

Institut für Architektur von Anwendungssystemen (IAAS)

Universität Stuttgart  
Universitätsstraße 38  
D - 70569 Stuttgart

Diplomarbeit Nr. 2944

Mapping Interconnection  
Choreography Models to Interaction  
Choreography  
Models

Fei Wu

Studiengang:	Informatik
Prüfer:	Prof. Dr. Frank Leymann
Betreuer:	Dipl.-Inf. Oliver Kopp
begonnen am:	June 1st, 2009
beendet am:	December 14st, 2009
CR-Klassifikation:	H.4.1

# Contents

<b>1 Introduction.....</b>	<b>7</b>
<b>2 Background .....</b>	<b>9</b>
2.1 Introduction to Service Choreographies .....	9
2.2 Interconnection Models using BPMN .....	9
2.2.1 BPMN .....	10
2.2.2 Drawbacks of interconnection model .....	14
2.2.3 Anti-patterns in BPMN .....	14
2.3 Interaction models using iBPMN .....	16
2.3.1 iBPMN .....	16
2.3.2 Avoiding anti-patterns .....	17
2.4 Intermediate models using interaction Petri nets .....	18
2.5 Related work .....	21
<b>3 Basic idea .....</b>	<b>22</b>
3.1 Fundamental Difference between BPMN and iBPMN .....	22
3.2 Approaches .....	22
<b>4 Mapping interconnection models to iPetri nets .....</b>	<b>26</b>
4.1 Syntax of BPMN Choreography Model .....	26
4.2 Mapping of Basic Constructs.....	28
4.3 Sequence of Activities .....	31
4.4 Message Flow .....	32
4.5 Looping Constructs.....	34
4.6 Reduction of the models .....	36
4.6.1 Background knowledge for reduction.....	38
4.6.2 Reduction rules .....	39
4.7 Restriction of the mapping.....	46
<b>5 Mapping iPetri nets to iBPMN .....</b>	<b>47</b>
5.1 Syntax of iBPMN Choreography.....	47
5.2 Mapping from iPetri nets onto iBPMN .....	49
5.2.1 Mapping of Basic Constructs.....	49
5.2.2 Normalized iPetri nets.....	50
5.2.3 Mapping of Data-based and Event-based gateway .....	52
5.2.4 Mapping of looping constructs .....	55
5.2.5 Algorithm.....	58

<b>6 Summary and outlook .....</b>	<b>61</b>
<b>7 Bibliography .....</b>	<b>62</b>

## List of Figures

Figure 2.1: BPMN language overview .....	10
Figure 2.2: Travel booking scenario .....	13
Figure 2.3 : D1. Incompatible branching behaviour.....	15
Figure 2.4:D2. Impossible data-based decisions .....	15
Figure 2.5:O1. Contradicting sequence flow .....	15
Figure 2.6:O3. Unilateral sequentialization.....	15
Figure 2.7: Overview of iPBMN language constructs .....	16
Figure 2.8: Travel booking in iPBMN.....	17
Figure 2.9: Travel booking in iPetri nets .....	20
Figure 3.1: Sketch of direct approach.....	24
Figure 4.1: Mapping of Basic Constructs .....	29
Figure 4.2: Algorithm for the mapping of basic constructs.....	31
Figure 4.3: Sequence of Activities in BPMN .....	32
Figure 4.4: Mapping to iPetri nets .....	32
Figure 4.5: Mapping of Message Flow.....	33
Figure 4.6 : incoming message as trigger of process.....	33
Figure 4.7: Mapping of message flow .....	34
Figure 4.8: A piece of algorithm for message flows.....	34
Figure 4.9: Looping constructs.....	35
Figure 4.10: Loop constructs in iPetri nets.....	35
Figure 4.11: Mapping of loop constructs.....	36
Figure 4.12: Travel booking in iPetri nets .....	37
Figure 4.13: Fusion of series places .....	40

Figure 4.14: Fusion of series transitions.....	40
Figure 4.15: Fusion of parallel places .....	41
Figure 4.16: Fusion of parallel transitions.....	41
Figure 4.17: Reduce redundant place .....	42
Figure 4.18: Reachability graph .....	43
Figure 4.19: Reduced iPetri net using reduction rules .....	43
Figure 4.20: Algorithm for iPetri nets reduction rule1 to rule4.....	45
Figure 4.21: Algorithm for removing redundant places .....	45
Figure 4.22: Algorithm of identifying redundant place.....	45
Figure 5.1: Mapping of basic constructs .....	49
Figure 5.2: Adding silent transition before labelled transition .....	51
Figure 5.3: Adding silent transition after labelled transition .....	51
Figure 5.4: Type 1 decisions gateway in iPetri nets and iBPMN .....	53
Figure 5.5: Type2 decision gateway in iPetri nets and iBPMN.....	54
Figure 5.6: Type 3 decision gateway in iPetri nets and iBPMN.....	54
Figure 5.7: Type 4 mixed choices.....	55
Figure 5.8: looping constructs representations with XOR gateway .....	56
Figure 5.9: looping constructs representation with marked task.....	57
Figure 5.10: Algorithm for adding silent transitions .....	58
Figure 5.11 presents an algorithm for the mapping from iPetri net onto iBPMN.....	60



# 1 Introduction

As an architecture style for developing and integrating software systems, service-oriented architecture (SOA for short) is widely accepted by IT developer and companies. Within the context of SOA functionalities as interoperable services are packaged. These services are loosely coupled, can be used by several organizations and communicate with each other. Through the communicating of the services data will be exchanged or an activity will be coordinated among the services. A common realization of SOA is the Web services platform [WCL+05].

As an independent module application a service can not accomplish a whole business process. It will be achieved by a flow of several service invocations. For more complex business collaboration services can be combined to a service. Such services can be offered as web services. To fulfil the complex business requirements web services are needed to be formed a Business Process. Using Business Process Execution Language (BPEL for short) [BPEL07] the dependency of service invocations and data flow among the services can be defined, and a business process can be formed. BPEL depicts the way how to orchestrate and coordinate web services.

While BPEL orchestrates the services from the view point of single business process, Service Choreographies describe the conversations between services from a global perspective [DKB08]. There are two approaches to model service choreography: interconnection model and interaction model. In this thesis these two approaches will be presented. While interconnection models are more on a detailed level, an interaction model is more on a higher-level view. Currently, only transformations from interaction models to interconnection models are available; there are no transformations from interconnection to interaction models. The goal of this thesis is to fill this gap.

The aim of this thesis is to realize the transformation from interconnection model to interaction model. The Business Process Management Notation (BPMN for short) [BPMN06] and interaction BPMN (iBPMN for short) [DB07] will be chosen as choreography languages. The both modelling languages are on the implementation-independent level. Moreover iBPMN is the extension of, and it reuses some basic constructs of BPMN. This will simplify the transformation.

While BPMN lists all interactions through the message flows interconnecting the different internal behaviour models, iBPMN uses atomic message exchanges i.e. interactions as the building blocks of the models. Hence we use iPetri nets as the intermediate models to achieve the transformation from interconnected models to interaction models. The transformation will be accomplished in two steps: i) mapping from BPMN models to iPetri nets models; ii) mapping from iPetri nets to iBPMN models.

This thesis is structured as follows:

Chapter 2 covers basic knowledge about Service Choreographies, two approaches modelling choreographies: interconnections models and interaction models; there exist different choreography languages. In Chapter 2 BPMN will be introduced as interconnection choreography language. And iBPMN will be introduced as interaction choreography language. As a formal model and intermediate model in the mapping, the definition and introduction of interaction Petri nets will be given. The related work is introduced in this chapter.

Chapter 3 presents the fundamental difference between the source and target models, i.e. BPMN and iBPMN. And then two approaches for the mapping are proposed. One is direct mapping from BPMN onto iBPMN. Another is indirect mapping approach with intermediate models. In this thesis we use the latter approach, which maps firstly BPMN onto iPetri nets, then maps iPetri nets onto iBPMN.

Chapter 4 presents the first step of mapping from BPMN onto iPetri nets. In order to reduce a large iPetri net after mapping, the reduction rules for iPetri nets are presented.

Chapter 5 presents the second step of the mapping from iPetri nets onto iBPMN. The corresponding algorithms are given.

Finally, we give a summary and outlook in Chapter 6.

## **2 Background**

Within a context of Service-oriented Architecture (SOA) Services are loosely coupled components and can communicate with each other. A Business process is formed through the combination of the services and can interact with other business processes. Using the BPEL the Business Processes orchestration can be described from a view point of a single process. In a context of collaboration of business processes more complex conversations will be handled. Therefore a global view point of modelling is needed. In next section this global modelling will be introduced.

### **2.1 Introduction to Service Choreographies**

As stated in [BDO05], a service choreography model is used to describe a business-to-business collaboration. The complex interactions are captured in the choreography models. In collaboration scenarios multiple participants are involved. In such scenarios like “travel booking” scenario, various messages are exchanged between several participants. In such scenarios the simple request/response interaction is not sufficient. More complex interactions are considered in collaboration scenarios. For instance, in “travel boeing” scenario, traveller, travel agency and airline services are involved. Traveller makes a travel plan firstly. Then he sends the request to travel agency. Travel agency request to airline services for the airline information and list the proper airline information to traveller. If traveller confirms the airline, the travel agency issues the itinerary for traveller. Meanwhile airline services issues the ticket for traveller. In this scenario the complex interactions are involved. In contrast to the service orchestrations all interactions and constrains for the collaboration in the context must be captured.

There exist two modelling approaches for service choreographies: interconnection models and interaction models. In the case of interconnection models processes are described from every participant’s view point, and they are interconnected by the messages between them. In the case of interaction models interactions between the processes form the service choreography [DKB08].

In next two sections the two modelling approaches and corresponding languages for describing service choreographies will be introduced.

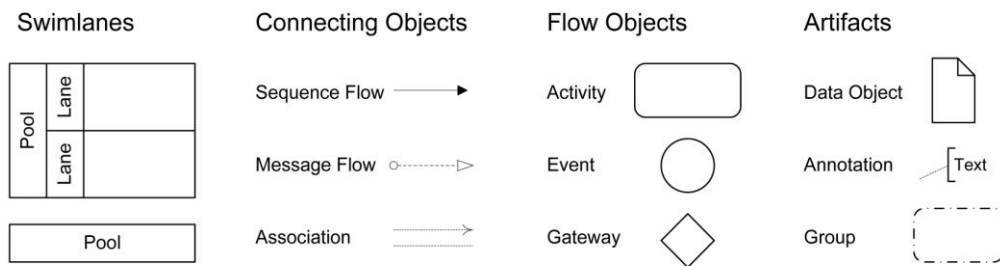
### **2.2 Interconnection Models using BPMN**

We use service choreographies to capture the interactions among the processes. An interconnection model does this in a way, that the processes are connected by send and receive messages. In an interconnection model it describes the observable behaviour for each participant and the send and receives messages that interconnect each participant. That means that the control flow is defined per participant [DKB08].

## 2.2.1 BPMN

The BPMN is the de-facto standard for graphical process modelling. It can be used to describe choreographies. It defines the control flow constructs with a set of graphical notations. Additionally it provides the definition of message flow, with which different processes are interconnected. In BPMN model there is the notion of pools, which represent different roles or entities. Therefore the different activities of the same process are assigned to different organizational units [DKB08]. And the message flows capture interactions of different roles. The diagram in BPMN is called Business Process Diagram (BPD), which is made up of the graphical elementary constructs of BPMN and presents the flowcharting diagram of the business process.

Figure 2.1 shows the elementary constructs of BPMN, which are divided in four basic categories: flow objects, connecting objects, swimlanes and artifacts.



**Figure 2.1: BPMN language overview according to [Dec09]**

With the core elements of *flow objects* it can make up a BPD. These three core elements are *Event*, *Activity* and *Gateway*. *Events* are denoted as single circles and represent something that happens during the course of business process. They can be a start event which is the start of the process, or an intermediate event which is in between the process, or an end event which is the end of the process. They may have a trigger to initiate the event or have a result. *Activities* are denoted as rounded rectangles and represent the work performed. A task is an elementary atomic activity within a process. Compounded activities within a process are called sub-processes. Gateways are denoted as diamonds and used to specify the behavioural dependencies in the business process except sequence flow i.e. advanced branching and concurrency.

*Connecting objects* are used to connect the flow objects. In BPMN three types of connectors are defined: sequence flow, message flow and association. The

sequence flow determines the order of the activities within the process. The message flow represents the flow of messages between the different participants.

In BPMN a concept of the *swimlanes* is defined to partition the activities in different organizations. There two types of *swimlanes*: *pools* and *lanes*. In BPMN a *pool* is used to represent a role or participant of a process. A lane is seen as a sub-partition of a pool and used to organize the activities in a pool. *Pools* represent several separate participants involved in a scenario. The flow diagram in a pool can be seen as a separate process. The message flows connect the pools to represent the interactions between the different participants. The lanes are used to partition the activities of different functional organizations in a pool. But message flows cannot connect the activities in different lanes but in different pools.

