow relations from a particular control flow pattern. Here we assume that the abstraction and aggregation are for user role r_1. The tasks related to r_1 are kept; the tasks not participated by r_1 are abstracted and aggregated and become abstracted nodes. In the following, the control flow patterns associated with elementary operations except **Single-Abs-Agg-1**, **Single-Abs-Agg-2**, **Single-Abs-Agg-3** are called **Basic BP Fragments**.

Single-Abs-Agg-1 in Figure 4.1 shows how an elementary operation realize the task abstraction and aggregation from a single task ($Task_A$) not involving r_1. In this situation, this task ($Task_A$) is abstracted as a single abstracted node (ABS_A).

Single-Abs-Agg-2 in Figure 4.1 shows how an elementary operation realize the task abstraction and aggregation from a single task ($Task_A$). And the task ($Task_A$) can be carried out by either r_1 or r_2, which is denoted by $r_1 \veebar r_2$. In this situation, the task ($Task_A$) is abstracted as two nodes: (1) a task ($Task_A$) that is performed only

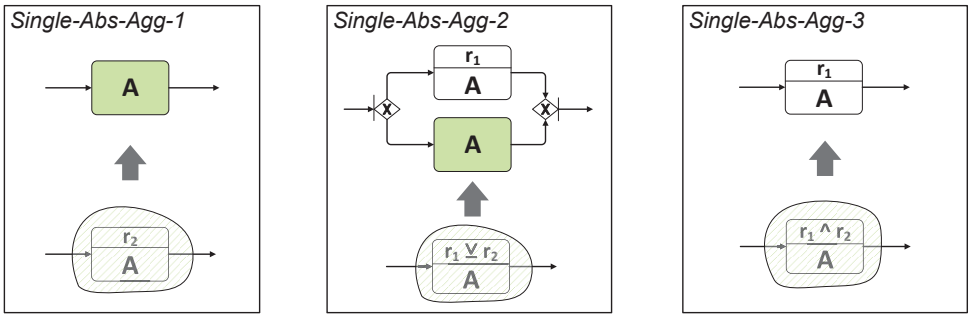


Figure 4.1: Elementary Operations Single-Abs-Agg-1,2,3 on Single Tasks

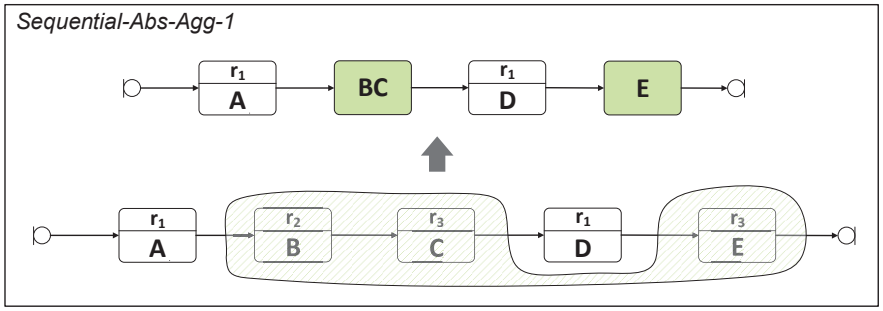


Figure 4.2: Elementary Operation Sequential-Abs-Agg-1 on Strict-order Sequential

by r_1 and (2) an abstracted node (ABS_A). These two nodes have the **Conditional** control flow relation.

Single-Abs-Agg-3 in Figure 4.1 shows how an elementary operation realize the task abstraction and aggregation from a single task ($Task_A$). And the task ($Task_A$) must be participated by both r_1 and r_2, which is denoted by $r_1 \wedge r_2$. In this situation, the task ($Task_A$) is abstracted as a new task ($Task_A$) that is only participated by r_1.

Sequential-Abs-Agg-1 in Figure 4.1 shows how an elementary operation realize the task abstraction and aggregation from a **Strict-order Sequential** control flow pattern. In this pattern, if the tasks ($Task_B$ and $Task_C$) not involving r_1 are adjacent, these tasks ($Task_B$ and $Task_C$) are abstracted as a single abstracted node (ABS_{BC}); if the tasks ($Task_E$) not involving r_1 are not adjacent, each of these tasks ($Task_E$) is abstracted as an individual abstracted node (ABS_E).

Sequential-Abs-Agg-2 in Figure 4.3 shows how an elementary operation realize the task abstraction and aggregation from a **Free-order Sequential** control flow

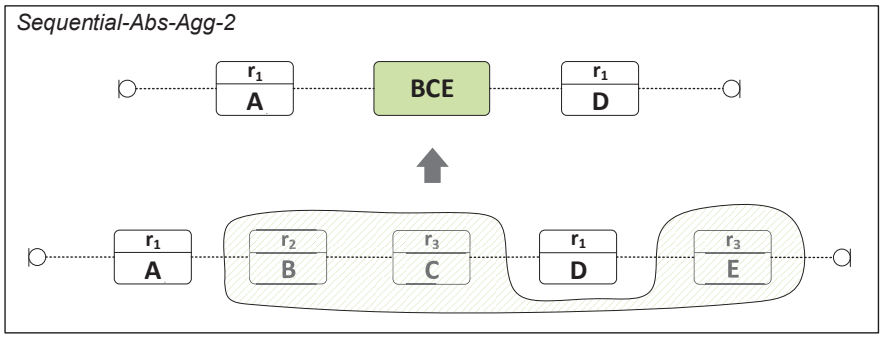


Figure 4.3: Elementary Operation Sequential-Abs-Agg-2 on Free-order Sequential

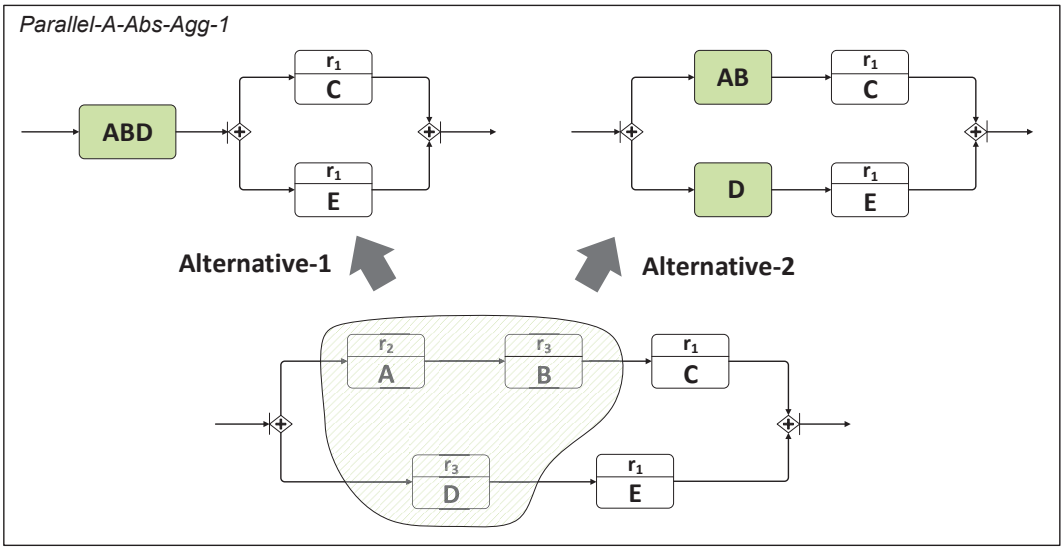


Figure 4.4: Elementary Operation Parallel-A-Abs-Agg-1 on Parallel-A

pattern. In this pattern, all the tasks ($Task_B$, $Task_C$, $Task_E$) not involving r_1 are abstracted as one single abstracted node (ABS_{BCE}).

Parallel-A-Abs-Agg-1 in Figure 4.4 shows how an elementary operation realize the task abstraction and aggregation from a **Parallel-A** control flow pattern. Each branch of the **Parallel-A** control flow pattern contains both tasks participated by r_1 ($Task_C$ on the upper branch, $Task_E$ on the lower branch) and adjacent tasks not involving r_1 ($Task_A$ and $Task_B$ on the upper branch, $Task_D$ on the lower branch); the **Parallel-A** entry gateway is adjacent to these adjacent tasks not involving r_1. Two alternatives are provided to deal with this situation in **Parallel-A**: (1) *Alternative 1*

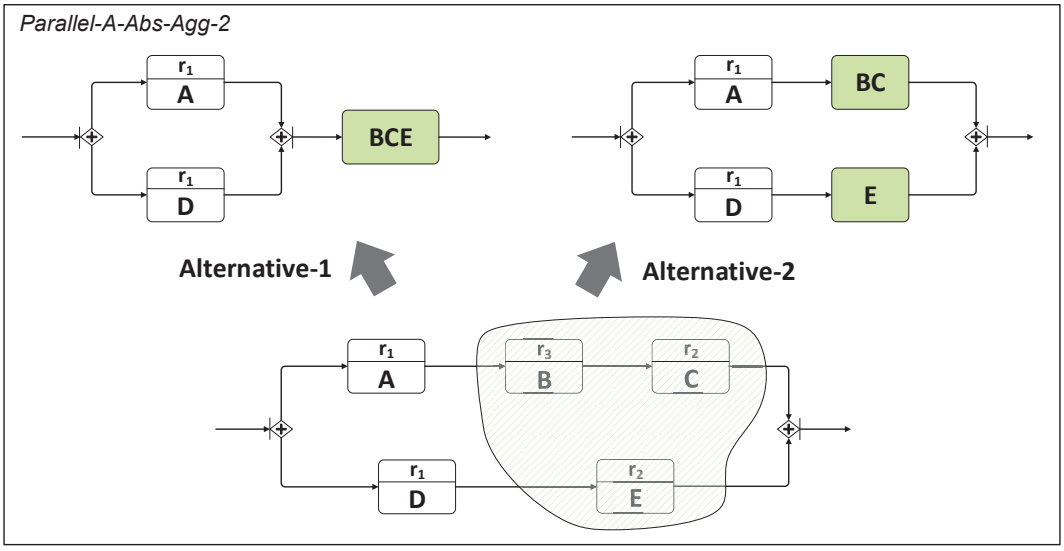


Figure 4.5: Elementary Operation Parallel-A-Abs-Agg-2 on Parallel-A

abstracts the adjacent tasks not involving r_1 from all branches ($Task_A$ and $Task_B$ on the upper branch, $Task_D$ on the lower branch) as a single abstracted node (ABS_{ABD}). This abstracted node is shifted out of the **Parallel-A** pattern and adjacent to the **Parallel-A** entry gateway. Differently, (2) *Alternative 2* abstracts the adjacent tasks not involving r_1 on each branch as single abstracted nodes ($Task_A$ and $Task_B$ on the upper branch as ABS_{AB}, $Task_D$ on the lower branch as ABS_D). And each of these abstracted nodes remains on the original branch (ABS_{AB} on the the upper branch, ABS_D on the lower branch). Both *Alternative 1* and *Alternative 2* can realize the same goal of task abstraction and aggregation. *Alternative 1* realize this goal by providing an option of structure changing. But the derived UI logic will not be affected.

Parallel-A-Abs-Agg-2 in Figure 4.5 shows how an elementary operation realize the task abstraction and aggregation from a **Parallel-A** control flow pattern. Each branch of the **Parallel-A** control flow pattern contains both tasks participated by r_1 ($Task_A$ on the upper branch, $Task_D$ on the lower branch) and adjacent tasks not involving r_1 ($Task_B$ and $Task_C$ on the upper branch, $Task_E$ on the lower branch); the **Parallel-A** exit gateway is adjacent to these adjacent tasks not involving r_1. Two alternatives are provided to deal with this situation in **Parallel-A**: (1) *Alternative 1*

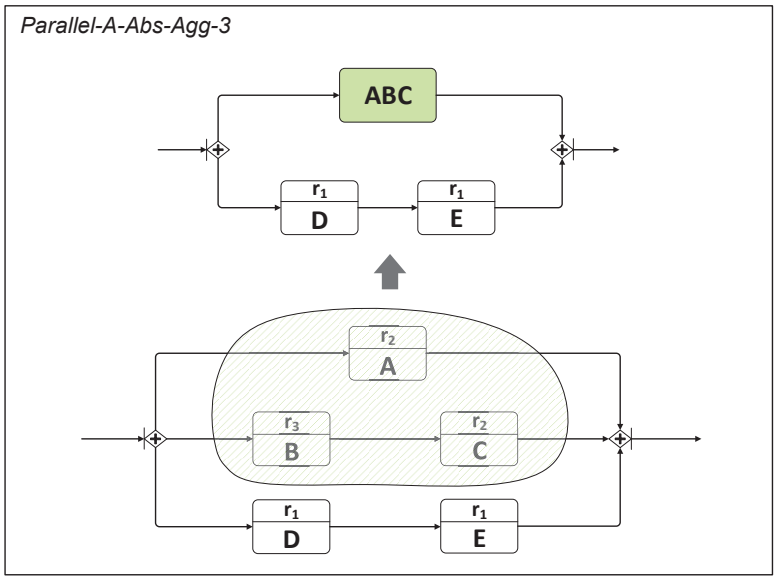


Figure 4.6: Elementary Operation Parallel-A-Abs-Agg-3 on Parallel-A

abstracts the adjacent tasks not involving r_1 from all branches ($Task_B$ and $Task_C$ on the upper branch, $Task_E$ on the lower branch) as a single abstracted node (ABS_{BCE}). This abstracted node is shifted out of the **Parallel-A** pattern and adjacent to the **Parallel-A** exit gateway. Differently, (2) *Alternative 2* abstracts the adjacent tasks not involving r_1 on each branch as single abstracted nodes ($Task_B$ and $Task_C$ on the upper branch as ABS_{BC}, $Task_E$ on the lower branch as ABS_E). And each of these abstracted nodes remains on the original branch (ABS_{BC} on the the upper branch, ABS_E on the lower branch). Both *Alternative 1* and *Alternative 2* can realize the same goal of task abstraction and aggregation. *Alternative 1* realize this goal by providing an option of structure changing. But the derived UI logic will not be affected.

Parallel-A-Abs-Agg-3 in Figure 4.6 shows how an elementary operation realize the task abstraction and aggregation from a **Parallel-A** control flow pattern containing three or more branches. In the **Parallel-A** control flow pattern, there exist at least two branches, each of which only contains tasks not participated by r_1 ($Task_A$ on the upper branch, $Task_B$ and $Task_C$ on the middle branch). All these branches (the upper branch containing $Task_A$, the middle branch containing $Task_B$ and $Task_C$) are abstracted as one single abstracted node (ABS_{ABC}). And this abstracted node (ABS_{ABC}) forms an

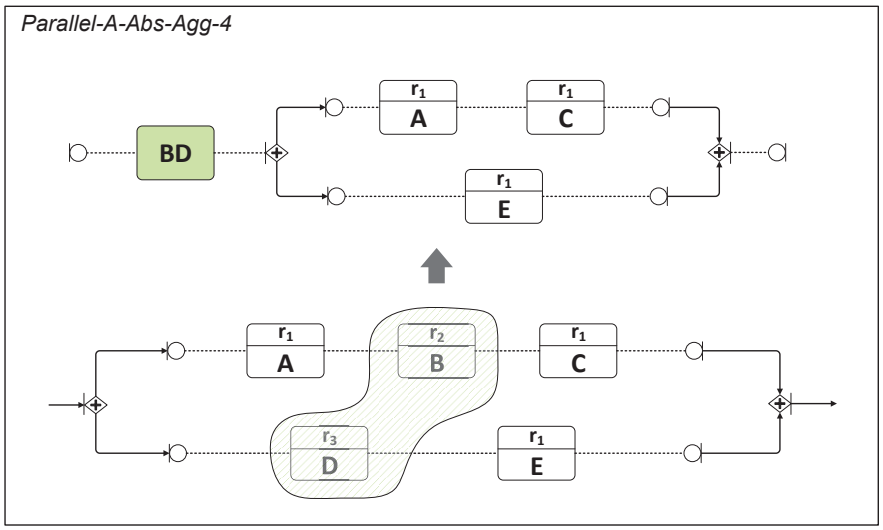


Figure 4.7: Elementary Operation Parallel-A-Abs-Agg-4 on Parallel-A

independent branch of the **Parallel-A** control flow pattern.

Parallel-A-Abs-Agg-4 in Figure 4.7 shows how an elementary operation realize the task abstraction and aggregation from a **Parallel-A** control flow pattern, each branch of which has a **Free-order Sequential** control flow pattern. On each branch of the **Parallel-A** control flow pattern, there exists at least one task not participated by r_1 ($Task_B$ on the upper branch, $Task_D$ on the lower branch). All these tasks ($Task_B$ and $Task_D$) are abstracted as one single abstracted node (ABS_{ABC}). This abstracted node (ABS_{ABC}) is shifted out of the **Parallel-A** pattern and has the **Free-order Sequential** control flow relation with the **Parallel-A** pattern.

Parallel-B-Abs-Agg-1 in Figure 4.8 shows how an elementary operation realizes the task abstraction and aggregation from a **Parallel-B** control flow pattern. Each branch of the **Parallel-B** control flow pattern contains both tasks participated by r_1 ($Task_B$ on the upper branch, $Task_E$ on the lower branch) and adjacent tasks not involving r_1 ($Task_A$ and on the upper branch, $Task_C$ and $Task_D$ on the lower branch). The adjacent tasks not involving r_1 on each branch are abstracted as a single abstracted node ($Task_C$ and $Task_D$ on the upper branch as ABS_{CD}); the non-adjacent tasks not involving r_1 on each branch are abstracted separately as abstracted nodes ($Task_A$ on

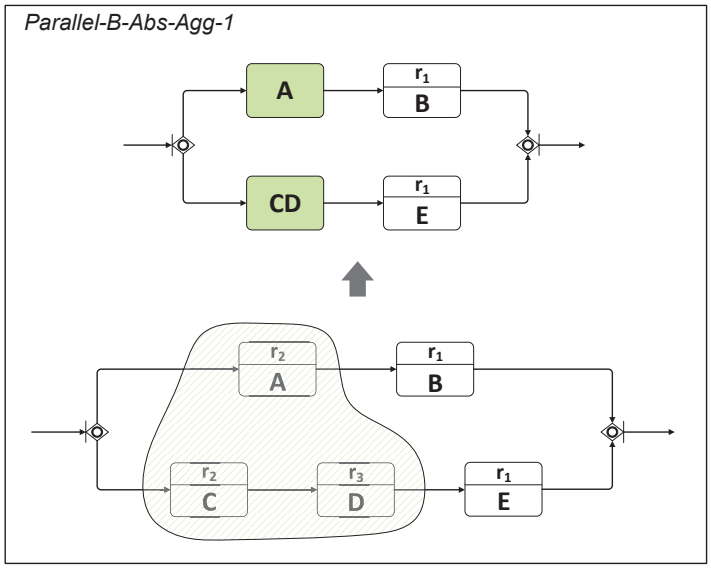


Figure 4.8: Elementary Operation Parallel-B-Abs-Agg-1 on Parallel-B

the upper branch as ABS_A). Each of these abstracted nodes remains on the original branch (ABS_A on the the upper branch, ABS_{CD} on the lower branch).

Parallel-B-Abs-Agg-2 in Figure 4.9 shows how an elementary operation realizes the task abstraction and aggregation from a **Parallel-B** control flow pattern containing three or more branches. In the **Parallel-B** control flow pattern, there exist at least two branches, each of which only contains tasks not participated by r_1 ($Task_C$ on the middle branch, $Task_D$ and $Task_E$ on the lower branch). All these branches (the middle branch containing $Task_C$, the lower branch containing $Task_D$ and $Task_E$) are abstracted as one single abstracted node (ABS_{CDE}). And this abstracted node (ABS_{CDE}) forms an independent branch of the **Parallel-B** control flow pattern.

Parallel-B-Abs-Agg-3 in Figure 4.10 shows how an elementary operation realizes the task abstraction and aggregation from a **Parallel-B** control flow pattern, each branch of which has a **Free-order Sequential** control flow pattern. On each branch of the **Parallel-A** control flow pattern, the tasks not participated by r_1 are abstracted as single abstracted nodes ($Task_A$ as ABS_A, $Task_C$ and $Task_D$ as ABS_{CD}). And each of these abstracted nodes (ABS_A, ABS_{CD}) remains on the original branch.

Parallel-C-Abs-Agg-1 in Figure 4.11 shows how an elementary operation realizes

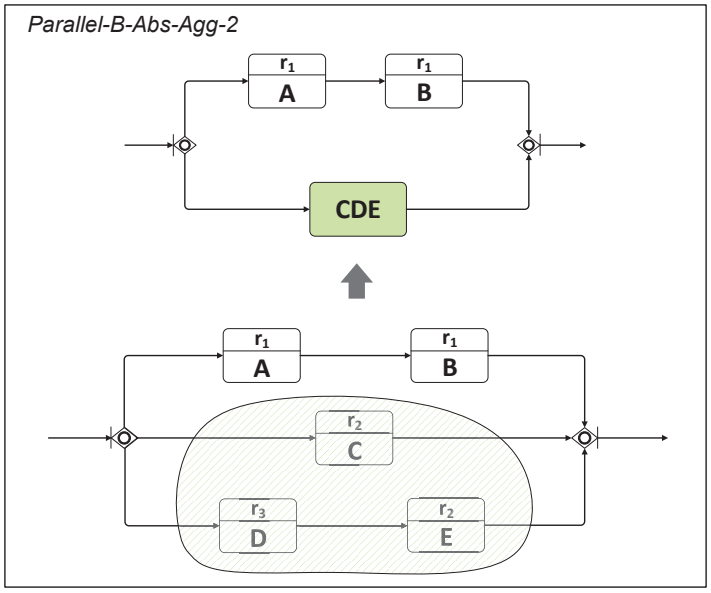


Figure 4.9: Elementary Operation Parallel-B-Abs-Agg-2 on Parallel-B

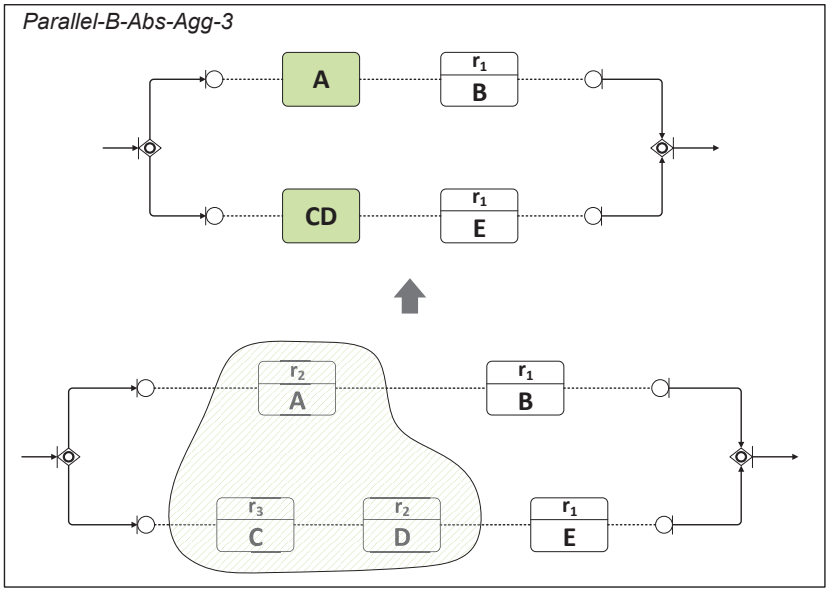


Figure 4.10: Elementary Operation Parallel-B-Abs-Agg-3 on Parallel-B

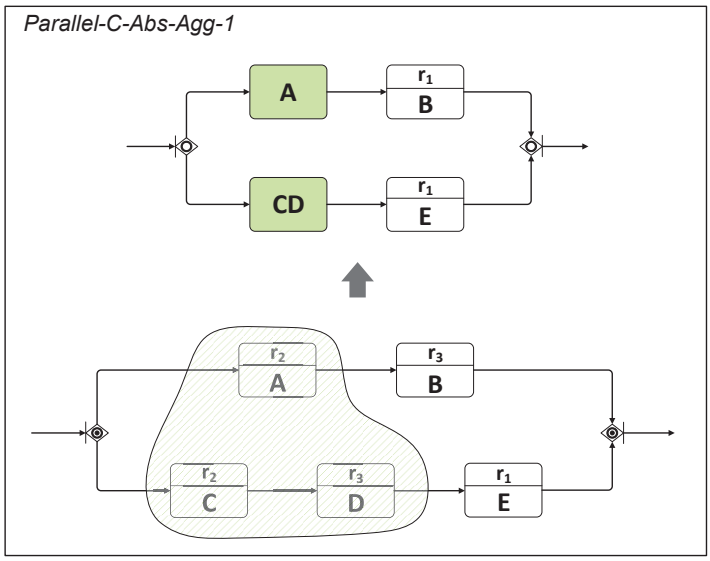


Figure 4.11: Elementary Operation Parallel-C-Abs-Agg-1 on Parallel-C

the task abstraction and aggregation from a **Parallel-C** control flow pattern. Each branch of the **Parallel-C** control flow pattern contains both tasks participated by r_1 ($Task_B$ on the upper branch, $Task_E$ on the lower branch) and adjacent tasks not involving r_1 ($Task_A$ and on the upper branch, $Task_C$ and $Task_D$ on the lower branch). The adjacent tasks not involving r_1 on each branch are abstracted as a single abstracted node ($Task_C$ and $Task_D$ on the upper branch as ABS_{CD}); the non-adjacent tasks not involving r_1 on each branch are abstracted separately as abstracted nodes ($Task_A$ on the upper branch as ABS_A). Each of these abstracted nodes remains on the original branch (ABS_A on the the upper branch, ABS_{CD} on the lower branch).

Parallel-C-Abs-Agg-2 in Figure 4.12 shows how an elementary operation realizes the task abstraction and aggregation from a **Parallel-C** control flow pattern containing three or more branches. In the **Parallel-C** control flow pattern, there exist at least two branches, each of which only contains tasks not participated by r_1 ($Task_B$ and $Task_C$ on the middle branch, $Task_D$ and $Task_E$ on the lower branch). All these branches (the middle branch containing $Task_B$ and $Task_C$, the lower branch containing $Task_D$ and $Task_E$) are abstracted as one single abstracted node (ABS_{BCDE}). And this abstracted node (ABS_{BCDE}) forms an independent branch of the **Parallel-C** control flow pattern.

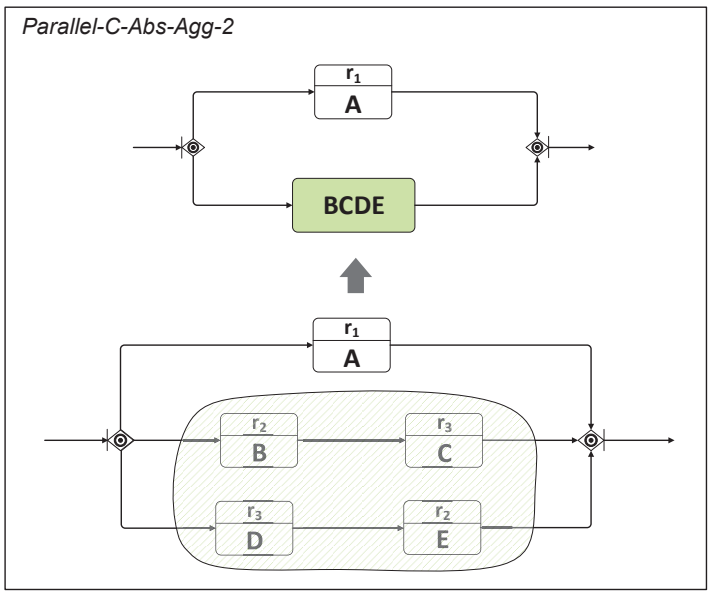


Figure 4.12: Elementary Operation Parallel-C-Abs-Agg-2 on Parallel-C

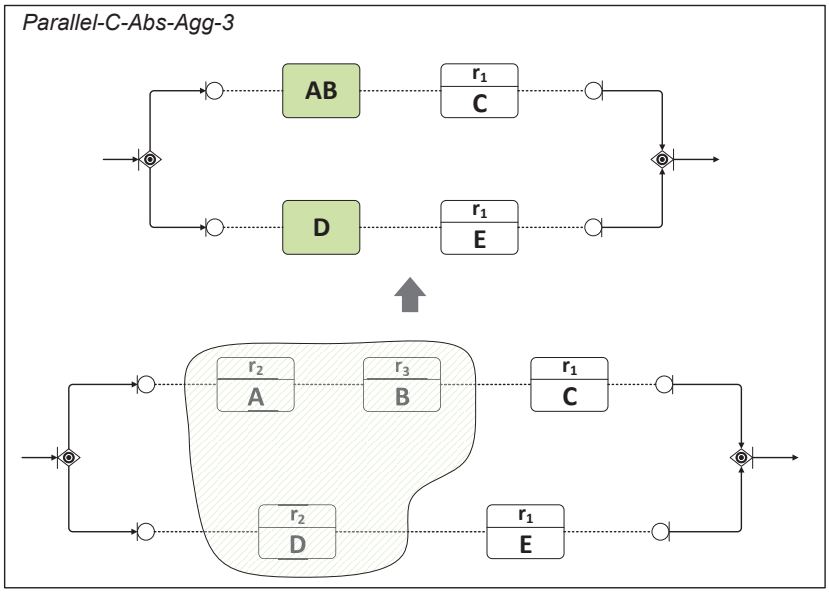


Figure 4.13: Elementary Operation Parallel-C-Abs-Agg-3 on Parallel-C

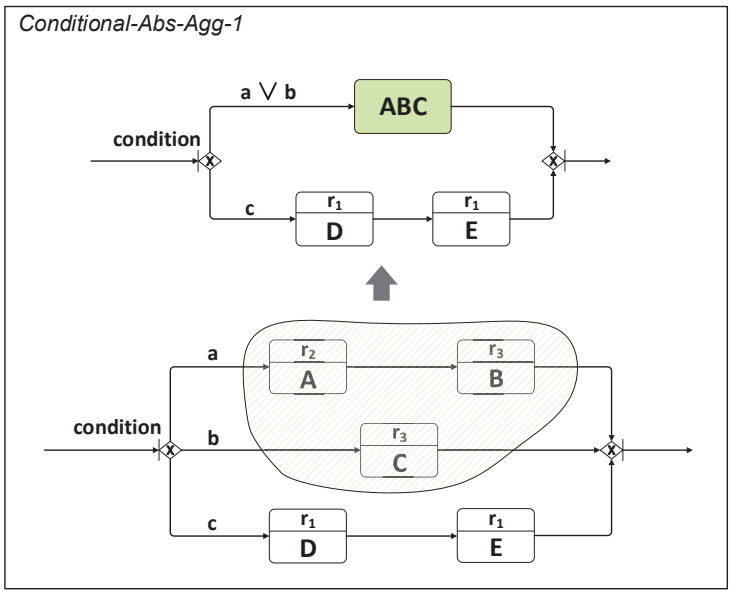


Figure 4.14: Elementary Operation Conditional-Abs-Agg-1 on Conditional

Parallel-C-Abs-Agg-3 in Figure 4.13 shows how an elementary operation realizes the task abstraction and aggregation from a **Parallel-C** control flow pattern, each branch of which has a **Free-order Sequential** control flow pattern. On each branch of the **Parallel-A** control flow pattern, the tasks not participated by r_1 are abstracted as single abstracted nodes ($Task_A$ and $Task_B$ as ABS_{AB}, $Task_D$ as ABS_D). And each of these abstracted nodes (ABS_{AB}, ABS_D) remains on the original branch.

Conditional-Abs-Agg-1 in Figure 4.14 shows how an elementary operation realizes the task abstraction and aggregation from a **Conditional** control flow pattern containing three or more branches. In the **Condtional** control flow pattern, there exist at least two branches, each of which only contains tasks not participated by r_1 ($Task_A$ and $Task_B$ on the upper branch, $Task_C$ on the middle branch). All these branches (the upper branch containing $Task_A$ and $Task_B$, the middle branch containing $Task_C$) are abstracted as one single abstracted node (ABS_{ABC}). And this abstracted node (ABS_{ABC}) forms an independent branch of the **Conditional** control flow pattern. The condition ($a \cup b$) on the branch of the abstracted node (ABS_{ABC}) is the union of the conditions from the two abstracted branches (a from the upper branch, b from the

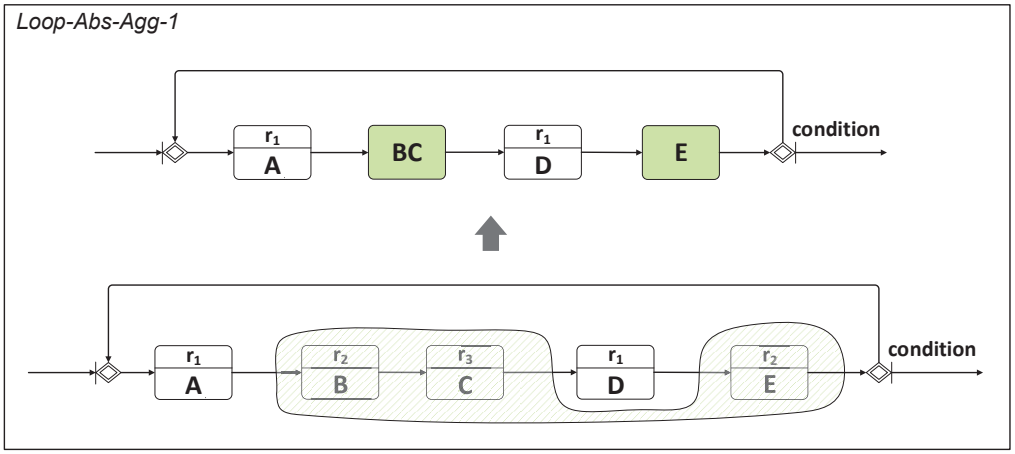


Figure 4.15: Elementary Operation Loop-Abs-Agg-1 on Loop

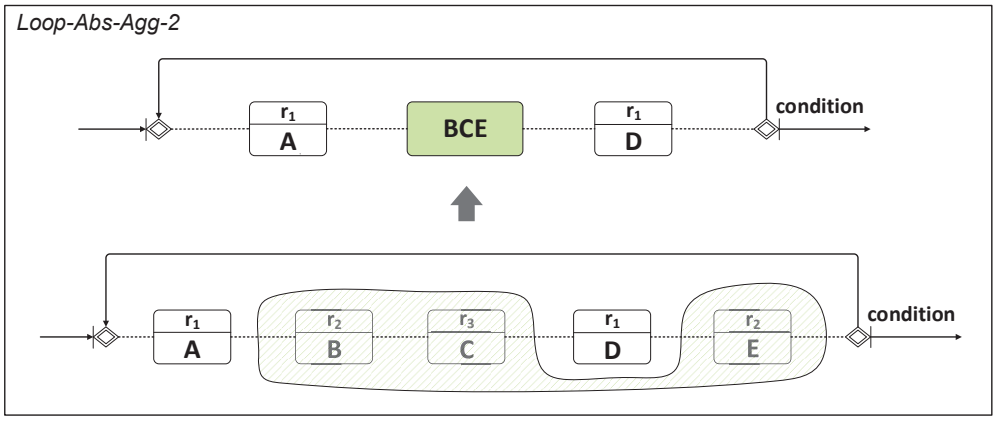


Figure 4.16: Elementary Operation Loop-Abs-Agg-2 on Loop

middle branch).

Loop-Abs-Agg-1 in Figure 4.15 shows how an elementary operation realizes the task abstraction and aggregation from a **Strict-order Loop** control flow pattern. In this pattern, if the tasks ($Task_B$ and $Task_C$) not involving r_1 are adjacent, these tasks ($Task_B$ and $Task_C$) are abstracted as a single abstracted node (ABS_{BC}); if the tasks ($Task_E$) not involving r_1 are not adjacent, each of these tasks ($Task_E$) is abstracted as an individual abstracted node (ABS_E).

Loop-Abs-Agg-2 in Figure 4.16 shows how an elementary operation realizes the task abstraction and aggregation from a **Free-order Sequential** control flow pattern.

In this pattern, all the tasks ($Task_B$, $Task_C$, $Task_E$) not involving r_1 are abstracted as one single abstracted node (ABS_{BCE}).

4.3.2 Algorithm for Task Abstraction and Aggregation

This subsection introduces the algorithms for task abstraction and aggregation, which calls the elementary operations introduced in Subsection 4.3.1. In the algorithms, the tasks related to a particular user role will be reserved in the output abstracted and aggregated BP, while the tasks irrelevant to this user role will be abstracted and aggregated into abstracted nodes in the output abstracted and aggregated BP. In order to realize this goal, we proceed recursively to detect and handle the elements on each granularity level of the role-enriched BP. In this procedure, the recursive function `BPAbsAgg` in **Algorithm 3** is the working horse of this algorithm. The control flow patterns at non-finest granularity level of the role-enriched BP belong to **Complex BP Fragments** and they are abstracted/aggregated by the function `AbsComplexFrag`. The control flow patterns at the finest granularity level of the role-enriched BP belong to **Basic BP Fragments** and they are abstracted/aggregated by applying the function `AbsBasicFrag`.

A **Basic BP Fragment** in a role-enriched BP must satisfy one of the following conditions:

- **if** the control flow pattern of the fragment is **Strict-order Sequential** or **Free-order Sequential**, **then** all the elements of this fragment must only be individual tasks. (For example in Figure 4.17, (a) and (b) are Basic BP Fragments, while (g) is a Complex BP Fragment.)

- **if** the control flow pattern of the fragment is **Parallel-A**, **Parallel-B**, or **Parallel-C**, **then** the control flow pattern of each branch of this fragment must only be **Strict-order Sequential** or **Free-order Sequential**. (For example in Figure 4.17, (c) are Basic BP Fragments, while (h) is a Complex BP Fragment.)

- **if** the control flow pattern of the fragment is **conditional**, **then**

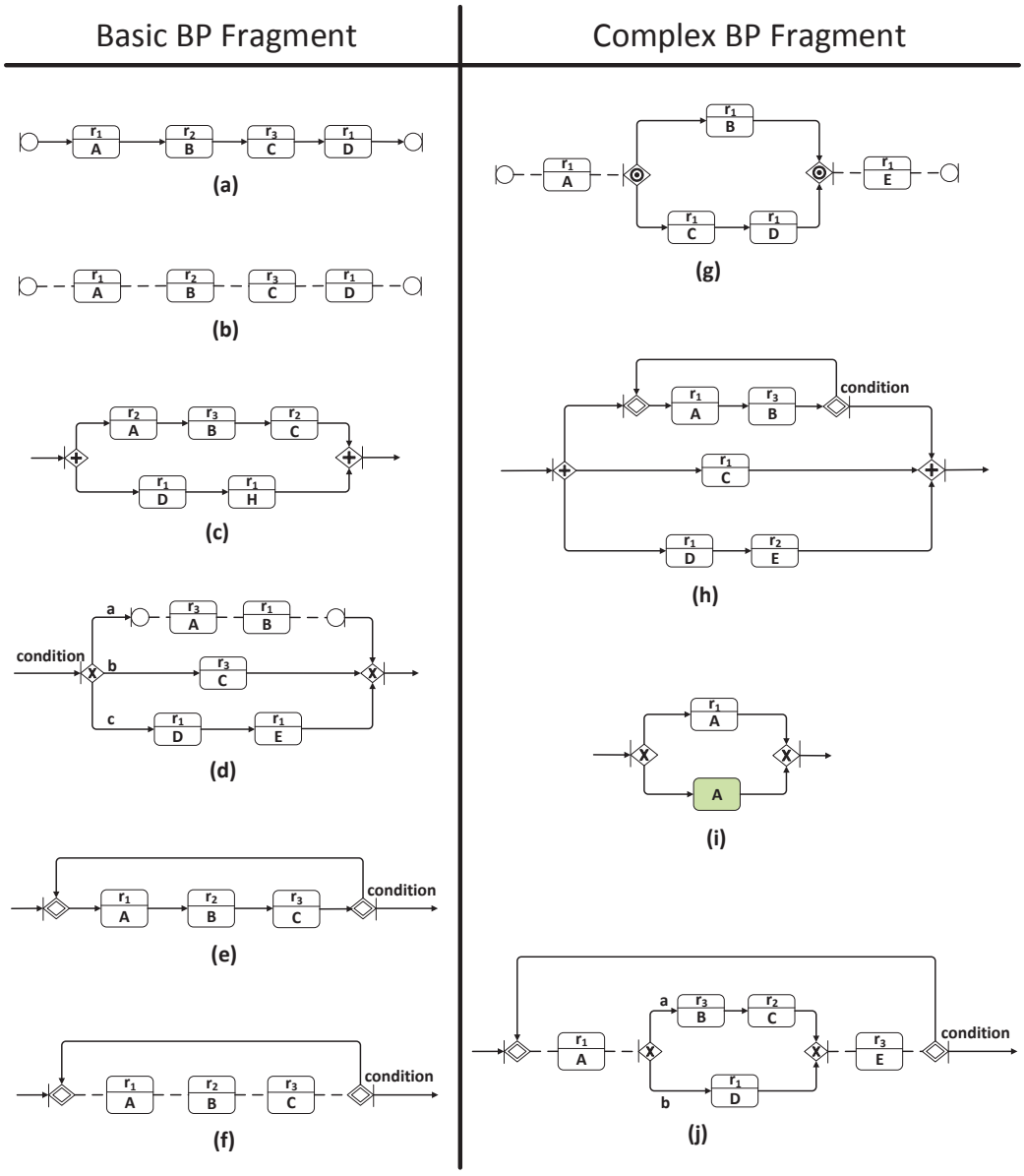


Figure 4.17: Examples of Basic BP Fragments and Complex BP Fragments

  - the control flow pattern of each branch of this fragment must only be **Strict-order Sequential** or **Free-order Sequential** (For example in Figure 4.17, (d) is a Basic BP Fragment), and
  - each branch of this fragment must not consist of a single task or an abstracted node of this task. ((i) in Figure 4.17 is an example of this situation. We notate (i) as *Task-Abs Block*.)

- **if** the control flow pattern of the fragment is **Strict-loop** or **Free-loop**, **then** all the elements of this fragment must only be individual tasks. (For example in Figure 4.17, (e) and (f) are Basic BP Fragments, and (j) is a Complex BP Fragment.)

In the following, we will firstly introduce the two functions `AbsComplexFrag` and `AbsBasicFrag`, then followed by **Algorithm 3** for task abstraction and aggregation. Lastly, an example business process with a complex structure is gone through to show the major steps of the algorithm for task abstraction and aggregation.

4.3.2.1 Handling Complex BP Fragments

We use function `AbsComplexFrag` to deal with the tasks in the Complex BP Fragments of a role-enriched BP. The input is a Complex BP Fragment and the output is the transformed Complex BP Fragment.

In order to abstract and aggregate the tasks in a complex BP fragment, the first step is to identify the control flow pattern $CFPattern$ and the related elements in $TaskSet$ on the coarsest granularity level of the input fragment $CFrag$ (line 2). We use different methods to the deal with different elements in $TaskSet$ according to the types of the elements.

We assume that the abstraction and aggregation is for user role r_1. In $TaskSet$, there are four categories of elements: (1) a task $Task_{Local}$ that is executed by r_1, (2) a task $Task_{Foreign}$ that is not executed by r_1, (3) a task $Task_{Multi-Or}$ that can be executed by r_1 or other user roles, (4) a task $Task_{Multi-And}$ that must be executed by both r_1 and other user roles, and (5) a block of tasks with one entry gateway and one

Algorithm 1: Function for Handling Complex BP Fragment

```
1  Function AbsComplexFrag(ComplexFrag CFrag)
2      identify CFPattern and TaskSet at the coarsest granularity level of CFrag;
3      if TaskSet does not contain MultiRoleTask then
4          if CFPattern matches elementaryOperation then
5              CFrag' =transform (CFrag, elementaryOperation);
6          else if CFPattern does not matches elementaryOperation then
7              CFrag' =transform (CFrag, Single − Abs − Agg − 1);
8              examine and combine adjacent abstracted nodes in CFrag';
9      else
10         CFrag' =transform (CFrag, Single − Abs − Agg − 1/ − 2/ − 3);
11         examine and combine adjacent abstracted nodes in CFrag';
12     return CFrag';
```

exit gateway, inside this block there exist complex control flow relations between/among these tasks.

Line 3 represents $TaskSet$ does not contain $Task_{Multi-Or}$ or $Task_{Multi-And}$. In this case, if $CFPattern$ matches one of the elementary operations, the input role-enriched BP fragment is transformed by this matched elementary operation (line 4 and line 5); if $CFPattern$ cannot match any of the elementary operations, each $Task_{Foreign}$ is abstracted by **Single-Abs-Agg-1** (line 6 and line 7). After that, the transformed fragment is examined to see if it has adjacent abstracted nodes or not. If there exist adjacent abstracted nodes, they are aggregated as single abstracted nodes (line 8). The other situation is $CFEleSet$ that contains $Task_{Multi-Or}$ or $Task_{Multi-And}$ (line 9), $Task_{Foreign}$ are abstracted by **Single-Abs-Agg-1**, and $Task_{Multi-Or}$ and/or $Task_{Multi-And}$ are abstracted by **Single-Abs-Agg-2** and/or **Single-Abs-Agg-3** (line 10). Then we examine the transformed fragment for adjacent abstracted nodes. If there exist adjacent abstracted nodes, they are aggregated as single abstracted nodes (line 11). Lastly, the transformed result is returned (line 12).

4.3.2.2 Handling Basic BP Fragments

We use function `AbsBasicFrag` to deal with the Basic BP Fragments of role-enriched BP. The input is a Basic BP Fragment, and the output is the transformed Basic BP Fragment.

In order to abstract and aggregate the tasks in a Basic BP Fragment $BFrag$, the first step is to identify the control flow pattern $cfPattern$ and the related elements in $cfEleSet$ of $BFrag$ (line 2). We use different methods to the deal with the $BFrag$ according to different $cfPattern$.

If $cfPattern$ is **Strict-order Sequential**, **Free-order Sequential**, **Strict-loop**, or **Free-loop** (line 3), we check whether $BFrag$ contains `MultiRoleTask`. If no `MultiRoleTask` is included, $BFrag$ is transformed by using corresponding elementary operations (line 4 and line 5); if there exist `MultiRoleTask`s in $BFrag$, each task in $cfEleSet$ of $BFrag$ is transformed by using **Single-Abs-Agg-1** or **Single-Abs-Agg-2**, and $BFrag'$ is the result. Then $BFrag'$ is transformed using `TransformByTree` as $BFrag''$. After that, we check the adjacent abstracted nodes in $BFrag''$ and aggregate them as single abstracted nodes (from line 6 to line 9).

If $cfPattern$ is **Parallel-A**, **Parallel-B**, **Parallel-C**, or **Conditional** (line 10), we check again whether $BFrag$ contains `MultiRoleTask`.

- When no `MultiRoleTask` is included (line 11), we check if $BFrag$ can be transformed directly by using elementary operations. If suitable *elementaryOperation* can apply, $BFrag$ is transformed accordingly as $BFrag''$ (line 12 and line 13); if no suitable elementary operation can apply, we only abstract single tasks on each branch of $BFrag$ using **Single-Abs-Agg-1** or **Single-Abs-Agg-2** and $BFrag''$ is the result. Then we check the adjacent abstracted nodes on each branch of $BFrag''$ and aggregate them as single abstracted nodes (from line 14 to line 18), the result is named as $BFrag''$.

- When `MultiRoleTask`(s) is/are included, single tasks on each branch of $BFrag$ is transformed by using **Single-Abs-Agg-1** or **Single-Abs-Agg-2** and $BFrag'$ is

Algorithm 2: Function for Handling Basic BP Fragment

```
1  Function AbsBasicFrag(BasicFrag BFrag)
2      identify cfPattern and cfEleSet of BFrag;
3      if cfPattern = StrictSeq, FreeSeq, StrictLoop, or FreeLoop then
4          if cfEleSet does not contain MultiRoleTask then
5              BFrag'' = transform (BFrag, Sequential − Abs − Agg − 1/ − 2
                or Loop − Abs − Agg − 1/ − 2);
6          else if cfEleSet contains MultiRoleTask then
7              BFrag' = transform (BFrag, Single − Abs − Agg − 1/ − 2);
8              BFrag'' = TransformWithTree (BFrag');
9              examine and combine adjacent abstracted nodes in BFrag'';
10     else if cfPattern = ParallelA, ParallelB, ParallelC, or Conditional then
11         if cfEleSet does not contain MultiRoleTask then
12             if cfPattern matches elementaryOperation then
13                 BFrag'' = transform (BFrag, elementaryOperation);
14             else
15                 foreach branch ∈ BFrag do
16                     branch' = transform (branch,
                        Single − Abs − Agg − 1/ − 2);
17                     BFrag'' is the result;
18                     examine and combine adjacent abstracted nodes in BFrag'';
19         else if cfEleSet contains MultiRoleTask then
20             foreach branch ∈ BFrag do
21                 branch' = transform (branch, Single − Abs − Agg − 1/ − 2);
22                 BFrag' is the result;
23                 BFrag'' = TransformWithTree (BFrag');
24                 examine and combine adjacent abstracted nodes in BFrag'';
25     return the transformed fragment BFrag'';
```

the result. Then $BFrag'$ is transformed by using `TransformByTree` and $BFrag''$ is returned. After that, we check the adjacent abstracted nodes in $BFrag''$ and aggregate them into single abstracted nodes (from line 19 to line 24).

Finally, the fragment $BFrag''$ is returned as the result (line 25).

In the above algorithm, the function `TransformByTree` is used to transform a **Sequential** fragment containing the Task-Abs Blocks in (i) of Figure 4.17 to a **Conditional** fragment using tree graph. The input of this function is a control flow pattern **Strict-order Sequential** or **Free-order Sequential**, where the element set contains: (1) at least one Task-Abs Block, (2) zero to many tasks performed by r_1, and (3) zero to many abstracted nodes. The output of this function is a **Conditional** fragment.

We use an example as shown in Figure 4.18 to demonstrate the transformation. The first step is to generate a tree diagram for the input block. In this step, single tasks and abstracted nodes are added onto the tree graph ($Task_B$ and ABS_{DE}); each Task-Abs Block is separated as individual branches on the tree graph ($Task_A$-ABS_{DE} block and $Task_C$-ABS_C block). The second step is to transform the tree graph to a **Conditional** fragment. In this step, each path in the tree graph is transformed as a branch of the **Conditional** fragment. For example in (b) of Figure 4.18, the path $Task_A$-$Task_B$-ABS_C-ABS_{DE} is transformed as one branch containing $Task_A$, $Task_B$, ABS_C, ABS_{DE} in **Conditional** fragment.

4.3.2.3 Abstracting and Aggregating Tasks in Role-enriched BP

In this section, we introduce **Algorithm 3** for task abstraction and aggregation. The input is a role-enriched BP, and the output is an abstracted and aggregated BP. This algorithm is realized by using a recursive function `BPAbsAgg`, which iteratively calls itself. In the recursive way, each granularity level of the role-enriched BP is reached, ranging from coarsest granularity level (the role-enriched BP itself) to the finest granularity level (the Basic BP Fragments). In doing so, tasks on each granularity level are dealt with by using functions `AbsComplexFrag` and `AbsBasicFrag`.

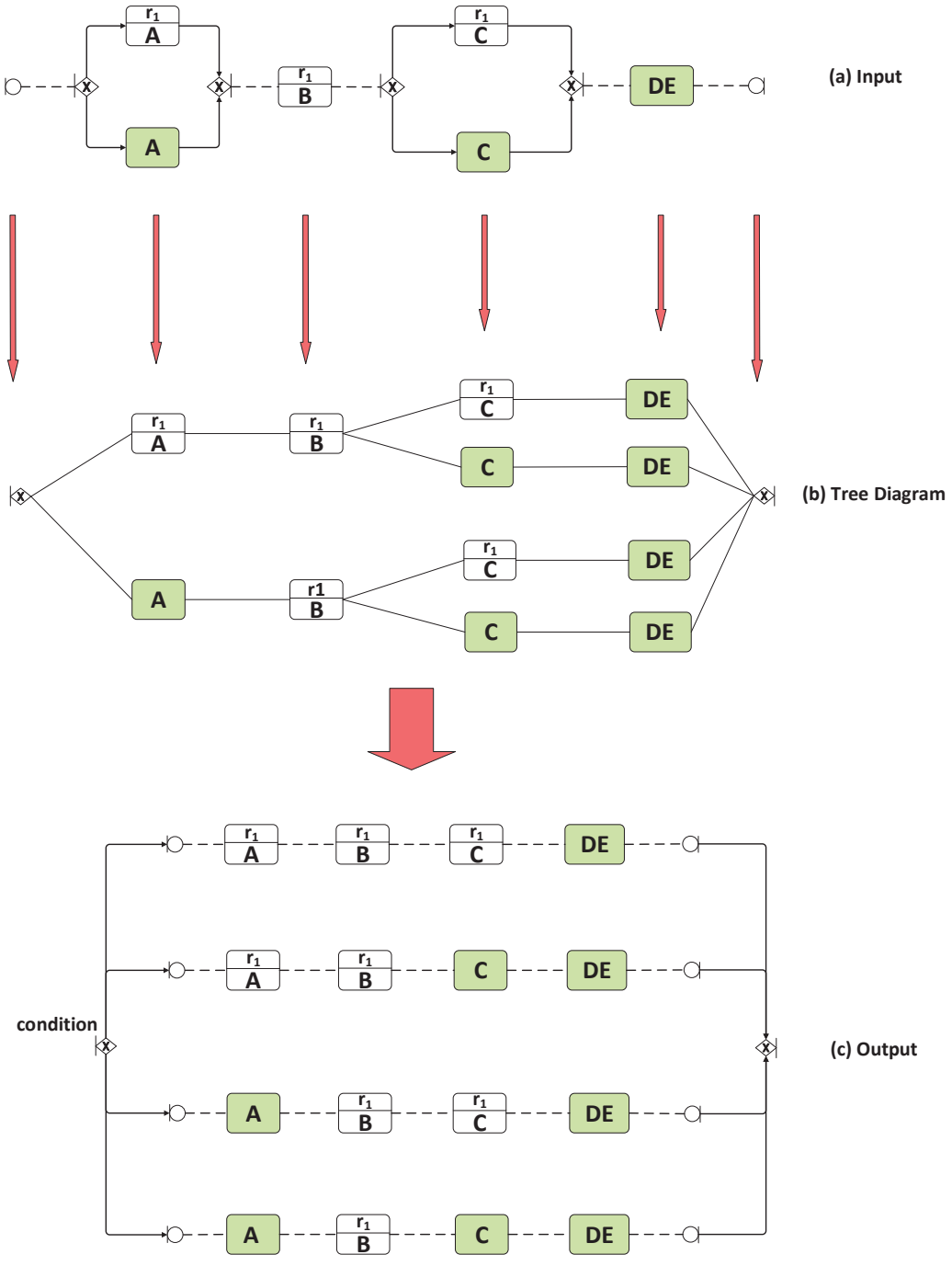


Figure 4.18: Transforming Task-Abs Block Using Tree Graph

Algorithm 3: Task Abstraction and Aggregation

Input : *RoleEnrichedBP*
Output: *AABP*

```
1  AABP = ∅;
2  BPAbsAgg(RoleEnrichedBP);
3  return AABP;
4  Function BPAbsAgg(RoleEnrichedBP RoleEnBP)
5      RoleEnBP' = AbsComplexFrag(RoleEnBP);
6      if AABP == ∅ then
7          add RoleEnBP' to AABP;
8      else
9          update RoleEnBP in AABP with RoleEnBP';
10     identify cfElementSet of RoleEnBP';
11     foreach element ∈ cfElementSet do
12         if element is BasicBPFragment then
13             element' = AbsBasicFrag(element);
14             update element in AABP with element';
15         else
16             BPAbsAgg(element);
```

When a role-enriched BP $RoleEnrichedBP$ is input into this recursive function, function `AbsComplexFrag` is used to deal with the elements on the coarsest granularity level of $RoleEnrichedBP$ (line 4). In case the input $RoleEnrichedBP$ has only one granularity level, where the elements on this level equal to those on the finest level, the function `AbsComplexFrag` deals with this $RoleEnrichedBP$ in the same way as the function `AbsBasicFrag` does. After handled by `AbsComplexFrag`, the handled $RoleEnrichedBP'$ is copied onto the initialized $AABP$ (line 6 and line 7). Then the element set $cfElementSet$ on the coarsest level of $RoleEnrichedBP'$ is identified (line 10) to deal with the tasks on a finer level L_1. Till this step, two possible results exist: (1) if L_1 is the finest level, the function `AbsComplexFrag` is used to deal with the tasks on this level L_1, and the dealt result replaces the counter parts in $AABP$; (2) if L_1 is not the finest level, the $RoleEnrichedBP'$ is re-input into the recursive function `BPAbsAgg` to handle the finer but non-finest level (line 15 and line 16). From the second iteration, the result of the function `AbsComplexFrag` in this iteration continuously replace the result in $AABP$ of last iteration. The recursive function `BPAbsAgg` finalizes after all the tasks on the finest level are handled. Then the final abstracted and aggregated BP $AABP$ is derived and returned (line 3).

4.3.2.4 Go-through Example of Task Abstraction and Aggregation

Figure 4.19 shows the major steps of the algorithm when going through an example business process with a complex structure. (a) Figure 4.19 shows the a role-enriched BP as the input of the algorithm. At **Step 1** ((b) Figure 4.19), the control flow pattern **Free-order Sequential** at the coarsest granularity level of the process is identified, and $Task_A$, $Task_R$, $Task_S$ are abstracted/aggregated by applying elementary operations. At **Step 2** ((c) Figure 4.19), the control flow pattern **Conditional** at the middle granularity level of the process is identified. $Task_C$, $Task_G$ and $Task_I$, $Task_Q$ are abstracted/aggregated separately. At **Step 3** ((d) Figure 4.19), the elementary operations *Loop-Abs-Agg-1*, *Parallel-B-Abs-Agg-1*, *Alternative-2* of *Parallel-A-Abs-Agg-2* are applied on **Basic BP Fragments** as **Loop** with $Task_{D,E,F}$, **Parallel-B** with $Task_{J,K,L}$,

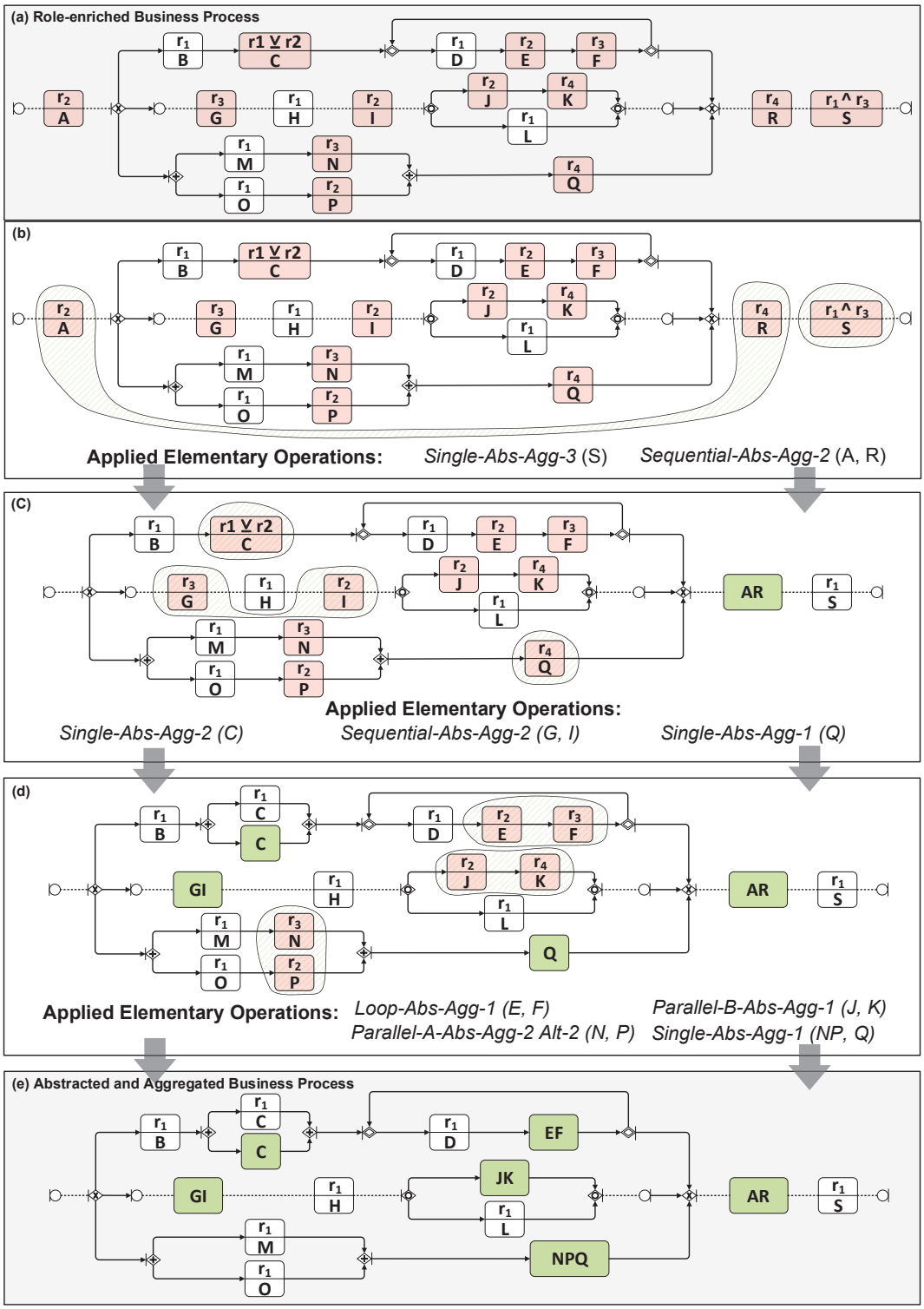


Figure 4.19: Task Abstraction and Aggregation of a BP with a Complex Structure

Parallel-A with $Task_{M,N,O,P}$ respectively. *Alternative-2* of *Parallel-A-Abs-Agg-2* generates an abstracted node Abs_{NP}, that is further aggregated with Abs_Q as Abs_{NPQ} by using *Single-Abs-Agg-1*. (e) Figure 4.19 shows the AABP as the task abstraction and aggregation result.

4.4 Analysis of Abstracted and Aggregated Business Processes

This section discusses a series of BP structural properties regarding the order and dependency between tasks of a BP. We use them to examine whether the AABP is consistent with the Role-enriched BP. The "consistency" means that the ordering of tasks from a role-enriched BP must be kept in the AABP, and no additional ordering between tasks are introduced in the AABP. In the context of change management of a BP, different change solutions for the BP could be judged based on the examination of these properties.

4.4.1 Order Between Tasks

Based on the definitions of role-enriched BP model and AABP model, we introduce two functions:

(1) A role mapping function is defined as $RoleMap$: $T \rightarrow R_T$, where $R_T = \bigcup_{i=1}^{n} R_{t_i}$, $i = 1, 2, ..., n$. It is used to obtain the set of roles R_t participating in task t.

(2) According to the elementary operations in Subsection 4.3.1, an abstraction mapping function $AbsMap$ is defined as:

$$AbsMap(t) = \begin{cases} \{t^r\} & \text{if } RoleMap(t) = \{r\} \\ \{abs^r\} & \text{if } RoleMap(t) \not\supseteq \{r\} \\ \{t^r, abs^r\} & \text{if } RoleMap(t) \supsetneq \{r\} \end{cases}$$

With the abstraction mapping function, the corresponding node in the AABP for each task of the Role-enriched BP can be found out.

Here we use three notations to represent the path between two tasks in a role-enriched BP bp: $t_1 \gg t_2$ denotes that there exists one path of **Strict-order Sequential**

in bp from t_1 to t_2; $t_1 \leftrightarrow t_2$ denotes that there exists one path of **Free-order Sequential** in bp from t_1 to t_2; and $t_1|t_2$ denotes that there exists no path between t_1 and t_2 in bp. Usually, two tasks, locating on different branches of a *Parallel* or *Conditional* pattern, have no path between each other. These notations are also used to express the path between two nodes in an AABP for user role r, where each node can be a task t^r, or an abstracted node abs^r. Based on the above functions and notations, the definitions of Order-Keeping, and Strict-Order-Keeping are built up.

Definition 3: Order-Keeping. Given a role-enriched BP bp with the task set T, and the AABP $aabp^r$ for user role $r \in R$, $aabp^r$ is defined as Order-Keeping, **iff** $\forall t_1, t_2 \in T$, where $t_1 \neq t_2$ and

1. $t_1 \gg t_2$, **then** $\neg(n_2^r \gg n_1^r)$, where $n_1^r = AbsMap(t_1)$ and $n_2^r = AbsMap(t_2)$; or
2. $t_1 \leftrightarrow t_2$, **then** $\neg(n_1^r | n_2^r)$, where $n_1^r = AbsMap(t_1)$ and $n_2^r = AbsMap(t_2)$.

The property of Order-Keeping describes that (1) if there exist two tasks t_1, t_2 which are **Strict-order Sequential** in a role-enriched BP, and t_1 is precedent of t_2, then in AABP, the abstracted node n_2^r (from t_2) cannot be precedent of the abstracted node n_1^r (from t_1); (2) if there exist two tasks t_1, t_2 which are **Free-order Sequential** in a role-enriched BP, then the corresponding nodes n_1^r, n_2^r in AABP must be **Free-order Sequential** or **Strict-order Sequential**. This is a loose constraint of orders between two nodes. Next, we specify a more strict constraint of orders between two nodes.

Definition 4: Strict-Order-Keeping. Given a role-enriched BP bp with the task set T, and the AABP $aabp^r$ for user role $r \in R$, $aabp^r$ is defined as Strict-Order-Keeping, **iff** $\forall t_1, t_2 \in T$, where $t_1 \neq t_2$ and

1. $t_1 \gg t_2$, **then** $n_1^r \gg n_2^r$, where $n_1^r = AbsMap(t_1)$ and $n_2^r = AbsMap(t_2)$; or
2. $t_1 \leftrightarrow t_2$, **then** $n_1^r \leftrightarrow n_2^r$, where $n_1^r = AbsMap(t_1)$ and $n_2^r = AbsMap(t_2)$; or
3. $t_1|t_2$, **then** $n_1^r | n_2^r$, where $n_1^r = AbsMap(t_1)$ and $n_2^r = AbsMap(t_2)$.

The property of Strict-Order-Keeping describes that for any two tasks t_1, t_2 in a role-enriched BP, the order between the corresponding nodes n_1^r, n_2^r must be kept in AABP. Definition 3 and Definition 4 provide the order constraints between two nodes at different degrees. The consistency between a role-enriched BP and the AABP requires that at least one of these two properties is satisfied.

4.4.2 Dependency Between Tasks

According to the definitions of role-enriched BP model and AABP model, the notions of dependency and dependency set are discussed in this subsection.

In a role-enriched BP, a dependency, consisting of two adjacent tasks, reflects the control flow relation between the two adjacent tasks. In an AABP, a dependency, consisting of two nodes, reflects the control flow relation between the two adjacent nodes, each of which can be a task, an abstracted node abstracted from one task in the role-enriched BP, or an abstracted node aggregated from multiple tasks in the role-enriched BP.

The dependency between two tasks A and B is annotated as $\lfloor A, B \rceil$; and the the dependency between a task A and an abstracted node BC is annotated as $\lfloor A, BC \rceil$. Note that there may exist gateways between the two adjacent tasks/nodes (see the abstracted node ABD and task C in the AABP fragment of the **Parallel-A-Abs-Agg-A Alt-1** in Subsection 4.3.1). We represent the dependency between these two nodes ABD and C as $\lfloor ABD, C \rceil$ even there exists a *Parallel-A Entry Gateway* between two nodes ABD and C.

The dependency set of role-enriched BP is $\Upsilon_{bp} = \{\lfloor t_1, t_2 \rceil \in T \times T | (t_1, t_2 \text{ are adjacent}) \wedge (t_1 \gg t_2 \vee t_1 \leftrightarrow t_2)\}$. The dependency set of an AABP is $\Upsilon_{aabp^r} = \{\lfloor n_1^r, n_2^r \rceil \in N_\alpha^r \times N_\alpha^r | (N_\alpha^r = T^r \cup ABS^r \cup AGGR^r) \wedge (n_1^r, n_2^r \text{ are adjacent}) \wedge (n_1^r \gg n_2^r \vee n_1^r \leftrightarrow n_2^r)\}$. For the **Parallel-B-Abs-Agg-1** (see Subsection 4.3.1), we can get $\Upsilon_{bp} = \{\lfloor A, B \rceil, \lfloor C, D \rceil, \lfloor D, E \rceil\}$, and $\Upsilon_{aabp^r} = \{\lfloor A, B \rceil, \lfloor CD, E \rceil\}$.

The comparison between Υ_{bp} and Υ_{aabp^r} shows how the dependency sets are influenced by task abstraction and aggregation. The dependency $\lfloor C, D \rceil$ in Υ_{bp} disappears

in Υ_{aabp^r}; the dependency $\lfloor D, E \rceil$ in Υ_{bp} turns into $\lfloor CD, E \rceil$ in Υ_{aabp^r}; and the dependency $\lfloor A, B \rceil$ of Υ_{bp} is reserved in Υ_{aabp^r}. After examining the elementary operations in Subsection 4.3.1, we classify four properties associated with the dependency between tasks as follows.

Definition 5: Dependency-Keeping. Given a role-enriched BP bp with its dependency set Υ_{bp}, and the AABP $aabp^r$ for user role $r \in R$ with its dependency set Υ_{aabp^r}, $aabp^r$ is Dependency-Keeping **iff** $\Upsilon_{bp} = \Upsilon_{aabp^r}$.

Definition 6: Dependency-Removing. Given a role-enriched BP bp with its dependency set Υ_{bp}, and the AABP $aabp^r$ for user role $r \in R$ with its dependency set Υ_{aabp^r}, $aabp^r$ is Dependency-Removing **iff** $\Upsilon_{bp} \not\supseteq \Upsilon_{aabp^r}$.

Definition 7: Dependency-Increasing. Given a role-enriched BP bp with its dependency set Υ_{bp}, and the AABP $aabp^r$ for user role $r \in R$ with its dependency set Υ_{aabp^r}, $aabp^r$ is Dependency-Increasing **iff** $\Upsilon_{bp} \subsetneq \Upsilon_{aabp^r}$.

Definition 8: Dependency-Updating. Given a role-enriched BP bp with its dependency set Υ_{bp}, and the AABP $aabp^r$ for user role $r \in R$ with its dependency set Υ_{aabp^r}, $aabp^r$ is Dependency-Updating **iff** in Υ_{aabp^r}, there exists at least one dependency in which one node is an abstracted node, and this abstracted node is aggregated from multiple tasks in bp.

4.4.3 Property Analysis of Elementary Operations

Table 4.1 provides a summary for property analysis on elementary operation in Subsection 4.3.1. **Parallel-A-Abs-Agg-1 Alt-1**, **Parallel-A-Abs-Agg-2 Alt-1**, and **Parallel-A-Abs-Agg-4** are Order-Keeping, and the rest elementary operations are Strict-Order-Keeping. **Single-Abs-Agg-1** and **Single-Abs-Agg-3** is Dependency-Keeping; **Single-Abs-Agg-2** is Dependency-Increasing; **Parallel-A-Abs-Agg-3**, **Parallel-B-Abs-Agg-2**, **Parallel-C-Abs-Agg-2**, and
Conditional-Abs-Agg-1 are Dependency-Removing; the rest of the elementary operations are Dependency-Updating.