Modeller can use *artifacts* to extend the diagram and add context information to the diagram, which don't impact the performance of the business process. There are three types of *artifacts* in current version of BPMN: *data object*, *group* and *annotation*. *Data objects* show what data is required or produced by activities. The input or output data can be distinguished by the direction of the connected association. With *group*, which is represented by a rounded rectangle with a dashed line, activities are grouped for documentation or analysis purpose. For the reader of the diagram, modeller can use *Annotations* mechanism to add additional text information in a BPD.

Figure 2.2 illustrates an example of a collaboration scenario using BPMN. For the simplifying in this scenario just three participants - traveller, travel agency and airline services are considered.

A traveller makes a travel plan, and sends a request to the travel agency. The request lists the destination, the favourite travel date and even budget.

When the Travel Agency gets the request from the Traveller, he sends a request of airline order to an airline service. After the acknowledgement of the airline service the travel agency requests the airline information which feats the traveller's requirements. Then the airline service queries the airlines and sends suitable airlines information to the travel agency.

The travel agency decides the airline for the Traveller. When there is no appropriate airline, the travel agency will cancel the airline booking, and inform respectively the airline service and traveller. When there is matched airline, Travel agency sends the order of ticket and meanwhile informs traveller. Airline service makes the reservation, and then sends confirmation to the travel agency. Later Airline service issues the ticket to traveller. After the confirmation of airline confirmation travel agency creates an itinerary for traveller and issues the itinerary to the traveller.

This example shows an interconnection choreography model using BPMN. In this Model the internal behaviour of the process is represented in a pool. And they are interconnected by the message flows.

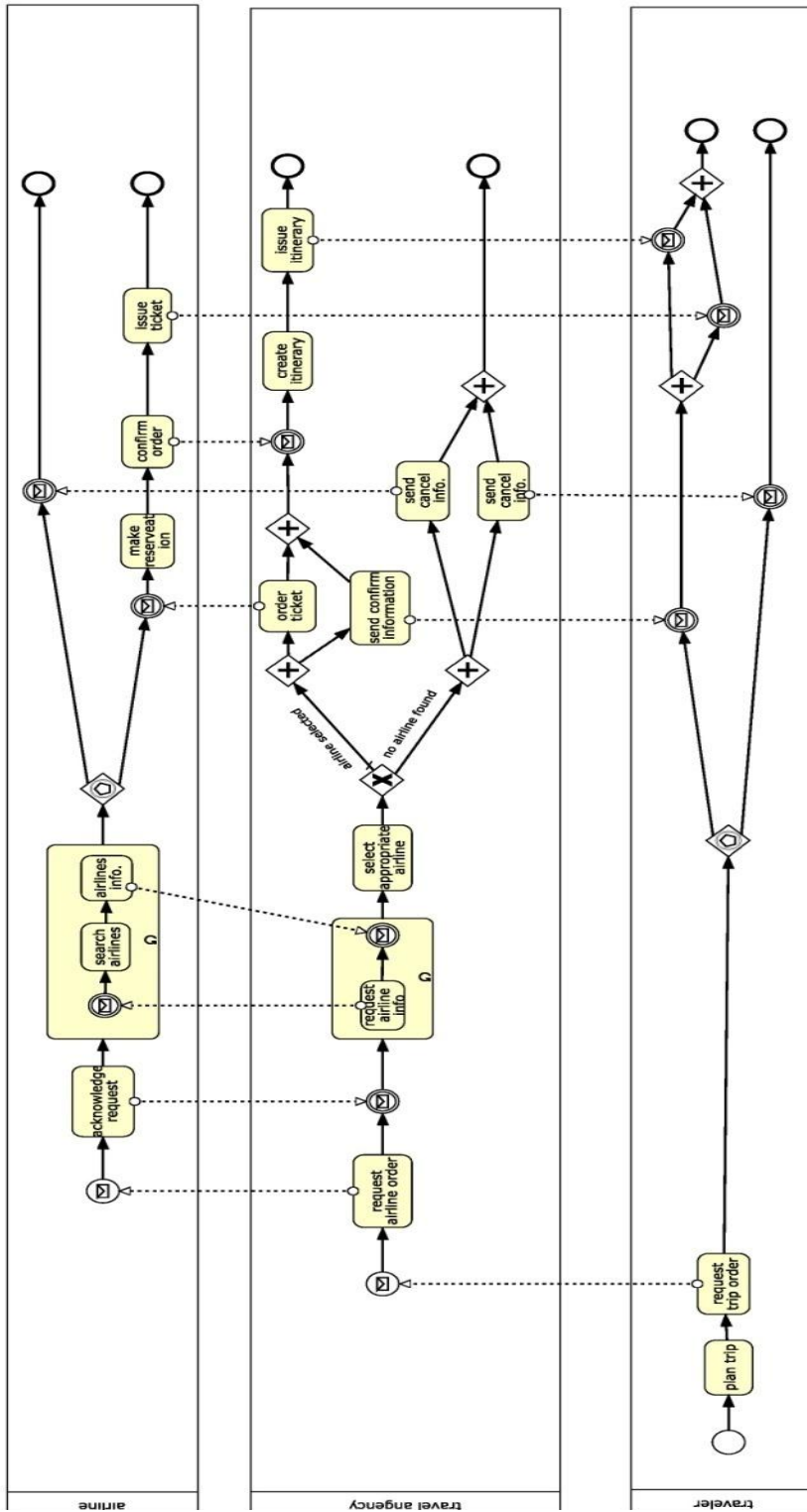


Figure 2.2: Travel booking scenario

### 2.2.2 Drawbacks of interconnection model

However there exist a number of drawbacks in this modelling style. These drawbacks are discussed in detail in [DB07]. Next we will discuss these drawbacks in short. In interconnection modelling style the ordering constraints are defined in respective role. Therefore if a modeller wants to define an ordering constraint, he must define it for both roles involved in the interaction. This leads to the redundancy of modelling. Another drawback in interconnection modelling style is potentially incompatible behaviour. A common error in this situation is deadlock. It occurs often in the case of data-based XOR-gateways. To avoid these drawbacks in modelling a better suited choreography language is required. In Section 2.3 we will introduce another modelling style and corresponding choreography language, which avoid the drawbacks in the interconnection modelling style.

### 2.2.3 Anti-patterns in BPMN

In this section we present some anti-patterns in BPMN, which are syntactical correct but lead to modelling errors. These anti-patterns are introduced in detail in the work of [Dec09]. In [Dec09] there are 8 anti-patterns presented. They are categorized into three classes by G.Decker : 1) class related to decision-making (*D*), 2) class related ordering constraints (*O*), 3) and class related to process instance creation and termination(*I*). The third class (*I*) is out of scope of this thesis, hereby we will not discuss the third class. In the other classes, we will discuss the part (*D1, D2, D3, O1 and O3 presented in [Dec09]* ), in which we are interested in this thesis .

***D1. Incompatible branching behaviour.*** In BPMN we have two types of decision making. One is data-based decision. That means that the branching decision is made by process data within a process instance. The other is event-based decision, which is made by external trigger e.g. receiving a message. D1 depicts a incompatible situation in interconnection model. As shown in Figure 2.3, two interconnected processes A and B make the branching decisions for own process respectively, e.g. they both contain data-based gateways. However the activities following gateway of one process are triggered by the activities following gateway of another process. It obviously leads to incompatibility.

***D2. Impossible data-based decisions.*** In this situation the data-based gateway is used with conditions that cannot be evaluated by corresponding role. That means that the role might not be able to make the decision for the branching. For example, as shown in Figure 2.4, a bidder decides whether he has won the auction. Actually the bidders cannot know the result of the auction until he has the notification from the auctioning service.

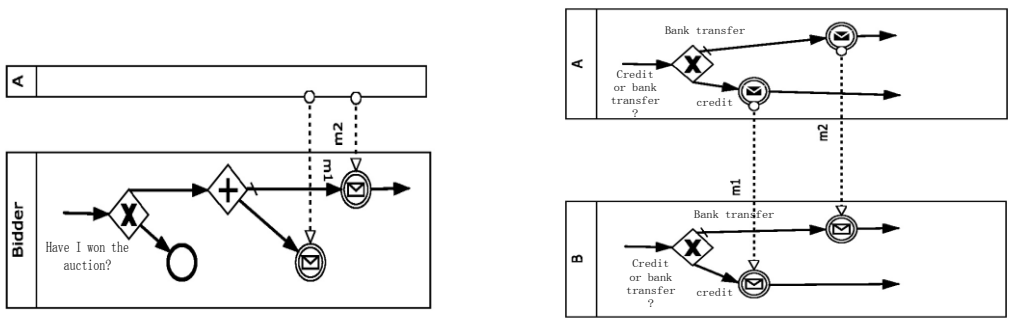


Figure 2.3 : D1. Incompatible branching behaviour [Dec09] Figure 2.4:D2. Impossible data-based decisions[Dec09]

**D3. Mixed choices.** Mixed choices appear in the cases where a gateway is followed by a sending activity and a receiving activity. That means that the role makes a decision to send or receive a message. But *Mixed choices* cannot be syntactically supported by BPMN. Hence, it will not be regarded as source model in this thesis. If readers are interested in this case, the detail about *Mixed choices* is presented in the work of [Dec09].

**O1. Contradicting sequence flow.** In connection modelling modeller must respectively define the flow relationships for different roles. Meanwhile modeller must to establish the ordering constraints of interactions. The redundant modelling of ordering constraints between interactions leads to contradicting sequence flow. In the following we present an example in Figure 2.5 to illustrate this situation. In this example, we notice that seller does not send the goods until he gets the payment from buyer. But buyer does not pay the bill until he gets the goods. It obviously leads to deadlock.

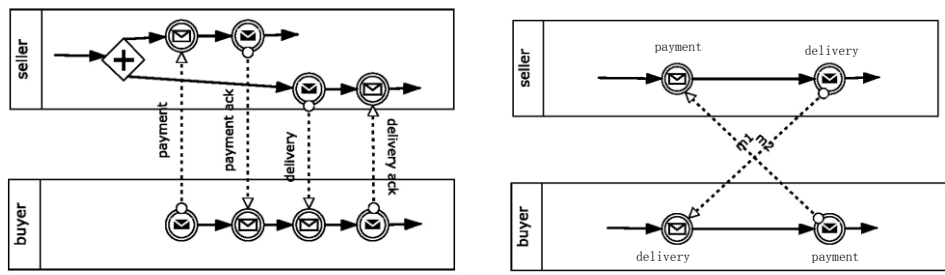


Figure 2.5:O1. Contradicting sequence flow [Dec09]

Figure 2.6:O3. Unilateral sequentialization[Dec09]

**O3. Unilateral sequentialization.** In this case, different ordering constraints are modelled for the message exchanging activities in pair in opposite roles. As shown in Figure 2.6, seller can process the payment and delivery in parallel, while buyer must process the payment first. It leads to problem when the delivery allowed processed before payment.

These anti-patterns show the drawbacks in interconnection modelling style. We will in Section 2.3.2 discuss whether these problems occur in interaction modelling style.

## 2.3 Interaction models using iBPMN

### 2.3.1 iBPMN

While in an interconnection model observable behaviour of different roles are interconnected by the message flows, the elementary interactions are building blocks in an interaction model. This section will introduce a modelling language which reuses the notations of BPMN and describes choreography in interaction modelling style. This language is the extension of BPMN, named iBPMN [DB07].

Figure 2.7 shows the elementary constructs of iBPMN. Roles are represented as rectangles. In iBPMN the internal behaviours of the participants are not described, i.e. the pools are empty. Elementary interaction is represented as a message event using a letter symbol. With directed message flow or undirected message flow the participants involved in an interaction and the elementary interaction are connected. Directed message flow explicitly shows who sends the message and who receives the message. Undirected message flow indicates the roles who involve in an interaction.

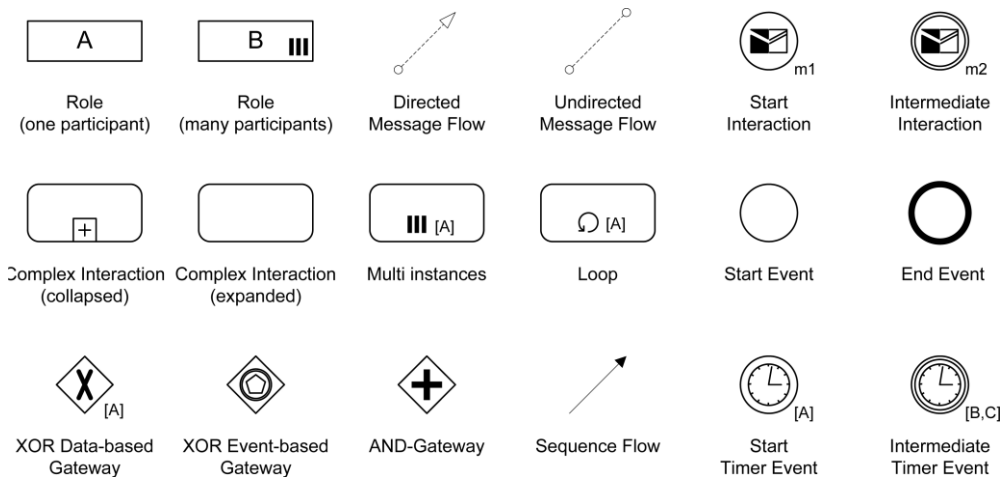


Figure 2.7: Overview of iBPMN language constructs according to [Dec08]

Complex interaction is made up several elementary interactions. There are two types of complex interactions in iBPMN, which both are represented by rounded rectangles. One is the collapsed complex interaction, in which all elementary interactions are hidden. The other is expanded complex interaction, in which

further elements can be placed. Additional loop and multi-instance can also be defined in complex interaction.

XOR Data-based Gateway, XOR event-based gateways and AND-gateways are the control flow branching in the iBPMN. Start and end events are the same as in BPMN. Timer events must be assigned owners. The same travel booking in Section 2.2 will be showed in Figure 2.8 using iBPMN.

This section will introduce a formal model that enables unambiguous semantics of interconnection model and interaction model. This model is an extension to Petri nets – interaction Petri nets (iPetri nets for short). And iPetri nets will be chosen as the intermediate models of the transformation from interconnection models to interaction models.

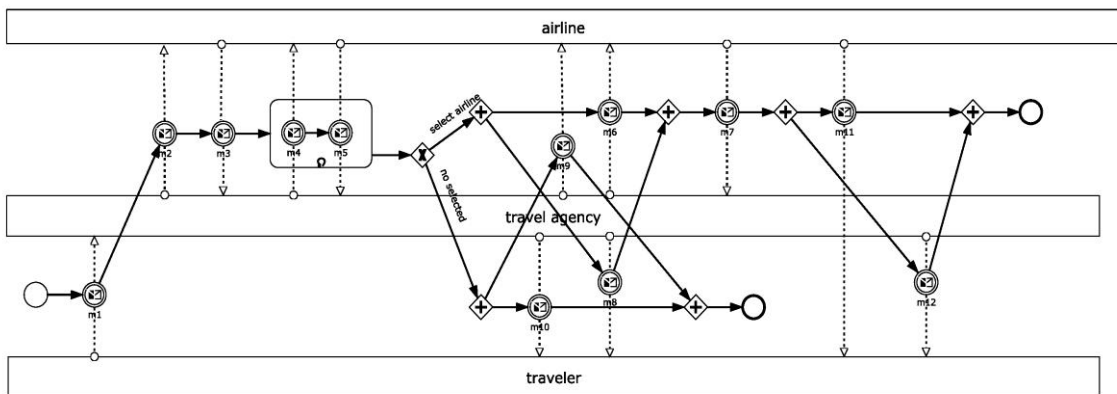


Figure 2.8: Travel booking in iBPMN

### 2.3.2 Avoiding anti-patterns

In this section we discuss the anti-patterns in BPMN will be avoided in iBPMN. Using interaction modelling style, some of anti-patterns in BPMN are largely avoided in iBPMN.

Incompatible branching behaviour (*D1*) comes from the modelling with wrong ownership of decision making. Using interaction modelling style, modeller largely avoids modelling the ownership of choices respectively for different roles in a conversation. The modeller models the decision making from a global view.

Impossible data-based decisions (*D2*) cannot completely be avoided in iBPMN. Solving this problem is out of scope of this thesis.

Mixed Choices (*D3*) are syntactically not supported in BPMN. However, using interaction modelling e.g. iBPMN style we can present *D3* correctly. For instance, we can use event-based gateway in iBPMN to model mixed choices. However, due to the syntactical wrong in BPMN, we will not use Mixed choices as source model. We assume that source model does not contain anti-pattern *D3*.

*O1* and *O3* will not occur in interaction models. All interactions and their flow dependencies will be modelled from a global view. The flow relations will be not redundant modelled. Hence, these anti-patterns cannot be modelled in iBPMN.

## 2.4 Intermediate models using interaction Petri nets

As presented in [Reis85] classical Petri nets is a place/transition net that is a directed bipartite graph, and where places and transitions are with arcs connected. There are zero or many tokens in place, which enables transition fire.

**Definition 1. (Petri nets)** A Petri net is a triple  $(P, T, F)$ , where

- $P$  is a finite set of places,  $T$  is a finite set of transitions, and  $T \cap P = \emptyset$ ;
- $F \in (P \times T) \cup (T \times P)$  is a flow relation;
- $p \in P, t \in T, \cdot t = \{p | (p, t) \in F\}$  is called the set of input places of transitions  $t$ ,  $t \cdot = \{t | (t, p) \in F\}$  is called the set of output places of transition  $t$ .

As introduced in [DeWe07], for capturing the interactions of choreographies the classical Petri nets must be extended. This extension of classical Petri nets is interaction Petri nets (iPetri nets for short). In iPetri nets transitions are extended to be labelled. The labelled transitions can represent message exchanges in the conversations. The following is the definition of iPetri nets according to [DeWe07].

**Definition 2. (Interaction Petri net)** A interaction Petri net is a tuple  $iPN = (P, T, F, R, s, r, timer, MT, t, M_0)$  where

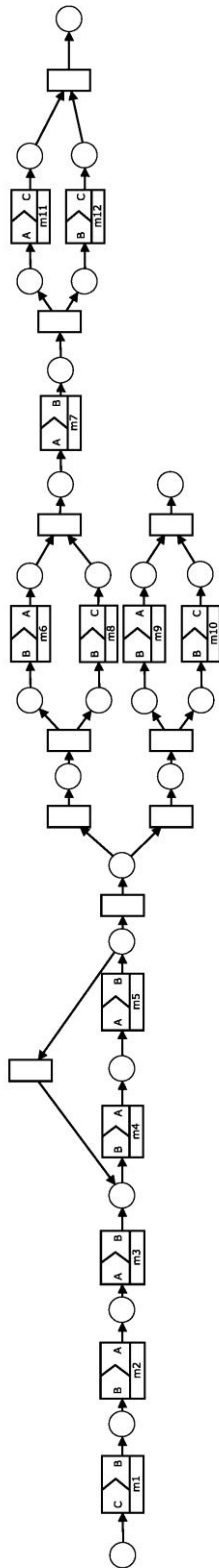
- $P$  is a set of places,
- $T$  is a set of transitions, where  $T = T_i \cup T_c \cup T_u$ ,  $T_i$  represents interaction occurrences.  $T_c$  is the set of controlled silent transitions, and  $T_u$  is the set of uncontrolled silent transitions,
- $F \subseteq (P \times T) \cup (T \times P)$  is a flow relation connecting places with interaction model occurrences,
- $R$  is a set of roles,

- $s, r, timer: T \rightarrow R$  are functions assigning the sender and the receiver role to an interaction model occurrence, or assigning the timer flag to a Transition to represent a Timer,
- $t: T \rightarrow MT$  is a function assigning a message type to an interaction model occurrence and
- $M_0: P \rightarrow \mathbb{N}$  is the initial marking.

The set of interaction models  $IM$  for an interaction Petri net is defined as  $IM := \{im \in (R \times R \times MT) \mid \exists t \in T(im = (s(t), r(t), t(t)))\}$ . We introduce the auxiliary function roles:  $T \rightarrow \wp(R)$ .

Figure 2.9 shows an example of the model using iPetri nets.

As defined above, we notice that the labelled transitions represent the interactions of the models. In the next chapter we will discuss the reduction rules, with which we will reduce those uncontrolled silent transitions, which are not involved in conversations, not timer or not related with gateways. Then we will achieve the models, which just contain interactions represented by labelled transition and some branch/merge constructs represented by silent transitions. The structure of the reduced models is similar with iBPMN's structure. While iPetri nets uses labelled transitions as interactions, message events as interactions in iBPMN are the building blocks. Consequently transformation from reduced iPetri nets onto iBPMN is only about transforming types. In next two chapters will discuss the transformation on the basis of concrete examples.



A: airline  
 B: Travel Agency  
 C: Traveler

Figure 2.9: Travel booking in iPetri nets

## 2.5 Related work

As introduced the goal of this thesis is to provide a conceptual mapping from interconnection models to interaction models. Currently there is no work dealing with it. In present works there exist mappings from interaction to interconnection models [DeWe08]. Decker et al. in [DeWe08] provide an approach to generate observable behaviour models for the different roles from iBPMN choreographies. In this approach interactions are converted into corresponding communicated activities. Other constructs are mapped to corresponding constructs. Meanwhile these constructs can be divided into two classifications. One is the class of constructs, which are not relevant for the observable behaviour model of a role. The other is the class of rest constructs. Such constructs which are not relevant for a role, are converted to  $\tau$ -nodes. And then  $\tau$ -nodes are removed. To preserve the control flow dependencies  $\tau$  nodes are used for us to achieve the inverse mapping. And an approach for formal semantic of iBPMN is introduced. An extension of Petri nets, interaction Petri nets (iPetri nets for short), is defined in [DeWe08]. In this thesis, our mapping uses the iPetri nets as the intermediate model.

In [DDO07a] and [DDO07b] the mapping from BPMN onto Petri nets is introduced. Petri nets model is regarded as a formal model for BPMN. In these papers Dijkman et al. provide the definition of well-formed BPMN model and mapping well-formed core BPMN models onto Petri nets. The basic idea is using labelled transitions to model tasks and events, and silent transitions to model internal control flow within a role. However there exist constructs, which are not fully supported. They are (i) multi-instances; (ii) concurrently executed exception handling; (iii) OR-gateway. In [AH02] and [AH04] a solution for the three problems will be introduced. The mapping in our thesis is based on this approach. But we use iPetri nets to model the BPMN choreography model.

The mapping from graph-oriented models (e.g. BPMN) onto block-oriented models (e.g. BPEL) is discussed in a lot of work. Especially in [ODH+07], Ouyang et al. provide an approach to translate BPMN into BPEL. These two languages have different structures. This is the challenge of the mapping from BPMN onto BPEL. The approach in [ODH+07] identifies the block-structured patterns in BPMN and then translates these patterns into blocks in BPEL. Not only well-structured pattern can be identified, but also the quasi-structured fragments can be also properly translated into block-structured constructs. In this work the concept of components is proposed to identify the patterns. This concept will be used in this thesis for identifying the looping construct in the mapping from iPetri nets onto iBPMN.

## 3 Basic idea

In this Chapter we introduce the basic ideas and reasons we choose the approach in this thesis.

### 3.1 Fundamental Difference between BPMN and iBPMN

As introduced in Chapter 2, we know that both of BPMN and iBPMN are graph-based modelling languages. iBPMN is an extension of BPMN. Some notations in BPMN are reused in iBPMN. However, the modelling using iBPMN is focused on interactions, which are the building blocks in interaction modelling. Hence, there are some notions in iBPMN that are similar to the notions in BPMN. It simplifies the mapping to reuse the notions. For example, the notions of control flows such as gateways are reused in iBPMN. And the syntax and semantics are not changed. However, as Decker summarized in [Dec09], there are fundamental differences between BPMN and iBPMN.

First, message exchanges using sending/receiving tasks or intermediate message events are replaced by atomic interactions in iBPMN. In iBPMN the interactions represent at the same time two activities in BPMN [Dec09].

Secondly, from a global viewpoint complex interactions can be decomposed into refined interactions. That means that information exchanges can be step by step refined into message exchanges, which are building blocks of interaction modelling. However, in BPMN decomposition can be done from the viewpoint of a process [Dec09].

Thirdly, while in BPMN control flow is modelled in a process, in iBPMN control flow dependency is not assigned to any role in choreography. That means that control flow dependency in BPMN will be assigned to the individual role. But in iBPMN the control flow dependency is not specified [Dec09].

Next difference is ownership of decisions. While in BPMN the ownership of gateway decisions is assigned to an individual role, more than one role is responsible to the ownership of a gateway decision [Dec09].

Process instance creation and termination is another difference between BPMN and iBPMN. Start event and end event in BPMN represent the instance creation and termination. In iBPMN there is no explicit specification [Dec09].

### 3.2 Approaches

According to the common character and difference, we propose two approaches to map the BPMN onto iBPMN. One approach is *direct* approach, another is *indirect* approach.

### *Direct approaches*

In this approach, we try to directly translate BPMN into iBPMN. As introduced in Chapter 2, the building blocks in iBPMN are elementary interactions in choreographies. Intuitively we will identify the corresponding constructs in BPMN and then map them onto iBPMN interactions. In BPMN we use message flows to represent the interactions. Hence, we can pack up the send task and receive task/ intermediate message event in BPMN to an interaction in iBPMN, i.e. the tasks or events involved in a conversation will be regarded as a whole. A whole packet in BPMN will be translated into an interaction in iBPMN. The timer events will be kept in iBPMN. But the controlling role of the timer events will be provided. The other tasks or events which are not involved in conversations will be ignored, because interaction choreographies only focus on the interactions among the participants, but not on the internal activities [DeWe08]. The identification of ordering constraint of interactions in iBPMN is the challenge in this approach. Considering the example in Figure 2.2, the first interaction in this scenario is the send task “request trip order” and intermediate message event in role “travel agency”. In role “traveller” we note that send task “request trip order” is followed by an event-based gateway which is also followed by two intermediate message events. Meanwhile the corresponding intermediate message event in “travel agency” is followed by a send task “request airline order”. The send task “request trip order” and the corresponding intermediate message event will be as a whole translated into an interaction in iBPMN. The challenge is which interaction following the first interaction will be executed first. The analysis of the flow dependencies is complicated.