Table 4.1: Overview of Properties for Elementary Operations

Properties	Strict-Order-Keeping	Order-Keeping	Dependency-Keeping	Dependency-Removing	Dependency-Increasing	Dependency-Updating
Single-Abs-Agg-1	+	+	+	-	-	-
Single-Abs-Agg-2	+	+	-	-	+	-
Single-Abs-Agg-3	+	+	+	-	-	-
Sequential-Abs-Agg-1	+	+	-	-	-	+
Sequential-Abs-Agg-2	+	+	-	-	-	+
Parallel-A-Abs-Agg-1 Alt-1	-	+	-	-	-	+
Parallel-A-Abs-Agg-1 Alt-2	+	+	-	-	-	+
Parallel-A-Abs-Agg-2 Alt-1	-	+	-	-	-	+
Parallel-A-Abs-Agg-2 Alt-2	+	+	-	-	-	+
Parallel-A-Abs-Agg-3	+	+	-	+	-	-
Parallel-A-Abs-Agg-4	-	+	-	-	-	+
Parallel-B-Abs-Agg-1	+	+	-	-	-	+
Parallel-B-Abs-Agg-2	+	+	-	+	-	-
Parallel-B-Abs-Agg-3	+	+	-	-	-	+
Parallel-C-Abs-Agg-1	+	+	-	-	-	+
Parallel-C-Abs-Agg-2	+	+	-	+	-	-
Parallel-C-Abs-Agg-3	+	+	-	-	-	+
Conditional-Abs-Agg-1	+	+	-	+	-	-
Loop-Abs-Agg-1	+	+	-	-	-	+
Loop-Abs-Agg-2	+	+	-	-	-	+

The following theorem is about the accompanying relation between the Order-Keeping property and the Dependency-Updating property of an AABP. **Theorem 1: Iff** an AABP has Order-Keeping property but has no Strict-Order-Keeping property, **Then** the AABP has Dependency-Updating property.

Proof Let bp be a role-enriched BP with the task set T, and $aabp^r$ be the AABP for user role $r \in R$, and $aabp^r$ has the Order-Keeping property. There are three elementary operations with Order-Keeping property and without Strict-Order-Keeping property (see Table 4.1). These operations are **Parallel-A-Abs-Agg-1 Alt-1**, **Parallel-A-Abs-Agg-2 Alt-1**, and **Parallel-A-Abs-Agg-4**. On each branch of the **Parallel-A** pattern, there are some tasks in which r does not participate. All the tasks not involving r are aggregated as one abstracted node, and this node is shifted out of the **Parallel-A** pattern. According to **Definition 8: Dependency-Updating**, an AABP as the output of anyone of these three operations has the Dependency-Updating property. Thus the claim holds. □ **Theorem 2 Let** $aabp^r$ be an AABP for user role $r \in R$,

- **iff** $aabp^r$ is Dependency-Keeping, **then** $aabp^r$ is Strict-Order-Keeping;
- **iff** $aabp^r$ is Dependency-Increasing, **then** $aabp^r$ is Strict-Order-Keeping;
- **iff** $aabp^r$ is Dependency-Removing, **then** $aabp^r$ is Strict-Order-Keeping.

Proof Let bp be a role-enriched BP with the task set T, and $aabp^r$ be an AABP for user role $r \in R$,

(1) $aabp^r$ is Dependency-Keeping. Then the dependency set $\Upsilon_{bp} = \Upsilon_{aabp^r}$. For any two tasks t_1 and t_2 in bp, there always exists at least one path from t_1 to t_2 in bp, and this path is *Strict-order Sequential* and/or *Free-order Sequential*. As $\Upsilon_{bp} = \Upsilon_{aabp^r}$, the path from t_1^r to t_2^r in $aabp^r$ is reserved. Therefore, the first claim holds; (2) $aabp^r$ is Dependency-Increasing. In all the elementary operations in Subsection 4.3.1, the only reason causing Dependency-Increasing is in bp, there exists at least one task which can be performed by either r or other user role. If there are three tasks t_3, t_4, t_5, t_4 is performed by user role r or r', t_3 is before t_4 and t_5 is after t_4. Corresponding to t_4 in bp, the $aabp^r$ has a *Conditional* pattern containing t_4 and t_4' (see *Single-Abs-Agg-2* in

Subsection 4.3.1). t'_4 is an abstracted node when the task is performed by a user role r'. Comparing to Υ_{bp}, the newly increased dependency in Υ_{aabp^r} are $\lfloor t_3, t'_4 \rceil$ and $\lfloor t'_4, t_5 \rceil$. And a path from t_3 via t'_4 to t_5 is generated in $aabp^r$. This increased dependency does not affect the orders among t_3, t_4, t_5 in $aabp^r$. Therefore, the second claim holds; (3) $aabp^r$ is Dependency-Removing. In all the elementary operations in Subsection 4.3.1, the only reason causing Dependency-Removing is that multiple branches of a pattern **Parallel-A**, **Parallel-B**, **Parallel-C**, or **Conditional** are abstracted as one single abstracted node. This abstraction causes associated dependencies in Υ_{bp} disappear in Υ_{aabp^r}. For any two tasks t_6 and t_7 in bp, there exists at least one path from t_6 to t_7. (a) If both t_6 and t_7 are abstracted, the path(s) from t'_6 to t'_7 will not appear in $aabp^r$; (b) if both t_6 and t_7 are not abstracted, the path(s) from t'_6 to t'_7 will be appear in $aabp^r$; (c) if t_6 is not abstracted and t_7 is abstracted, t_6 and t_7 must be in different branches of the pattern and there is no path from t_6 to t_7 in bp. In $aabp^r$, t'_7 and other tasks except t'_6 are aggregated as a single abstracted node, and this abstracted node forms a single branch. Thus there is no path from t'_6 to t'_7 in $aabp^r$. Therefore, the third claim holds. □

4.5 Scenario Example

In this section, we use the scenario example introduced in Figure 1.4 of Section 1.2 to demonstrate the derived AABPs for three user roles `personnel officer`, `referee`, and `applicant` participating in the recruitment process.

(a) Figure 4.20 shows the AABP for the user role `personnel officer`. The tasks for the `personnel officer` are all kept, and the tasks for the `applicant`/`referee` are abstracted. In the abstraction and aggregation, the tasks `Submit Application` and `Writing Reference Letter` are abstracted as ABS_1; the task `Confirm Interview` is abstracted as ABS_2).

(b) Figure 4.20 shows the AABP for the user role for `referee`. The tasks for the `referee` are all kept, and the tasks for the `personnel officerapplicant` are abstracted. In the abstraction and aggregation, the tasks `Announce A Job` and `Submit`

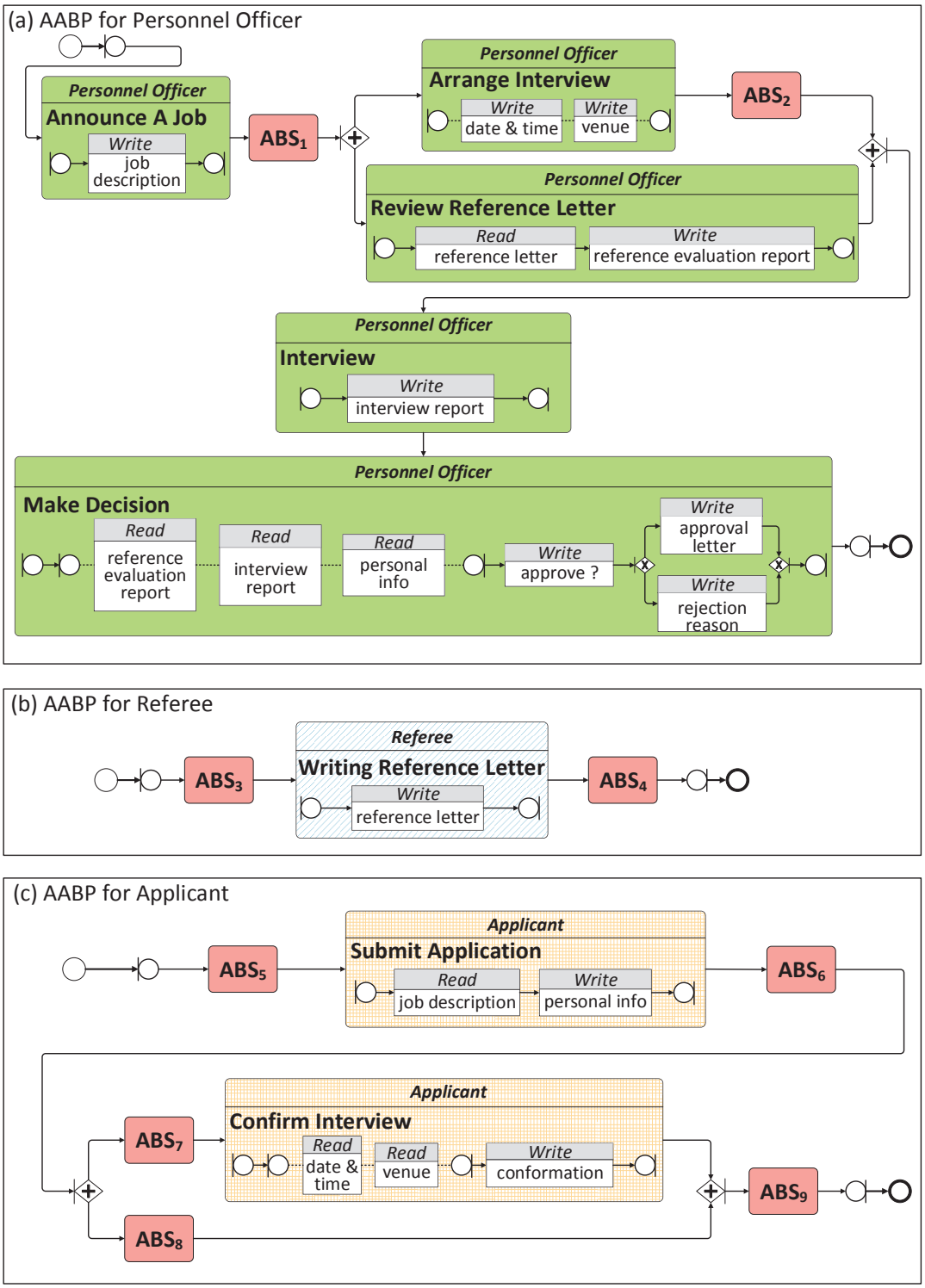


Figure 4.20: AABPs for User Roles Participating in Recruitment Process

`Application` are abstracted as ABS_3; the tasks `Arrange Interview`, `Confirm Interview`, `Review Reference Letter`, `Interview, Make Decision` are abstracted as ABS_4.

(c) Figure 4.20 shows the AABP for the user role for `applicant`. The tasks for the `applicant` are all kept, and the tasks for the `personnel officer` textttreferee are abstracted. In the abstraction and aggregation, the task `Announce A Job` is abstracted as ABS_5; the task `Writing Reference Letter` is abstracted as ABS_6; the task `Arrange Interview` is abstracted as ABS_7; the task `Review Reference Letter` is abstracted as ABS_8; the tasks `Interview` and `Make Decision` are abstracted as ABS_9.

4.6 Summary and Discussion

This section proposes an approach of task abstraction and aggregation in BPs based on a role-enriched BP model. This is the first step of our proposed UI derivation approach. The approach of task abstraction and aggregation can be used to support analysing, developing, and updating software components such as user interfaces related to different user roles. According to these patterns identified from the role-enriched business process, a series of elementary operations are coined as cornerstones of the algorithm for task abstraction and aggregation. The structural consistency between the role-enriched BP and the AABP is illustrated by providing details of the analysis on both the orders and dependencies between/among tasks. As a future research direction, the change management of BPs and AABPs can be studied in a unified framework. Note that there exist tasks which do not need to have user interfaces. In all the proposed elementary operations, the tasks without UIs are treated the same as the tasks performed by other roles. Therefore, the tasks without UIs are abstracted/aggregated.

Chapter 5

Data Relationship Extraction

In this chapter, we discuss the data relationship extraction, which is the second step of our UI derivation approach as shown in Figure 1.5. Section 5.1 provides an introduction of this chapter. Section 5.2 introduces the data relationships. Two subsections are included as tree graph that specifies the data relationships, and JSON Schema that records the data relationships. Section 5.3 discusses the approach of data relationship extraction. In this section, two aspects are included as the elementary operations and algorithms. Section 5.4 introduces a scenario example. Section 5.5 provides a summary and discussion on this chapter.

5.1 Introduction

After the AABP is generated for a user role, the data relationships, including operated data and data operation flow, are extracted from the AABP for this specific user role. With the extracted data relationships, the UI ligic for this user role can be derived. We use tree graph to represent the data relationships. And the JSON Strings are used to record all the details represented in the tree graph.

(a) Attribute Node (b) Information Node (c) CDE Node (d) Virtual Node

Figure 5.1: Nodes in a Tree Graph

5.2 Data Relationships

5.2.1 Tree Graph

A tree graph is a graph consisting of a collection of interrelated tree graph fragments. Each tree graph fragment comprises a set of nodes and the data relationships between these nodes. The reason why the tree graph is used to represent the data relationship is because the tree graph can explicitly specify a hierarchy structure, which is the characteristics of process data. In the following, we introduce the tree graph in detail.

- `ProcessStructRef` is to represent the root of a tree graph. In a tree graph, there exist one and only one `ProcessStructRef`.In JSON Strings, it is represented as

```
{"structRefType": "ProcessStructRef"}
```

- `Normal StructRef` is to indicate the reference of a tree graph fragment. In JSON Strings, it is represented as

```
{"structRefType": "NormalStructRef"}
```

In a tree graph, there are four types of nodes (see Figure 5.1) as follows.

(1) `Attribute Node` is represented by "Attr". It refers to a data item operated inside a single task of an AABP. In the example of (a) Figure 5.1, the subscript represents that the data entity represented by this `attribute node` is derived from attribute in *TaskA* of an AABP. In JSON Strings, it is represented as

```
{"node":{
    "nodeName": "attr_A",
    "nodeType": "attr",
    "preAbsOfNode": "null",
    "postAbsOfNode": "null"
```

```
 }
}
```

(2) `Information Node` is represented by "INFO". It contains a piece of information for a specific user role. In the example of (b) Figure 5.1, the subscript A indicates the data entity represented by this `information node` is derived from $TaskA$ of an AABP. In JSON Strings, it is represented as

```
{"node":{
    "nodeName": "INFO_A",
    "nodeType": "info",
    "preAbsOfNode": "null",
    "postAbsOfNode": "null"
 }
}
```

(3) `CDE Node` is represented by "CDE". It refers to a fragment of the tree graph. In the example of (c) Figure 5.1, the subscript A indicates the tree graph fragment named A. In JSON Strings, it is represented as

```
{"node":{
    "nodeName": "CDE_A",
    "nodeType": "cde",
    "preAbsOfNode": "null",
    "postAbsOfNode": "null"
 }
}
```

(4) `Virtual Node` is represented by a dashed circle (see (d) Figure 5.1). It starts a conditional data relationship pattern. In JSON Strings, the `virtual node` is not represented, as we can directly represent the **Conditional** block using the type **Conditional** and the related branches.

There are six data relationship patterns: **Strict-order Sequential**, **Free-order Sequential**, **Parallel-A**, **Conditional**, **Strict-Order Loop**, and **Free-Order Loop**. Figure 5.2 shows examples of these patterns with `CDE Nodes`.

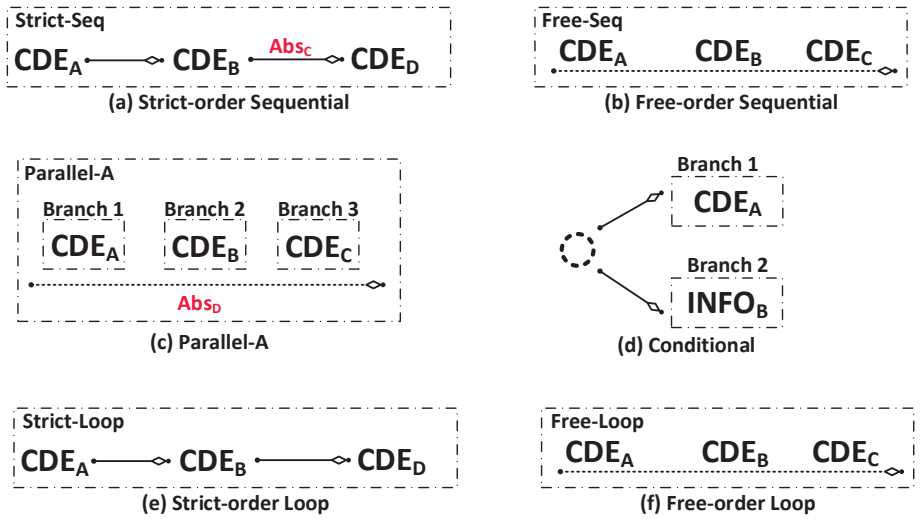


Figure 5.2: Data Relationship Patterns

- **Strict-order Sequential** ((a) Figure 5.2) is represented with a dot-dash rectangle, a pattern name `Strict-Seq` on the upper left of the rectangle, and node relationships. The node relationships are represented using an edge with two dots on each end to connect two nodes. A diamond on one end of the line is used to point to the following node. In JSON Strings, it is represented as

```
1  {"temporalRelation": "Strict_Seq",
2   "graphNodes": {
3       {"nodeName": "CDE_A", "nodeType": "cde", "
           preAbsOfNode": "null", "postAbsOfNode": "
           null"},
4       {"nodeName": "CDE_B", "nodeType": "cde", "
           preAbsOfNode": "null", "postAbsOfNode": "
           Abs_C"},
5       {"nodeName": "CDE_D", "nodeType": "cde", "
           preAbsOfNode": "Abs_C", "postAbsOfNode": "
           null"}
6    }
7  }
```

- **Free-order Sequential** ((b) Figure 5.2) is represented with a dot-dash rectangle,

a pattern name `Free-Seq` on the upper left of the rectangle, and node relationships. The node relationships are represented using a dashed edge with two dots on each end, and the nodes are above the dashed edge. There is a diamond on one end of the dashed edge. In JSON Strings, it is represented as

```
{"temporalRelation": "Free_Seq",
 "graphNodes": {
     {"nodeName": "CDE_A", "nodeType": "cde", "
        preAbsOfNode": "null", "postAbsOfNode": "
        null"},
     {"nodeName": "CDE_B", "nodeType": "cde", "
        preAbsOfNode": "null", "postAbsOfNode": "
        null"},
     {"nodeName": "CDE_C", "nodeType": "cde", "
        preAbsOfNode": "null", "postAbsOfNode": "
        null"}
 }
}
```

- **Parallel-A** ((c) Figure 5.2) is represented with a dot-dash rectangle, a pattern name `Parallel-A` on the upper left of the rectangle, and node relationships. The node relationships are represented using a dashed edge with two dots on each end, and the data entities are above the dashed edge. There is a diamond on one end of the dashed edge. Each data entity is represented with a dot-dash rectangle, a node inside the rectangle, and branch name above the rectangle. In JSON Strings, it is represented as

```
{"temporalRelation": "Parallel_A",
 "graphNodes": {
     {"branchName": "Branch_1",
      "branchType": "singleNode",
      "preAbsOfBranch": "Abs_D",
      "postAbsOfBranch": "Abs_D",
      "branchNodes": {
          "nodeNameInBranch": "CDE_A",
```

```
            "nodeTypeInBranch": "cde",
            "preAbsInBranch": "null",
            "postAbsInBranch": "null"
        },
        {"branchName": "Branch_2",
        "branchType": "singleNode",
        "preAbsOfBranch": "Abs_D",
        "postAbsOfBranch": "Abs_D",
        "branchNodes": {
            "nodeNameInBranch": "CDE_B",
            "nodeTypeInBranch": "cde",
            "preAbsInBranch": "null",
            "postAbsInBranch": "null"
        },
        {"branchName": "Branch_3",
        "branchType": "singleNode",
        "preAbsOfBranch": "Abs_D",
        "postAbsOfBranch": "Abs_D",
        "branchNodes": {
            "nodeNameInBranch": "CDE_C",
            "nodeTypeInBranch": "cde",
            "preAbsInBranch": "null",
            "postAbsInBranch": "null"
        }
      }
    }
  }
}
```

- **Conditional** ((d) Figure 5.2) is represented with a `Virtual Node` and a set of branches, each of which holds a data entity. Each data entity is represented with a dot-dash rectangle, a node inside the rectangle, and branch name above the rectangle. In JSON Strings, it is represented as

```
{"temporalRelation": "Conditional",
 "graphNodes": {
```

```
        {"branchName": "Branch_1",
         "branchType": "labelOnly",
         "preAbsOfBranch": "null",
         "postAbsOfBranch": "null",
         "branchNodes": {
             {"nodeNameInBranch": "CDE_A",
              "nodeTypeInBranch": "cde",
              "preAbsInBranch": "null",
              "postAbsInBranch": "null"
             }
         },
         {"branchName": "Branch_2",
         "branchType": "Strict_Seq",
         "preAbsOfBranch": "null",
         "postAbsOfBranch": "null",
         "branchNodes": {
             {"nodeNameInBranch": "INFO_B",
             "nodeTypeInBranch": "info",
             "preAbsInBranch": "null",
             "postAbsInBranch": "null"
             }
         }
     }
    }
```

- **Strict-order Loop** ((e) Figure 5.2) is represented with a dot-dash rectangle, a pattern name `Strict-Loop` on the upper left of the rectangle, and node relationships. The node relationships are represented by using an edge with two dots on each end to connect two nodes. A diamond on one end of the line is used to point to the following node. In JSON Strings, it is represented as

```
    {"temporalRelation": "Strict_Loop",
     "graphNodes": {
         {"nodeName": "CDE_A", "nodeType": "cde", "
            preAbsOfNode": "null", "postAbsOfNode": "
            null"},
```

```
4          {"nodeName": "CDE_B", "nodeType": "cde", "
               preAbsOfNode": "null", "postAbsOfNode": "
               null"},
5          {"nodeName": "CDE_C", "nodeType": "cde", "
               preAbsOfNode": "null", "postAbsOfNode": "
               null"}
6       }
7     }
```

- **Free-order Loop** ((f) Figure 5.2) is represented with a dot-dash rectangle, a pattern name `Free-Loop` on the upper left of the rectangle, and node relationships. The node relationships are represented by using a dashed edge with two dots on each end, and the nodes are above the dashed edge.There is a diamond on one end of the dashed edge. In JSON Strings, it is represented as

```
1     {"temporalRelation": "Free_Loop",
2      "graphNodes": {
3          {"nodeName": "CDE_A", "nodeType": "cde", "
               preAbsOfNode": "null", "postAbsOfNode": "
               null"},
4          {"nodeName": "CDE_B", "nodeType": "cde", "
               preAbsOfNode": "null", "postAbsOfNode": "
               null"},
5          {"nodeName": "CDE_C", "nodeType": "cde", "
               preAbsOfNode": "null", "postAbsOfNode": "
               null"}
6       }
7     }
```

An `Abs Label` is used to represent control flow relations between tasks and abstracted nodes in the AABP. In a tree graph, an `Abs Label` only turns up above a solid edge (see Abs_C in (a) Figure 5.2), or under a dashed edge (see Abs_D in (c) Figure 5.2).

5.2.2 Data Relationships Recorded using JSON Schema

In this section, we introduce the JSON Schema that represents all the details represented in the tree graph. This JSON Schema is written according to the draft v4 specification.

```
{"$schema": "http://json-schema.org/draft-04/schema#",
 "title": "Tree Graph",
 "type": "array",
 "items": {
     "title": "Tree graph fragment",
     "description": "A ree graph fragment in tree graph",
     "type": "object",
     "properties": {
         "id": {
             "description": "The unique identifier for the
                tree graph fragment",
             "type": "integer"
         },
         "treeName": {
             "description": "Name of the tree graph
                fragment",
             "type": "string"
         },
         "structRefType": {
             "description": "The StructRef Type of the tree
                 graph fragment",
             "type": "string",
             "enum": ["Process StructRef", "Normal
                StructRef"]
         },
         "temporalRelation": {
             "description": "The type of temporal
                raletaionships between the nodes within the
                 tree graph fragment",
             "type": "string",
             "enum": ["Strict_Seq", "Free_Seq", "Parallel_A
```

```
                ", "Conditional", "Strict_Loop", "Free_Loop
                "]
        },
        "graphNodes": {
            "description": "All the node in the the tree
               graph fragment",
            "type": "array",
            "items": {
                "oneOf": [
                    {"$ref": "#/definitions/node"}, {"$ref
                       ": "#/definitions/branch"}
                ]
            },
            "minItems": 1,
            "uniqueItems": true
        }
    },
    "required": ["nodeName", "nodeType", "structRef", "
       temporalRelation", "graphNodes"]
},
"definitions": {
    "node": {
        "type": "object",
        "properties": {
            "nodeName": {"type": "string"},
            "nodeType": {"type": {"enum": [ "cde", "attr",
                "info" ]}},
            "preAbsOfNode":   {"type": "string"},
            "postAbsOfNode":  {"type": "string"}
        },
        "required": ["nodeName", "nodeType", "preAbsOfNode
           ", "postAbsOfNode"]
    }
},
"definitions": {
    "branch": {
```

```
55          "type": "object",
56          "properties": {
57              "branchName":  {"type": "string"},
58              "branchType":  {"type": {"enum": ["Strict_Seq"
                    , "Free_Seq", "singleNode", "labelOnly"]}},
59              "preAbsOfBranch":   {"type": "string"},
60              "postAbsOfBranch":  {"type": "string"},
61              "branchNodes": {
62                  "oneOf": [
63                      {"$ref": "#/definitions/
                            nodesInsideBranch"}, {"$ref": "#/
                            definitions/singleAbsInsideBranch"}
64                  ]
65              }
66          },
67          "required": ["branchName", "branchType", "
                preAbsOfBranch", "postAbsOfBranch", "
                branchNodes"]
68      }
69  },
70  "definitions": {
71      "nodesInsideBranch":{
72          "type": "array",
73          "items": {
74              "type": "object",
75              "properties": {
76                  "nodeNameInBranch": {"type": "string"},
77                  "nodeTypeInBranch": {"type": {"enum": ["
                        cde", "attr", "info"]}},
78                  "preAbsInBranch":   {"type": "string"},
79                  "postAbsInBranch":  {"type": "string"}
80              },
81              "required": ["nodeNameInBranch", "
                    nodeTypeInBranch", "preAbsInBranch", "
                    postAbsInBranch"]
82          },
```

```
            "minItems": 1,
            "uniqueItems": true
        }
    },
    "definitions": {
        "singleAbsInsideBranch":{
            "type": "object",
            "properties": {
                "absLabelName": {"type": "string"}
            },
            "required": ["absLabelName"]
        }
    }
}
```

5.2.3 Well-formness of Tree Graph

In this subsection, we introduce the regulations of a well-formed tree graph. These regulations can be transformed into algorithms to check whether the data relationships represented by JSON Strings are well formed or not.

- In a tree graph, there exists one and only one tree graph fragment, which has `Process StructRef`.
- In a tree graph fragment with `Process StructRef`, no `Attribute nodes` can be contained.
- In a tree graph fragment, the `Attribute nodes` must not co-exist with the `INFO nodes`.
- In a tree graph fragment that does not contain `CDE nodes`, there must exist `Attribute nodes`; there might exist `INFO nodes` and `Virtual nodes`.

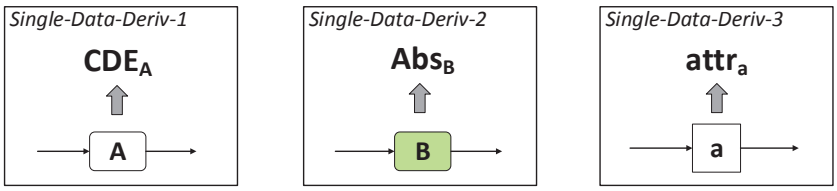


Figure 5.3: Elementary Operations Single-Data-Deriv-1/2/3 on Task/Abstracted Node/-Data Item

5.3 Data Relationship Extraction

This section introduces the data relationship extraction from an AABP for a particular user role. The data relationships are extracted for each user role involved in the role-enriched BP. To extract the data relationships, a series of elementary operations are specified. The algorithm for data relationship extraction is developed by utilizing these elementary operations.

5.3.1 Elementary Operations

In this section, 21 elementary operations for data relationship extraction are specified. Among these operations, 3 elementary operations extract the data relationships from a task, an abstracted node and an attribute; 13 elementary operations extract the data relationships from the control flow patterns of an AABP; and 5 elementary operations extract the data relationships from the data operation patterns inside individual tasks of an AABP.

5.3.1.1 Elementary Operations on Task, Abstracted Node and Data Item of AABP

Fig.5.3 captures three types of elementary operations based on a task and abs/agg node, respectively. And they are denoted as **Single-Data-Deriv-1**, **Single-Data-Deriv-2**, and **Single-Data-Deriv-3**, respectively.

Single-Data-Deriv-1 ((a) Figure 5.3) shows how an elementary operation generates data relationships from a task in an AABP. In this situation, the task ($Task_A$) is represented as a CDE node (CDE_A).

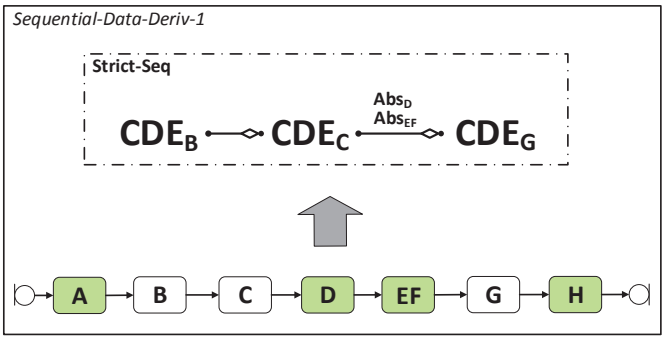


Figure 5.4: Elementary Operations Sequential-Data-Deriv-1 on Strict-order Sequential

Single-Data-Deriv-2 ((b) Figure 5.3) shows how an elementary operation generates data relationships from an abstracted node in an AABP. In this situation, the abstracted node (ABS_B) is represented as an Abs Label (Abs_B).

Single-Data-Deriv-3 ((c) Figure 5.3) shows how an elementary operation generates data relationships from a data item inside a task of an AABP. In this situation, the data item ($Attribute_a$) is represented as an Attribute Node ($Attribute_a$).

5.3.1.2 Elementary Operations on Control Flow Pattern of AABP

Sequential-Data-Deriv-1 (Figure 5.4) shows how an elementary operation generates data relationships from a **Strict-order Sequential** control flow pattern in an AABP. In this situation, this control flow pattern is inherited by the **Strict-order Sequential** data relationship pattern. The tasks ($Task_B$, $Task_C$, $Task_G$) in the control flow pattern are represented as CDE nodes (CDE_B, CDE_C, CDE_G), and the order between these CDE nodes inherits the order between the tasks. The abstracted nodes (ABS_D, ABS_EF) between tasks are represented as Abs Labels (Abs_D, Abs_EF), and the Abs Labels are placed on the edge between the CDE nodes originated from the corresponding tasks. The abstracted nodes (ABS_A, ABS_H) at both ends of a branch are removed away.

Sequential-Data-Deriv-2 (Figure 5.5) shows how an elementary operation generates data relationships from a **Free-order Sequential** control flow pattern in an AABP. In this situation, this control flow pattern is inherited by the **Free-order Se-**

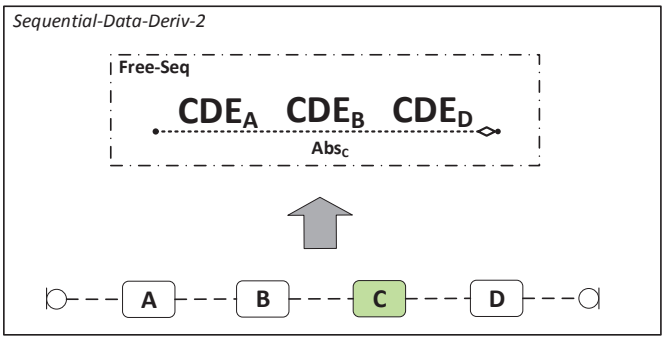


Figure 5.5: Elementary Operations Sequential-Data-Deriv-2 on Free-order Sequential

quential data relationship pattern. The tasks ($Task_A$, $Task_B$, $Task_D$) in the control flow pattern are represented as CDE nodes (CDE_A, CDE_B, CDE_D). The abstracted nodes (ABS_C) between tasks are represented as Abs Labels (Abs_C) under the dashed line in the data relationship pattern.

In the following, we introduce the transformations of single branches in **Parallel-A**, **Parallel-B** and **Parallel-C** control flow patterns of AABPs (see Figure 5.6).

- The **Transformation 1** ((a) Figure 5.6) shows how the data relationships are extracted from a branch of **Strict-order Sequential** control flow pattern. And this pattern only contains tasks. In this situation, the branch ($Branchk$) is inherited by a data entity named ($Branchk$) that has **Strict-order Sequential** data relationship pattern. The tasks ($Task_{k1}$, $Task_{k2}$, $Task_{k3}$) in the control flow pattern are represented as CDE nodes (CDE_{k1}, CDE_{k2}, CDE_{k3}), and the order between these CDE nodes inherits the order between the tasks.

- The **Transformation 2** ((b) Figure 5.6) shows how the data relationships are extracted from a branch of **Strict-order Sequential** control flow pattern. And this pattern contains tasks and abstracted nodes. In this situation, the branch ($Branchk$) is inherited by a data entity named ($Branchk$) that has **Strict-order Sequential** data relationship pattern. The tasks ($Task_{k1}$, $Task_{k3}$) in the control flow pattern are represented as CDE nodes (CDE_{k1}, CDE_{k3}), and the order between these CDE nodes inherits the order between the tasks. The abstracted

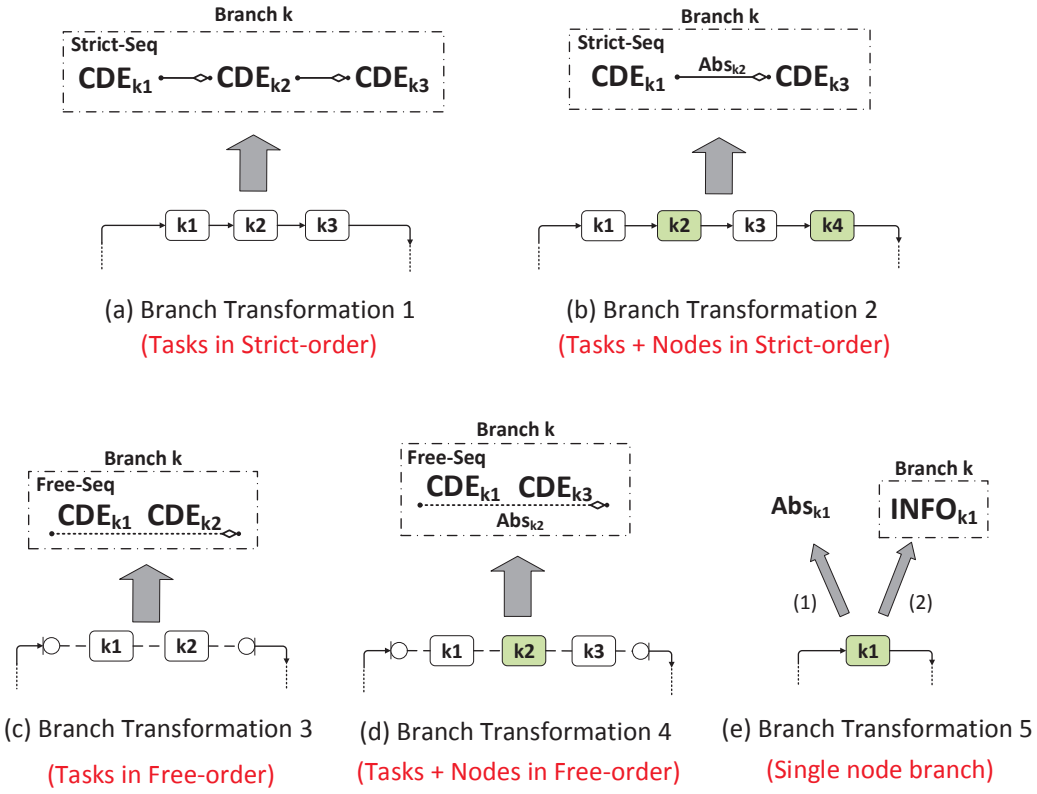


Figure 5.6: Transformations of Individual Branches from Parallel-A, B, C Control Flow Patterns In AABP

nodes (ABS_{k2}) between tasks are represented as Abs Labels (Abs_{k2}), and the Abs Labels are placed on the edge between the CDE nodes originated from the corresponding tasks. The abstracted nodes (ABS_{k4}) at both ends of a branch are removed away.

- The **Transformation 3** ((c) Figure 5.6) shows how the data relationships are extracted from a branch of **Free-order Sequential** control flow pattern. And this pattern only contains tasks. In this situation, the branch ($Branchk$) is inherited by a data entity named ($Branchk$) that has **Free-order Sequential** data relationship pattern. The tasks ($Task_{k1}$, $Task_{k2}$) in the control flow pattern are represented as CDE nodes (CDE_{k1}, CDE_{k2}).