Now we discuss another direct approach, which might make the analysis of the flow dependencies not so complicated. We sketch this approach in Figure 3.1. We can analyse and list all interactions one by one in different processes. We go through first process, and find all activities which are involved in conversations and mark them as interactions. The sender and receiver roles are attached to these interactions. Meanwhile the activities which communicate with these interactions in other processes are also connected with these marked interactions. The first class of interactions in first pool are arranged according to the ordering constraints of the first process. Then we go through the second process. We analyse and list all communicated activities which are not listed in first step. As in first step, we mark these activities as interactions with information of interacting roles. The central problem in this step is to integrate the listed interactions in first step into the control flow of interactions listed in the second step. It is also a challenge to integrate two control flows.

To solve this problem we propose another approach, which use a third-party model as an intermediate model of the mapping. In the following we introduce this approach.

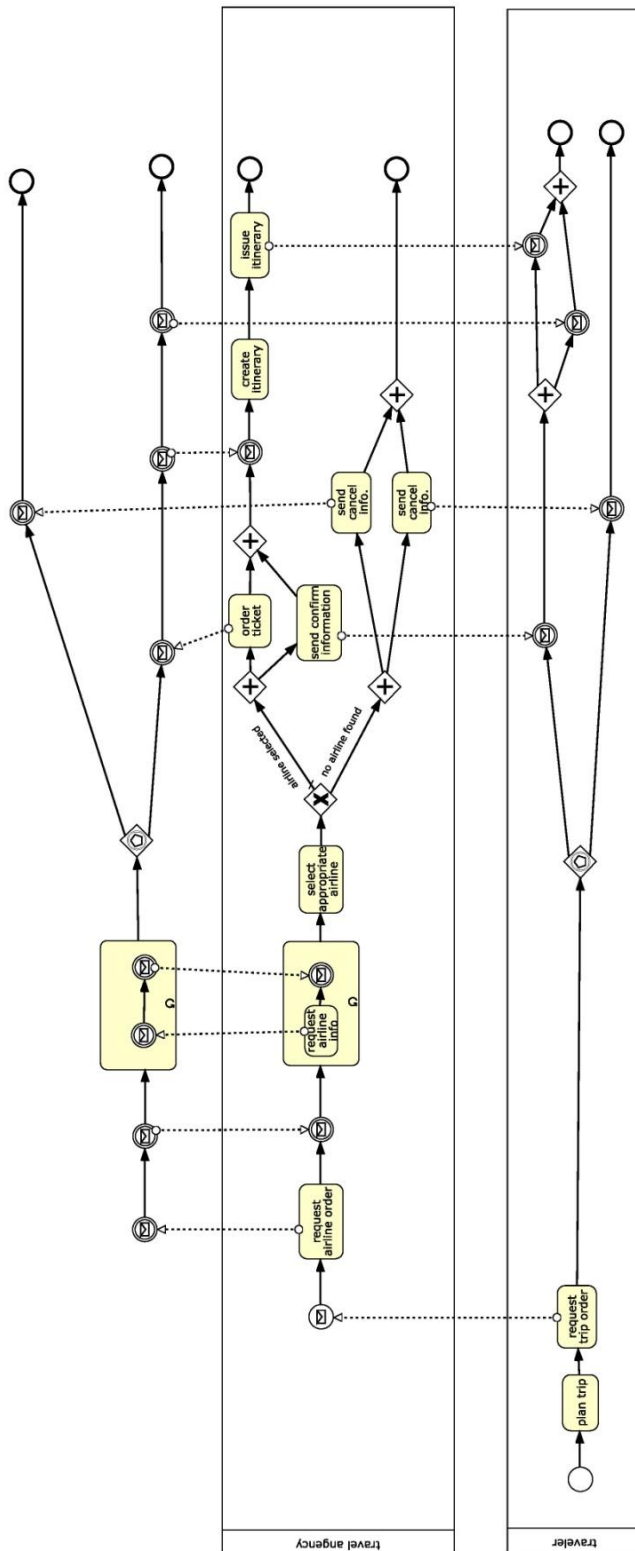


Figure 3.1: Sketch of direct approach (The Airline has been transformed to an interaction model. The travel agency and traveller processes have not.)

### *Indirect approach*

In this approach we use a third-party modelling language to facilitate the mapping from BPMN onto iBPMN. This third-party modelling language is iPetri nets introduced in section 2.4. As BPMN and iBPMN, iPetri nets is also a graph-oriented modelling language. There are several reasons, why are Petri nets chosen as an intermediate model. Like BPMN Petri nets are used as a simple graphical language. As a graphical and mathematical modelling tool, Petri Nets are widely used to describe and study information processing systems that are characterized as being concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic [Mur89]. Hence Petri nets can used to simulate and check the validation of the Process modelling. Especially as a graphical modelling tool, Petri nets can be similar to flow charts, block diagrams, which are useful for the mapping from the BPMN to iBPMN. BPMN provides the graphical notations to depict the control flow for business process modelling and collaboration processes. However iBPMN uses interactions as building blocks to depict choreography from a global view point. iPetri nets can fill in the gap between BPMN and iBPMN.

In the following we explain the basic idea of this approach. To accomplish the mapping in this approach, we firstly map the BPMN choreographies onto iPetri nets models. In this step, we translate the tasks/events involved in a conversation as a whole into labelled transitions in iPetri nets. The silent transitions in iPetri nets are used to represent the task/events not involved in conversations and routing in BPMN. After this mapping we get a large net. Using reduction rules in iPetri nets we get a reduced iPetri nets model, which preserves the same firing sequence and desired properties. And then we translate the reduced iPetri nets model into iBPMN choreography. The labelled transitions in iPetri nets can be directly mapped onto interactions in iBPMN. According to the flow relations in iPetri nets we can derive the flow dependencies in iBPMN.

In this thesis we choose indirect approach to accomplish the mapping. In the following Chapter we will introduce this approach in detail.

## 4 Mapping interconnection models to iPetri nets

As we introduced in Chapter 2, we use Interaction Petri nets (iPetri nets for short) as a formal model and an intermediate model for the mapping from interconnection models to interaction models. Therefore the mapping from interconnection models to interaction models will be accomplished in two steps: i) mapping from interconnection models (BPMN) to iPetri nets; ii) mapping from iPetri nets to interaction models (iBPMN).

In this chapter we will establish a mapping from interconnection models to iPetri nets. As introduced in previous chapter iPetri nets are a subset of the classical Petri Nets. In the mapping from BPMN to iPetri nets the transformations are straightforward. The constructs of sequence flows of BPMN are similar to iPetri nets.

This chapter is structured as follows: in Section 4.1 we provide the syntax of BPMN choreography model; then in Section 4.2 we introduce the mapping of basic constructs in BPMN; next in Section 4.3 we state how to translate the sequence activities; and then we introduce the mapping of the message flows in Section 4.5; the mapping of advanced constructs e.g. looping constructs will be introduced in Section 4.6; in Section 4.6 we provide the reduction rules for the large iPetri nets after the mapping; at last the restrictions of the mapping are given.

### 4.1 Syntax of BPMN Choreography Model

Before the mapping we first introduce a definition “well-formed BPMN process”, which is introduced in [DDO07]. With this definition we can simplify the mapping from BPMN to Petri nets. We introduce firstly the definition of core BPMN process.

**Definition 3. (core BPMN process)** *A core process is a tuple where:  $\mathcal{P} = (\mathcal{T}, S, \mathcal{E}, \mathcal{G}, \mathcal{F}, \text{Cond}, \text{Excp})$*

- *$O$  is a set of objects which can be partitioned into disjoint sets of activities  $\mathcal{T}$ , events  $\mathcal{E}$  and gateways  $\mathcal{G}$ ,*
- *$\mathcal{T}^R \subseteq \mathcal{T}$  is a set receive tasks,*
- *$\mathcal{E}$  can be partitioned into disjoint sets of start event  $\mathcal{E}^S$ , intermediate events  $\mathcal{E}^I$ , and end event  $\mathcal{E}^E$ ,*
- *$\mathcal{E}^I$  can be partitioned into disjoint sets of intermediate message events  $\mathcal{E}^{Im}$ , intermediate timer events  $\mathcal{E}^{It}$ , and intermediate error events  $\mathcal{E}^{Ir}$ ,*

- $\mathcal{G}$  can be partitioned into disjoint sets of parallel fork gateways  $\mathcal{G}^f$ , parallel joint gateways  $\mathcal{G}^j$ , data-based XOR decision gateways  $\mathcal{G}^x$ , event-based XOR decision gateways  $\mathcal{G}^v$ , and XOR merge gateways  $\mathcal{G}^m$ ,
- $\mathcal{F} \subseteq O \times O$  is the control flow relation, i.e. a set of sequence flows connecting objects,
- *Cond*:  $\mathcal{F} \cap (\mathcal{G}^x \times O) \rightarrow C$  is a function which maps sequence flows emanating from data-based XOR gateways to conditions, and
- *Excp*:  $\mathcal{E}^i \rightarrow \mathcal{T}$  is a function assigning an intermediate event to an activity such that the occurrence of the event signals an exception and thus interrupts the performance of the activity.

For the further definition, we provide a couple of auxiliary functions. As in [DDO07] defined, for any node  $x \in O$ , input nodes of  $x$  are given by  $\text{in}(x) = \{y \in O \mid y\mathcal{F}x\}$  and output nodes of  $x$  are given by  $\text{out}(x) = \{y \in O \mid x\mathcal{F}y\}$ .

A core BPMN process is a well-formed if relation  $\mathcal{F}$  satisfies the following requirements [DDO07]:

- $\forall s \in \mathcal{E}^s \cup \text{dom}(\text{Excp}), \text{in}(s) = \emptyset \wedge |\text{out}(s)| = 1$ , i.e. start events and exception events have no incoming flow and have only one outgoing flow,
- $\forall e \in \mathcal{E}^e, \text{out}(e) = \emptyset \wedge |\text{in}(e)| = 1$ , i.e., end events have no outgoing flow but only one incoming flow,
- $\forall x \in \mathcal{T} \cup (\mathcal{E}^i \setminus \text{dom}(\text{Excp})), |\text{in}(x)| = 1 \wedge |\text{out}(x)| = 1$ , i.e. activities, and non-exception intermediate events have only one incoming and outgoing flow.
- $\forall g \in \mathcal{G}^f \cup \mathcal{G}^x \cup \mathcal{G}^v (\mathcal{E}^i \setminus \text{dom}(\text{Excp}))$ :  $|\text{in}(g)| = 1 \wedge |\text{out}(g)| > 1$ , i.e. fork or decision gateways have only one incoming flow but more than one outgoing flows.
- $\forall g \in \mathcal{G}^j \cup \mathcal{G}^m, |\text{out}(g)| = 1 \wedge |\text{in}(g)| > 1$ , i.e. join or merge gateways have an outgoing flow but more than one incoming flows,

- $\forall g \in \mathcal{G}^v, out(g) \subseteq \mathcal{E}^m \cup \mathcal{E}^t \cup \mathcal{T}^k$ , i.e. event-based XOR decision gateways must be followed by intermediate message or timer event or receive tasks.
- $\forall g \in \mathcal{G}^x, \exists$  an order  $<_g$  which is a strict total order over the set of flows  $\{g\} \times out(g)$ , and  $\exists x \in out(g)$  such that  $\neg \exists f \in \{g\} \times out(g) ((g, x) <_g f)$ , i.e.  $(g, x)$  is the default flow among all the outgoing flows from  $g$ ,
- $\forall x \in O, \exists s \in \mathcal{E}^s \cup dom(Excp), \exists e \in \mathcal{E}^e, s \mathcal{F}^* x \wedge s \mathcal{F}^* e$ , i.e. every object is on a path from a start event or an exception event to an end event.

Then we will introduce the definition of a well formed choreography model in BPMN:

**Definition 4. (well formed BPMN choreography model)** A well formed BPMN choreography model is a tuple  $\mathcal{M} = (Q, \mathcal{R}, \mathcal{F}^M)$  where,

- $Q$  is a set of well formed core BPMN processes,
- $\mathcal{R}$  is a function assigning a Role to a Process,
- $\mathcal{F}^M \subseteq \bigcup_{p,q \in Q, p \neq q} \mathcal{T}_p \times (\mathcal{E}_q^{Im} \cup \mathcal{T}_q^R)$  is the set of message flows between processes.

To accomplish the mapping, we need to translate the constructs introduced above to the iPetri nets constructs. In the Section 4.2 the translation of the basic constructs will be introduced.

We introduce the definitions and conditions above for simplifying the mapping from a BPMN choreography model onto iPetri nets model. A core BPMN model consists of a set of core BPMN processes. A core BPMN model is well-formed if these processes are well-formed as defined in definition 4. In the following sections we will provide the mapping from the well-formed core BPMN choreography models onto iPetri nets. In this mapping we use labelled transitions to represent timer events and message exchanges i.e. interactions, and unlabelled transitions to represent the tasks or branches which are not involved in a conversation.

## 4.2 Mapping of Basic Constructs

For the mapping we first split BPMN model into several classes. First class is a set of basic constructs. Sequence activities are in second class. The third class is message flow. The fourth class is looping constructs.