- The **Transformation 4** ((d) Figure 5.6) shows how the data relationships are extracted from a branch of **Free-order Sequential** control flow pattern. And this pattern contains tasks and abstracted nodes. In this situation, the branch ($Branchk$) is inherited by a data entity named ($Branchk$) that has **Free-order Sequential** data relationship pattern. The tasks ($Task_{k1}$, $Task_{k3}$) in the control flow pattern pattern are represented as CDE nodes (CDE_{k1}, CDE_{k3}). The abstracted nodes (ABS_{k2}) between tasks are represented as Abs Labels (Abs_{k2}) under the dashed line in the data relationship pattern.

- The **Transformation 5** ((e) Figure 5.6) shows how the data relationships are extracted from a branch ($Branchk$) that only has one abstracted node (ABS_{k1}). (1) if this branch is from a **Parallel-A** pattern, this branch is represented as an Abs Labels (Abs_{k1}); (2) if this branch is from a **Parallel-B** or **Parallel-B** pattern, this branch is inherited by a data entity named ($Branchk$), and the abstracted node (ABS_{k1}) on the branch is represented as an Information node ($INFO_{k1}$) in the data entity to show the execution status of of this branch.

Parallel-A-Data-Deriv-1 (Figure 5.7) shows how an elementary operation generates data relationships from a **Parallel-A** control flow pattern in an AABP. Each branch of the pattern contains one or multiple tasks. In this situation, all branches of

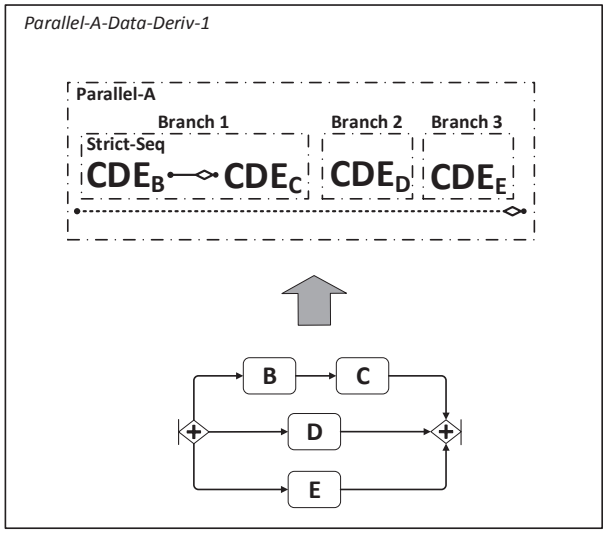


Figure 5.7: Elementary Operations Parallel-A-Data-Deriv-1 on Parallel-A

the **Parallel-A** control flow pattern are inherited by the corresponding data operation pattern. Each branch is transformed by **Transformation 1**.

Parallel-A-Data-Deriv-2 (Figure 5.8) shows how an elementary operation generates data relationships from a **Parallel-A** control flow pattern in an AABP. Each branch of the pattern contains one or multiple tasks/abstracted nodes. In this situation, all branches of the **Parallel-A** control flow pattern are inherited by the corresponding data operation pattern. A branch containing only an abstracted node is transformed by (1) **Transformation 5**; a branch containing only tasks is transformed by **Transformation 1**; a branch containing tasks and abstracted nodes is transformed by **Transformation 2**.

Parallel-A-Data-Deriv-3 (Figure 5.9) shows how an elementary operation generates data relationships from a **Parallel-A** control flow pattern in an AABP. Each branch of the pattern contains one or multiple tasks, which have **Free-order Sequential** control flow pattern. In this situation, all branches of the **Parallel-A** control flow pattern are inherited by the corresponding data operation pattern. Each branch is transformed by **Transformation 3**.

Parallel-A-Data-Deriv-4 (Figure 5.10) shows how an elementary operation gen-

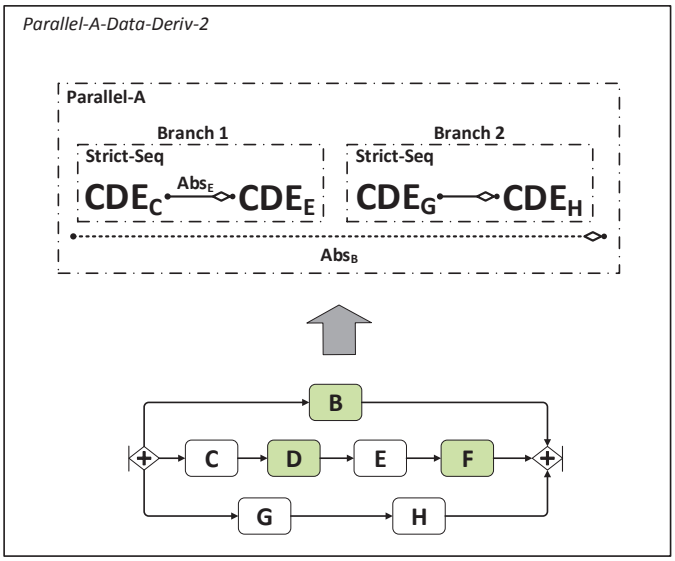


Figure 5.8: Elementary Operations Parallel-A-Data-Deriv-2 on Parallel-A

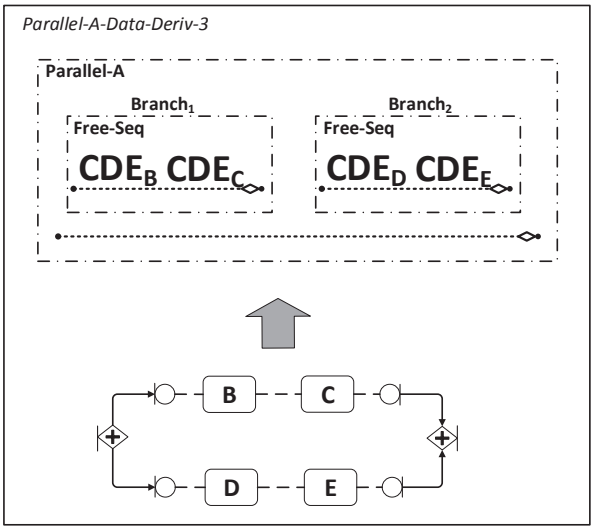


Figure 5.9: Elementary Operations Parallel-A-Data-Deriv-3 on Parallel-A

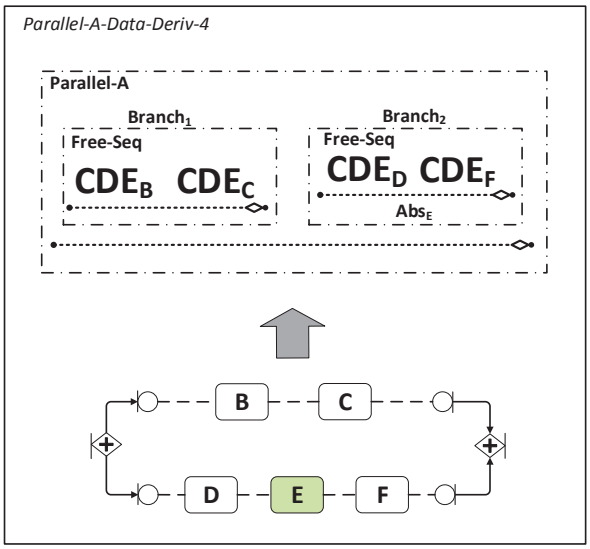


Figure 5.10: Elementary Operations Parallel-A-Data-Deriv-4 on Parallel-A

erates data relationships from a **Parallel-A** control flow pattern in an AABP. Each branch of the pattern contains one or multiple tasks and zero or multiple abstracted nodes which have **Free-order Sequential** control flow pattern. In this situation, all branches of the **Parallel-A** control flow pattern are inherited by the corresponding data operation pattern. A branch containing only tasks is transformed by **Transformation 3**; a branch containing tasks and abstracted nodes is transformed by **Transformation 4**.

Parallel-B-Data-Deriv-1 (Figure 5.11) shows how an elementary operation generates data relationships from a **Parallel-B** control flow pattern in an AABP. Each branch of the pattern contains one or multiple tasks. In this situation, all branches of the **Parallel-B** control flow pattern are inherited by the **Conditional** data operation pattern. Each branch is transformed by **Transformation 1**.

Parallel-B-Data-Deriv-2 (Figure 5.12) shows how an elementary operation generates data relationships from a **Parallel-B** control flow pattern in an AABP. Each branch of the pattern contains one or multiple tasks/abstracted nodes. In this situation, all branches of the **Parallel-B** control flow pattern are inherited by the **Conditional** data operation pattern. A branch containing only an abstracted node is

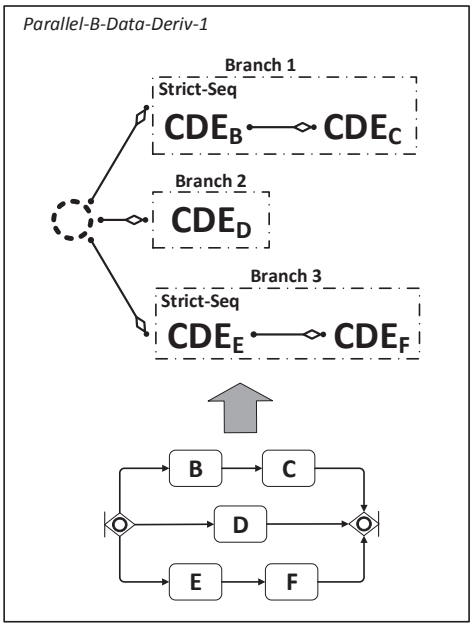


Figure 5.11: Elementary Operations Parallel-B-Data-Deriv-1 on Parallel-B

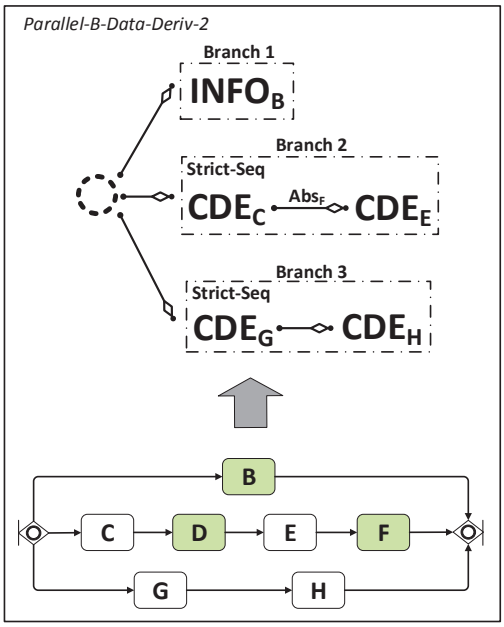


Figure 5.12: Elementary Operations Parallel-B-Data-Deriv-2 on Parallel-B

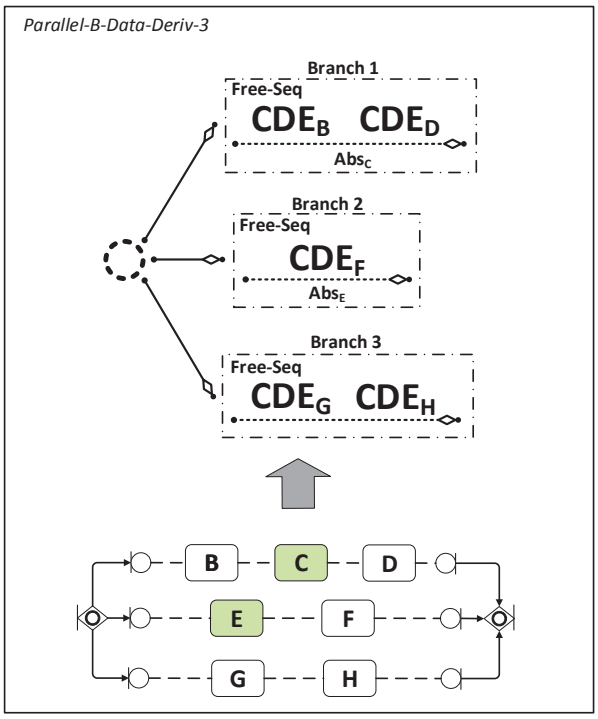


Figure 5.13: Elementary Operations Parallel-B-Data-Deriv-3 on Parallel-B

transformed by (1) **Transformation 5**; a branch containing only tasks is transformed by **Transformation 1**; a branch containing tasks and abstracted nodes is transformed by **Transformation 2**.

Parallel-B-Data-Deriv-3 (Figure 5.13) shows how an elementary operation generates data relationships from a **Parallel-B** control flow pattern in an AABP. Each branch of the pattern contains one or multiple tasks and zero or multiple abstracted nodes, which have **Free-order Sequential** control flow pattern. In this situation, all branches of the **Parallel-A** control flow pattern are inherited by the corresponding data operation pattern. A branch containing only tasks is transformed by **Transformation 3**; a branch containing tasks and abstracted nodes is transformed by **Transformation 4**.

Parallel-C-Data-Deriv-1 (Figure 5.14) shows how an elementary operation generates data relationships from a **Parallel-C** control flow pattern in an AABP. The **Parallel-C** pattern has n paths in total, and any m paths out of the n paths must

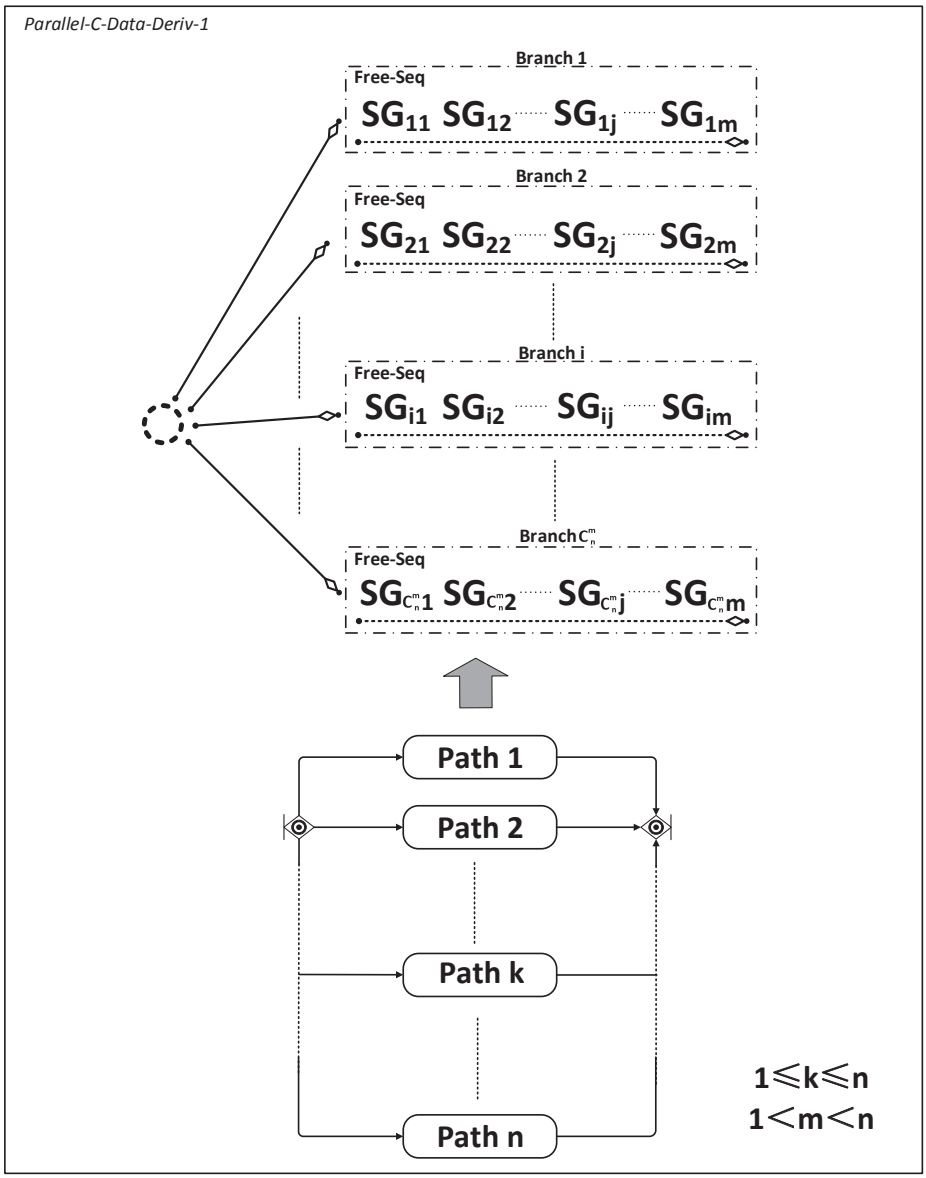


Figure 5.14: Elementary Operations Parallel-C-Data-Deriv-1 on Parallel-C

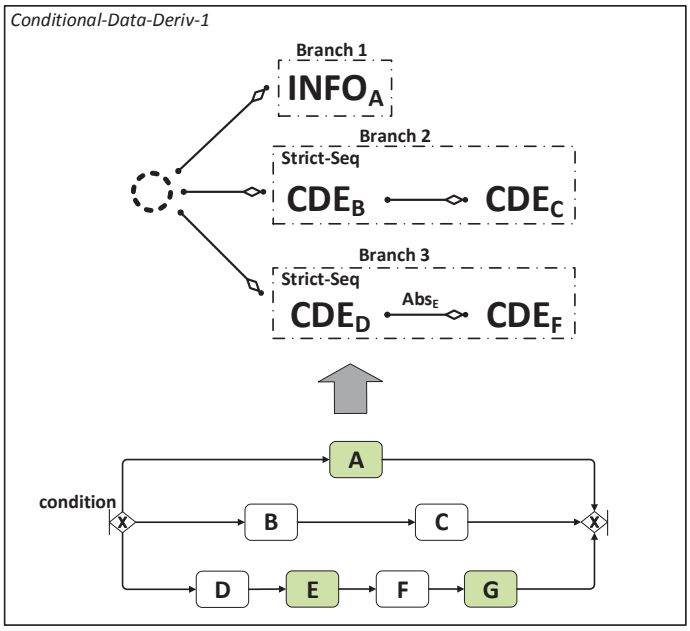


Figure 5.15: Elementary Operations Conditional-Data-Deriv-1 on Conditional

be completed. There are five types for a path $Path_k$: (1) a **Strict-order Sequential** path only containing tasks, (2) a **Strict-order Sequential** path containing tasks and abstracted nodes, (3) a **Free-order Sequential** path only containing tasks, (4) a **Free-order Sequential** path containing tasks and abstracted nodes, (5) a path containing a single abstracted node.

The transformation of **Parallel-C** control flow pattern comprises three steps. The first step is to transform each path $Path_k$ into a segment SG_k using the corresponding branch transformation in Figure 5.6. The second step is to build up the **Free-order Sequential** pattern inside each `Branch` block $\{Branch_i | 1 \leq i \leq C_n^m\}$ using the segments $\{SG_k | 1 \leq k \leq n\}$. Each $Block_i$ contains m segments notated as SG_{i1}, SG_{i2},..., SG_{ij},..., SG_{im}, which represents a possible way to select m segments from $\{SG_k | 1 \leq k \leq n\}$. The third step is that each `Branch` block $Branch_i$ follows the virtual node.

Conditional-Data-Deriv-1 in Figure 5.15 shows how an elementary operation generates data relationships from a **Conditional** control flow pattern in an AABP. Each branch of the **Conditional** control flow pattern is inherited by the corresponding data relationship pattern. For a branch, there are rules: (1) If it is composed of

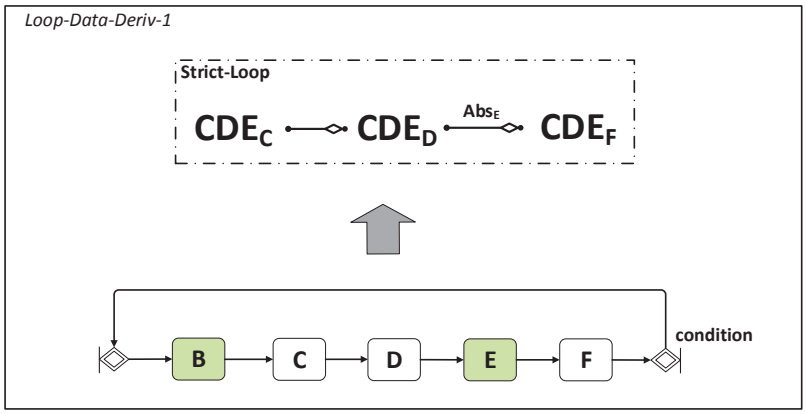


Figure 5.16: Elementary Operations Loop-Data-Deriv-1 on Loop

only one abstracted node ($AbsNode_A$), an `Information Node` ($INFO_A$) is used to represent the execution status of this branch. (2) If a branch is composed of only tasks, `CDE Nodes` (CDE_B and CDE_C) are used to represent these tasks ($Task_B$ and $Task_C$). (3) If a branch is composed of both tasks and abstracted nodes, a `CDE Node` is used to represent a task; an `Abs Label`(Abs_E) is used to represent an abstracted node ($AbsNode_E$) between tasks; and an abstracted node ($AbsNode_G$) at one end of a branch is removed away.

Loop-Data-Deriv-1 (Figure 5.16) shows how an elementary operation generates data relationships from a **Loop** control flow pattern in an AABP. In this situation, this control flow pattern is inherited by the **Strict-order Loop** data relationship pattern. The tasks ($Task_C$, $Task_D$, $Task_F$) in the control flow pattern pattern are represented as CDE nodes (CDE_C, CDE_D, CDE_F), and the order between these CDE nodes inherits the order between the tasks. The abstracted nodes (ABS_E) between tasks are represented as Abs Labels (Abs_E), and the Abs Labels are placed on the edge between the CDE nodes originated from the corresponding tasks. The abstracted nodes (ABS_B) at both ends of a branch are removed away.

Loop-Data-Deriv-2 (Figure 5.17) shows how an elementary operation generates data relationships from a **Loop** control flow pattern in an AABP. All the tasks and abstracted nodes inside this loop have **Free-order Sequential** pattern. In this situation, this control flow pattern is inherited by the **Free-order Loop** data relationship

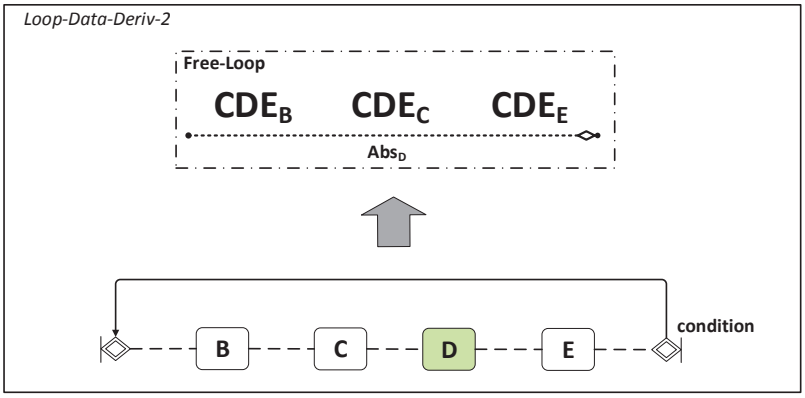


Figure 5.17: Elementary Operations Loop-Data-Deriv-2 on Loop

pattern. The tasks ($Task_B$, $Task_C$, $Task_E$) in the control flow pattern pattern are represented as CDE nodes (CDE_B, CDE_C, CDE_E). The abstracted nodes (ABS_D) between tasks are represented as Abs Labels (Abs_D) under the dashed line in the data relationship pattern.

5.3.1.3 Elementary Operations on Data Operation Patterns inside Individual Tasks of AABP

Sequential-Attr-Data-Deriv-1 in Figure 5.18 shows how an elementary operation generates data relationships from a **Strict-order Sequential** data operation pattern inside a task of an AABP. In this situation, this data operation pattern is inherited by the **Strict-order Sequential** data relationship pattern. The data items (DI_a, DI_b, DI_c) in the data operation pattern are represented as attribute nodes ($attr_a$, $attr_b$, $attr_c$), and the order between these attribute nodes inherits the order between the data items.

Sequential-Attr-Data-Deriv-2 in Figure 5.18 shows how an elementary operation generates data relationships from a **Free-order Sequential** data operation pattern pattern inside a task of an AABP. In this situation, this data operation pattern is inherited by the **Free-order Sequential** data relationship pattern. The data items (DI_a, DI_b, DI_c) in the data operation pattern are represented as attribute nodes ($attr_a$, $attr_b$, $attr_c$).

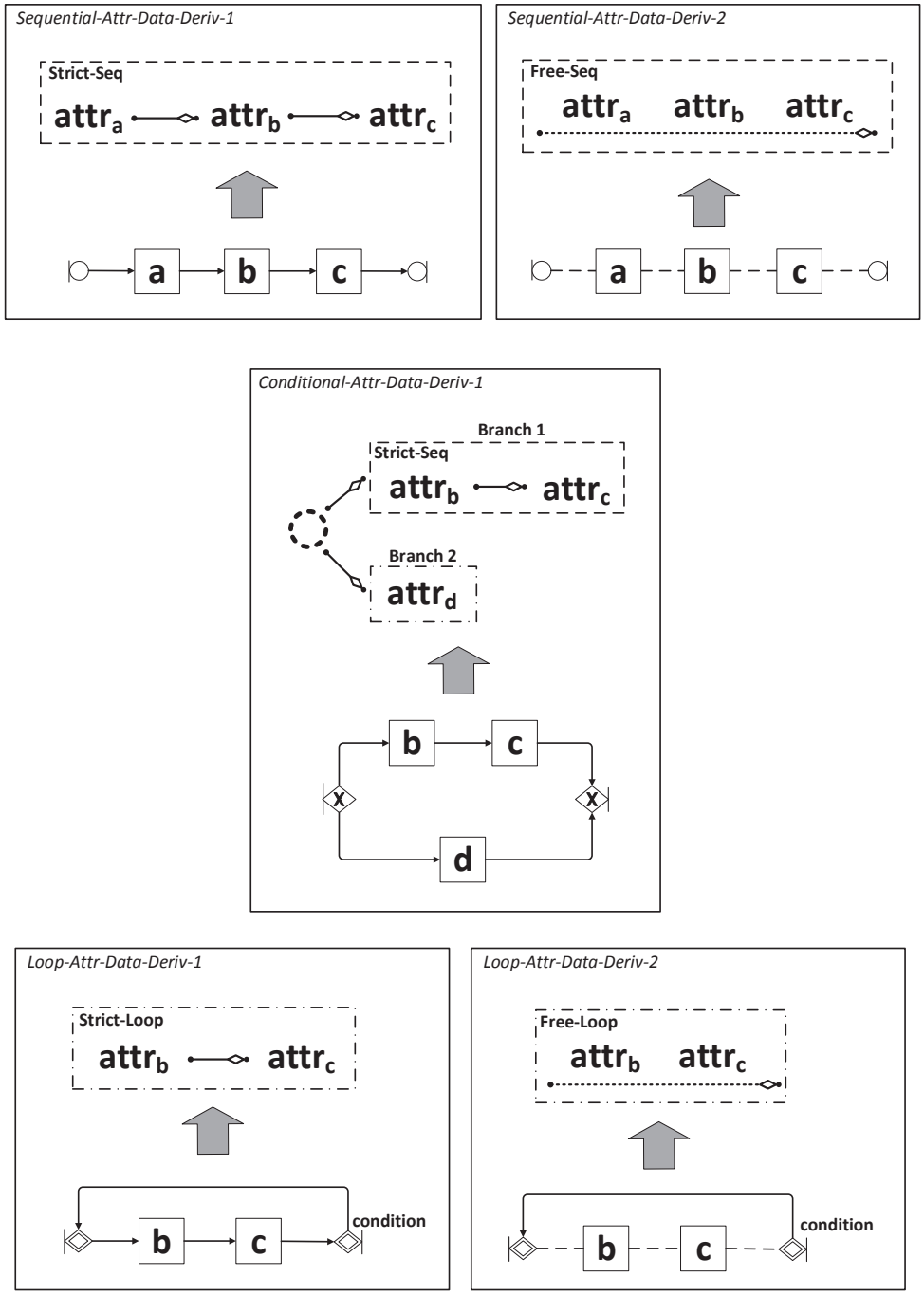


Figure 5.18: Elementary Operations on Data Operation Patterns inside Individual Tasks of AABP

Conditional-Attr-Data-Deriv-1 in Figure 5.18 shows how an elementary operation generates data relationships from a **Conditional** data operation pattern inside a task of an AABP. Each branch of the **Conditional** data operation pattern is inherited by the corresponding data relationship pattern. The data items on each branch (DI_b and DI_c on the upper branch, DI_d on the lower branch) is represented as attribute nodes (DI_b and DI_c as $attr_b$ and $attr_c$, DI_d as $attr_d$).

Loop-Attr-Data-Deriv-1 Figure 5.18 shows how an elementary operation generates data relationships from a **Loop** data operation pattern in an AABP. In this situation, this data operation pattern is inherited by the **Strict-order Loop** data relationship pattern. The data items (DI_b, DI_c) in the data operation pattern are represented as attribute nodes ($attr_b$, $attr_c$), and the order between these attribute nodes inherits the order between the data items.

Loop-Attr-Data-Deriv-2 Figure 5.18 shows how an elementary operation generates data relationships from a **Loop** data operation pattern in an AABP. All the data items inside this loop have **Free-order Sequential** pattern. In this situation, this data operation pattern is inherited by the **Free-order Loop** data relationship pattern. The data items (DI_b, DI_c) in the data operation pattern pattern are represented as attribute nodes ($attr_b$, $attr_c$).

5.3.2 Algorithm for Data Relationship Extraction

In order to extract the data relationships from an AABP, the control flow patterns of the AABP and the data operation patterns inside each individual task are identified and elementary operations are applied accordingly. A tree graph representing the extracted data relationships is built up by using **Algorithm 4**. This algorithm comprises two functions `DeriveDataRelationships` that deals with the control flow relations between tasks and abstracted nodes in a AABP, and `HandleDataOperationFlowInTask` that deals with the data operation flow inside tasks of the AABP.

The function `DeriveDataRelationships` is utilized to extract the data relationships from the an AABP. The input is an AABP for user role r, and the output is a tree fragment set that specifies the data relationships extracted from the input AABP.

Algorithm 4: Data Relationship Extraction

Input : an $AABP$ for user role r

Output: a $TreeGraph$

```
1  Function DeriveDataRelationships(AbsAggBusinessProcess aabp)
2      TreeGraph = ∅;
3      identify cfPattern and cfElementSet at coarsest granularity level of aabp;
4      foreach cfElement in cfElementSet do
5          if cfElement is Task then
6              get data operation flow df_cfElement from cfElement;
7              result = HandleDataOperationFlowInTask(df_cfElement);
8              add result to TreeGraph;
9          else if cfElement is ComplexStructure then
10             cfElement = DeriveDataRelationships(cfElement);
11     transform cfElementSet to treeFragment using elementary operation
        according to cfPattern;
12     if cfPattern is on the coarsest granularity level then
13         assign ProcessStructRef to treeFragment;
14     else
15         assign NormalStructRef to treeFragment;
16     add treeFragment to TreeGraph;
17     return TreeGraph;
18 Function HandleDataOperationFlowInTask(DataOperationFlow df)
19     TreeFragmentSet = ∅;
20     identify dfPattern and dfElementSet at coarsest granularity level of df;
21     foreach dfElement in dfElementSet do
22         if dfElement is not DataItem then
23             dfElement = HandleDataOperationFlowInTask(dfElement);
24     transform dfElementSet to treeFragment' using elementary operation
        according to dfPattern;
25     assign NormalStructRef to treeFragment';
26     add treeFragment' to TreeFragmentSet;
27     return TreeFragmentSet;
```

The data relationship extraction of the input AABP is realized in a recursive way. When an abstracted and aggregated business process $AABP$ is input (line 1), the control flow pattern $cfPattern$ on the coarsest granularity level of $AABP$ and the related elements $cfElementSet$ constituting the $cfPattern$ are both identified (line 3). A tree graph $TreeGraph$ is initialized (line 19). If an identified element $cfElement$ in $cfElementSet$ is task (line 5), we find the data operation flow $df\_cfElement$ from this task $cfElement$ (line 6). The function `TransformDataFlowInTask` is used to extract the tree graph fragment set from the $df\_cfElement$ (line 7), and this set is added to the initialized $TreeGraph$ (line 8). When $cfElementSet$ does not contain any complex control flow structure, we transform the control flow on the coarsest granularity level of the $AABP$ into a tree graph fragment. The first step is to find suitable elementary operation from Subsection 5.3.1 according to the identified $cfPattern$. This found elementary operation extracts the data relationships $treeFragment$ from $cfElementSet$ (line 11). The second step is to assign a structure reference to the the extracted $treeFragment$: if the $cfPattern$ is on the coarsest granularity level of $AABP$, the $ProcessStructRef$ is assigned to the $treeFragment$ (line 12 and line 13); otherwise, a $NormalStructRef$ is assigned to the $treeFragment$ (line 14 and line 15). The third step is to add the $treeFragment$ to the $TreeGraph$ (line 16). Till now, the data relationship extraction of the coarsest granularity level of $AABP$ is completed. If an identified element $cfElement$ in $cfElementSet$ is a complex control flow structure (line 9), we input this $cfElement$ into the function `DeriveDataRelationships` for recursive processing. The recursively processed result $cfElement$ from the function `TransformDataFlowInTask` replaces the input $cfElement$ (line 24). After all the complex structure are handled, the extracted $TreeGraph$ is returned as the output of `DeriveDataRelationships` (line 17).

The function `TransformDataFlowInTask` is adopted to extract the data relationships from the data operation flow inside a task of the AABP. The input is a data operation flow that specifies the execution flows of data items operated inside a tasks of the AABP, and the output is a tree fragment set that specifies the extracted data relationships.

The data relationship extraction inside a task of the AABP is realized in a recursive way. When a data operation flow df is input (line 18), the data operation pattern $dfPattern$ on the coarsest granularity level of df and the related elements $dfElementSet$ constituting the $dfPattern$ are both identified (line 20). A set of tree graph fragments $TreeFragmentSet$ are also initialized (line 19). If each of identified element in $dfElementSet$ is data item (line 21), we transform the data operation flow on the coarsest granularity level into a tree graph fragment. The first step is to find suitable elementary operation from Subsection 5.3.1 according to the identified $dfPattern$. This found elementary operation extracts the data relationships $treeFragment'$ from $dfElementSet$ (line 24). The second step is to assign $NormalStructRef$ to the extracted $treeFragment'$. The third step is to add the $treeFragment'$ to the initialized $TreeFragmentSet$ (line 26). Till now, the data relationship extraction of the coarsest granularity level of df is completed. If an identified element $dfElement$ in $dfElementSet$ is a complex data operation flow structure (line 21 and line 22), we input this $dfElement$ into the function `TransformDataFlowInTask` for recursive processing. The recursively processed result $dfElement$ from the function `TransformDataFlowInTask` replaces the input $dfElement$ (line 23). After all the complex structure are handled, the extracted $TreeFragmentSet$ is returned as the output of `TransformDataFlowInTask` (line 27).

5.4 Scenario Example

In this section, we use the scenario example introduced in Figure 1.4 of Section 1.2 to demonstrate the extracted data relationships for three user roles `personnel officer`, `referee`, and `applicant` participating in the recruitment process.

(a) Figure 5.19 shows the data relationships for the user role `personnel officer`. According to elementary operation **Sequential-Data-Deriv-1** in Subsection 5.3.1, ABS_1 in (a) Figure 4.20 is inherited by label Abs_1 in (a) Figure 5.19; and according to elementary operation **Parallel-A-Data-Deriv-3** in Subsection 5.3.1, ABS_2 in (a) Figure 4.20 does not appear in (a) Figure 5.19.

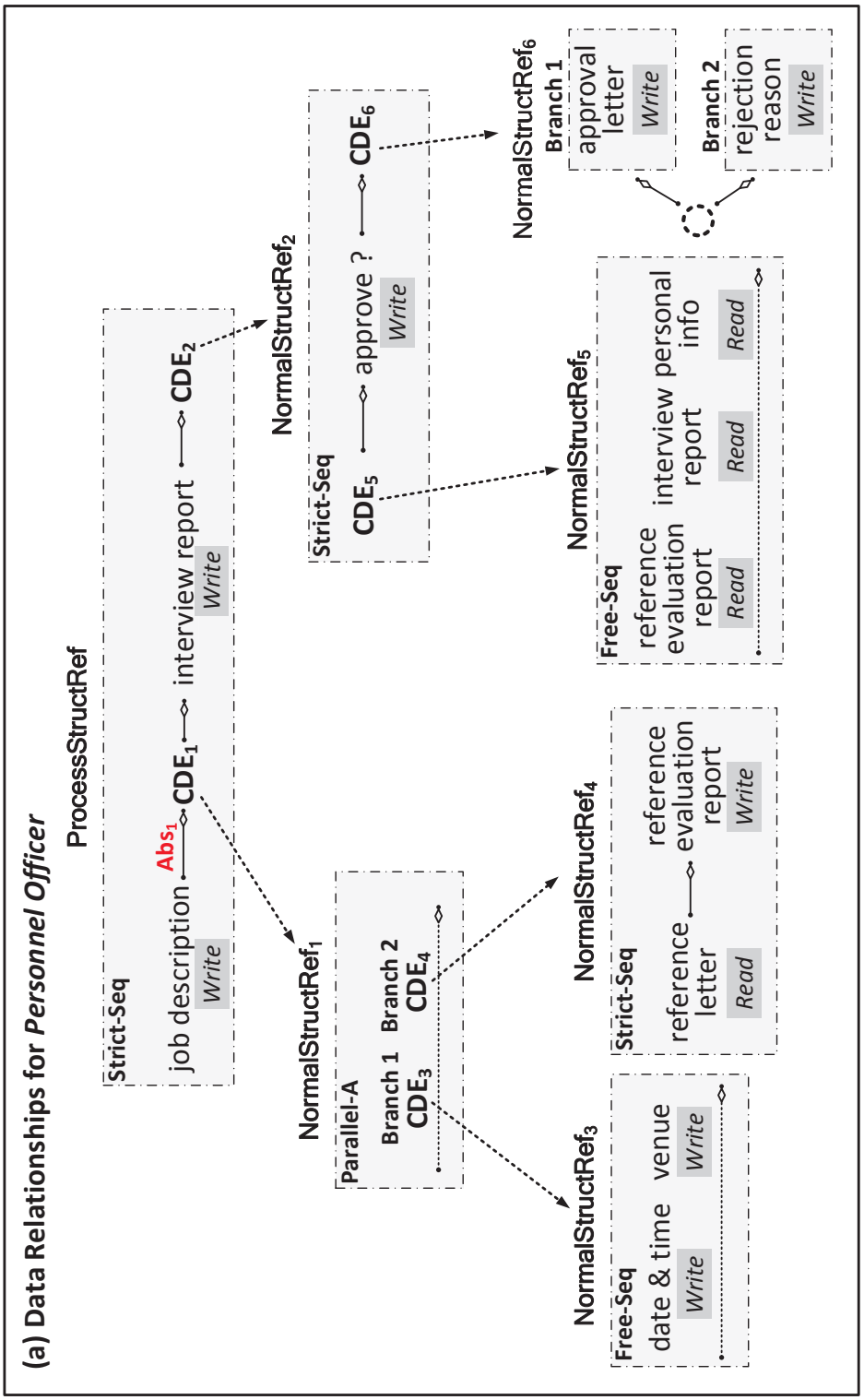


Figure 5.19: Extracted Data Relationships for Personnel Officer

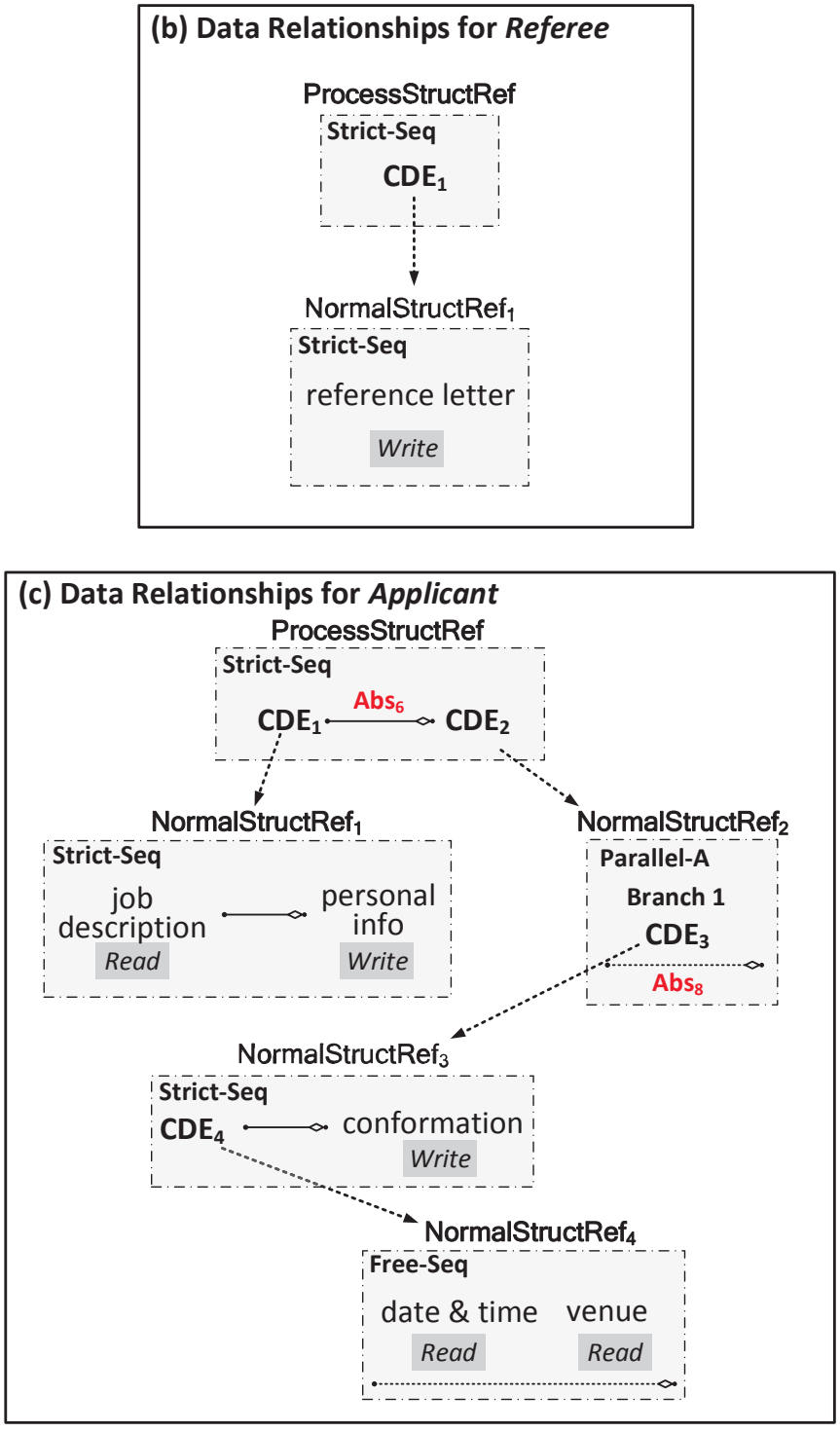


Figure 5.20: Extracted Data Relationships for Referee and Applicant

(b) Figure 5.20 shows the data relationships for the user role for `referee`. According to elementary operation **Sequential-Data-Deriv-1** in Subsection 5.3.1, ABS_3 and ABS_4 in (b) Figure 4.20 do not appear in (b) Figure 5.20.

(c) Figure 5.20 shows the data relationships for the user role for `applicant`. According to elementary operation **Sequential-Data-Deriv-1** in Subsection 5.3.1, ABS_5 and ABS_9 in (c) Figure 4.20 do not appear in (c) Figure 5.20, and ABS_6 in (c) Figure 4.20 is inherited by label Abs_6 in (c) Figure 5.19; according to elementary operation **Parallel-A-Data-Deriv-3** in Subsection 5.3.1, ABS_7 in (c) Figure 4.20 does not appear in (c) Figure 5.19; according to elementary operation **Parallel-A-Data-Deriv-1** in Subsection 5.3.1, ABS_8 in (c) Figure 4.20 is inherited by label Abs_8 in (c) Figure 5.19.

5.5 Summary and Discussion

In this chapter, the method for data relationship extraction is proposed, which is the second step of our UI derivation approach. The data relationships are extracted from the abstracted and aggregated business process for each user role. The data relationships for a particular user role are represented as a tree graph, and we use JSON Strings to record all the details in the tree graph. A set of elementary operations are developed according to the data operations inside individual tasks and the identified control flow patterns in the AABP. The algorithm for data relationship extraction is built up with the elementary operations as cornerstones. We also use the recruitment process as a scenario example to demonstrate the extracted data relationships for three participating user roles. The extracted data relationships are the foundation to analyze and derive the UI logic.

Chapter 6

User Interface Derivation

In this chapter, we discuss the user interface derivation, which is the third step of our UI derivation approach as shown in Figure 1.5. Section 6.1 provides an introduction of this chapter. Section 6.2 introduces the user interface flow. Two subsections are included as formal specification of the user interface flow, and operation flow relations between UI containers. Section 6.3 discusses the rules of UI derivation including constraints and recommendations. Section 6.4 provides the algorithm for UI derivation. Section 6.5 introduces a scenario example. Section 6.6 provides a summary and discussion on this chapter.

6.1 Introduction

After the data relationships are extracted from an AABP for a user role, the UI logic for the user role is able to be analyzed and derived. The UI logic for a user role is represented as a user interface flow. A UI flow comprises a set of UI containers that holds the maximum amount of data items to be operated by a user role, and the operation flow relations between the UI containers. In order to derive such UI flow, a set of UI derivation rules are introduced as constraints and recommendations. The algorithm for UI derivation is developed by utilizing these rules.

6.2 User Interface Flow

The UI flow has two granularity levels: the operation flow between UI containers, and data items included inside each UI container. Each data item needs to be specified with the **access type** including *read* and *write*. A UI container holds the maximum amount of data items to be operated by a user role. Basically, a UI container holds a set of data items that can be directly implemented as a single web page containing the data items at its upper limit. In this situation, the entire set of data items of the container will be shown to end users. Alternatively, the UI designers can divide this container into sub-containers, and implement these sub-containers as separate pages. The UI containers can have operation flow relations as: **Strict-order Sequential**, **Free-order Sequential**, **Conditional**, **Strict-order Loop**, or **Free-order Loop**.

6.2.1 Formal Specification

Specification 1: UI Container. A UI container is denoted as $con = (DI,$ $AccessType)$, where:

- $DI = \{di_1, di_2, ..., di_n\}$ is a finite set of data items.
- $AccessType : DI \rightarrow \{read, write\}$ assigns each data item an access type.

Specification 2: UI Flow. A UI flow for user role r is denoted as $UIF^r = (CON^r,$ $UIRelation^r_{Flow})$, where:

- CON^r is a finite set of UI containers.
- $UIRelation^r_{Flow} \subseteq (CON^r \times CON^r)$ is a finite set of operation flow relations between UI containers. An operation flow relation is **Strict-order Sequential**, **Free-order Sequential**, **Conditional**, **Strict-order Loop**, or **Free-order Loop**.

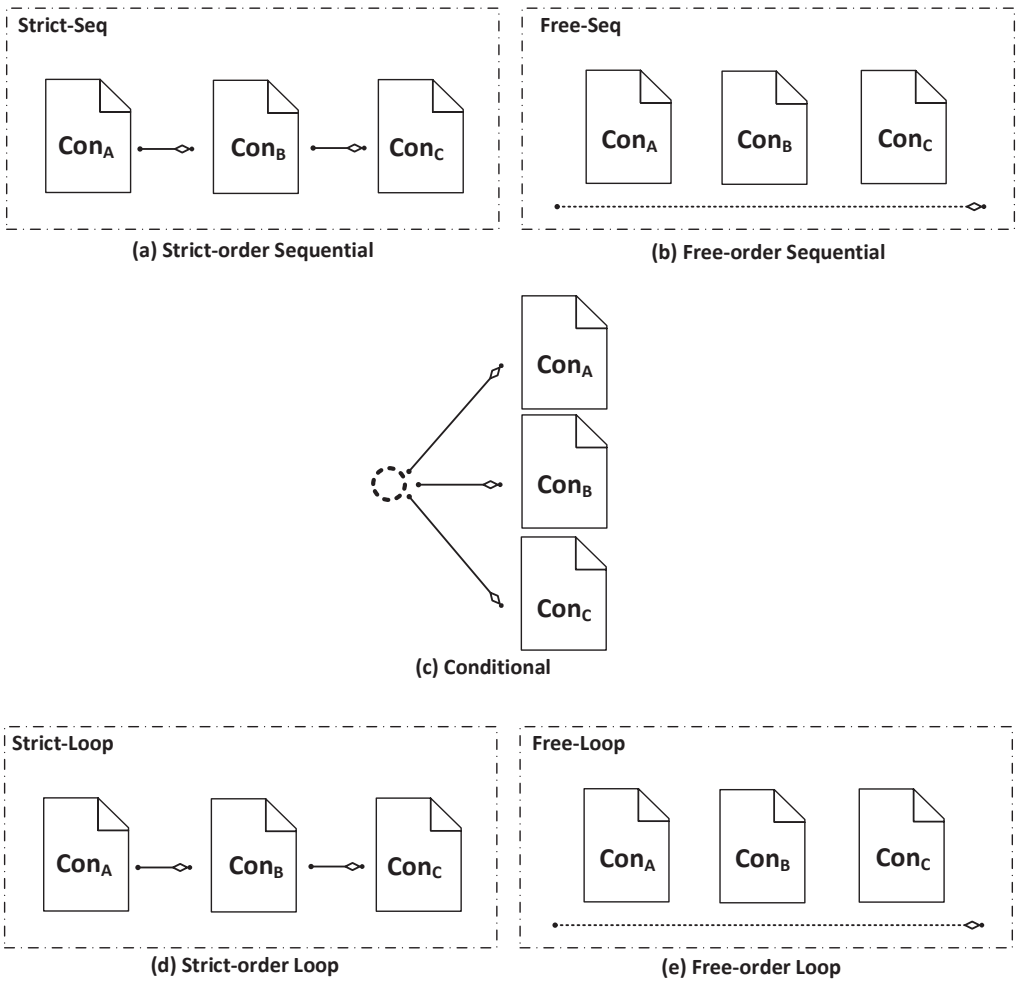


Figure 6.1: Operation Flow Relations between UI Containers

6.2.2 Operation Flow Relations between UI Containers

Figure 6.1 describes the graphical representations of five types of operation flow relations between UI containers.

- **Strict-order Sequential** ((a) Figure 6.1) is represented with a dot-dash rectangle, a pattern name `Strict-Seq` on the upper left of the rectangle, and the relationships between UI containers. The container relationships are represented by using an edge with two dots on each end to connect two containers. A diamond on one end of the line is used to point to the following container.

- **Free-order Sequential** ((b) Figure 6.1) is represented with a dot-dash rectangle, a pattern name `Free-Seq` on the upper left of the rectangle, and the relationships between UI containers. The container relationships are represented by using a dashed edge with two dots on each end, and the containers are above the dashed edge. There is a diamond on one end of the dashed edge.

- **Conditional** ((c) Figure 6.1) is represented with a `Virtual Node` and a set of branches. Each branch is a UI container.

- **Strict-order Loop** ((d) Figure 6.1) is represented by using with a dot-dash rectangle, a pattern name `Strict-Loop` on the upper left of the rectangle, and the relationships between UI containers. The container relationships are represented by using an edge with two dots on each end to connect two containers. A diamond on one end of the line is used to point to the following container.

- **Free-order Loop** ((e) Figure 6.1) is represented with a dot-dash rectangle, a pattern name `Free-Loop` on the upper left of the rectangle, and the relationships between UI containers. The container relationships are represented by using a dashed edge with two dots on each end, and the containers are above the dashed edge. There is a diamond on one end of the dashed edge.

6.3 Rules of UI Derivation

This section coins a set of rules for deriving the UI flow from the tree graph. Three principles are enacted in these rules: (1) the `Parallel-A` data relationship pattern is transformed into `Free-order Sequential` operation flow relations between UI containers in the UI flow; (2) the branch entities in **Parallel-A**, **Parallel-B**, **Parallel-C**, **Conditional** data relationship patterns are inherited by UI containers in the UI flow; (3) the abstracted nodes in the tree graph are parsed as the rules of constraints.

These UI derivation rules can be classified into two categories as **Constraints** and **Recommendations**. The **Constraints** include rules that must be followed by the UI designers. The **Recommendations** include rules that are recommended to be followed by the UI designers. This means if the recommended rules are not implemented, the derived UI logic will not be affected and will still reflect the process logic. If the recommended rules are applied, the data relationships in the UI logic, which do not determine the process logic, will sufficiently reflect the data relationships in the process. Each of these rules has been formalized. In the following, we will introduce these rules in detail. Note that the "Node" label in each figure of the UI derivation rule can represent either an `Attribute Node`, an `Information Node`, or a `CDE Node` in the tree graph.

6.3.1 Constraints

Sequential-Constraint-1:

$\forall m$: "$TreeGraph[m]$"."$dataRelationshipPattern$" == "$Strict - Seq$",

iff $\forall i, j$: "$TreeGraph[m]$"."$graphNodes[i]$"."$postAbsOfNode$" ==

"$TreeGraph[m]$"."$graphNodes[j]$"."$preAbsOfNode$",

then ("$TreeGraph[m]$"."$graphNodes[i]$" $\in con_i$) $\wedge$

("$TreeGraph[m]$"."$graphNodes[j]$" $\in con_j$) $\wedge$ $((con_i, con_j) ==$ "$Strict - Seq$").

Sequential-Constraint-1 (Figure 6.2) shows how a rule of constraint derives UI flow from the **Strict-order Sequential** data relationship pattern. In this situation, if there exists an `Abs Label` between two adjacent nodes (Abs_B between $Node_A$ and

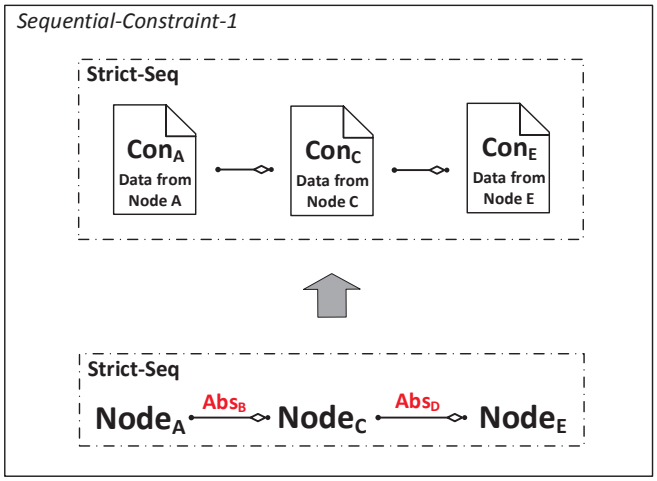


Figure 6.2: Constraint Sequential-Constraint-1 on Strict-order Sequential

$Node_C$, Abs_D between $Node_C$ and $Node_E$), these two nodes must be separated into different containers (Data in $Node_A$, $Node_C$, $Node_E$ are put into Con_A, Con_C, Con_E respectively). And these containers inherit the data relationship pattern between the nodes. In a business process, $Node_A$, $Node_C$, and $Node_E$ correspond to three individual tasks (say t_1, t_3, and t_5) participated by one user role (say r_1. Abs_B and Abs_D correspond to the tasks (say t_2 and t_4) not participated by r_1. Due to duplication, here we just take t_1, t_2, and t_3 for analysis. When r_1 completes t_1, t_2 must be done by other user roles; only if t_2 is completed, can r_1 carry out t_3. This means that r_1 cannot operate the data in t_1 and t_3 in parallel within a single container. Therefore the operated data in t_1 and t_3 must be put in different containers. And the data from the nodes $Node_A$ and $Node_C$ must be separated into different containers Con_A, Con_C. This also works with $Node_C$ and $Node_E$.

Sequential-Constraint-2:

$\forall m$: “$TreeGraph[m]$”.“$dataRelationshipPattern$” == “$Free - Seq$”,

iff $\exists i$: (“$TreeGraph[m]$”.“$graphNodes[i]$”.“$preAbsOfNode$” $\neq$ “$null$”),

then $\forall k$ **where** $(k \geq 0) \wedge$ (“$TreeGraph[m]$”.“$graphNodes[k{+}1]$” $\in$ “$TreeGraph[m]$”):

(“$TreeGraph[m]$”.“$graphNodes[k]$” $\in con_k$) $\wedge$

(“$TreeGraph[m]$”.“$graphNodes[k{+}1]$” $\in con_{k+1}$) $\wedge$ $((con_k, con_{k+1}) ==$ “$Free{-}Seq$”).

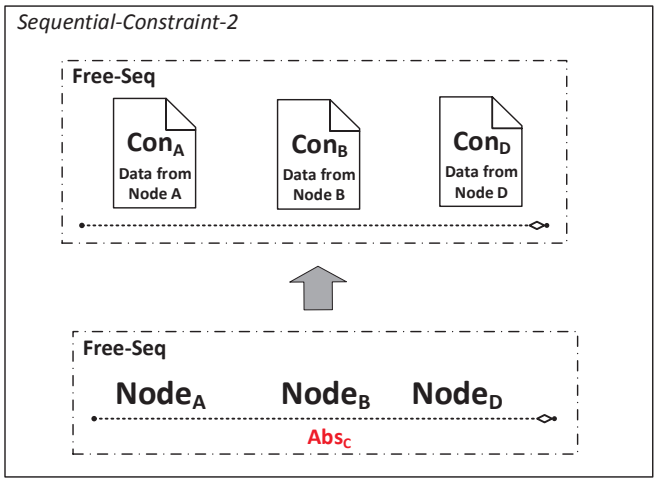


Figure 6.3: Constraint Sequential-Constraint-2 on Free-order Sequential

Sequential-Constraint-2 (Figure 6.3) shows how a rule of constraint derives UI flow from the **Free-order Sequential** data relationship pattern. In this situation, if there exists an `Abs Label` (Abs_C) in the **Free-order Sequential** data relationship pattern, all the nodes in this pattern must be separated into different containers (Data in $Node_A$, $Node_B$, $Node_D$ are put into Con_A, Con_B, Con_D respectively). And these containers inherit the data relationship pattern between the nodes. In a business process, $Node_A$, $Node_B$, and $Node_D$ correspond to three individual tasks (say t_1, t_2, and t_4) participated by one user role (say r_1). Abs_C corresponds to the task (say t_3) not related to r_1. During BP execution, the three tasks t_1, t_2, and t_4 must be executed sequentially, but the execution orders between the three tasks are decided during runtime. Only if the operated data t_1, t_2, and t_4 are put into different containers (Con_A, Con_B, Con_D) which have **Free-order Sequential** relation, can the execution orders between t_1, t_2, and t_4 be decided during runtime. Therefore, this rule holds.

Parallel-A-Constraint-1:

$\forall m$: "$TreeGraph[m]$"."$dataRelationshipPattern$" == "$Parallel - A$",

iff $\exists i$: "$TreeGraphS[m]$"."$graphNodes[i]$"."$preAbsOfBranch$" $\neq$ "$null$",

then $\forall k$ **where** $(k \geq 0) \wedge$ ("$TreeGraph[m]$"."$graphNodes[k{+}1]$" $\in$ "$TreeGraph[m]$"):

("$TreeGraph[m]$"."$graphNodes[k]$" $\in con_k$) $\wedge$

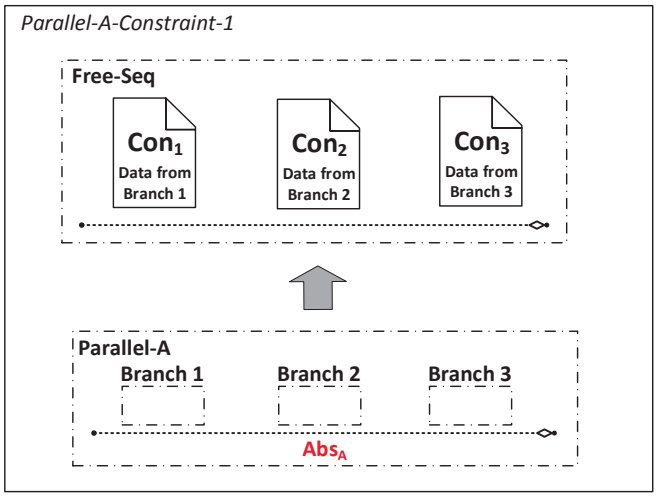


Figure 6.4: Constraint Parallel-A-Constraint-1 on Parallel-A

($\text{“}TreeGraph[m]\text{”}.\text{“}graphNodes[k{+}1]\text{”} \in con_{k+1}) \wedge ((con_k, con_{k+1}) == \text{“}Free{-}Seq\text{”}$).

Parallel-A-Constraint-1 (Figure 6.4) shows how a rule of constraint derives UI flow from the **Parallel-A** data relationship pattern. In this situation, if there exists an `Abs Label` (Abs_A) in the the **Parallel-A** data relationship pattern, the data entities from all the branches of the **Parallel-A** data relationship pattern must be separated into different containers (Data in $Branch_1$, $Branch_2$, $Branch_3$ are put into Con_1, Con_2, Con_3 respectively). And these containers have **Free-order Sequential** operation flow relations. In a business process, the `Branch` data entities ($Branch_1$, $Branch_2$, and $Branch_3$) of the tree graph correspond to the branches ($Branch_A$, $Branch_B$, and $Branch_C$) of the **Parallel-A** control flow pattern, and each of the branches of the pattern contains at least one task participated by one user role (say r_1). The abstracted node Abs_A represents a branch (say $Branch_D$), whose tasks are not participated by r_1, and this implies the business rules in $Branch_A$, $Branch_B$, $Branch_C$, $Branch_D$ of **Parallel-A** control flow pattern are strongly independent from each other. Therefore the operated data in each branch must be put into different containers, so that we can avoid non-related data to appear in the same container. On the other hand, $Branch_A$, $Branch_B$ and $Branch_C$ are executed by r_1. r_1 is only able to execute $Branch_A$, $Branch_B$, and $Branch_C$ one by one, which equals to that $Branch_A$, $Branch_B$ and

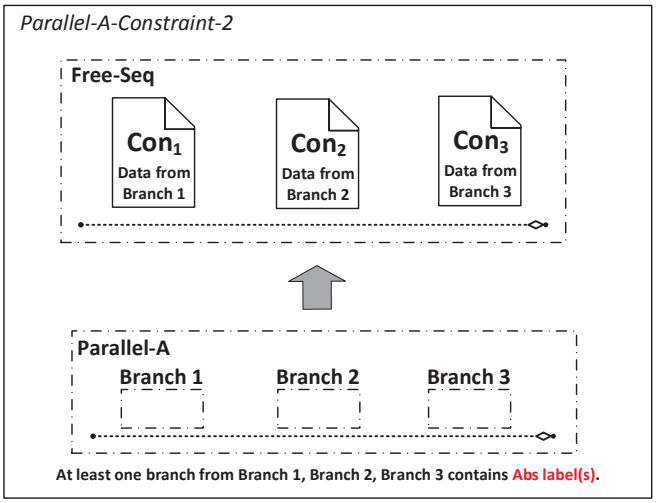


Figure 6.5: Constraint Parallel-A-Constraint-2 on Parallel-A

$Branch_C$ have **Free-order Sequential** relation for r_1. Therefore, the containers Con_A, Con_B, Con_C holding the data from $Branch_A$, $Branch_B$ and $Branch_C$ must have **Free-order Sequential** relation.

Parallel-A-Constraint-2:

$\forall m$: "$TreeGraph[m]$"."$dataRelationshipPattern$" == "$Parallel-A$",

iff $\exists i, j$: ("$TreeGraph[m]$"."$graphNodes[i]$"."$preAbsOfBranch$" == "$null$") $\wedge$

[("$TreeGraph[m]$"."$graphNodes[i]$"."$branchNodes[j]$"."$preAbsInBranch$" $\neq$ "$null$") $\vee$

("$TreeGraph[m]$"."$graphNodes[i]$"."$branchNodes[j]$"."$postAbsInBranch$" $\neq$ "$null$")],

then $\forall k$ **where** $(k \geq 0) \wedge$ ("$TreeGraph[m]$"."$graphNodes[k{+}1]$" $\in$ "$TreeGraph[m]$"):

("$TreeGraph[m]$"."$graphNodes[k]$" $\in con_k$) $\wedge$

("$TreeGraph[m]$"."$graphNodes[k{+}1]$" $\in con_{k+1}$) $\wedge$ $((con_k, con_{k+1})$ == "$Free{-}Seq$").

Parallel-A-Constraint-2 (Figure 6.5) shows how a rule of constraint derives UI flow from the **Parallel-A** data relationship pattern, where at least one branch data entity ($Branch_1$, $Branch_2$, $Branch_3$) contains `Abs Label`. In this situation, the data entities from all the branches of the **Parallel-A** data relationship pattern must be separated into different containers (Data in $Branch_1$, $Branch_2$, $Branch_3$ are put into Con_1, Con_2, Con_3 respectively). And these containers have **Free-order Sequential** operation flow relations. In a business process, the `Branch` data entities ($Branch_1$, $Branch_2$,

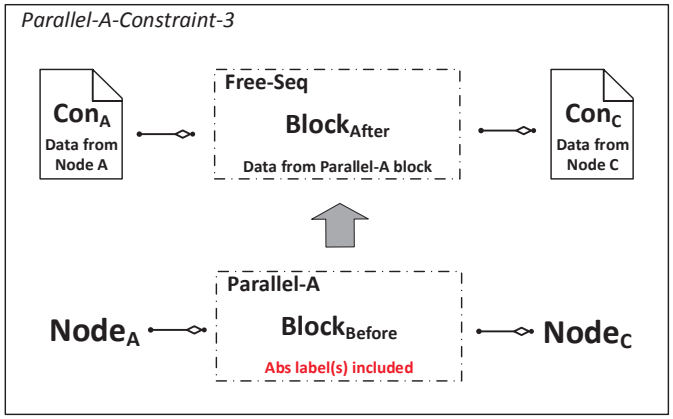


Figure 6.6: Constraint Parallel-A-Constraint-3 on Parallel-A

and $Branch_3$) of the tree graph corresponds to the branches ($Branch_A$, $Branch_B$, and $Branch_C$) of the **Parallel-A** control flow pattern. There exists at least one branch (say $Branch_1$) containing at least one task not participated by one user role (say r_1). This implies the business rules in $Branch_A$, $Branch_B$, $Branch_C$ of **Parallel-A** control flow pattern are strongly independent from each other. Therefore the operated data in each branch must be put into different containers, so that we can avoid non-related data to appear in the same container. On the other hand, both $Branch_A$, $Branch_B$ and $Branch_C$ are executed by r_1. r_1 is only able to execute $Branch_A$, $Branch_B$ and $Branch_C$ one by one, which equals to that $Branch_A$, $Branch_B$ and $Branch_C$ are have **Free-order Sequential** relation for r_1. Therefore, the containers holding the data from $Branch_1$, $Branch_2$ and $Branch_3$ are have **Free-order Sequential** relation.

Parallel-A-Constraint-3:

$\forall m$: “$TreeGraph[m]$”.“$dataRelationshipPattern$” == “$arbitary$”,

iff $\forall i \wedge \exists n, p$(“$TreeGraph[m]$”.“$graphNodes[i]$”.“$nodeType$” == “$cde$”) $\wedge$

(“$TreeGraph[n]$”.“$dataRelationshipPattern$” == “$Parallel-A$”,

where “$TreeGraph[n]$”.“id” == “$TreeGraph[m]$”.“$graphNodes[i]$”) $\wedge$

(“$TreeGraph[n]$”.“$graphNodes[p]$”.$preAbsOfBranch \neq$ “$null$”),

then $\forall j \neq i$: (“$TreeGraph[m]$”.“$graphNodes[i]$” $\in con_i$) $\wedge$

(“$TreeGraph[m]$”.“$graphNodes[j]$” $\in con_j$) $\wedge$

$((con_i, con_j) ==$ “$dataRelationshipPattern$”).

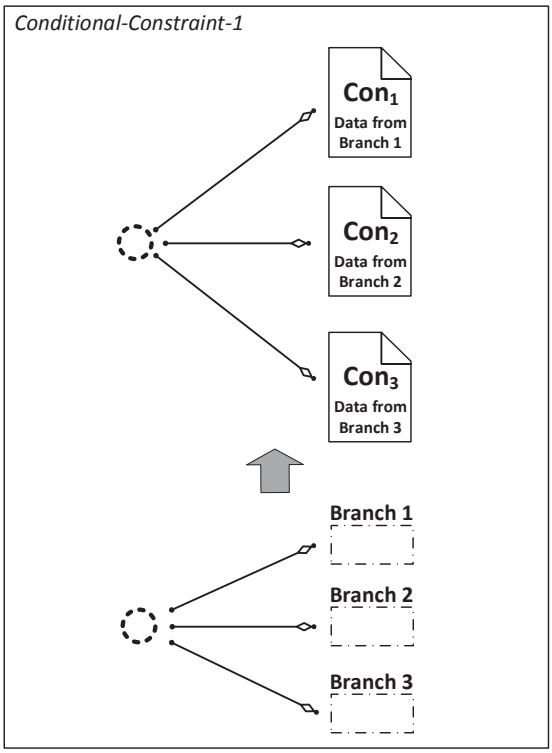


Figure 6.7: Constraint Conditional-Constraint-1 on Conditional

Parallel-A-Constraint-3 (Figure 6.6) shows how a rule of constraint derives UI flow from the **Parallel-A** data relationship pattern and its adjacent nodes. In this situation, if there exists an `Abs Label` in the the **Parallel-A** data relationship pattern, the precedent ($Node_A$) and subsequent ($Node_C$) nodes of the data relationship pattern must be separated into different containers (Data in $Node_A$, $Node_C$ are put into Con_A, Con_C respectively). According to **Parallel-A-Constraint-1** and **Parallel-A-Constraint-2**, the data in `Parallel-A` data operation pattern are put in separate containers. This means that the data from Con_A, $Block_{Before}$, Con_C will never be put into the same container. Therefore the data from the outside nodes $Node_A$, $Node_C$ must be also be put in separate containers.

Conditional-Constraint-1:

$\forall m$, **iff** "$TreeGraph[m]$"."$dataRelationshipPattern$" == "$Conditional$",

then $\forall k$: "$TreeGraph[m]$"."$graphNodes[k]$" $\in con_k \wedge$

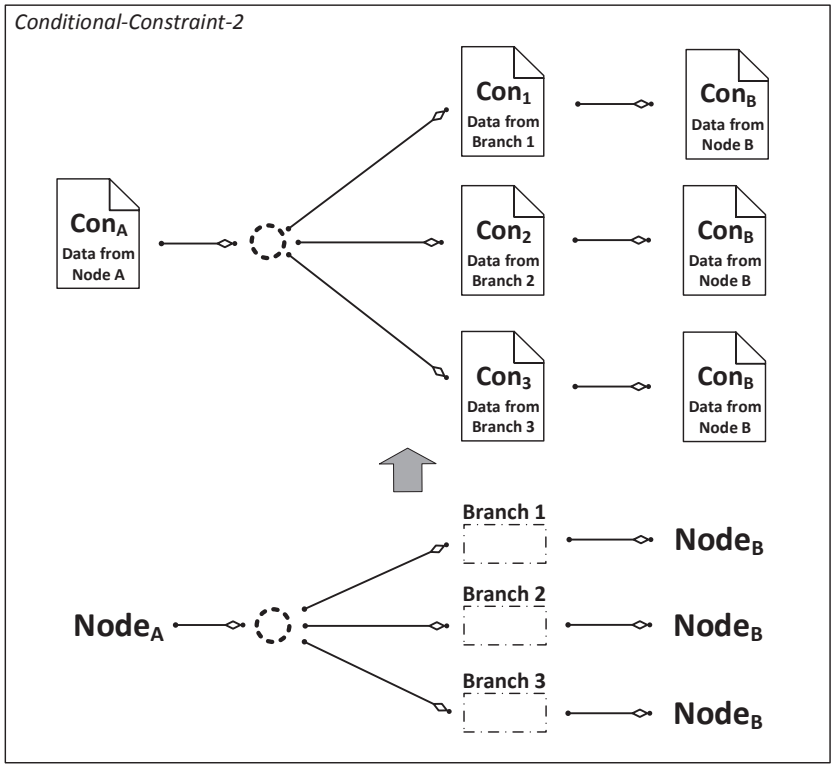


Figure 6.8: Constraint Conditional-Constraint-2 on Conditional

$((con_k, con_{k+1}) == \text{“}Conditional\text{”})$.