First class of the mapping is the class of basic constructs of BPMN. These constructs are tasks, events and gateways except event-based gateways, which are not directly involved in any conversation. Figure 4.1 shows the mapping of the first class of the BPMN constructs.

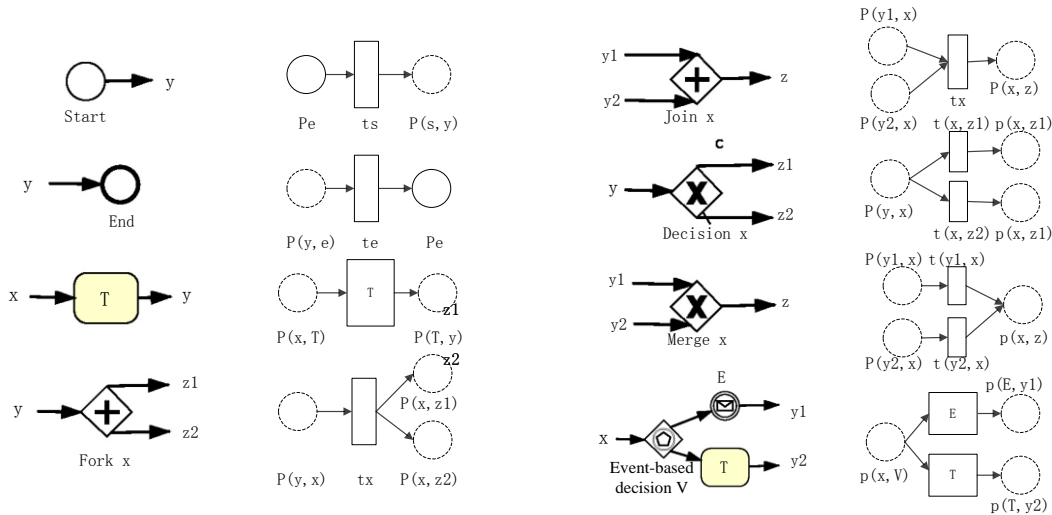


Figure 4.1: Mapping of Basic Constructs [DDO07a]

The mapping of the class of basic constructs is intuitive. A transition can be used to model the task or event. Hence, a start event can be mapped to a place without incoming arcs and one outgoing silent transition. And the input place as the start place will be marked through the initial marking. An end event is like start event mapped onto a transition with which implies the end of the process. A task or an intermediate event is mapped to similar construct like a start or end event, but it shares the places with the predicted/succeeded task/event.

Gateways are mapped onto the corresponding iPetri net constructs as shown in Figure 4.1. But event-based decision gateways or the gateways that followed by the send/receive task or message events are special cases. Because the event-based gateways must be followed by intermediate message event, timer event or receive task as the trigger of the routing. These tasks or events followed by gateways are connected with message flows. Hence the detailed mapping of these gateways with interactions will be discussed with the mapping of message flow together.

However we can map the event-based gateways onto corresponding iPetri nets constructs if not considering of the interactions. An event-based gateway will be mapped to one incoming place followed several transitions which model the tasks or events following the gateway. This construct captures the race conditions of event-based gateway. But the actual mapping of an event-based gateway will be in the Section 4.4 in detail discussed.

The other gateways without interactions can be intuitively represented by iPetri nets. As shown in Figure 4.1, a parallel fork gateway can be represented by a silent transition with one input place and several output places. If the input place has a token, the followed transitions are enabled. That means every output place can have respectively own token. It captures the property of the parallel fork gateway, i.e. the activities following the parallel gateway are synchronized executed. A parallel join gateway is mapped in a similar way. Several incoming places join in one silent transition with one outgoing place. That means that the transition can be fired only if every incoming place must respectively own token.

A data-based XOR decision gateway is modelled by several silent transitions with one input place. These transitions compete for one token from the input place. In this way race condition between the tasks or events is captured. A XOR merge gateway is mapped in a similar way. Several incoming transitions join in one output place. And every transition has own incoming place.

To simplify the mapping algorithm, we use  $\mathcal{M} = (Q, \mathcal{R}, \mathcal{F}^M)$  to represent a BPMN choreography model. As defined we know that:  $Q$  is a set of well-formed BPMN processes, where  $P = (\mathcal{T}, S, \mathcal{E}, \mathcal{G}, \mathcal{F}, Cond, Excp)$ .  $\mathcal{R}$  is a set of Roles.  $\mathcal{F}^M$  is a set of message flows. And in the following algorithm we use  $iPN = (P, T, F, R, s, r, timer, MT, t, M_0)$  to present an iPetri nets model, which contains no interactions.

The following algorithm shows how to map the basic constructs of BPMN model onto corresponding iPetri nets model. Line 4 loops all objects in  $O$  to map them onto corresponding constructs. Line 5 adds places which connect the transitions in iPetri nets. Lines 6 to 8 handle start events, and transform start event to start transition. Line 7 adds a new transition and line 8 connects this transition to the start place and to the places shared with subsequent activities of start event. Meanwhile line 8 adds the flow relations in  $F$ . In a similar way, lines 9 to 11 handle end event. Lines 12 to 14 handle tasks or intermediate events. Lines 14 to 17 handle parallel fork/join gateway. Lines 18 to 21 handle data-based XOR decision. Lines 22 to 25 handle XOR merge gateway. At the end it returns iPetri net iPN.

<pre> 1:  Input: P = (T, E, G, F)  Output: IPN 2:  Var: x,y,z; 3:  Init: P=∅, T= ∅, F= ∅; 4:  foreach  x in O do 5:      P:=P∪{p_s}∪{p_e}∪{p_(x,y)} (x,y) ∈ F}; 6:      if x ∈ E^s, then                                // start event 7:          T := T ∪ {t_s}; 8:          F := F ∪ {(p_s, t_s)} ∪ {(t_s, p_(s,y)} y ∈ out(x)} 9:      if x ∈ E^e, then                                // end event </pre>
--

```

10:    $T := T \cup \{t_e\};$ 
11:    $F := F \cup \{(t_e, p_e)\} \cup \{(p_{(y,e)}, t_e \mid y \in in(x))\}$ 
12:   if  $x \in \mathcal{T} \cup \mathcal{E}^I \wedge x \notin \mathcal{E}^S$ , then // task or intermediate event
13:    $T := T \cup \{x\};$ 
14:    $F := F \cup \{(p_{(z,x)}, x) \mid z \in \mathcal{T} \cup \mathcal{E}^I \wedge z \in in(x) \setminus \mathcal{G}^V\} \cup \{(x, p_{(x,y)}) \mid y \in$   

 $\mathcal{T} \cup \mathcal{E}^I \wedge z \in out(x)\}$ 
15:   if  $x \in \mathcal{G}^F \cup \mathcal{G}^J$ , then //parallel fork/join gateway
16:    $T := T \cup \{t_x\};$ 
17:    $F := F \cup \{(p_{(z,x)}, t_x) \mid z \in in(x)\} \cup \{(t_x, p_{(x,y)}) \mid y \in out(x)\}$ 
18:   if  $x \in \mathcal{G}^X, z \in in(x)$ , then //data-based XOR decision gateway
19:   foreach  $y \in out(x)$ 
20:    $T := T \cup \{t_{(x,y)}\}$ 
21:    $F := F \cup \{(p_{(z,x)}, t_{(x,y)}) \mid z \in in(x)\} \cup \{(t_{(x,y)}, p_{(x,y)})\}$ 
22:   if  $x \in \mathcal{G}^M, z \in out(x)$ , then //XOR merge gateway
23:   foreach  $y \in in(x)$ 
24:    $T := T \cup \{t_{(y,x)}\}$ 
25:    $F := F \cup \{(p_{(y,x)}, t_{(y,x)}) \mid y \in in(x)\} \cup \{(t_{(y,x)}, p_{(x,z)})\};$ 
26: return  $IPN := (P, T, F)$ 

```

**Figure 4.2: Algorithm for the mapping of basic constructs adapted from [DDO07b]**

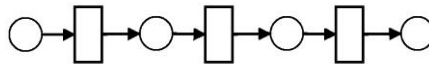
Note that, the mapping of an event-based gateway is not separately listed. However it is contained here as construct of normal task or events. In the next sections we will introduce the mapping of complex constructs.

### 4.3 Sequence of Activities

Sequential flow is the fundamental construct of a process. Here, a set of activities are connected by sequence flows without gateways and splitting and executed sequentially. According to the mapping of the basic BPMN constructs, we can intuitively map the sequence activities to a sequence of place and transitions. In the mapping transitions represent the tasks or events. Figure 4.3 shows a sequence of activities in BPMN and corresponding iPetri nets model is shown in Figure 4.4. These sequence activities have one entry node and one entry exit node, which can be a task or an event. The entry node and the exit node are not same. Such a sequence of activities will be directly mapped onto a sequence of transitions in iPetri nets.



**Figure 4.3: Sequence of Activities in BPMN**



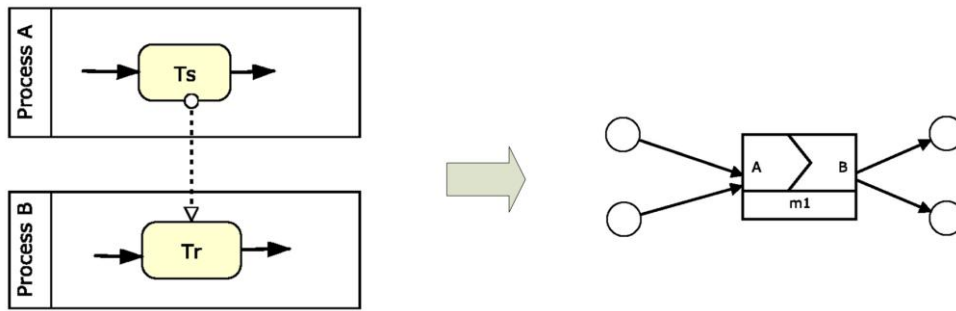
**Figure 4.4: Mapping to iPetri nets**

#### 4.4 Message Flow

In this section we discuss the mapping of the constructs, which are involved in a conversation. A message flow is used to describe an interaction between two participants in choreography.

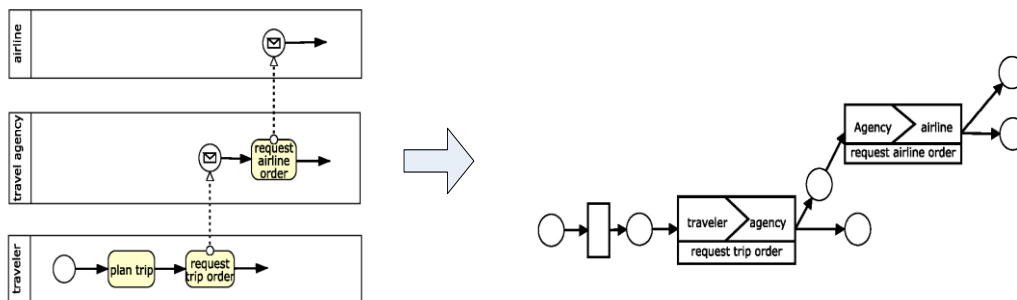
In an abstract way, the message exchanges can be mapped to a labelled transition, which represents the interaction in the iPetri nets. As defined in chapter 2, the label of the transition contains the information about sender, receiver and message type. In an interconnected model a send task is connected with a receive task or an intermediate message event through a message flow. In iPetri nets the interactions are represented by the labelled transitions. Hence a send task and the corresponding connected receive task or intermediate event will be in the mapping integrated to one transition. As defined in chapter2, the role of sender, receiver and message type will be attached to this transition. The sending and receiving nodes will be transformed to corresponding transitions and places, which all connect the labelled transition.

Figure 4.5 shows the mapping rules. In this Figure, Ts is a send task, Tr is a receive task. After the integration of send and receive task into one transition, the former nodes will directly connect to the new labelled transition.



**Figure 4.5: Mapping of Message Flow**

The mapping of a message flow to a start event is a special case of the mapping. That means that the message is the trigger of start of another process. The start message event will be not like the normal start event to be mapped to a place with an initial token and a silent transition, but with the send task as a whole to be mapped onto a labelled transition. The input place of the start event will be thrown away. Figure 4.6 shows a part of travel booking, where the process Travel Agency and Airline will be triggered by incoming messages. And the corresponding iPetri nets model is shown in Figure 4.6



**Figure 4.6 : incoming message as trigger of process**

The gateways involved in a conversation are in a similar way mapped onto the iPetri nets. For example an event-based gateway is always followed by receive tasks, intermediate message events or timer. An event-based gateway is mapped to several labelled transitions outgoing from one input place. The outgoing labelled transitions represent, like before discussed, interactions with other process. The outgoing flows from one place are used to capture the race condition. As introduced interactions are integrated with send and receive task or send and intermediate message event. As shown in Figure 4.7 the corresponding input place of interconnected send task connects directly to the labelled transition. After the transformation of gateways with message flows the model will be complicate. And it will generate many crossed arcs. The refinement of the complicated model will in Section4.6 discussed.