Conditional-Constraint-1 (Figure 6.7) shows how a rule of constraint derives UI flow from the **Conditional** data relationship pattern. In this situation, the data entities from all the branches of the **Conditinal** data relationship pattern must be separated into different containers (Data in $Branch_1$, $Branch_2$, $Branch_3$ are put into Con_1, Con_2, Con_3 respectively). And these containers have the **Conditional** data relationship pattern. The data from $Branch_1$, $Branch_2$ and $Branch_3$ are related to all the possible results after making choices. During runtime, after a choice is made by a user role, the data from one and only one branch will be available for this user role. Therefore, this rule holds.

Conditional-Constraint-2:

$\forall m$: $\text{“}TreeGraph[m]\text{”}.\text{“}dataRelationshipPattern\text{”} == \text{“}arbitrary\text{”}$,

iff $\forall i \ \wedge \ \exists \ n$: $(\text{“}TreeGraph[m]\text{”}.\text{“}graphNodes[i]\text{”}.\text{“}nodeType\text{”} == \text{“}cde\text{”}) \wedge$

(“$TreeGraph[n]$”.“$dataRelationshipPattern$” == “$Conditional$”,
where “$TreeGraph[n]$”.“id” == “$TreeGraph[m]$”.“$graphNodes[i]$”),
then $\forall j \neq i$: (“$TreeGraph[m]$”.“$graphNodes[i]$” $\in con_i$) $\wedge$
(“$TreeGraph[m]$”.“$graphNodes[j]$” $\in con_j$) $\wedge$
$((con_i, con_j)$ == “$dataRelationshipPattern$”).

Conditional-Constraint-2 (Figure 6.8) shows how a rule of constraint derives UI flow from the **Conditional** data relationship pattern and its adjacent nodes. In this situation, the precedent ($Node_A$) and subsequent ($Node_C$) nodes of the data relationship pattern must be separated into different containers (Data in $Node_A$ are put into Con_A, data in three $Node_B$s are put into three Con_Bs respectively). The data from $Node_A$ are related to the the data about how to making choice from the different options in the `Conditional` branches; the data from $Branch_1$, $Branch_2$ and $Branch_3$ are related to all the possible results after making choices; and The data from $Node_Bs$ are related to the the data about the operation after the choice is made from the different options in the `Conditional` branches. During runtime, when a user role is operating the data from $Node_A$, the data from each branch should not be available for this user until a choice is made. Meanwhile, when a user role is operating the data from one of his choices ($Branch_1$, $Branch_2$, or $Branch_3$), this user role cannot touch the data until the operation in the choice is completed. Therefore, this rule holds.

Loop-Constraint-1:
$\forall m$: “$TreeGraph[m]$”.“$dataRelationshipPattern$” == “$Strict - Loop$”,
iff $\forall i, j$: “$TreeGraph[m]$”.“$graphNodes[i]$”.“$postAbsOfNode$” ==
“$TreeGraph[m]$”.“$graphNodes[j]$”.“$preAbsOfNode$”,
then (“$TreeGraph[m]$”.“$graphNodes[i]$” $\in con_i$) $\wedge$
(“$TreeGraphSet[m]$”.“$graphNodes[j]$” $\in con_j$) $\wedge$ $((con_i, con_j)$ == “$Strict - Loop$”).

Loop-Constraint-1 (Figure 6.9) shows how a rule of constraint derives UI flow from the **Strict-order Loop** data relationship pattern. In this situation, if there exists an `Abs Label` between two adjacent nodes (Abs_B between $Node_A$ and $Node_C$, Abs_D between $Node_C$ and $Node_E$), these two nodes must be separated into different containers (Data in $Node_A$, $Node_C$, $Node_E$ are put into Con_A, Con_C, Con_E respectively).

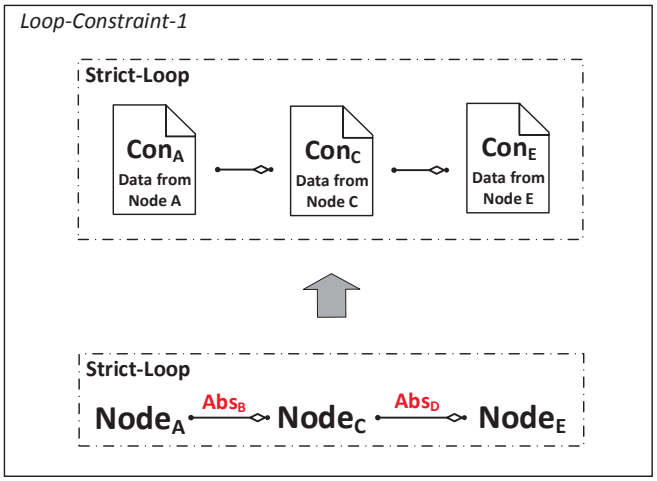


Figure 6.9: Constraint Loop-Constraint-1 on Strict-Order Loop

And these containers inherit the data relationship pattern between the nodes. In a business process, $Node_A$, $Node_C$, and $Node_E$ correspond to three individual tasks (say t_1, t_3, and t_5) participated by one user role (say r_1). Abs_B and Abs_D correspond to the tasks (say t_2 and t_4) not participated by one r_1. Due to duplication, here we just take t_1, t_2, and t_3 for analysis. When r_1 completes t_1, t_2 has to be done by other user roles; only if t_2 is completed, can r_1 carry out t_3. This means that r_1 cannot operate the data in t_1 and t_3 in parallel within a single container. Therefore the operated data t_1 and t_3 must be put in different containers. And the data from the corresponding nodes $Node_A$ and $Node_C$ must be separated into different containers Con_A, Con_C. This also works with $Node_C$ and $Node_E$.

Loop-Constraint-2:

$\forall m$: “$TreeGraph[m]$”.“$dataRelationshipPattern$” == “$arbitrary$”,

iff $\forall i \ \wedge \ \exists \ n$: (“$TreeGraph[m]$”.“$graphNodes[i]$”.“$nodeType$” == “$cde$”) $\wedge$

(“$TreeGraph[n]$”.“$dataRelationshipPattern$” == “$Strict - Loop$”,

where “$TreeGraph[n]$”.“id” == “$TreeGraph[m]$”.“$graphNodes[i]$”),

then $\forall j \neq i$: (“$TreeGraph[m]$”.“$graphNodes[i]$” $\in con_i$) $\wedge$

(“$TreeGraph[m]$”.“$graphNodes[j]$” $\in con_j$) $\wedge$

$((con_i, con_j)$ == “$dataRelationshipPattern$”).

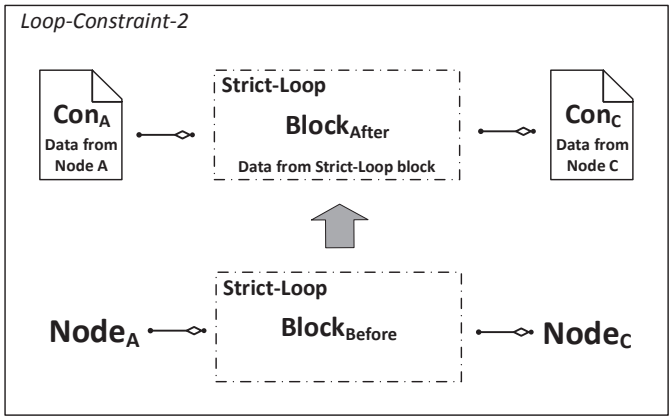


Figure 6.10: Constraint Loop-Constraint-2 on Strict-Order Loop

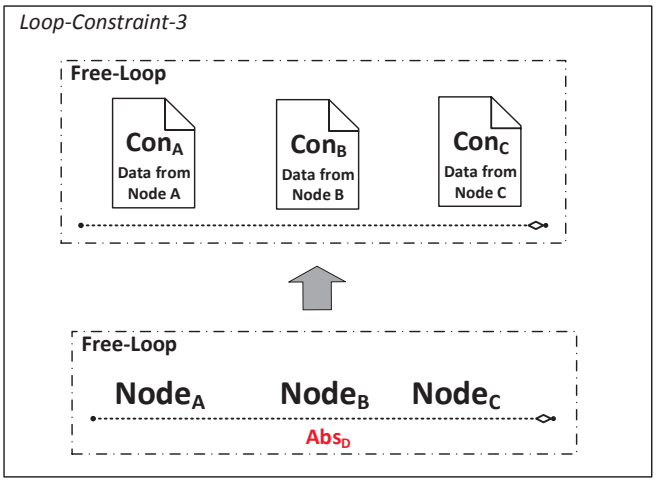


Figure 6.11: Constraint Loop-Constraint-3 on Free-Order Loop

Loop-Constraint-2 (Figure 6.10) shows how a rule of constraint derives UI flow from the **Strict-order Loop** data relationship pattern and its adjacent nodes. In this situation, the precedent ($Node_A$) and subsequent ($Node_C$) nodes of the data relationship pattern must be separated into different containers (Data in $Node_A$, $Node_C$ are put into Con_A, Con_C respectively). In a business process, the data inside the `Strict-Loop` control flow pattern are executed iteratively, and the times of iteration are decided during runtime. However, the data from the nodes ($Node_A$ and $Node_C$) outside the `Strict-Loop` control flow pattern are executed only once. Therefore, the data from $Block_{After}$, Con_A, and Con_C will never be put in the same UI container.

Loop-Constraint-3:

$\forall m$: "$TreeGraph[m]$"."$dataRelationshipPattern$" == "$Free - Loop$",

iff $\forall i$("$TreeGraph[m]$"."$graphNodes[i]$"."$preAbsOfNode$" $\neq$ "$null$"),

then $\forall k$ **where** $(k \geq 0) \wedge$

("$TreeGraph[m]$"."$graphNodes[k+1]$" $\in$ "$TreeGraph[m]$"):

("$TreeGraph[m]$"."$graphNodes[k]$" $\in con_k$) $\wedge$

("$TreeGraph[m]$"."$graphNodes[k+1]$" $\in con_{k+1}$) $\wedge$

$((con_k, con_{k+1})$ == "$Free - Loop$").

Loop-Constraint-3 (Figure 6.11) shows how a rule of constraint derives UI flow from the **Free-order Loop** data relationship pattern. In this situation, if there exists an `Abs Label` (Abs_D) in the **Free-order Loop** data relationship pattern, all the nodes in this pattern must be separated into different containers (Data in $Node_A$, $Node_B$, $Node_C$ are put into Con_A, Con_B, Con_C respectively). And these containers inherit the data relationship pattern between the nodes. In a business process, $Node_A$, $Node_B$, and $Node_C$ correspond to three individual tasks (say t_1, t_2, and t_3) participated by user role (say r_1). Abs_D corresponds to the tasks (say t_4) not participated by r_1. During each iteration of the loop execution, the three tasks t_1, t_2, and t_3 must be executed sequentially, but the execution orders between the three tasks are decided during runtime. Only if the operated data t_1, t_2, and t_3 are put into different containers (Con_A, Con_B, Con_C) that have **Free-order Loop** relation, can the execution orders between t_1, t_2, and t_3 be decided during runtime. Therefore, this rule holds.

Loop-Constraint-4:

$\forall m$: "$TreeGraph[m]$"."$dataRelationshipPattern$" == "$arbitrary$",

iff $\forall i \wedge \exists n$: ("$TreeGraph[m]$"."$graphNodes[i]$"."$nodeType$" == "$cde$") $\wedge$

("$TreeGraph[n]$"."$dataRelationshipPattern$" == "$Free - Loop$",

where "$TreeGraph[n]$"."id" == "$TreeGraph[m]$"."$graphNodes[i]$"),

then $\forall j \neq i$: ("$TreeGraph[m]$"."$graphNodes[i]$" $\in con_i$) $\wedge$

("$TreeGraph[m]$"."$graphNodes[j]$" $\in con_j$) $\wedge$

$((con_i, con_j)$ == "$dataRelationshipPattern$").

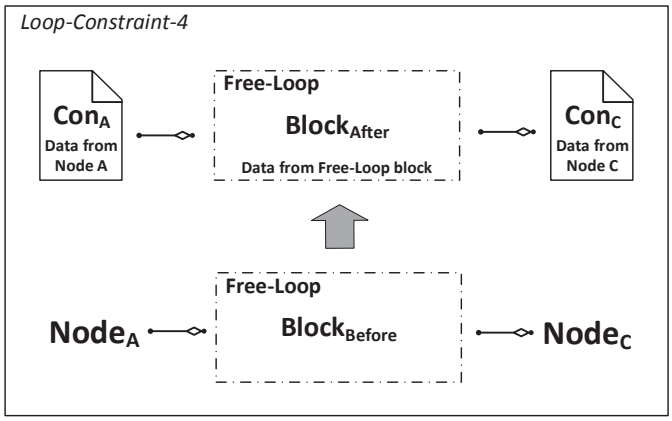


Figure 6.12: Constraint Loop-Constraint-4 on Free-Order Loop

Loop-Constraint-4 (Figure 6.12) shows how a rule of constraint derives UI flow from the **Free-order Loop** data relationship pattern and its adjacent nodes. In this situation, the precedent ($Node_A$) and subsequent ($Node_C$) nodes of the data relationship pattern must be separated into different containers (Data in $Node_A$, $Node_C$ are put into Con_A, Con_C respectively). In a business process, the data inside the `Free-Loop` control flow pattern are executed iteratively, and the times of iteration are decided during runtime. However, the data from the nodes ($Node_A$ and $Node_C$) outside the `Free-Loop` control flow pattern are executed only once. Therefore, the data from $Block_{After}$, Con_A, and Con_C will never be put in the same UI container.

6.3.2 Recommendations

Sequential-Recommendation-1:

$\forall m$: "$TreeGraph[m]$"."$dataRelationshipPattern$" == "$Strict - Seq$",

iff $\forall i$: ("$TreeGraph[m]$"."$graphNodes[i]$"."$preAbsOfNode$" == "$null$") $\wedge$

("$TreeGraph[m]$"."$graphNodes[i]$"."$postAbsOfNode$" == "$null$"),

then $\forall j$ **where** $(j \geq 0) \wedge$

("$TreeGraph[m]$"."$graphNodes[j+1]$" $\in$ "$TreeGraph[m]$"):

("$TreeGraph[m]$"."$graphNodes[j]$" $\in con_j$) $\wedge$

("$TreeGraph[m]$"."$graphNodes[j+1]$" $\in con_{j+1}$) $\wedge$

$((con_j, con_{j+1})$ == "$Strict - Seq$").

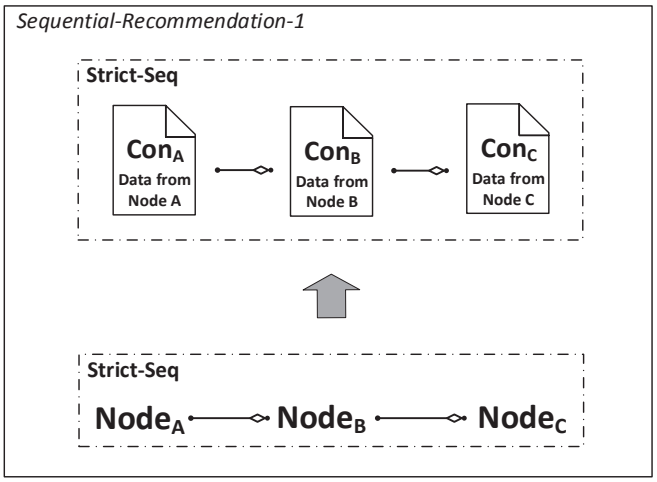


Figure 6.13: Recommendation Sequential-Recommendation-1 on Strict-Order Sequential

Sequential-Recommendation-1 (Figure 6.13) shows how a rule of recommendation derives UI flow from the **Strict-order Sequential** data relationship pattern, which does not contain any `Abs Label`. In this situation, these nodes are recommended to be separated into different containers (Data in $Node_A$, $Node_C$, $Node_E$ are put into Con_A, Con_C, Con_E respectively). And these containers inherit the data relationship pattern between the nodes. The recommendation reason is that in a business process, $Node_A$, $Node_C$, and $Node_C$ correspond to three individual tasks participated by one user role (say r_1), the three tasks indicate three different business rules. Different business rules are recommended to be put in different containers for r_1. In doing so, each separate container can provide r_1 with the data that are strongly related to each other.

Sequential-Recommendation-2:

$\forall m$, **iff** $(\text{“}TreeGraph[m]\text{”}.\text{“}dataRelationshipPattern\text{”} == \text{“}Free - Seq\text{”}) \wedge$

$(\exists i$: $\text{“}TreeGraph[m]\text{”}.\text{“}graphNodes[i]\text{”}.\text{“}preAbsOfNode\text{”} == \text{“}null\text{”})$,

then $\forall j$ **where** $(j \geq 0) \wedge$

$(\text{“}TreeGraph[m]\text{”}.\text{“}graphNodes[j+1]\text{”} \in \text{“}TreeGraph[m])$:

$(\text{“}TreeGraph[m]\text{”}.\text{“}graphNodes[j]\text{”} \in con_j) \wedge$

$(\text{“}TreeGraph[m]\text{”}.\text{“}graphNodes[j+1]\text{”} \in con_{j+1}) \wedge ((con_j, con_{j+1}) == \text{“}Free-Seq\text{”})$.

Sequential-Recommendation-2 (Figure 6.14) shows how a rule of recommen-

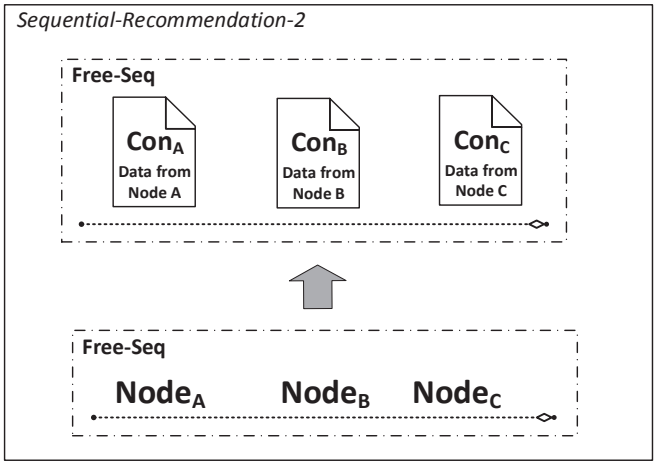


Figure 6.14: Recommendation Sequential-Recommendation-2 on Free-Order Sequential

dation derives UI flow from the **Free-order Sequential** data relationship pattern, which does not contain any `Abs Label`. In this situation, all the nodes in this pattern are recommended to be separated into different containers (Data in $Node_A$, $Node_B$, $Node_D$ are put into Con_A, Con_B, Con_D respectively). And these containers inherit the data relationship pattern between the nodes. The recommendation reason is that in a business process, $Node_A$, $Node_C$, and $Node_C$ correspond to three individual tasks participated by one user role (say r_1), the three tasks indicate three different business rules. Different business rules are recommended to be put in different containers for r_1. In doing so, each separate container can provide r_1 with the data that are strongly related to each other.

Parallel-A-Recommendation-1:

$\forall m$: "$TreeGraph[m]$"."$dataRelationshipPattern$" == "$Parallel - A$",

iff $\forall i, j$: ("$TreeGraph[m]$"."$graphNodes[i]$"."$preAbsOfBranch$" == "$null$") $\wedge$

("$TreeGraph[m]$"."$graphNodes[i]$".

"$branchNodes[j]$"."$preAbsInBranch$" == "$null$") $\wedge$

("$TreeGraph[m]$"."$graphNodes[i]$".

"$branchNodes[j]$"."$postAbsInBranch$" == "$null$"),

then $\forall k$ **where** $(k \geq 0) \wedge$

("$TreeGraph[m]$"."$graphNodes[k+1]$" $\in$ "$TreeGraphSet[m]$"):

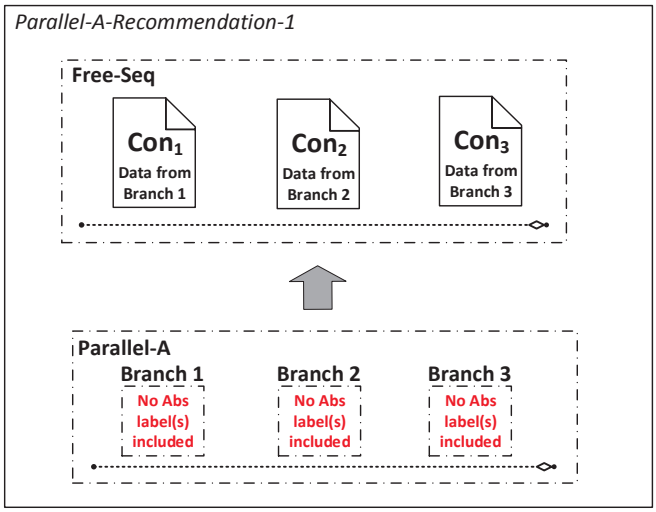


Figure 6.15: Recommendation Parallel-A-Recommendation-1 on Parallel-A

$(\text{“}TreeGraphSet[m]\text{”}.\text{“}graphNodes[k]\text{”} \in con_k) \wedge$

$(\text{“}TreeGraph[m]\text{”}.\text{“}graphNodes[k+1]\text{”} \in con_{k+1}) \wedge$

$((con_k, con_{k+1}) == \text{“}Free - Seq\text{”}).$

Parallel-A-Recommendation-1 (Figure 6.15) shows how a rule of recommendation derives UI flow from the **Parallel-A** data relationship pattern, which does not contain any `Abs Label`. In this situation, the data entities from all the branches of the **Parallel-A** data relationship pattern are recommended to be separated into different containers (Data in $Branch_1$, $Branch_2$, $Branch_3$ are put into Con_1, Con_2, Con_3 respectively). And these containers have **Free-order Sequential** operation flow relations. The recommendation reason is that in a business process, $Branch_1$, $Branch_2$, $Branch_3$ of the tree graph correspond to three individual branches of the **Parallel-A** control flow pattern, and the entire **Parallel-A** control flow pattern is participated by one user role (say r_1). However, the three branches of the **Parallel-A** control flow pattern indicate three different business rules. Different business rules are recommended to be in different containers for r_1. In doing so, each separate container can provide r_1 with the data that are strongly related to each other.

Parallel-A-Recommendation-2:

$\forall m$: $\text{“}TreeGraph[m]\text{”}.\text{“}dataRelationshipPattern\text{”} == \text{“}arbitary\text{”},$

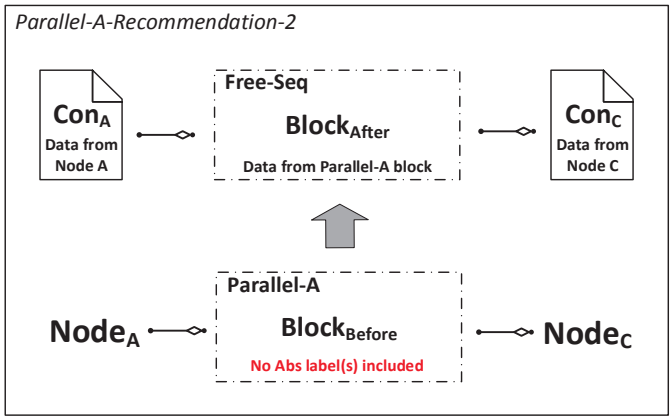


Figure 6.16: Recommendation Parallel-A-Recommendation-2 on Parallel-A

iff $\forall i \ \wedge \ \exists\, n, p$: ($\text{“}TreeGraph[m]\text{”}.\text{“}graphNodes[i]\text{”}.\text{“}nodeType\text{”} == \text{“}cde\text{”}$) $\wedge$
($\text{“}TreeGraph[n]\text{”}.\text{“}dataRelationshipPattern\text{”} == \text{“}Parallel - A\text{”}$, **where**
$\text{“}TreeGraph[n]\text{”}.\text{“}id\text{”} == \text{“}TreeGraph[m]\text{”}.\text{“}graphNodes[i]\text{”}$) $\wedge$
($\text{“}TreeGraph[n]\text{”}.\text{“}graphNodes[p]\text{”}.\text{“}preAbsOfBranch\text{”} == \text{“}null\text{”}$),
then $\forall j \neq i$: ($\text{“}TreeGraph[m]\text{”}.\text{“}graphNodes[i]\text{”} \in con_i$) $\wedge$
($\text{“}TreeGraph[m]\text{”}.\text{”}graphNodes[j]\text{”} \in con_j$) $\wedge$
$((con_i, con_j) == \text{“}dataRelationshipPattern\text{”})$.

Parallel-A-Recommendation-2 (Figure 6.16) shows how a rule of recommendation derives UI flow from the **Parallel-A** data relationship pattern and its adjacent nodes. The **Parallel-A** data relationship pattern does not contain any `Abs Label`. In this situation, the precedent ($Node_A$) and subsequent ($Node_C$) nodes of the data relationship pattern are recommended to be separated into different containers (Data in $Node_A$, $Node_C$ are put into Con_A, Con_C respectively). The recommendation reason is that in a business process, $Block_{Before}$, $Node_A$, and $Node_C$ of the tree graph correspond to the **Parallel-A** control flow pattern, the task t_A before the **Parallel-A** block and the task t_C after **Parallel-A** block. The **Parallel-A** control flow pattern, t_A, t_C are all participated by one user role (say r_1). However, the **Parallel-A** control flow pattern, t_A, t_C indicate three different business rules. Different business rules are recommended to be in different containers for r_1. In doing so, each separate container can provide r_1 with the data that are strongly related to each other.

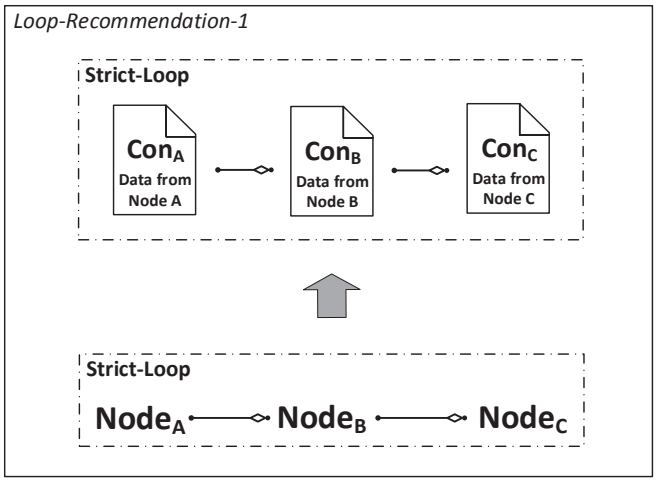


Figure 6.17: Recommendation Loop-Recommendation-1 on Strict-order Loop

Loop-Recommendation-1:

$\forall m$: "$TreeGraph[m]$"."$dataRelationshipPattern$" == "$Strict-Loop$",

iff $\forall i$"$TreeGraph[m]$"."$graphNodes[i]$"."$preAbsOfNode$" == "$null$" $\wedge$

"$TreeGraph[m]$"."$graphNodes[i]$"."$postAbsOfNode$" == "$null$",

then $\forall k$ **where** $(k \geq 0) \wedge$

("$TreeGraph[m]$"."$graphNodes[k+1]$" $\in$ "$TreeGraphSet[m]$"):

("$TreeGraph[m]$"."$graphNodes[k]$" $\in con_k$) $\wedge$

("$TreeGraph[m]$"."$graphNodes[k+1]$" $\in con_{k+1}$) $\wedge$

$((con_k, con_{k+1})$ == "$Strict-Loop$").

Loop-Recommendation-1 (Figure 6.17) shows how a rule of recommendation derives UI flow from the **Strict-order Loop** data relationship pattern, which does not contain any `Abs Label`. In this situation, all the nodes in this pattern are recommended to be separated into different containers (Data in $Node_A$, $Node_B$, $Node_C$ are put into Con_A, Con_B, Con_C respectively). And these containers inherit the data relationship pattern between the nodes. The recommendation reason is that in a business process, $Node_A$, $Node_C$, and $Node_C$ correspond to three individual tasks participated by one user role (say r_1). The three tasks indicate three different business rules. Different business rules are recommended to be in different containers for r_1. In doing so, each

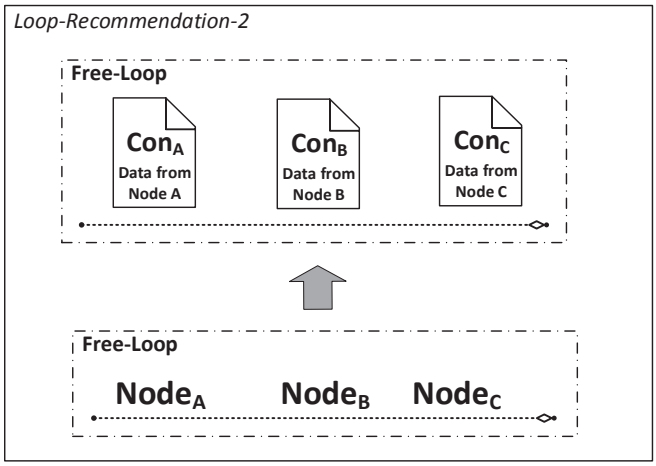


Figure 6.18: Recommendation Loop-Recommendation-2 on Free-order Loop

separate container can provide r_1 with the data that are strongly related to each other.

Loop-Recommendation-2:

$\forall m$: "$TreeGraph[m]$"."$dataRelationshipPattern$" == "$Free - Loop$",

iff $\exists i$: ("$TreeGraph[m]$"."$graphNodes[i]$"."$preAbsOfNode$" == "$null$"),

then $\forall k$ **where** $(k \geq 0) \wedge$

("$TreeGraphSet[m]$"."$graphNodes[k+1]$" $\in$ "$TreeGraphSet[m]$"):

("$TreeGraphSet[m]$"."$graphNodes[k]$" $\in con_k$) $\wedge$

("$TreeGraphSet[m]$"."$graphNodes[k+1]$" $\in con_{k+1}$) $\wedge$

$((con_k, con_{k+1}) ==$ "$Free - Loop$").

Loop-Recommendation-2 (Figure 6.18) shows how a rule of recommendation derives UI flow from the **Free-order Loop** data relationship pattern, which does not contain any `Abs Label`. In this situation, all the nodes in this pattern are recommended to be separated into different containers (Data in $Node_A$, $Node_B$, $Node_C$ are put into Con_A, Con_B, Con_C respectively). And these containers inherit the data relationship pattern between the nodes. The recommendation reason is that in a business process, $Node_A$, $Node_C$, and $Node_C$ correspond to three individual tasks participated by one user role (say r_1). The three tasks indicate three different business rules. Different business rules are recommended to be in different containers for r_1. In doing so, each separate container can provide r_1 with the data that are strongly related to each other.

6.4 Algorithm for UI Derivation

In this section, we introduce the algorithm for UI derivation, which operate the rules introduced in Section 6.3. All these rules are categorized as two groups (*Group 1* and *Group 2*).

Rules in *Group 1* are about the separation of data inside a data operation pattern. This group includes **Sequential-Constraint-1**, **Sequential-Constraint-2**, **Parallel-A-Constraint-1**, **Parallel-A-Constraint-2**, **Conditional-Constraint-1**, **Loop-Constraint-1**, **Loop-Constraint-3**, **Sequential-Recommendation-1**, **Sequential-Recommendation-2**, **Parallel-A-Recommendation-1**, **Loop-Recommendation-1**, **Loop-Recommendation-2**.

Rules in *Group 2* are about the separation of inside and outside data of a data operation pattern. This group includes **Parallel-A-Constraint-3**, **Conditional-Constraint-2**, **Loop-Constraint-2**, **Loop-Constraint-4**, **Parallel-A-Recommendation-2**.

The **Algorithm 5** is built up to derive the UI from a tree graph. The input is a tree graph set that specifies the data relationships for a particular user role; and the output is a the user interface flow which specifies the derived BP UI for the user role. The function `UIDerivation` is the working horse of this algorithm. When a tree graph $TreeGraph$ is input into the algorithm, the first step is to retrieve the the tree fragment $headTreeFragment$, which has $ProcessStructRef$ from $TreeGraph$ (line 1). According to the data relationship pattern of $headTreeFragment$, suitable rule from *Group 1* is found to derive the UI flow fragment $UIFlowFrag$ from $headTreeFragment$ (line 2). This $UIFlowFrag$ is copied to the initialized UI Flow $UIFlow$ (line 3). Then $headTreeFragment$ is input into the function `UIDerivation` (line 4). In this function, the $structRef$ of the input tree fragment is found for judgement (line 8). If $structRef$ is not $ProcessStructRef$ (line 9), the input tree fragment must not be the $headTreeFragment$. Under this condition, $UIFlowFrag$ is derived from the input tree fragment according to suitable rule in *Group 1* (line 10). Then the the *container* holding $structRef$ in $UIFlow$ is replaced with $UIFlowFrag$ (line 11). Next, the nodes

Algorithm 5: UI Derivation

Input : $TreeGraph$

Output: $UIFlow$

```
1  retrieve headTreeFragment with ProcessStructRef from TreeGraph;
2  generate UIFlowFrag from headTreeFragment by applying rules in Group 1;
3  set UIFlow = UIFlowFrag;
4  UIDerivation(headTreeFragment);
5  process UIFlow by applying rules in Group 2;
6  return UIFlow;
7  Function UIDerivation(TreeFragment treeFrag)
8      get structRef from treeFrag;
9      if structRef is not ProcessStructRef then
10         generate UIFlowFrag from treeFrag by applying rules in Group 1;
11         update the container holding structRef with UIFlowFrag;
12     foreach node of treeFrag do
13         if node is CDENode then
14             get the value structRef' from node;
15             retrieve treeFrag' with structRef' from TreeGraph;
16             UIDerivation(treeFrag');
```

inside the input tree fragment is examined. If there exist any CDE node $CDENode$ (line 13), the tree graph fragment $treeFrag'$ pointed by this $CDENode$ is found (line 14 and 15). This $treeFrag'$ is re-input into `UIDerivation` for recursive processing (line 16). After all the tree fragments in the $TreeGraph$ are handled, the rules in *Group 2* are used in $UIFlow$ for secondary separation (line 5). Then the final result $UIFlow$ is returned as the output of the algorithm (line 6).

6.5 Scenario Example

In this section, we use the scenario example introduced in Figure 1.4 of Section 1.2 to demonstrate the derived UI flows for three user roles `personnel officer`, `referee`, and `applicant` participating in the recruitment process.

(a) Figure 6.19 shows the data relationships for the user role `personnel officer`. **Sequential-Constraint-1** and **Sequential-Recommendation-1** are used to generate the two containers `Describe Job` and `Comment Interview`. **Parallel-A-Recommendation-1** is used to generate the two containers `Organize Interview` and `Review Reference`. **Sequential-Recommendation-1** and **Conditional-Constraint-2** to generate the container `Make Decision`. **Conditional-Constraint-1** is used to generate the two containers `Notify Approval` and `Notify Rejection`.

(b) Figure 6.19 shows the data relationships for the user role `referee`. **Sequential-Recommendation-1** is used to generate the container `Review Reference`.

(c) Figure 6.19 shows the data relationships for the user role `applicant`. **Sequential-Constraint-1** is used to generate the two containers `Submit Application` and `Confirm Interview`.