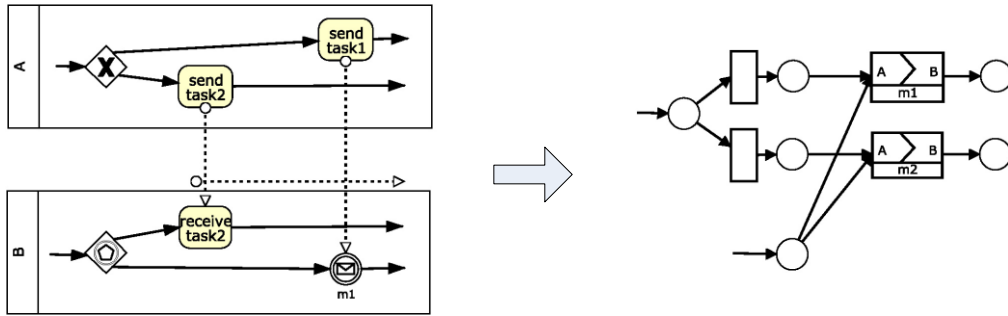


Figure 4.7: Mapping of message flow

Next the algorithm handling the message exchanges will be introduced. This piece of algorithm handles the mapping of message flow. Line 1 loops all activities in a process which are involved in conversations. Lines 2 to 6 integrate the sending/receiving activities into one labelled transition attached information of sender, receiver and message type. Lines 7 to 9 handle the start event which is triggered by external message. Line 11 adds the new flow relations in  $F$ .

```

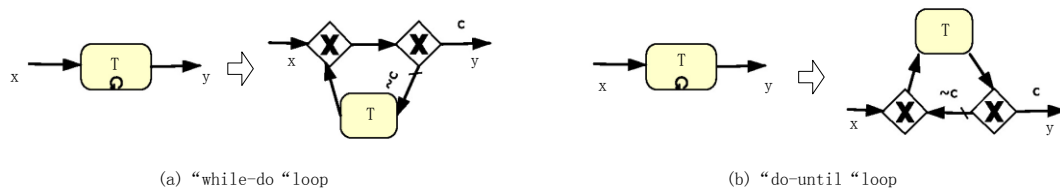
1:  foreach  $x$  in  $O_p, \mathcal{P} \in \mathcal{Q} \wedge \exists y \wedge (x, y) \in \mathcal{F}^M$  do
2:     $P := P \cup \{p_s\} \cup \{p_e\} \cup \{p_{(x,y)} \mid (x, y) \in \mathcal{F}\}$ ;
3:     $T := T \cup \{t_{xy}\}$ ;
4:     $t_{xy}.sender := Role(x)$ ;
5:     $t_{xy}.receiver := Role(y)$ ;
6:     $t_{xy}.mtype := MT(t_{xy})$ ;
7:  if  $x \in \mathcal{E}^s$ , then
8:     $P := P \setminus \{p_s\}$ ;
9:     $F := F \cup \{(t_{xy}, p_{(xy,z)} \mid z_A \in out(x))\} \cup \{(t_{xy}, p_{(xy,z)} \mid z_B \in out(y))\}$ ;
10: elseif  $x \in \mathcal{T}^R \cup \mathcal{E}^I \wedge x \notin \mathcal{E}^s$ , then
11:    $F := F \cup \{(t_{xy}, p_{(xy,z)} \mid z \in out(x))\} \cup \{(t_{xy}, p_{(xy,z)} \mid z \in out(y))\}$ 
       $\cup \{(p_{(z,xy)} \mid z \in in(x))\} \cup \{(p_{(z,xy)}, p_{(z,xy)} \mid z \in in(y))\}$ ;

```

Figure 4.8: A piece of algorithm for message flows

## 4.5 Looping Constructs

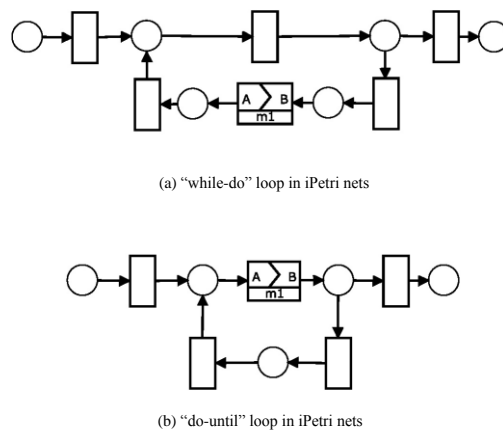
For simplifying the mapping, the looping constructs can also first transform to the basic constructs. Looping constructs have two forms --- a “while-do” or a “do-until” loop. Figure 4.9 shows the corresponding mapping of the two constructs.



**Figure 4.9: Looping constructs**

For simplifying the mapping, we consider the case that the opposite constructs (receive task or intermediate message event) of the looping send tasks are symmetric. That means, that here we do not consider constructs of multiple instances or multiple participants. We assume that the receive constructs have the same structure as the send constructs. In other words, receive constructs have the same loop type, loop condition and loop frequency.

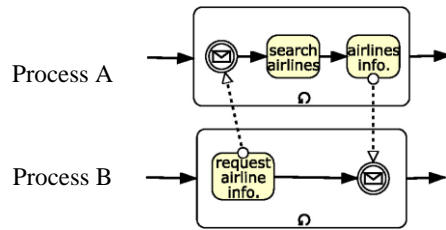
In case looping constructs are involved in the conversation, the internal constructs will be mapped onto iPetri nets like the non-looping constructs. That means that message exchanges will be mapped to labelled transitions firstly. Then the looping constructs will be transformed corresponding constructs in iPetri nets. Especially a “while-do” or a “do-until” loop are represented by XOR fork gateways and merge join gateways. Then we can use corresponding iPetri nets constructs model the loop. Figure 4.10 shows the looping constructs in iPetri nets.



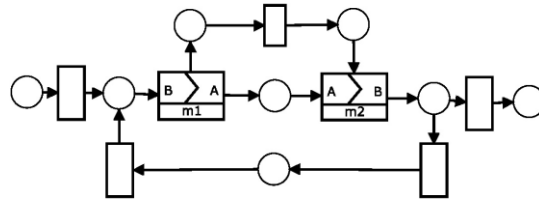
**Figure 4.10: Loop constructs in iPetri nets**

The following figure shows a part of travel booking scenario. It shows that travel agency request the airline information and airlines feedback the airline

information. The internal tasks and events will be first mapped onto the corresponding iPetri nets modules, which represent the message exchanges between the two processes. And then the mapped iPetri module will be regarded as a whole, which will be as above introduced transformed to corresponding iPetri loop model.



(a) loop in travel booking scenario



(b) loop in travel booking scenario

**Figure 4.11: Mapping of loop constructs**

#### 4.6 Reduction of the models

After the mapping we get a large iPetri nets model which has redundant places and transitions. In this section we will propose some reduction rules to reduce the large net and simplify the mapping from iPetri nets onto iBPMN. There are a lot of related work in which the reduction rules for Petri nets are introduced. The reduction rules in this thesis are based on the basic rules for Petri nets [Mur89].

Firstly, an iPetri nets model after the mapping will be large and complicated as shown in the following Figure 4.12. We can see a large complicated net with many crossed arcs. And for the final result of the mapping --- iBPMN choreography model, we care about the interactions of the model. However in the accomplished iPetri nets model there are many silent transitions, which are not concerned about interactions or routing. We will reduce the silent transitions, which do not represent patterns such as the branching, merging or loop constructs. With the reduction rules we will fusion the parallel places and transitions, or reduce the redundant places and transitions. With the result we get a reduced model. But we do not change the semantic of the original model.

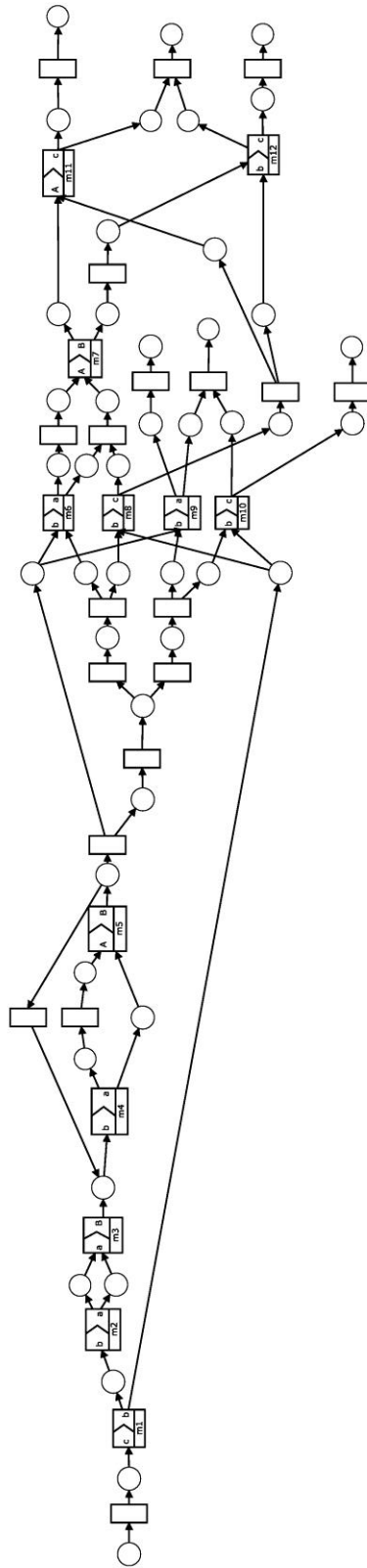


Figure 4.12: Travel booking in iPetri nets

### 4.6.1 Background knowledge for reduction

Before we discuss the reduction rules for the iPetri nets, some desirable properties that the iPetri nets must preserve after the reduction will be introduced. In a body of related work the nature and character of Petri nets or workflow Petri nets are introduced. In Chapter 2 we have defined the classical Petri nets and iPetri nets; we summarize some basic properties for them, which apply in classical Petri nets and WF-nets in the related works [Aa98] and [AaLa05a]. From the definition we know iPetri nets are extension of classical Petri nets. The reduction rules in [Mur89] and [WVA+06] can apply in iPetri nets to preserve the properties we summarized.

We firstly introduce some definition adapted from [SMD96]:

**Definition 5. (Transition Firing Sequence)** For a iPetri net  $iPN$ , a marking  $M$  is said to be reachable from  $M'$ , if there exists a sequence of markings  $M_j, M_{j+1} \dots M_{j+k}$ , where  $M_j = M, M_{j+k} = M'$ , and  $M_{j+i-1}$  leads to  $M_{j+i}$  through the firing of some enabled transition, where  $i=1, 2, \dots k$ . Such a transition firing sequence from  $M$  to  $M'$  is named *transition firing sequence*.

**Definition 6. (Reachable Marking)** For an iPetri net  $iPN$ , the set of reachable Marking is denoted by  $R(iPN, M_0)$ , and  $R(iPN, M_0) = \{M \mid M \text{ is a marking reachable from } M_0\}$ .

**Definition 7. (Safe)** An iPetri net,  $iPN$ , is said *safe* if the number of tokens in every place does not exceed one for any reachable marking. i.e.  $\forall M \in R(iPN, M_0) \text{ and } \forall p \in P, M(p) \leq 1$ .

Before we propose the reduction rules, we will first introduce a theorem about redundancy of place. As stated in [SMD96], if a place in a Petri net can be removed without changing the dependencies of transition firing, it can be regarded as a *Redundant Place*. When a place can be considered as a redundant place, it and its arcs can be removed. With the identifying of redundant places a large iPetri nets model can be further reduced. In [Ber85] Berthelot introduced a generalized formal definition of redundant place for classical Petri net. In this thesis we focus on the iPetri nets, whose arc weights are always equal to one. We say that such iPetri nets are *ordinary* [SMD96]. Hence, iPetri nets can apply the theorem presented in [SMD96]. The reduction rule 5 in Section 4.6.2 is based on this theorem. In the following we present the theorem for iPetri nets.

**Theorem 1 (Place Redundancy) (adapted from [SMD96])**

*Let  $iPN$  be a safe and ordinary iPetri net, place  $p \in P$  is redundant if the following conditions are satisfied:*

- $M_0(p)=0$  and  $|p^\bullet|=r$  ( $r \geq 1$ ), i.e. in initial marking  $p$  contains no token; and place  $p$  has  $r$  outgoing transitions.
- for every transition  $t_{ok} \in p^\bullet$ ,  $k=1, 2, \dots, r$ , there is a place  $q_{ok} \neq p$ , where  $t_{ok}$  is subsequent transition, i.e.  $q_{ok}^\bullet = \{t_{ok}\}$  and if  $\forall M \in R(iPN, M_0)$ , if  $M(q_{ok})=1$  for any  $k \in \{1, 2, \dots, r\}$ , then  $M(p)=1$ , i.e., if any  $q_{ok}$  place is marked in any state, then  $p$  is also marked in that same state.

There exist various reduction techniques for general Petri nets analysis e.g. the reduction rules presented in [Mur89] and [Ber85]. In this thesis we apply these reduction rules to redundant places and silent transitions. In other words, labelled transitions will be remained after reduction. We will prove that the firing of labelled transition will not be affected after reducing redundant places and silent transitions.