6.6 Summary and Discussion

In this chapter, the method for user interface derivation is proposed, which is the third step of our UI derivation approach. The UI logic is extracted from the data relationships for each user role. The UI logic for a particular user role is represented as a UI flow. A set of UI derivation rules including 11 constraints and 6 recommendations are developed

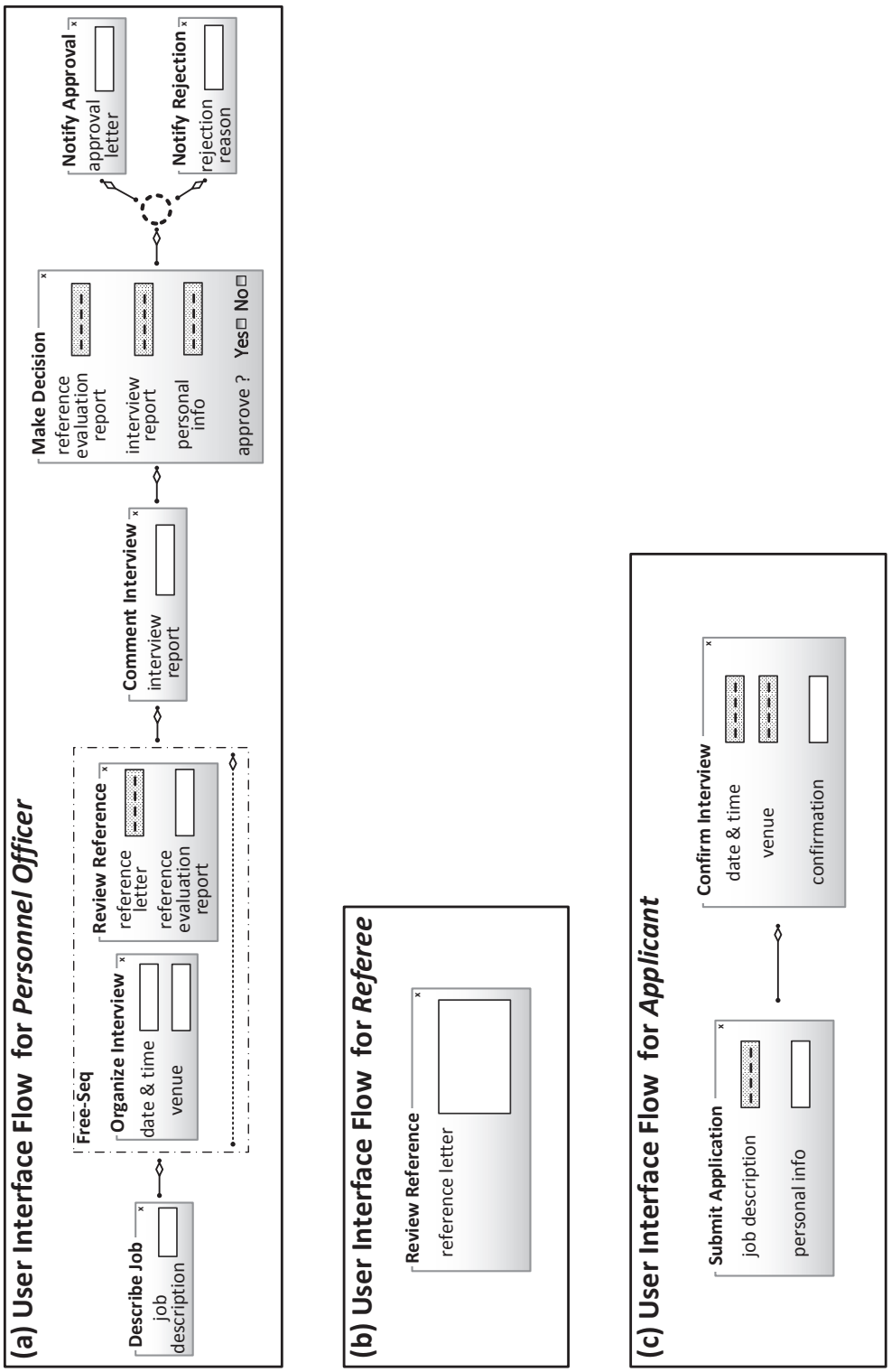


Figure 6.19: Derived UIs for Personnel Officer, Referee and Applicant

according to the data relationship patterns in the tree graph. The algorithm for UI derivation is built up with the UI derivation rules as cornerstones. We also use the recruitment process as a scenario example to demonstrate the derived UI flows for three participating user roles. The derived UI flows can help UI developers to analyze and develop graphical UIs of BPs.

Chapter 7

Implementation

This chapter describes a UI Derivation Tool (UIDrvTool). The UIDrvTool implements our proposed UI derivation approach based on the role-enriched BP model. The current developed UIDrvTool is a Proof-of-Concept prototype, which demonstrates the functionality of each UI derivation step presented in the previous chapters. The UIDrvTool is realized with the following features:

- The UIDrvTool is developed based on our proposed role-enriched business process model, where both the user roles participating in a BP and the data operations inside each task of a BP are considered. The UIDrvTool applies to a process specified using our role-enriched BP model, and generates the UIs related to the process.

- Three UI derivation steps as *Task Abstraction and Aggregation*, *Data Relationship Extraction*, and *User Interface Derivation* are implemented as three independent modules. In every module, each of the elementary operations/UI derivation rules introduced in the previous chapters is realized and implemented as a individual function.

- A GUI Generator is developed in the UIDrvTool to visualize the derived UI logics. For each user role participating in a BP, a set of Windows Forms is generated, which provides information to and requires inputs from the user role.

The remainder of this chapter is organized as follows. Section 7.1 explains the system architecture of the UIDrvTool. Section 7.2 introduces a recruitment process as a scenario example, which goes through the UIDrvTool to testify its availability. Section 7.3 provides a summary and discussion.

7.1 System Architecture

Figure 7.1 represents the system architecture that realizes the UI derivation approach from the role-enriched business process model. The input (① in Figure 7.1) is a business process specified with JavaScript Object Notation (JSON), which is a lightweight data-interchange format. The output (⑫ in Figure 7.1) is the derived BP UIs for each user role for each user role participating in the input business process.

Module A (② in Figure 7.1) is designed to realize the first UI derivation step - task abstraction and aggregation. The input of this module is a business process specified with JSON (① in Figure 7.1). For each user role participating in the input BP, the tasks not involving the user role are abstracted and aggregated as abstracted nodes, and the tasks involving the user role are reserved. The AABPs are produced for the participating user roles (④ in Figure 7.1). The functions of this module are introduced as follows.

`Task_Abs_Agg Controller`: The input of Task_Abs_Agg Controller function is a BP specified with JSON. This function implements a recursive algorithm to identify the control flow pattern on each granularity level of the input BP by calling the Role_BP_CFPattern Identifier function. The tasks in the identified pattern on each level are abstracted and aggregated by calling the suitable elementary operation from ③ in Figure 7.1. The AABPs for the participating user roles are generated as the output of Task_Abs_Agg Controller function.

`Role_BP_CFPattern Identifier`: The Role_BP_CFPattern Identifier function is to identify the control flow pattern on the coarsest granularity level of an input BP. The type of the identified pattern and the elements constituting the pattern are returned as the output of the function.

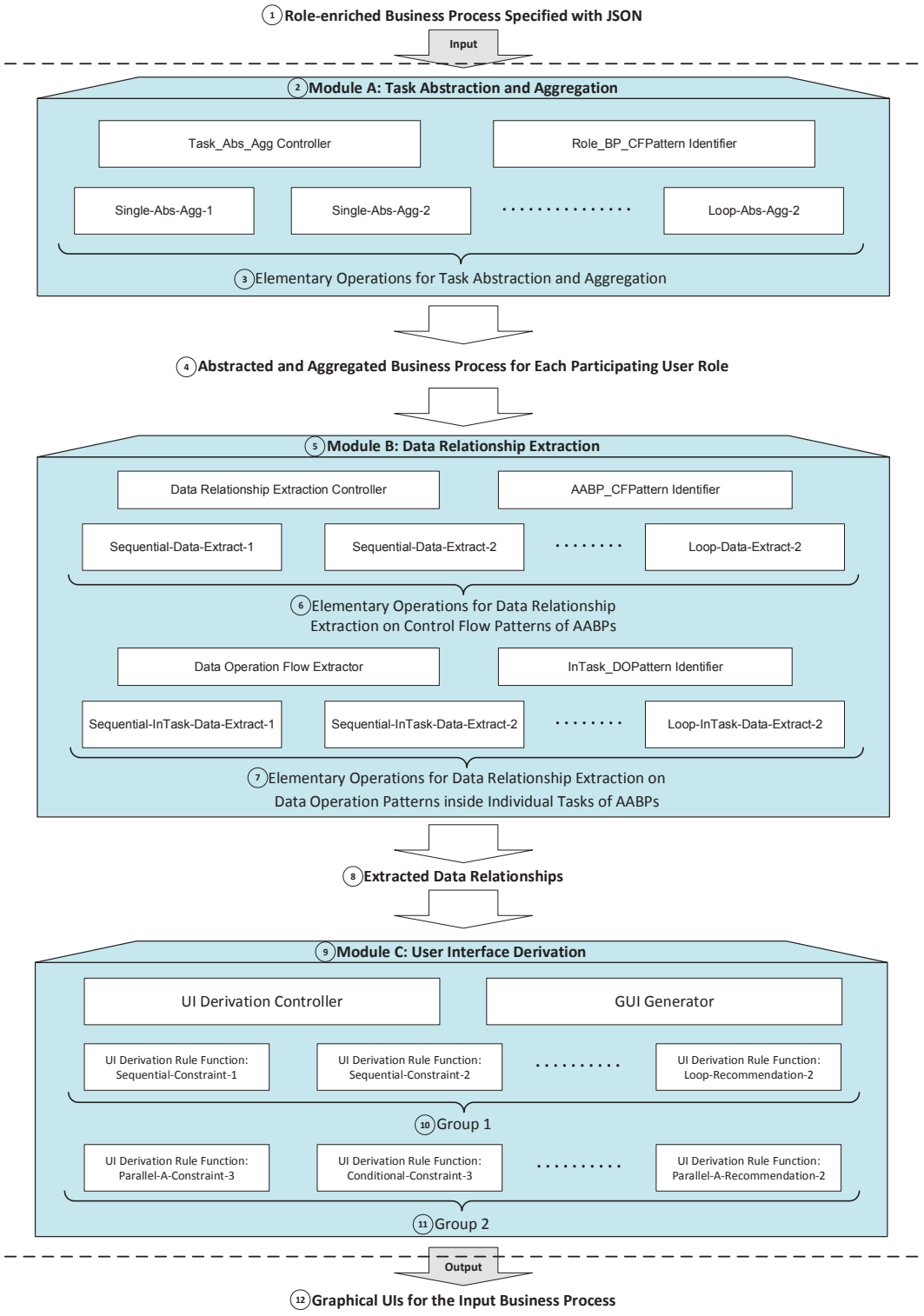


Figure 7.1: System Architecture

`Elementary Operations for Task Abstraction and Aggregation`: Each elementary operation is a function that abstracts and aggregates tasks and their control flow relations from a particular control flow pattern. Table 7.1 lists the names of all the elementary operations in ③ in Figure 7.1. Each elementary operation implements the corresponding rule of task abstraction and aggregation demonstrated in Chapter 4.

Table 7.1: List of Elementary Operations for Task Abstraction and Aggregation

Single-Abs-Agg-1	Parallel-A-Abs-Agg-4
Single-Abs-Agg-2	Parallel-B-Abs-Agg-1
Single-Abs-Agg-3	Parallel-B-Abs-Agg-2
Sequential-Abs-Agg-1	Parallel-B-Abs-Agg-3
Sequential-Abs-Agg-2	Parallel-C-Abs-Agg-1
Parallel-A-Abs-Agg-1 Alt-1	Parallel-C-Abs-Agg-2
Parallel-A-Abs-Agg-1 Alt-2	Parallel-C-Abs-Agg-3
Parallel-A-Abs-Agg-2 Alt-1	Conditional-Abs-Agg-1
Parallel-A-Abs-Agg-2 Alt-2	Loop-Abs-Agg-1
Parallel-A-Abs-Agg-3	Loop-Abs-Agg-2

Module B (⑤ in Figure 7.1) is designed to realize the second UI derivation step - data relationship extraction. The input of this module is the AABPs for the participating user roles (④ in Figure 7.1). The data relationships in each user role's AABP are analyzed and extracted as the output of the Module B (⑧ in Figure 7.1). The functions of this module are introduced as follows.

`Data Relationship Extraction Controller`: The input of Data Relationship Extraction Controller Controller function is the AABPs for the participating user roles.

A recursive algorithm is realized to identify the control flow pattern on each granularity level of an AABP by calling the AABP_CFPattern Identifier function. The control flow relations between/among the tasks in the identified pattern are analyzed and the data relationships are extracted by calling the suitable elementary operation from ⑥ in Figure 7.1. Inside each task, the data relationships are extracted from the data operation flows by calling the Data Operation Flow Extractor function.

`AABP_CFPattern Identifier`: The AABP_CFPattern Identifier function is to identified the control flow pattern on the coarsest granularity level of an input BP. The type of the identified pattern and the elements constituting the pattern are returned as the output of the function.

`Elementary Operations for Data Relationship Extraction on Control Flow Patterns of AABPs`: Each elementary operation is a function that extracts the data relationships from tasks and their control flow relations in a particular control flow pattern. Table 7.2 lists the names of all the elementary operations in ⑥ in Figure 7.1. Each elementary operation implements the corresponding rule of data relationship extraction demonstrated in Chapter 5.

`Data Operation Flow Extractor`: the Data Operation Flow Extractor function is to extract the data relationships from the data operations inside an individual task of an AABP. A recursive algorithm is realized to identify the data operation pattern on each granularity level of data operations inside an individual task of an AABP by calling the InTask_DOPattern Identifier function. The data operation flows between/among the data items in the identified pattern are analyzed and the data relationships are extracted by calling the suitable elementary operation from ⑦ in Figure 7.1.

`InTask_DOPattern Identifier`: The InTask_DOPattern Identifier function is to identified the data operation pattern on the coarsest granularity level of the data operations inside an individual task of an AABP. The type of the identified pattern and the elements constituting the pattern are returned as the output of the function.

Table 7.2: List of Elementary Operations for Data Relationship Extraction on Control Flow Patterns of AABPs

Sequential-Data-Deriv-1	Parallel-B-Data-Deriv-3
Sequential-Data-Deriv-2	Parallel-C-Data-Deriv-1
Parallel-A-Data-Deriv-1	Parallel-C-Data-Deriv-2
Parallel-A-Data-Deriv-2	Parallel-C-Data-Deriv-3
Parallel-A-Data-Deriv-3	Conditional-Data-Deriv-1
Parallel-A-Data-Deriv-4	Loop-Data-Deriv-1
Parallel-B-Data-Deriv-1	Loop-Data-Deriv-2
Parallel-B-Data-Deriv-2	

`Elementary Operations for Data Relationship Extraction on Data Operation Patterns inside Individual Tasks of AABPs`: Each elementary operation is a function that extracts the data relationships from a particular data operation pattern inside an individual task of an AABP. Table 7.3 lists the names of all the elementary operations in ⑦ in Figure 7.1. Each elementary operation implements the corresponding rule of data relationship extraction demonstrated in Chapter 5.

Table 7.3: List of Elementary Operations for Data Relationship Extraction on Data Operation Patterns inside Individual Tasks of AABPs

Sequential-InTask-Data-Deriv-1	Loop-InTask-Data-Deriv-1
Sequential-InTask-Data-Deriv-2	Loop-InTask-Data-Deriv-2
Conditional-InTask-Data-Deriv-1	

Module C (⑨ in Figure 7.1) is designed to realize the third UI derivation step -

user interface derivation. The input of this module are the data relationships for the participating user roles (⑧ in Figure 7.1). A series of Windows Forms representing the derived UIs (⑫ in Figure 7.1) are generated for each participating user role through analyzing the data relationships. The functions of this module are introduced as follows.

`UI Derivation Controller`: The UI Derivation Controller Controller function recursively identify the data relationship pattern of each fragment of the input tree graph. Two groups of UI derivation rules (⑩ and ⑪ in Figure 7.1) are applied to derive the UI flows, and the Windows Forms are generated by calling the GUI Generator function.

`GUI Generator`: the GUI Generator function is used to generate Windows Forms according to the derived UI flows.

`UI Derivation Rule Function`: Each UI derivation rule function derives the UI flows from a particular data relationship pattern. Table 7.4 lists the names of all the UI derivation rule functions from ⑩ and ⑪ in Figure 7.1. Each function implements the corresponding UI derivation rule demonstrated in Chapter 6.

7.2 Go-through Example

As an example, a role-enrich BP is input into the UIDrvTool, and the graphical UIs of this role-enrich BP are generated as output.

As the execution environment of the UIDrvTool, the CPU of the computer used for the implementation is Intel Core i5-4200U (1.60GHz), and the random-access memory size of the computer is 4.00 GB. The operating system installed on the computer is Windows 10 Home (64 bits). The utilized programming language is Visual Basic 2015, and the integrated development environment (IDE) is Microsoft Visual Studio Community 2015. Two class libraries .NET Framework 4.6.1 (by Microsoft) and Json.NET 9.0.1 (by Newtonsoft) are used to support the implementation.

As the input of the UIDrvTool, Figure 7.2 illustrates a recruitment process at the human resource department of a company. This process is represented with the role-

Table 7.4: List of Two Groups of UI Derivation Rule Functions

Group 1	Group 2
Sequential-Constraint-1	Parallel-A-Constraint-3
Sequential-Constraint-2	Conditional-Constraint-2
Parallel-A-Constraint-1	Loop-Constraint-3
Parallel-A-Constraint-2	Loop-Constraint-4
Conditional-Constraint-1	Parallel-A-Recommendation-2
Loop-Constraint-1	
Loop-Constraint-2	
Sequential-Recommendation-1	
Sequential-Recommendation-2	
Parallel-A-Recommendation-1	
Loop-Recommendation-1	
Loop-Recommendation-2	

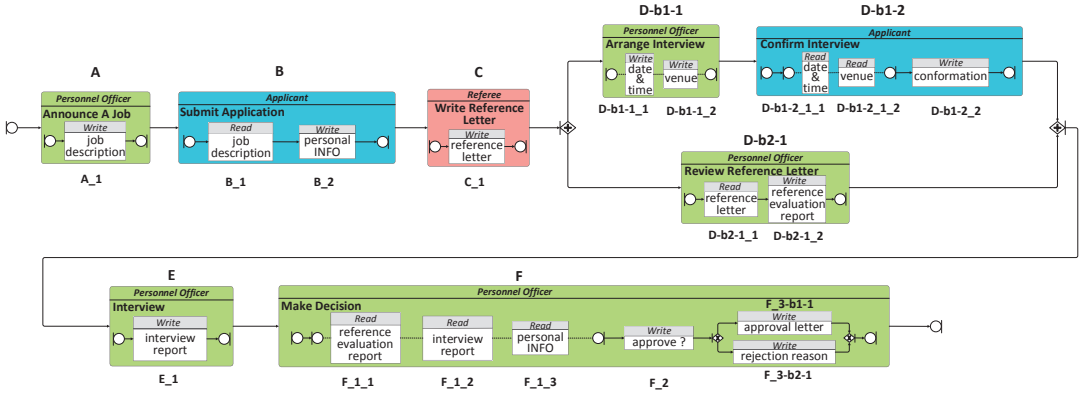


Figure 7.2: Recruitment Business Process Specified with Role-enriched BP Model

Figure 7.3: Role-enriched Business Process in JSON format - Part One

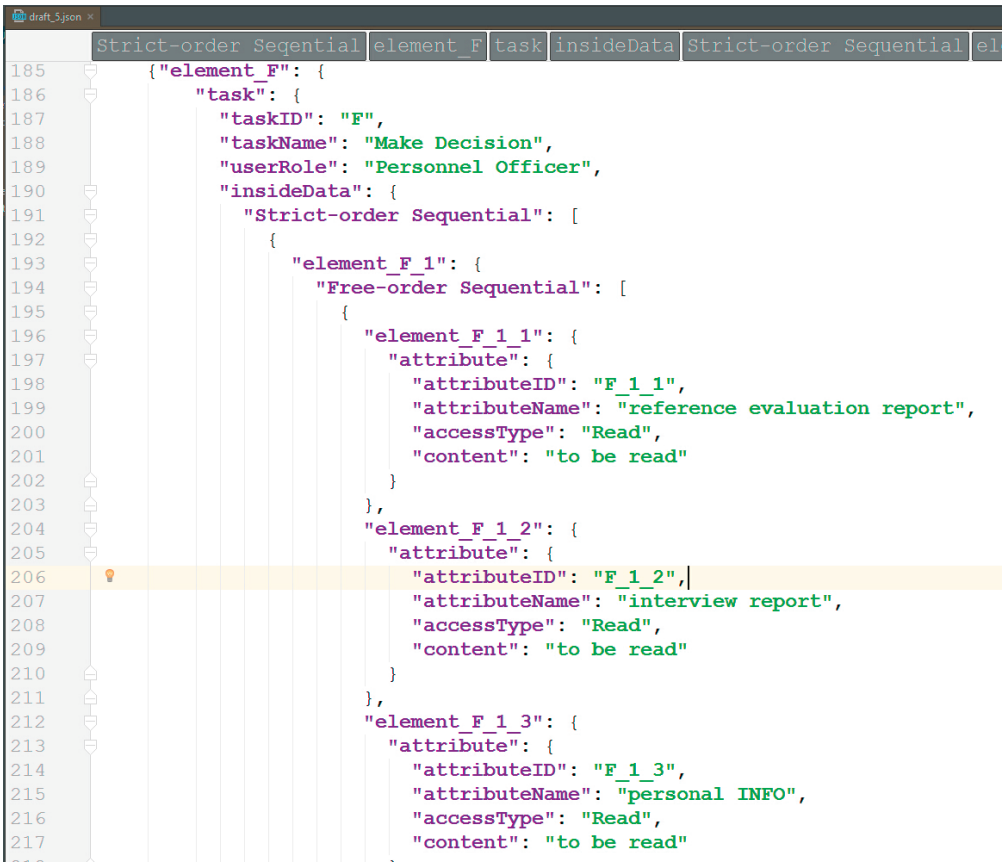


Figure 7.4: Role-enriched Business Process in JSON format - Part Two

enriched BP model. Three user roles as `personnel officer`, `applicant`, and `referee` are involved.

In the recruitment process, there are tasks as: (A) the `personnel officer` announces a job vacancy; (B) an `applicant` lodges his application; (C) a `referee` writes a reference letter to support the application; (D-b1-1) the `personnel officer` arranges an interview for the `applicant`; (D-b1-2) the `applicant` confirms the interview; (D-b2-1) the `personnel officer` reviews the reference letter; (E) the `personnel officer` conducts the interview; (F) the `personnel officer` makes the decision according to the evaluation of the reference letter and the interview report. Task A, B, C, D-b1-1 and D-b1-2 and D-b2-1, E, F have the **Strict-order Sequential** relations. Task D-b1-1 and D-b1-2 have the **Strict-order Sequential** relations. Task D-b1-1, D-b1-2 have the **Parallel-A** relations with task D-b2-1.

In task A, the `personnel officer` needs to provide the recruitment information for the job in the data item A_1. In task B, there exist two data items B_1 and B_2, which have the **Strict-order Sequential** relations. To perform the task B, the `applicant` firstly reads the recruitment information from the data item B_1, then provides the personal information in the data item B_2. In task C, the `referee` is required to write a reference letter into the data item C_1. In task D-b1-1, there exist two data items D-b1-1_1 and D-b1-1_2, which have the **Free-order Sequential** relations. To perform the task D-b1-1, the `personnel officer` must decide the date, time, and venue of the interview in the data items D-b1-1_1 and D-b1-1_2. In task D-b1-2, there exist three data items D-b1-2_1_1, D-b1-2_1_2, and D-b1-2_2. D-b1-2_1_1 and D-b1-2_1_2 have the **Free-order Sequential** relations. D-b1-2_1_1 and D-b1-2_1_2 have **Strict-order Sequential** relations with D-b1-2_2. To perform task D-b1-2, the `applicant` needs to read the interview information about date, time and venue from the data items D-b1-2_1_1, D-b1-2_1_2, then input the confirmation into the data item D-b1-2_2. In task D-b2-1, there exist two data items D-b2-1_1 and D-b2-1_2, which have the **Strict-order Sequential** relations. To perform the task D-b2-1, the `personnel officer` needs to read the reference letter from the data item D-b2-1_1, then provide the evaluation on the reference letter in the data item D-b2-1_2. In task E, the `personnel officer` needs

to complete the interview report in the data item E_1. In task F, there exist six data items F_1_1, F_1_2, F_1_3, F_2, F_3-b1-1, and F_3-b2-1. F_1_1, F_1_2, F_1_3 have **Free-order Sequential** relations. F_3-b1-1, and F_3-b2-1 have **Conditional** relations. The group of F_1_1, F_1_2, F_1_3, and F_2, and the the group of F_3-b1-1, and F_3-b2-1 have **Strict-order Sequential** relations. To perform the task F, the `personnel officer` needs to firstly read the information of the evaluation of the reference letter, the interview report, and the `applicant`'s personal information from the data items F_1_1, F_1_2, F_1_3, then makes the decision about whether to approve this applicant in the data item F_2, lastly provides the approval letter/rejection reason in the data item F_3-b1-1/F_3-b2-1 according to the decision.

Figure 7.3 and Figure 7.4 show the role-enriched business process in JSON format. In these JSON files, both a control flow pattern and a data operation pattern of the BP are modelled with JSON objects, and the elements of a control flow/data operation pattern are held in a JSON array. A task is represented as a JSON object that containing four properties: "`taskID`", "`taskName`", "`userRole`", and "`insideData`". An attribute, representing a data item inside a task, is specified as a JSON object that containing four properties: "`attributeID`", "`attributeName`", "`accessType`", and "`content`".

When the recruitment process is input into the UIDrvTool, **Module A** will firstly deal with this process. `Task_Abs_Agg Controller` identifies all the user roles `personnel officer`, `applicant`, and `referee` participating in the process. And then the UI derivation will be performed for each user role. Here we only explain the derivation process for the `personnel officer` in detail. After the user role `personnel officer` is identified, `AABP_CFPattern Identifier` is called to identify the control flow pattern on the coarsest granular level of the process, and the identified pattern is **Strict-order Sequential**. Then according to the pattern type, the function `Sequential-Abs-Agg-1` in Table 7.1 is called to perform task abstraction and aggregation on the pattern **Strict-order Sequential**. And the tasks B and C (not involving the `personnel officer`) are abstracted as an abstracted node (see Abs_1 in Figure 7.5) by `Sequential-Abs-Agg-1`. In the pattern **Strict-order Sequential**, there exists a complex structure comprising

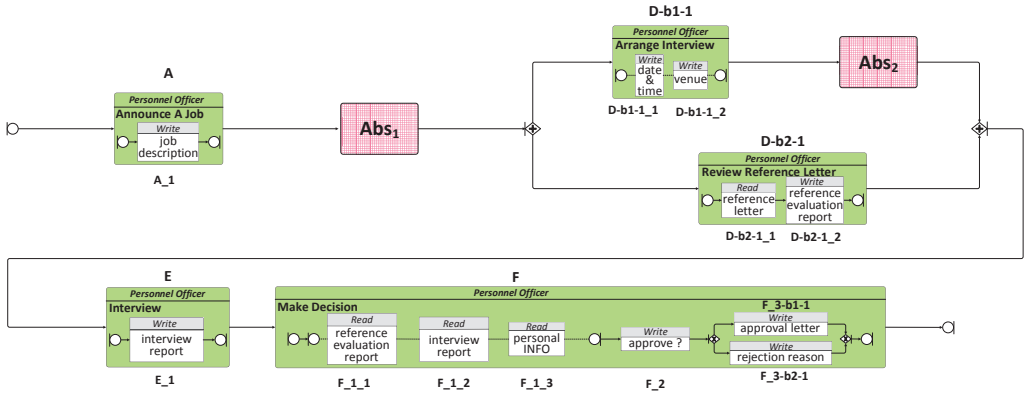


Figure 7.5: AABP for Personnel Officer

the tasks D-b1-1, D-b1-2, D-b2-1. Then the control flow pattern of this complex structure is identified through calling `AABP_CFPattern Identifier`, and the pattern type is **Parallel-A**. Then accordingly, the function **Parallel-A-Abs-Agg-2 Alt-2** is called to abstract the task D-b1-2 (not involving the `personnel officer`) as an abstract node (see Abs_2 in Figure 7.5). The resulted process is the AABP for `personnel officer` as shown in Figure 7.5.

When the AABP is input into **Module B**, the `Data Relationship Extraction Controller` calls the function `AABP_CFPattern Identifier` to identify the control flow pattern on the coarsest granular level of the AABP, and the identified pattern is **Strict-order Sequential**. In order to deal with this pattern, the function `Sequential-Data-Extract-1` is called to extract the relationships (see (A) in Figure 7.6) among tasks A, E, F, abstracted node Abs_1, and a complex structure composed of D-b1-1, Abs_1, D-b2-1. For each of the tasks A, E, F, the function `Data Operation Flow Extractor` extracts the inside data operation flow, and the data operation pattern on the coarsest granular level of data operation flow is identified by the function `InTask_DOPattern Identifier`. The identified data operation patterns for the tasks A, E, F are all **Strict-order Sequential**. Then accordingly, the function `Sequential-InTask-Data-Deriv-1` extracts data relationships as A-1 in Figure 7.6 from task A, A-2 in Figure 7.6 from task E, C in Figure 7.6 from task F. Next, the data operation patterns of two data operation flow structures (F_1_1, F_1_2, F_1_3 and F_3-b1-1, F_3-b2-1) in task F are identified

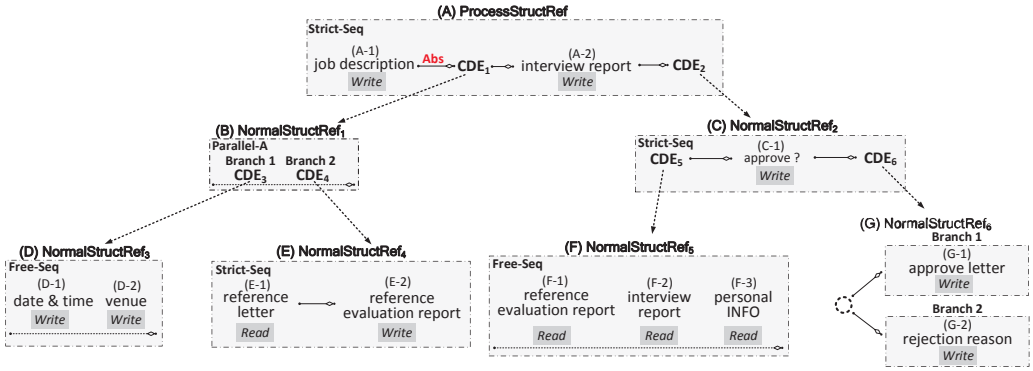


Figure 7.6: Data Relationships Extracted from the AABP for Personnel Officer

by `InTask_DOPattern Identifier` as **Strict-order Sequential** and **Conditionally** respectively. `Sequential-InTask-Data-Deriv-2` extracts F in Figure 7.6 from the structure of F_1_1, F_1_2, F_1_3, and `Conditional-InTask-Data-Deriv-1` extracts G in Figure 7.6 from the structure of F_3-b1-1, F_3-b2-1. Then the control flow pattern of the structure (D-b1-1, Abs_2, D-b2-1) of the AABP is identified by `AABP_CFPattern Identifier` as **Parallel-A**, whose data relationships is extracted as B in Figure 7.6 by `Parallel-A-Data-Deriv-3`. Next, `Data Operation Flow Extractor` extracts the data operation flows inside the two tasks D-b1-1 and D-b2-1 and the data operation patterns of the two extracted flows are identified as **Free-order Sequential** and **Strict-order Sequential** respectively. Then `Sequential-InTask-Data-Deriv-2` extracts D in Figure 7.6 from the data operation flows inside task D-b1-1, and `Sequential-InTask-Data-Deriv-1` extracts E in Figure 7.6 from the data operation flows inside task D-b2-1. As the final output of **Module B**, Figure 7.6 shows the data relationships extracted from the AABP for `personnel officer`.

When the data relationships are input into **Module C**, the `UI Derivation Controller` retrieves the `ProcessStructRef` in Figure 7.6 to start the UI derivation. The `ProcessStructRef` is the root of the tree graph representing the data relationships. Due to that the data relationship pattern of A is **Strict-order Sequential**, the functions `Sequential-Constraint-1` and `Sequential-Constraint-2` are called to separate the four nodes in A into four different UI containers which have **Strict-order**

Sequential relations. And the `GUI Generator` generates two UI forms (a) and (d) in Figure 7.7 for the container holding A-1 and A-2 in Figure 7.6 respectively. To deal with the container holding CED_1, `Parallel-A-Recommendation-1` is called to separate D and E of Figure 7.6 in two UI containers, which have **Parallel-A** relations. And the `GUI Generator` generates two UI forms (b) and (c) in Figure 7.7 for the container holding D and E in Figure 7.6 respectively. To deal with the container holding CED_2, `Sequential-Recommendation-1` is called to separate CED_5 and C-1, CED_6 in two UI containers which have **Strict-order Sequential** relations. And the `GUI Generator` generates a UI form (e) in Figure 7.7 for the container holding CED_5 and C-1 in Figure 7.6. Then to deal with CED_6, `Conditional-Constraint-1` is called to separate G-1 and G-2 of Figure 7.6 in two containers, which have **Conditional** relations. And the `GUI Generator` generates two UI forms (f) and (g) in Figure 7.7 for the container holding G-1 and G-2 in Figure 7.6 respectively.

Figure 7.7 illustrates a series of Windows Forms generated by the UIDrvTool. These forms represent the UI flow derived for the user role `Personnel Officer`. There exist seven forms with involved operation flow relations as **Strict-order Sequential**, **Free-order Sequential**, and **Conditional**. In Figure 7.7, Form (b) and Form (c) have **Free-order Sequential** relation; Form (a), Form (b) and Form (c), Form (d), Form (e) have **Strict-order Sequential** relation; Form (e), Form (f), Form (g) have **Conditional** relation where Form (f) and Form (g) are two branches following Form (e).

7.3 Summary and Discussion

This chapter discusses a UI Derivation Tool named as UIDrvTool, which is developed as a Proof-of-Concept for the proposed approach of UI derivation based on the role-enriched BP model. In the UIDrvTool, the elementary operations for Task Abstraction and Aggregation, Data Extraction, and the UI derivation rules are all implemented to support each UI derivation step. The GUI Generator is developed in the UIDrvTool to generate Windows Forms, which are able to graphically illusttrate the derived UI logic for each participating user role. A recruitment process specified with the role-enriched

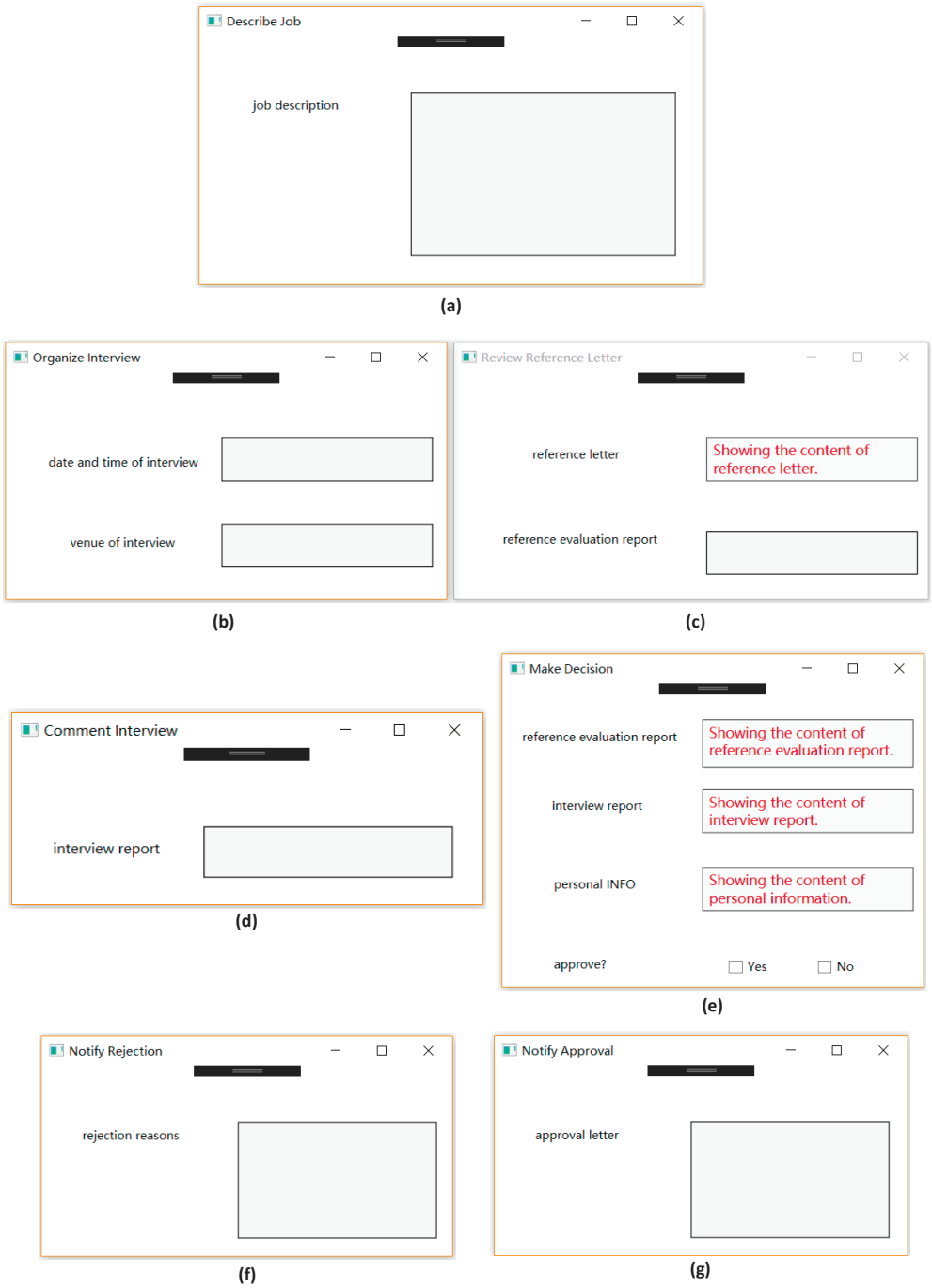


Figure 7.7: Derived Graphical User Interfaces

BP model goes through the UIDrvTool to demonstrate how the UI derivation is performed. With this UIDrvTool, the UI logic of a business process can be automatically derived. The UI engineers are able to directly design the GUI of the process based on derived UI logic, without analysing the process logic. This work greatly eases the UI design.

Chapter 8

Conclusion and Future Work

In this final chapter, we conclude the thesis by summarizing the main contributions in Section 8.1 and providing a discussion on open problems and an outlook on possible future research in Section 8.2.

8.1 Contributions

In this thesis, we proposed a user interface derivation approach to automatically derive the UI logic of a BP for each involved user role. This derivation approach has been built up based on a role-enriched business process model and is composed of three derivation steps. With this approach, the hard coding efforts on the UI development for BPs can be reduced. This approach helps BP changes to be easily adapted to the UIs without recoding. The main contributions of this approach are summarized as follows.

- We have proposed a role-enriched business process model to derive the UI logic of a BP for each involved user role. This BP model has the capabilities to specify (1) how user roles are involved in tasks; (2) how data are operated in individual tasks; (3) how complex control flow patterns affect data relationships. The formalism of role-enriched business process model has been introduced. The well-formness of the process model has been analyzed. The process model has been specified by extending the BPMN. A set of control flow patterns and data operation patterns have been identified and summarized as the cornerstones of building up the

elementary operations for task abstraction and aggregation. This role-enriched BP model lays the foundation for building up the UI derivation approach. The role-enriched BP model is a extension to the activity-centric paradigm in terms of expressing the data operated within a single task and how user roles are involved in tasks.

- We have proposed a method for task abstraction and aggregation as the first step of the UI derivation approach. The formalism of abstracted and aggregated business process model has been introduced. A set of elementary operations for task abstraction and aggregation have been developed based on the identified control flow patterns in the role-enriched BP. The algorithm for task abstraction and aggregation has been built up with the elementary operations as cornerstones. The structural consistency between the role-enriched BP and the AABP has been discussed through the analysis on the orders and dependencies between/among tasks. Two theorems about the structural consistency have been identified and proved. This method can allow each user role involved in a BP to have a customized UI logic. Besides, the method for task abstraction and aggregation also has the following significance: (1) firstly, the details of BP tasks can be hidden and abstracted from certain users to meet the information security requirements such as privacy, confidentiality, and conflict of interest; (2) secondly, task abstraction and aggregation can lay a foundation for deriving customized descriptions of a BP for participating users according to the users' requirements and intentions. The customized BP descriptions can play an important role in the modelling of BP collaboration, BP visualization, and authority control; (3) thirdly, abstracted and aggregated BPs are able to highlight the requirements associated with a specific user role and preserve some information of other user roles for the effective control flow in a BP. AABPs can be used to enable the development and updating of software components such as UIs related to different user roles.

- We have proposed a method for data relationship extraction as the second step of the UI derivation approach. We have used tree graph to represent the extracted

data relationships, and the JSON Strings to record all the details represented in the tree graph. Five data relationship patterns have been identified. A well-formed tree graph has been analyzed. A set of elementary operations for data relationship have been developed based on the identified control flow patterns and data operation patterns in the AABP. The algorithm for data relationship extraction has been built up with the elementary operations as cornerstones. With the data relationship extraction method, the data relationships, including the data operated in the process and data operation flow, can be extracted from the AABP. The extracted data relationships allow for analyzing the UI logic from the abstracted and aggregated BP for each participating user role. The well-formness regulations can be implemented to automatically check well-formness of a tree graph.

- We have proposed a method for UI derivation as the last step of the UI derivation approach. The derived UI logic for a particular user role has been represented as a user interface flow, which has two granularity levels: the operation flow between UI containers, and data items included inside each UI container. Each data item has been specified with the access type including read and write. Five types of operation flow relation between UI containers have been summarized. A set of UI derivation rules have been coined including constraints and recommendations. The algorithm for UI derivation has been built up with the rules as cornerstones. With the extracted data relationships, the UI logic is able to be derived for each participating user role. The derived UI logic allows UI developers to develop graphical user interfaces of the BP for each involved user role.

- Our proposed user interface derivation approach has been implemented as a tool named UIDrvTool for Proof-of-Concept. The role-enriched BP model has been specified using JSON. Each UI derivation step has been implemented as a independent module. In each module, the elementary operations for task abstraction and aggregation, the elementary operations for data relationship extraction, and the UI derivation rules have been implemented as individual functions. A GUI

Generator has been developed in the UIDrvTool to visualize the derived UI logics. For each user role participating in a BP, a set of Windows Forms have been generated by the GUI Generator, which can provide information to and require inputs from the user role.

Comparing to existing works, our proposed approach has successfully derived the UI logics for each participating user role. Complicated control flow patterns have been covered in our approach, such as Parallel-B and Parallel-C.

8.2 Future Work

In this thesis, we have provided a comprehensive discussion on the UI derivation approach based on role-enriched business process model. We believe that this research area will draw attentions from both the academic and industrial communities. There are a number of open issues for further investigation.

In our work, we only focused on deriving the UIs for business processes. The responding to changes from either the BP or the related UI is a critical challenge and therefore requires to be further explored. Changes may range from momentary (ad-hoc) modifications of the BP for a single user to a complete restructuring for the process to improve efficiency. In order to have the capability of adapting the changes, two major issues must be addressed: (1) the identification of potential changes in BPs and their related UIs, (2) the analysis on which parts will be and will not be affected by these potential changes [141, 142, 143, 144, 145, 146]. In our proposed role-enriched BP, we identified seven control flow patterns and four data operation patterns. These patterns are the most commonly-used. Complex workflow patterns such as inclusive branching and merging, synchronization of multiple process instances must be considered in the control flow patterns and data operation patterns of our work. Solutions to the complex patterns need to be provided in the future [86, 90, 91, 92]. Our UI derivation approach is built up on a role-enriched business process model, which follows the traditional activity-centric process modelling paradigm. Other than this paradigm, the declarative process modelling paradigm is another widely used modelling approach, which treats the

data objects in BPs as the first-class citizens. The artifact-centric modelling approach is a representative declarative modelling paradigm, which focuses on specifying the life cycles of business artifacts and their inter-relations. As a future exploration, the support to the artifact-centric BP model needs to be enriched in our UI derivation approach [33, 133, 134]. In our proposed UI derivation approach, we have not paid attention on the development of the formal BP modelling language. We used the BPMN (an informal language) to specify our role-enriched business process model. In the future work, a formal process modelling language may be required to formally specify the semantics of our proposed BP model. In doing so, the process properties such as correctness and reliability can be verified. Apart from the process verification, the verification of our proposed business rules needs to be conducted through specifying the rules in a unified language.

References

[1] James OBrien. **Introduction to information systems: Essentials for the internet worked e-business enterprise**. *McGrawhill Public Company*, 2001. 2

[2] Tim Allen. **Improve your business processes for ERP efficiency**. *Strategic Finance*, **92**(11):54, 2011. 2

[3] Vijay Gurbaxani and Seungjin Whang. **The impact of information systems on organizations and markets**. *Communications of the ACM*, **34**(1):59–73, 1991. 2

[4] Nariman Abdi, Behrouz Zarei, Jamshid Vaisy, and Badieahe Parvin. **Innovation models and business process redesign**. *International Business and Management*, **3**(2):147–152, 2011. 2

[5] Christoph Hienerth, Peter Keinz, and Christopher Lettl. **Exploring the nature and implementation process of user-centric business models**. *Long Range Planning*, **44**(5):344–374, 2011. 2

[6] Mathias Weske. *Business process management: concepts, languages, architectures*. Springer Science & Business Media, 2012. 2, 18, 42

[7] A Faye Borthick, Gary P Schneider, and Anthony O Vance. **Preparing graphical representations of business processes and making inferences from them**. *Issues in Accounting Education*, **25**(3):569–582, 2010. 2

[8] Mathias Weske. **Business process management architectures**. In *Business Process Management*, pages 333–371. Springer, 2012. 3

[9] Ksenia Ryndina, Jochen M Küster, and Harald Gall. **Consistency of business process models and object life cycles**. In *International Conference on Model Driven Engineering Languages and Systems*, pages 80–90. Springer, 2006. 3, 52

[10] Rong Liu, Frederick Y Wu, and Santhosh Kumaran. **Transforming activity-centric business process models into information-centric models for soa solutions**. *Cross-Disciplinary Models and Applications of Database Management: Advancing Approaches: Advancing Approaches*, page 336, 2011. 3

REFERENCES

[11] Object Management Group. **Business Process Model and Notation**. https://en.wikipedia.org/wiki/Business_Process_Model_and_Notation, 2016. [Online; accessed 02-November-2016]. 3, 19

[12] Jan Mendling. **Event-driven process chains (epc)**. In *Metrics for Process Models*, pages 17–57. Springer, 2008. 3

[13] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, et al. **Business process execution language for web services**, 2003. 3, 27

[14] Wil MP Van Der Aalst and Arthur HM Ter Hofstede. **YAWL: yet another workflow language**. *Information Systems*, **30**(4):245–275, 2005. 3

[15] Sidney L Smith and Jane N Mosier. **The user interface to computer-based information systems: A survey of current software design practice**. *Behaviour & Information Technology*, **3**(3):195–203, 1984. 4

[16] Wikipedia. **User interface**. https://en.wikipedia.org/wiki/User_interface, 2016. [Online; accessed 04-December-2016]. 4

[17] Edsger Wybe Dijkstra, Edsger Wybe Dijkstra, Edsger Wybe Dijkstra, Etats-Unis Informaticien, and Edsger Wybe Dijkstra. *A discipline of programming*, **1**. prentice-hall Englewood Cliffs, 1976. 4

[18] Hubert Zimmermann. **OSI reference model-the ISO model of architecture for open systems interconnection**. *IEEE Transactions on Communications*, **28**(4):425–432, 1980. 5

[19] Jens Kolb, Paul Hübner, and Manfred Reichert. **Model-driven user interface generation and adaptation in process-aware information systems**. 2012. 5, 42, 48

[20] Manfred Reichert, Jens Kolb, Ralph Bobrik, and Thomas Bauer. **Enabling personalized visualization of large business processes through parameterizable views**. 2012. 10, 36, 40

[21] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A Reijers. *Fundamentals of business process management*. Springer, 2013. 17, 18

[22] Ruth Sara Aguilar-Saven. **Business process modelling: Review and framework**. *International Journal of Production Economics*, **90**(2):129–149, 2004. 18, 23

[23] Ryan KL Ko, Stephen SG Lee, and Eng Wah Lee. **Business process management (BPM) standards: a survey**. *Business Process Management Journal*, **15**(5):744–791, 2009. 18, 27

[24] Wil MP van der Aalst. **Business process management: a comprehensive survey**. *ISRN Software Engineering*, **2013**, 2013. 18, 22, 26

[25] Stephen A White. **Introduction to BPMN**. *IBM Cooperation*, **2**(0):0, 2004. 18

[26] Wikipedia. **Business Process Model and Notation**. https://en.wikipedia.org/wiki/Business_Process_Model_and_Notation, 2016. [Online; accessed 02-November-2016]. 18

[27] Business Process Model. **Notation (BPMN) Version 2.0**. *OMG Specification, Object Management Group*, 2011. 19, 52, 60

[28] Michele Chinosi and Alberto Trombetta. **BPMN: An introduction to the standard**. *Computer Standards & Interfaces*, **34**(1):124–134, 2012. 19

[29] BPM Offensive Berlin. **BPMN 2.0 Business Process Model and Notation, Poster**, 2011. 19

[30] Andreas Meyer, Luise Pufahl, Dirk Fahland, and Mathias Weske. *Modeling and enacting complex data dependencies in business processes*. Springer, 2013. 20

[31] Andreas Meyer, Luise Pufahl, Dirk Fahland, and Mathias Weske. **Enacting Complex Data Dependencies from Activity-Centric Business Process Models.** In *BPM (Demos)*, 2013. 20

[32] Niels Lohmann and Martin Nyolt. **Artifact-centric modeling using BPMN**. In *Service-Oriented Computing-ICSOC 2011 Workshops*, pages 54–65. Springer, 2012. 20

[33] Anil Nigam and Nathan S Caswell. **Business artifacts: An approach to operational specification**. *IBM Systems Journal*, **42**(3):428–445, 2003. 21, 44, 49, 195

[34] Kamal Bhattacharya, Cagdas Gerede, Richard Hull, Rong Liu, and Jianwen Su. **Towards formal analysis of artifact-centric business process models**. In *Business Process Management*, pages 288–304. Springer, 2007. 21, 75

[35] Hagen Völzer. **A new semantics for the inclusive converging gateway in safe processes**. In *International Conference on Business Process Management*, pages 294–309. Springer, 2010. 21

[36] David Raymond Christiansen, Marco Carbone, and Thomas Hildebrandt. **Formal semantics and implementation of BPMN 2.0 inclusive gateways**. In *International Workshop on Web Services and Formal Methods*, pages 146–160. Springer, 2010. 21

[37] Remco M Dijkman, Marlon Dumas, and Chun Ouyang. **Semantics and analysis of business process models in BPMN**. *Information and Software Technology*, **50**(12):1281–1294, 2008. 21

[38] WMP Van der Aalst and KM Van Hee. **Business process redesign: a Petri-net-based approach**. *Computers in Industry*, **29**(1):15–26, 1996. 21, 25

[39] Peter YH Wong and Jeremy Gibbons. **A process semantics for BPMN**. In *International Conference on Formal Engineering Methods*, pages 355–374. Springer, 2008. 21

[40] Peter YH Wong and Jeremy Gibbons. **Formalisations and applications of BPMN**. *Science of Computer Programming*, **76**(8):633–650, 2011. 21

REFERENCES

[41] Jim Woodcock and Jim Davies. *Using Z: specification, refinement, and proof*, **39**. Prentice Hall Englewood Cliffs, 1996. 21

[42] Bill Roscoe. **The theory and practice of concurrency**. 1998. 21

[43] OMG Uml. **2.0 Superstructure Specification**. *OMG, Needham*, 2004. 21

[44] Martin Fowler. *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional, 2004. 22

[45] Wikipedia. **Unified Modelling Laguguage - Activity Diagram**. https://en.wikipedia.org/wiki/Activity_diagram, 2016. [Online; accessed 03-November-2016]. 22

[46] Tutorialspoint. **Learn UML - Activity Diagrams**. https://www.tutorialspoint.com/uml/uml_activity_diagram.htm, 2016. [Online; accessed 03-November-2016]. 22

[47] Marlon Dumas and Arthur HM Ter Hofstede. **UML activity diagrams as a workflow specification language**. In *International Conference on the Unified Modeling Language*, pages 76–90. Springer, 2001. 22

[48] Petia Wohed, Wil MP van der Aalst, Marlon Dumas, Arthur HM ter Hofstede, and Nick Russell. **Pattern-based analysis of UML activity diagrams**. *Beta, Research School for Operations Management and Logistics, Eindhoven*, 2004. 22

[49] Nick Russell, Wil MP van der Aalst, Arthur HM Ter Hofstede, and Petia Wohed. **On the suitability of UML 2.0 activity diagrams for business process modelling**. In *Proceedings of the 3rd Asia-Pacific Conference on Conceptual Modelling-Volume 53*, pages 95–104. Australian Computer Society, Inc., 2006. 22

[50] John E Hopcroft. *Introduction to Automata Theory, Languages and Computation: For VTU, 3/e*. Pearson Education India, 1979. 22

[51] Alain Girault, Bilung Lee, and Edward A Lee. **Hierarchical finite state machines with multiple concurrency models**. *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, **18**(6):742–760, 1999. 23

[52] Wikipedia. **Finite-state machine**. https://en.wikipedia.org/wiki/Finite-state_machine, 2016. [Online; accessed 03-November-2016]. 23

[53] Khodakaram Salimifard and Mike Wright. **Petri net-based modelling of workflow systems: An overview**. *European Journal of Operational Research*, **134**(3):664–676, 2001. 24

[54] Wil Van Der Aalst and Kees Max Van Hee. *Workflow management: models, methods, and systems*. MIT Press, 2004. 24

[55] Tadao Murata. **Petri nets: Properties, analysis and applications**. *Proceedings of the IEEE*, **77**(4):541–580, 1989. 24

[56] Kurt Jensen. *Coloured Petri nets: basic concepts, analysis methods and practical use*, **1**. Springer Science & Business Media, 2013. 25

[57] Kurt Jensen. **A brief introduction to coloured petri nets**. In *International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, pages 203–208. Springer, 1997. 25

[58] Mohammed Elkoutbi and Rudolf K Keller. **Modeling interactive systems with hierarchical colored petri nets**. In *Advanced Simulation Technologies Conference*, pages 432–37. Citeseer, 1998. 25

[59] Wil MP Van der Aalst. **The application of Petri nets to workflow management**. *Journal of Circuits, Systems, and Computers*, **8**(01):21–66, 1998. 25

[60] Wil MP Van Der Aalst. **Three good reasons for using a Petri-net-based workflow management system**. In *Information and Process Integration in Enterprises*, pages 161–182. Springer, 1998. 25

[61] Wil MP van der Aalst and Kees M van Hee. **Framework for business process redesign.** In *WETICE*, pages 36–45, 1995. 25

[62] WMP Van der Aalst. **A class of Petri net for modeling and analyzing business processes**. *Computing Science Reports*, **95**:26, 1995. 25

[63] Wil MP van der Aalst. **Structural characterizations of sound workflow nets**. *Computing Science Reports*, **96**(23):18–22, 1996. 25

[64] Wil MP Van der Aalst. **Verification of workflow nets**. In *International Conference on Application and Theory of Petri Nets*, pages 407–426. Springer, 1997. 25

[65] Wil MP Van Der Aalst. **Workflow verification: Finding control-flow errors using petri-net-based techniques**. In *Business Process Management*, pages 161–183. Springer, 2000. 25

[66] Dimitra Giannakopoulou and Klaus Havelund. **Automata-based verification of temporal properties on running programs**. In *Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on*, pages 412–416. IEEE, 2001. 25

[67] Klaus Havelund and Grigore Roşu. **Synthesizing monitors for safety properties**. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 342–356. Springer, 2002. 25

[68] Wikipedia. **Linear Temporal Logic**. https://en.wikipedia.org/wiki/Linear_temporal_logic, 2016. [Online; accessed 03-November-2016]. 25

[69] Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen. *Principles of model checking*. MIT Press, 2008. 26

[70] Wil MP van Der Aalst, Maja Pesic, and Helen Schonenberg. **Declarative workflows: Balancing between flexibility and support**. *Computer Science-Research and Development*, **23**(2):99–113, 2009. 26, 75

[71] Wil MP Van Der Aalst and Maja Pesic. *DecSerFlow: Towards a truly declarative service flow language*. Springer, 2006. 26

[72] Wikipedia. **Extensible Markup Language**. https://en.wikipedia.org/wiki/XML, 2016. [Online; accessed 04-November-2016]. 27

[73] Anders Møller and Michael I Schwartzbach. *An introduction to XML and Web technologies*. Pearson Education, 2006. 27

[74] Diane Jordan, John Evdemon, Alexandre Alves, Assaf Arkin, Sid Askary, Charlton Barreto, Ben Bloch, Francisco Curbera, Mark Ford, Yaron Goland, et al. **Web services business process execution language version 2.0**. *OASIS standard*, **11**(120):5, 2007. 27

[75] Niels Lohmann, Eric Verbeek, and Remco Dijkman. **Petri net transformations for business processes–a survey**. In *Transactions on Petri Nets and Other Models of Concurrency II*, pages 46–63. Springer, 2009. 27

[76] Wikipedia. **Business Process Execution Language**. https://en.wikipedia.org/wiki/Business_Process_Execution_Language, 2016. [Online; accessed 04-November-2016]. 27

[77] Jan C Recker and Jan Mendling. **On the translation between BPMN and BPEL: Conceptual mismatch between process modeling languages**. In *The 18th International Conference on Advanced Information Systems Engineering. Proceedings of Workshops and Doctoral Consortium*, pages 521–532. Namur University Press, 2006. 28

[78] Michael Rosemann and Peter Green. **Developing a meta model for the Bunge–Wand–Weber ontological constructs**. *Information Systems*, **27**(2):75–91, 2002. 28

[79] Chun Ouyang, Marlon Dumas, Wil MP Van Der Aalst, Arthur HM Ter Hofstede, and Jan Mendling. **From business process models to process-oriented software systems**. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, **19**(1):2, 2009. 28

[80] Chun Ouyang, Marlon Dumas, Arthur HM Ter Hofstede, et al. **From business process models to process-oriented software systems: The BPMN to BPEL way**. 2006. 28

[81] Wil MP Van Der Aalst and Kristian Bisgaard Lassen. **Translating unstructured workflow processes to readable BPEL: Theory and implementation**. *Information and Software Technology*, **50**(3):131–159, 2008. 28

[82] Mika Koskela and Jyrki Haajanen. **Business Process Modeling and Execution**. *Tools and Technologies Report for SOAMeS Project*, 2007. 28

[83] Jan Mendling, Gustaf Neumann, and Markus Nüttgens. **A comparison of XML interchange formats for business process modelling**. *Workflow Handbook*, pages 185–198, 2005. 28

[84] Dirk Riehle and Heinz Züllighoven. **Understanding and using patterns in software development**. *TAPOS*, **2**(1):3–13, 1996. 28

[85] Martin Fowler. *Analysis patterns: reusable object models*. Addison-Wesley Professional, 1997. 28

[86] Wil MP van Der Aalst, Arthur HM Ter Hofstede, Bartek Kiepuszewski, and Alistair P Barros. **Workflow patterns**. *Distributed and Parallel Databases*, **14**(1):5–51, 2003. 28, 29, 75, 194

[87] Stephen A White. **Process modeling notations and workflow patterns**. *Workflow Handbook*, **2004**:265–294, 2004. 28

[88] Erich Gamma. *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995. 28

[89] The Workflow Patterns Initiative. **Workflow Patterns**. http://workflowpatterns.com, 2010. [Online; accessed 15-October-2016]. 29, 75

[90] Wil MP van der Aalst, Alistair P Barros, Arthur HM ter Hofstede, and Bartek Kiepuszewski. **Advanced workflow patterns**. In *International Conference on Cooperative Information Systems*, pages 18–29. Springer, 2000. 29, 194

[91] Wil MP van der Aalst and Arthur HM ter Hofstede. **Workflow patterns: On the expressive power of (petri-net-based) workflow languages**. In *Proceedings of the Fourth Workshop on the Practical Use of Coloured Petri Nets and CPN Tools (CPN 2002)*, **560**, pages 1–20. Citeseer, 2002. 29, 194

[92] Nick Russell, Arthur HM Ter Hofstede, and Nataliya Mulyar. **Workflow controlflow patterns: A revised view**. 2006. 29, 75, 194

[93] Workflow Manage Coalition. **Terminology & glossary**. *WFMC Document WFMCTC-1011, Workflow Management Coalition, Avenue Marcel Thiry*, **204**:1200, 1996. 29

[94] Jae Choi, Derek L Nazareth, and Hemant K Jain. **The impact of SOA implementation on IT-business alignment: A system dynamics approach**. *ACM Transactions on Management Information Systems (TMIS)*, **4**(1):3, 2013. 36

[95] Axel Martens. **Consistency between executable and abstract processes**. In *2005 IEEE International Conference on E-Technology, E-Commerce and E-Service*, pages 60–67. IEEE, 2005. 36

[96] Xiaohui Zhao, Chengfei Liu, Sira Yongchareon, Marek Kowalkiewicz, and Wasim Sadiq. **Role-based process view derivation and composition**. *ACM Transactions on Management Information Systems (TMIS)*, **6**(2):7, 2015. 36, 40

[97] Artem Polyvyanyy, Sergey Smirnov, and Mathias Weske. **Business process model abstraction**. In *Handbook on Business Process Management 1*, pages 147–165. Springer, 2015. 39

[98] Sergey Smirnov, Hajo A Reijers, and Mathias Weske. **A semantic approach for business process model abstraction**. In *Advanced Information Systems Engineering*, pages 497–511. Springer, 2011. 39

[99] Christian W Günther and Wil MP Van Der Aalst. **Fuzzy mining–adaptive process simplification based on multi-perspective metrics**. In *International Conference on Business Process Management*, pages 328–343. Springer, 2007. 39

[100] Rik Eshuis and Paul Grefen. **Constructing customized process views**. *Data & Knowledge Engineering*, **64**(2):419–438, 2008. 39

[101] H Eshuis and PWPJ Grefen. **Constructing customized process views**. 2007. 39

[102] Jens Kolb and Manfred Reichert. **A flexible approach for abstracting and personalizing large business process models**. *ACM SIGAPP Applied Computing Review*, **13**(1):6–18, 2013. 40

[103] Jens Kolb, Klaus Kammerer, and Manfred Reichert. **Updatable process views for user-centered adaption of large process models**. In *Service-Oriented Computing*, pages 484–498. Springer, 2012. 40

[104] Jens Kolb, Paul Hübner, and Manfred Reichert. **Automatically generating and updating user interface components in process-aware information systems**. In *On the Move to Meaningful Internet Systems: OTM 2012*, pages 444–454. Springer, 2012. 40, 42, 48

[105] Ralph Bobrik, Manfred Reichert, and Thomas Bauer. **View-based process visualization**. In *International Conference on Business Process Management*, pages 88–95. Springer, 2007. 40

[106] Sira Yongchareon, Chengfei Liu, Xiaohui Zhao, and Marek Kowalkiewicz. **BPMN process views construction**. In *Database Systems for Advanced Applications*, pages 550–564. Springer, 2010. 40

[107] Sira Yongchareon and Chengfei Liu. **A process view framework for artifact-centric business processes**. In *On the Move to Meaningful Internet Systems: OTM 2010*, pages 26–43. Springer, 2010. 41

[108] Duen-Ren Liu and Minxin Shen. **Workflow modeling for virtual processes: an order-preserving process-view approach**. *Information Systems*, **28**(6):505–532, 2003. 41

[109] Minxin Shen and Duen-Ren Liu. **Discovering role-relevant process-views for disseminating process knowledge**. *Expert Systems with Applications*, **26**(3):301–310, 2004. 41

[110] Vera Künzle and Manfred Reichert. **PHILharmonicFlows: towards a framework for object-aware process management**. *Journal of Software Maintenance and Evolution: Research and Practice*, **23**(4):205–244, 2011. 43, 49

[111] Vera Künzle and Manfred Reichert. **A modeling paradigm for integrating processes and data at the micro level**. In *Enterprise, Business-Process and Information Systems Modeling*, pages 201–215. Springer, 2011. 43, 49

[112] Kênia Sousa, Hildeberto Mendonça, Jean Vanderdonckt, Els Rogier, and Joannes Vandermeulen. **User interface derivation from business processes: a model-driven approach for organizational engineering**. In *Proceedings of the 2008 ACM Symposium on Applied Computing*, pages 553–560. ACM, 2008. 43, 48

[113] Quentin Limbourg, Jean Vanderdonckt, Benjamin Michotte, Laurent Bouillon, and Víctor López-Jaquero. **USIXML: a language supporting multi-path development of user interfaces**. In *International Workshop on Design, Specification, and Verification of Interactive Systems*, pages 200–220. Springer, 2004. 43, 48

[114] Quentin Limbourg and Jean Vanderdonckt. **Addressing the mapping problem in user interface design with UsiXML**. In *Proceedings of the 3rd Annual Conference on Task Models and Diagrams*, pages 155–163. ACM, 2004. 43, 48

[115] Jean Vanderdonckt. **A MDA-compliant environment for developing user interfaces of information systems**. In *International Conference on Advanced Information Systems Engineering*, pages 16–31. Springer, 2005. 43, 48

[116] Jean Vanderdonckt et al. **Model-driven engineering of user interfaces: Promises, successes, and failures**. In *5th Annual Romanian Conf. on Human-Computer Interaction ROCHI2008*, 2008. 43, 48

[117] David Cohn and Richard Hull. **Business artifacts: A data-centric approach to modeling business operations and processes**. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, **32**(3):3–9, 2009. 44, 49, 75

[118] Sira Yongchareon, Chengfei Liu, Xiaohui Zhao, and Jiajie Xu. **An artifact-centric approach to generating web-based business process driven user interfaces**. In *Web Information Systems Engineering–WISE 2010*, pages 419–427. Springer, 2010. 44, 49

[119] David Cohn, Pankaj Dhoolia, Fenno Heath Iii, Florian Pinel, and John Vergo. **Siena: From powerpoint to web app in 5 minutes**. In *Service-Oriented Computing–ICSOC 2008*, pages 722–723. Springer, 2008. 44, 49

[120] Fenno Terry Heath III, David Boaz, Manmohan Gupta, Roman Vaculín, Yutian Sun, Richard Hull, and Lior Limonad. **Barcelona: A design and runtime environment for declarative artifact-centric BPM**. In *Service-Oriented Computing*, pages 705–709. Springer, 2013. 44, 49

[121] Noi Sukaviriya, Senthil Mani, and Vibha Sinha. **Reflection of a year long model-driven business and ui modeling development project**. In *Human-Computer Interaction–INTERACT 2009*, pages 749–762. Springer, 2009. 45, 47

[122] Noi Sukaviriya, Vibha Sinha, Thejaswini Ramachandra, Senthil Mani, and Markus Stolze. **User-centered design and business process modeling: cross road in rapid prototyping tools**. In *Human-Computer Interaction–INTERACT 2007*, pages 165–178. Springer, 2007. 45, 47

[123] Wikipedia. **Principles of User Interface Design**. https://en.wikipedia.org/wiki/Principles_of_user_interface_design, 2016. [Online; accessed 13-November-2016]. 45

[124] Larry L Constantine and Lucy AD Lockwood. *Software for use: a practical guide to the models and methods of usage-centered design*. Pearson Education, 1999. 45

[125] Larry L Constantine and Lucy AD Lockwood. **Structure and style in use cases for user interface design**. *Object Modeling and User Interface Design*, pages 245–280, 2001. 45

[126] Larry L Constantine and Lucy AD Lockwood. **Usage-centered engineering for web applications**. *IEEE Software*, **19**(2):42, 2002. 45

[127] Larry L Constantine and Lucy AD Lockwood. **Usage-centered software engineering: an agile approach to integrating users, user interfaces, and usability into software engineering practice**. In *Proceedings of the 25th International Conference on Software Engineering*, pages 746–747. IEEE Computer Society, 2003. 45

[128] Alin Deutsch, Richard Hull, Fabio Patrizi, and Victor Vianu. **Automatic verification of data-centric business processes**. In *Proceedings of the 12th International Conference on Database Theory*, pages 252–267. ACM, 2009. 47

[129] Alin Deutsch, Liying Sui, and Victor Vianu. **Specification and verification of data-driven web applications**. *Journal of Computer and System Sciences*, **73**(3):442–474, 2007. 47

[130] Noi Sukaviriya, Vibha Sinha, Thejaswini Ramachandra, and Senthil Mani. **Model-driven approach for managing human interface design life cycle**. In *Model Driven Engineering Languages and Systems*, pages 226–240. Springer, 2007. 47

[131] Joan Fons, Vicente Pelechano, Manoli Albert, and Oscar Pastor. **Development of web applications from web enhanced conceptual schemas**. In *International Conference on Conceptual Modeling*, pages 232–245. Springer, 2003. 48

[132] Oscar Pastor, Joan Fons, Vicente Pelechano, and Silvia Abrahão. **Conceptual modelling of web applications: the OOWS approach**. In *Web Engineering*, pages 277–302. Springer, 2006. 48

[133] Richard Hull, Elio Damaggio, Fabiana Fournier, Manmohan Gupta, Fenno Terry Heath III, Stacy Hobson, Mark Linehan, Sridhar Maradugu, Anil Nigam, Piyawadee Sukaviriya, et al. **Introducing the guard-stage-milestone approach for specifying business entity lifecycles**. In *International Workshop on Web Services and Formal Methods*, pages 1–24. Springer, 2010. 49, 195

[134] Richard Hull, Elio Damaggio, Riccardo De Masellis, Fabiana Fournier, Manmohan Gupta, Fenno Terry Heath III, Stacy Hobson, Mark Linehan, Sridhar Maradugu, Anil Nigam, et al. **Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events**. In *Proceedings of the 5th ACM International Conference on Distributed Event-based System*, pages 51–62. ACM, 2011. 49, 195

[135] Andreas Meyer and Mathias Weske. **Activity-centric and artifact-centric process model roundtrip**. In *International Conference on Business Process Management*, pages 167–181. Springer, 2013. 51

[136] Rik Eshuis and Pieter Van Gorp. **Synthesizing object life cycles from business process models**. In *International Conference on Conceptual Modeling*, pages 307–320. Springer, 2012. 52

[137] Cagdas E Gerede and Jianwen Su. *Specification and verification of artifact behaviors in business process models*. Springer, 2007. 75

[138] Thongchai Srivardhana and Suzanne D Pawlowski. **ERP systems as an enabler of sustained business process innovation: A knowledge-based view**. *The Journal of Strategic Information Systems*, **16**(1):51–69, 2007. 78

[139] Gautam Ray, Jay B Barney, and Waleed A Muhanna. **Capabilities, business processes, and competitive advantage: choosing the dependent variable in empirical tests of the resource-based view**. *Strategic Management Journal*, **25**(1):23–37, 2004. 78

[140] SB Rinderle, Ralph Bobrik, Manfred Reichert, and Thomas Bauer. **Business process visualization-use cases, challenges, solutions**. 2006. 78

[141] Wil MP van der Aalst and Stefan Jablonski. **Dealing with workflow change: identification of issues and solutions**. *Computer Systems Science and Engineering*, **15**(5):267–276, 2000. 194

[142] T Koulopoulos. **The Workflow Imperative**, 1995. 194

[143] Shazia W Sadiq. **Handling dynamic schema change in process models**. In *Database Conference, 2000. ADC 2000. Proceedings. 11th Australasian*, pages 120–126. IEEE, 2000. 194

[144] Shazia W Sadiq, Olivera Marjanovic, and Maria E Orlowska. **Managing change and time in dynamic workflow processes**. *International Journal of Cooperative Information Systems*, **9**(01n02):93–116, 2000. 194

[145] Fabio Casati, Stefano Ceri, Barbara Pernici, and Giuseppe Pozzi. **Workflow evolution**. *Data & Knowledge Engineering*, **24**(3):211–238, 1998. 194

[146] Matthias Weidlich, Mathias Weske, and Jan Mendling. **Change propagation in process models using behavioural profiles**. In *Services Computing, 2009. SCC'09. IEEE International Conference on*, pages 33–40. IEEE, 2009. 194