#### 4.6.2 Reduction rules

Next we propose some reduction rules for iPetri nets which can be used to reduce some certain silent transitions and places in a large net. Thereby we could cut down the size of model. As a result, the mapping from iPetri nets onto iBPMN can be performed efficiently. These rules are stated and proofed in [Mur89] and [WVA+06] for Petri nets and WF-nets. We will apply these rules to reduce the silent transitions and corresponding places. Hence, the structure and dependencies of the interaction occurrences (labelled transitions) will be not changed. Note that, if we explain specially, all the reduction rules for transitions are applied to silent transitions.

Let  $iPN$  be an iPetri net and  $x \in (P \cup T)$ , we use  $\bullet x$  and  $x \bullet$  to denote the set of inputs and outputs.  $M_0$  denotes an initial marking of  $iPN$ .

##### **Rule 1 Fusion of Series Places [adapted from Mur89]**

*Let  $iPN$  be an iPetri net. If there exists a transition  $t \in T_u$ , and two place  $p_1$  and  $p_2 \in P$ , and  $\bullet t = \{p_1\} \wedge t \bullet = \{p_2\} \wedge |\bullet t| = |t \bullet| = 1$ , and  $p_1$  or  $p_2$  has only one entry and one exit node, then  $p_1$  and  $p_2$  can be fused into one place  $\langle p_1 p_2 \rangle$ , such that  $\bullet \langle p_1 p_2 \rangle = \bullet p_1$ , and  $\langle p_1 p_2 \rangle \bullet = p_2 \bullet$*

The Rule1 allows for the fusion of two places in a sequence. The condition is that there is only one transition between the two places. And one of places has only one entry node and one exit node. That means that the transition has single input place  $p_1$  and single output place  $p_2$ . And there are no other arcs from  $p_1$  to  $p_2$ . The transition between the places and the connected arcs can be removed. The figure illustrates the rule1. The Figure 4.13 a) shows the original iPetri net, where the grey part can be applicable to the rule1. The Figure 4.13 b) shows the new iPetri net after the reducing.

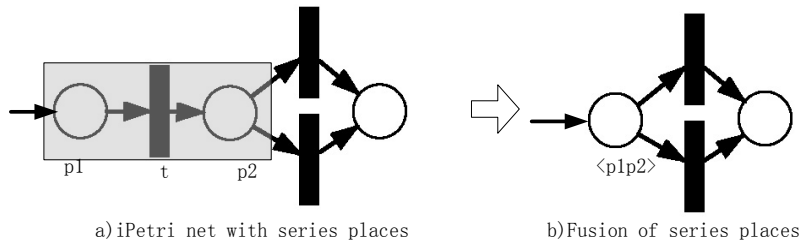


Figure 4.13: Fusion of series places

**Rule 2 Fusion of Series Transitions [adapted from Mur89]**

Let IPN be an iPetri net. If there exists a single-input/output place  $p \in P$ , and  $t_1, t_2 \in (T_c \cup T_u)$ ,  $\bullet p = \{t_1\} \wedge p \bullet = \{t_2\} \wedge |\bullet p| = |p \bullet| = 1$ , and one of  $t_1$  and  $t_2$  has only one entry and one exit node, then  $t_1$  and  $t_2$  can be fused into  $\langle t_1 t_2 \rangle$  such that  $\bullet \langle t_1 t_2 \rangle = \bullet t_1$ , and  $\langle t_1 t_2 \rangle \bullet = t_2 \bullet$ .

The Rule2 allows for the fusion of two transitions in a sequence. In the Rule2, there is a place between two transitions. And this place has only one input transition  $t_1$  and only one output transition  $t_2$ . There is only one path through the middle place from input transition to the output transition. One of the two transitions has only one entry node and one exit node. The two transitions will be fused in one transition. The place between them and the connected arcs can be removed. The Figure 4.14 illustrates the Merging of Transitions.

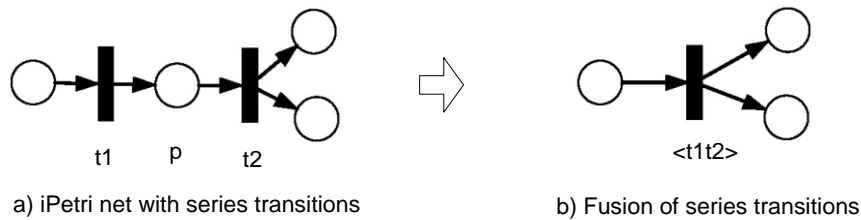
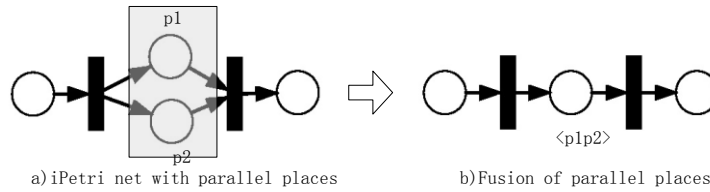


Figure 4.14: Fusion of series transitions

**Rule 3 Fusion of parallel Places [Mur89]**

Let iPN be an iPetri net. If for every pair  $p_1, p_2 \in P$ , there exist  $\bullet p_1 = \bullet p_2 \wedge p_1 \bullet = p_2 \bullet$ , then  $p_1, p_2$  can be fused into one new place  $\langle p_1 p_2 \rangle$ , and  $\bullet \langle p_1 p_2 \rangle = \bullet p_1$ , and  $\langle p_1 p_2 \rangle \bullet = p_1 \bullet$ . The parallel places  $p_1, p_2$  will be removed and corresponding arcs will be reduced.

*Rule 3* allows for the fusion of two parallel places, which have identical input transition set and identical output transition set. Actually multiple places which satisfied the parallel condition (identical input transition set and identical output transition set) can be fused into one place. Then the redundant arcs can be reduced. The Figure 4.15 shows the reduction rules 3.

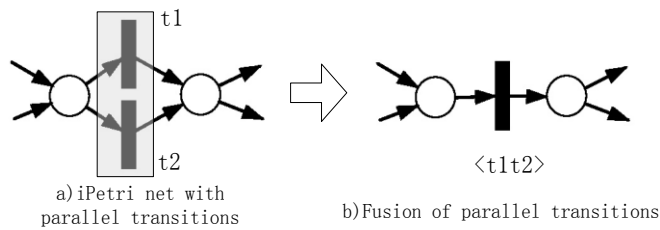


**Figure 4.15: Fusion of parallel places**

***Rule 4 Fusion of parallel Transitions [Mur89]***

*Let  $iPN$  be an iPetri net. If for every pair  $t_1, t_2 \in (T_c \cup T_u)$ , there exist  $\bullet t_1 = \bullet t_2 \wedge t_1 \bullet = t_2 \bullet$ , then  $t_1, t_2$  can fused into one transition  $\langle t_1 t_2 \rangle$ , and  $\bullet \langle t_1 t_2 \rangle = \bullet t_1$ , and  $\langle t_1 t_2 \rangle \bullet = t_1 \bullet$ . The reduced net has a new transition, and  $t_1, t_2$  will be reduced and corresponding arcs will be removed.*

*Rule 4* allows for the merging of two parallel transitions, which have identical input places and identical output places. After the reduction all the parallel transitions will be fuse into one new transition, which has identical input and output place set with the original parallel transition. The Figure 4.16 illustrates the reduction of parallel transition.

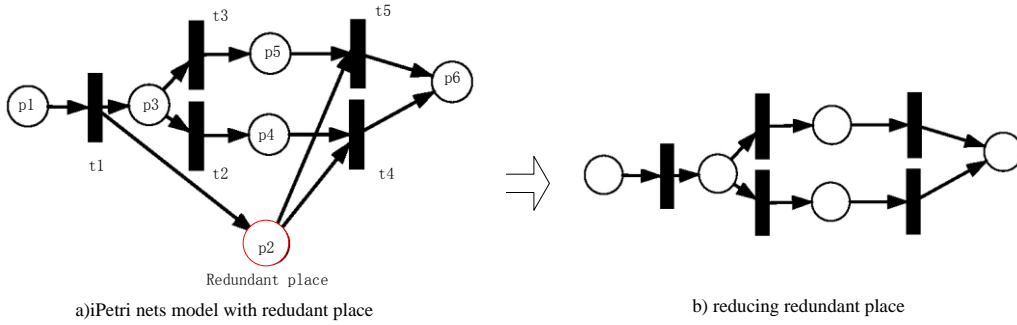


**Figure 4.16: Fusion of parallel transitions**

***Rule5 Reduce the redundant places [SMD96]***

*Let  $iPN$  be an iPetri net. If a place  $p \in P$  is a redundant place which satisfies the conditions in Theorem 1, then this place can be removed from iPetri net, and corresponding arcs connected to this place will be removed too.*

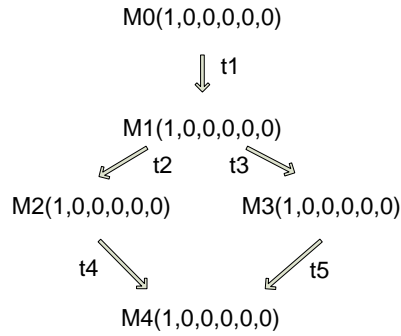
*Rule5* allows removing the redundant places from an iPetri net. Figure 4.17 a) shows a part of iPetri net without interaction occurrences, which has a redundant place  $p_2$ . The right part of the Figure 4.17 b) shows the iPetri net reducing the redundant place.



**Figure 4.17: Reduce redundant place**

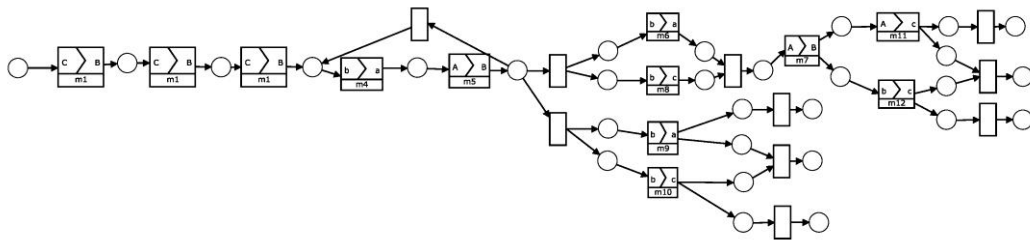
Figure 4.17 illustrates an example of redundant place and the reduced iPetri net. And in the following we will verify the redundancy of the place using the reachable marking.

To verify the redundancy of the place  $p_2$ , we set  $M_0 = (1, 0, 0, 0, 0, 0)$ . And we denote  $M_i$  as reachable marking from  $M_0$ ,  $R(iPN, M_0) = \{M_i | \text{all reachable Marking from } M_0.\}$ . We specify all  $R(iPN, M_0)$ . They are:  $M_0 = (1, 0, 0, 0, 0, 0)$ ,  $M_1 = (0, 1, 1, 0, 0, 0)$ ,  $M_2 = (0, \underline{1}, 0, \underline{1}, 0, 0)$ ,  $M_3 = (0, \underline{1}, 0, 0, \underline{1}, 0)$ ,  $M_4 = (0, 0, 0, 0, 0, 1)$ . The reachability graph is shown in Figure 4.18. It is clear that  $p_2$  is redundant place, which fulfils the conditions of redundant place: 1)  $M_0(p_2) = 0$  and  $|p_2^\bullet| = 2$  ( $2 \geq 1$ ); 2) for every output transition of  $p_2$   $\{t_4, t_5\}$ , there is a place  $q \in \{p_4, p_5\}$ , where  $q \neq p_2$ , and  $q^\bullet = \{t_4, t_5\}$ . And  $sM \in R(iPN, M_0)$ , if  $M(q) = 1 \wedge q \in \{p_4, p_5\}$ , then  $M(p_2) = 1$ . In our example, we can see  $M_2(p_4) = 1$ , and  $M_2(p_2) = 1$ ; if  $M_3(p_5) = 1$ , then  $M_3(p_2) = 1$ . Hence, we can conclude, that  $p_2$  is redundant place. Figure 4.19 presents the reduced net.



**Figure 4.18: Reachability graph**

Using the reduction rules, we can simplify the Figure 4.12. Thereby we cut down the size of the large net, and get a reduced model in iPetri nets. The following figure illustrates the reduced iPetri nets model from Figure 4.12. With the reduced iPetri net, we can further map the model onto iBPMN model.



**Figure 4.19: Reduced iPetri net using reduction rules**

In the following we present an algorithm for the iPetri nets reduction. The algorithm implements the transformation in Figure 4.20. Lines 2 to 25 loop through all places. Line 3 checks, whether the current place has only one entry and exit node. If yes, then line5 checks, whether outgoing transition of current place is an uncontrolled silent transition and a single entry single exit node. If yes, then we delete this transition and current place. Meanwhile we delete corresponding arcs. Line 9 checks, whether current place is a start place. If not, we must add a new arc from the incoming transition of current place to the outgoing place of deleted transition. Line 14 to 20 handles parallel places. Line16 loop through all places which have the same input and output transitions with current place. Line 17 checks, whether these transitions are uncontrolled silent and have single entry and exit nodes. If yes, we can delete all these transitions and corresponding arcs. In a similar way, line 20 to 26 deal with end places. Line27 to 41 loop through all transitions. Line 28 checks whether current transition is single entry single exit node. If yes, line 31 checks whether another transition  $t_2$  exists, which is in a sequence with current transition. Between  $t_1$  and  $t_2$  there is only one place. Then line 34 to 36 delete current transition from  $T$  and corresponding arcs.

Meanwhile a new arc from input place of t1 to transition t2 must be added in flow relations. Line37 to 41 check whether exist a parallel transition to the current transition. If yes, the parallel transition will be deleted.

```

1: input:  $iPN$  output:  $iPN$ 
2: foreach  $p1 \in P$  // Fusion of Places
3:   if  $|in(p1)| \leq 1 \wedge |out(p1)| = 1$  then
4:      $t2 := out(p1)$ ;
5:     if  $t2 \in T_u \wedge |in(t2)| = 1 \wedge |out(t2)| = 1$  then // series places
6:        $p2 := out(t2)$ ;
7:        $P := P \setminus \{p1\}$ 
8:        $T := T \setminus \{t2\}$ ;
9:        $F := F \setminus \{(p1, t2), (t2, p2)\}$ ;
10:      if  $|in(p1)| = 1$ , then // not start place
11:         $t1 := in(p1)$ ;
12:         $F := F \setminus \{(t1, p1)\}$ ;
13:         $F := F \cup \{(t1, p2)\}$ ;
14:      if  $t2 \in T_u \wedge |in(t2)| \neq 1 \wedge in(in(t2)) == in(p1) \wedge |in(p1)| = 1$  then
15:         $t1 := in(p1)$ ; //parallel places
16:        foreach  $p2 \in \{in(t2)\}, p2 \neq p1$ 
17:          if  $|in(p2)| = 1 \wedge |out(p2)| = 1$  then
18:             $P := P \setminus \{p2\}$ ;
19:             $F := F \setminus \{(t1, p2), (p2, t2)\}$ ;s
20:          if  $|in(p1)| = 1 \wedge out(p1) = 0$  then //end place
21:             $t1 := in(p1)$ ;
22:            if  $t1 \in T_u \wedge |in(t1)| = 1 \wedge |out(t1)| = 1$  then
23:               $p2 := in(t1)$ ;
24:              if  $|out(p2)| = 1$  then
25:                 $T := T \setminus \{t1\}$ ;
26:                 $F := F \setminus \{(p2, t1), (t1, p1)\}$ ;
27:          foreach  $t1 \in T_u$ 
28:            if  $|in(t1)| = 1 \wedge |out(t1)| = 1$  then
29:               $p1 := in(t1)$ ;
30:               $p2 := out(t1)$ ;
31:              if  $|out(p2)| = 1$  then // series transitions
32:                 $t2 := out(p2)$ ;
33:                if  $t2 \in T_u \wedge |in(t2)| = 1$  then
34:                   $T := T \setminus \{t1\}; P := P \setminus \{p2\}$ ;
35:                   $F := F \setminus \{(p1, t1), (t1, p2), (p2, t2)\}$ ;
36:                   $F := F \cup \{(p1, t2)\}$ ;
37:                if  $|out(p1)| > 1 \wedge |in(p2)| > 1 \wedge out(p1) == in(p2)$  then
38:                  foreach  $t2 \in \{out(p1)\}$ 
39:                    if  $t2 \in T_u \wedge t2 \neq t1$  then
40:                       $T := T \setminus \{t2\}$ ;
41:                       $F := F \setminus \{(p1, t2), (t2, p2)\}$ ;

```

```
42: return (iPN);
```

**Figure 4.20: Algorithm for iPetri nets reduction rule1 to rule4**

Next, we present algorithm for reduction rule 5.

```
1: DeleteRedudant(iPN)  
2: caculate Reachability Graph  $G = (V, E, M_0)$ ;  
3: for each  $\exists p \in P$   
4:   if redundant(p) then  
5:     remove (p);  
6: return iPN;
```

**Figure 4.21: Algorithm for removing redundant places**

In [May81] there is the algorithm for calculating the reachability graph for a Petri net in detail.

```
1: Redundant( $p \in P, M_0$ )  
2: if ( $|p \cdot| = 0$ ), return false;  
3: foreach  $t \in \{p \cdot\}$   
4:   if  $|\cdot t| = 1$   
5:     return false;  
6:   else  
7:     if  $q \in \{\cdot t\}$  then  
8:       if ( $q == p$ ) then return false  
9:       else  
10:        foreach  $M \in V$   
11:          if  $M(q) < M(p)$   
12:            return false;  
13:   return true;
```

**Figure 4.22: Algorithm of identifying redundant place**

In the algorithm for reduction 5, we remove the redundant place *p*, if *p* satisfied conditions in theorem1. The *redundant* (*p*,*M*<sub>0</sub>) checks whether *p* is redundant. If it returns true, the place *p* is redundant, then *p* will be removed. Otherwise it is not redundant place. Line 2 checks whether *p* has outgoing place. If yes, it returns false. If not, line 3 loops through all outgoing transitions of place *p*. Line 4 checks whether these transitions have other incoming place. If not, it returns false. If yes, lines 7 to 12 check whether a *q* exist, if *p* is marked, *q* is also marked. If yes, then it returns true. Otherwise it returns false.

## 4.7 Restriction of the mapping

Note that, there exist some construct in BPMN, which cannot correctly be transformed to iPetri nets construct, although iPetri nets are expressive model language. In this section we will focus on some patterns which cannot exactly expressed in iPetri nets. As discussed in [AH02] and [AH04] there are some problems dealing with workflow patterns using high-level Petri nets. We will in the following summarize these limitations of iPetri nets in the mapping.

Consider for example in the context of an auction scenario. The seller starts the auction by sending an auction creation request to the auction service. For the bidding the bidders will participant in the auction after the auctioning service starts the auction. At design time designer cannot determine how many potential bidders will take part in the auction. This is also known as a variant of the one-from-many-receive service interaction pattern (Ref. to SIP-paper). It is impossible to model a set of an unknown size of multiple participants using iPetri nets. The reason is that participants have to be explicitly enumerated in Petri nets[LKLR07]. When designer at design time knows the definite number of the participants, we can copy the same process  $n$  times. Otherwise, we cannot properly map the multiple participants in BPMN onto iPetri nets model.

Consider the travel booking scenario introduced in this thesis. When the travel agency requires the airline information of airline service, in fact, it will be instantiated many times for representing the different airlines providing information. However, using iPetri nets the part of multiple instances cannot be mapped properly. The low-level Petri nets (classical Petri nets, iPetri nets) cannot handle the synchronized different threads. The lack of supporting the multiple instances has been in [AH02] and [AH04] discussed. Van der Aalst et al. also proposal the method handling the multi instances with high-level Petri nets (colored Petri nets [Jen92]). And in [AH02] and [AH04] a Petri nets based *Yet Another Workflow Language* (YAWL) as a solution of multiple instances is introduced.

Note that there might be multiple start points and multiple end points in iPetri nets. Because the inclusive merge cannot be correctly mapped onto Petri nets construct [DDO07b]. It cannot be mapped onto iPetri nets constructs. In the thesis of kirstin [PK07] it is also discussed. And kirstin also introduced another method using colored Petri nets to translate the inclusive merge. Hence, we cannot merge the multiple end points. That means we have multiple end places in the iPetri nets model after the mapping.

## 5 Mapping iPetri nets to iBPMN

In this chapter we will discuss how to map iPetri nets model onto iBPMN model. As introduced in Chapter 3, we transform the BPMN choreography model to iPetri nets model firstly. Then we get a large size iPetri nets model. After the reduction we get a simplified iPetri nets model, which is the source model in the mapping introduced in this Chapter. Ultimately we obtain the target model in iBPMN in this Chapter.

This Chapter is structured as follows: Section 4.1 introduces the syntax of iBPMN choreography; then in Section 4.2 we introduce the mapping from iPetri nets onto iBPMN. In Section 4.2.1 we firstly introduce the fundamental difference between the BPMN and iBPMN; then we discuss the mapping of basic constructs in Section 4.2.2; however the data-based decision gateway and event-based decision gateway will be distinguished by their structure; this will be in Section 4.2.3 discussed; in the following section we discuss the mapping of looping constructs; and in Section 4.2.5 we will present the algorithm of the mapping.

### 5.1 Syntax of iBPMN Choreography

In Chapter 2 we introduced the overview of iBPMN language and the basic constructs in iBPMN (see Figure 2.7). We knew that the inclusion internal behaviours are not supported in iBPMN [KL09]. Now, we introduce the syntax of core iBPMN choreography in according to the specification in [Dec09].

**Definition 8. ( Core iBPMN choreography )** A core iBPMN choreography is a tuple  $C=(R, I, E, G, F, F^m, Cond)$  where:

- $R$  is a set of different Roles,
- $I$  is divided into disjoint sets of start interactions  $I^s$  and intermediate interactions  $I^i$  and complex interactions  $I^c$ ,
- $E$  is divided into disjoint sets of start events  $E^s$ , end events  $E^e$ , intermediate timer events  $E^t$ ,
- $E^s$  is divided into disjoint sets of non-trigger start events  $E_n^s$  and start timer events  $E_t^s$ ,
- $G$  is divided into disjoint sets of parallel fork gateways  $G^f$ , parallel join gateways  $G^j$ , data-based exclusive decision gateways  $G^d$ , event-based exclusive gateways  $G^v$ , exclusive merge gateways  $G^m$ ,
- $F \subseteq ((E \cup I \cup G) \times (E \cup I \cup G))$  is the control flow relation, i.e., a set of sequence flows connecting objects,

- $F^m \subseteq R \times I \times R$  is the message flow relation, i.e., a set of message flows connection objects,
- $Cond : F \cap (G^d \times (E^t \cup E^e \cup I^i \cup G)) \rightarrow \mathcal{B}$  is a function mapping sequence flows emanating from data-based exclusive decision gateways to the set of all possible conditions( $\mathcal{B}$ ).

In the definition, we define two types of flow relation. One is the dependency of the interactions; another is the message flow which defines the relation of different roles. In a similar way like BPMN, we define the facilitate functions for the definition of iBPMN. For simplify the notation we define a set of objects  $O = E \cup I \cup G$ . For any  $x \in O$ , input nodes of  $x$  are represented by  $in(x) = \{y \in O \mid yFx\}$  and output nodes of  $x$  are given by  $out(x) = \{y \in O \mid xFy\}$ . The definition 8 allows a choreography having no start or end events etc. In the following we give some restriction to the Definition 8.

**Definition 9. (well-formed iBPMN choreography)** *An iBPMN Choreography  $C$  in Definition10 is well-formed, if the following requirements are satisfied:*

- $\forall x \in E^s, in(x) = \emptyset \wedge |out(x)| = 1$ , i.e. start events has no incoming flow and has only one outgoing flow,
- $\forall x \in E^e, out(x) = \emptyset \wedge |in(x)| = 1$ , i.e., end events have no outgoing flow but only one incoming flow,
- $\forall x \in I \cup E^t, |in(x)| = 1 \wedge |out(x)| = 1$ , i.e. interactions and intermediate timer events have only one incoming and outgoing flow.
- $\forall g \in G^f \cup G^d \cup G^v: |in(g)| = 1 \wedge |out(g)| > 1$ , i.e. fork or decision gateways have only one incoming flow but more than one outgoing flows.
- $\forall g \in G^j \cup G^m, |out(g)| = 1 \wedge |in(g)| > 1$ , i.e. join or merge gateways have an outgoing flow but more than one incoming flows,
- $\forall g \in G^v, out(g) \subseteq I^i \cup E^t$ , i.e. event-based XOR decision gateways must be followed by interactions or intermediate timer events.
- $\forall g \in G^x, \exists$  an order  $<_g$  which is a strict total order over the set of flows  $\{g\} \times out(g)$ , and  $\exists x \in out(g)$  such that  $\neg \exists f \in \{g\} \times out(g) ((g, x) <_g f)$ , i.e.  $(g, x)$  is the default flow among all the outgoing flows from  $g$ ,

- $\forall x \in O, \exists s \in E^s, \exists e \in E^e, sF^*x \wedge sF^*e$ , i.e. every object is on a path from a start event to an end event.
- $I^c = (R_c, \{e^s\}, \{e^e\}, I^i, F_c, F_c^m)$ , i.e., the sub-choreography in a complex interaction has only one start event and one end event. That means, there exist no start interactions but only intermediate interactions in sub-choreography.

## 5.2 Mapping from iPetri nets onto iBPMN

In this section we introduce the mapping from iPetri nets models onto iBPMN models. The source language is now iPetri nets, which contains places, silent transitions and labelled transitions. The target language is iBPMN, whose building blocks are interactions. To achieve the mapping, we must identify in iPetri nets the patterns corresponding to the constructs in iBPMN. This section is structured in the following: Firstly, we introduce the fundamental difference between BPMN and iBPMN; then we introduce the mapping from iPetri nets onto iBPMN basic constructs; at last we illustrate the example of travel booking, which is mapped from the reduced iPetri nets model in Figure 4.19.

### 5.2.1 Mapping of Basic Constructs

In this section we will list all basic constructs, which can be mapped from iPetri nets. Figure 5.1 illustrates the patterns, which must be identified in iPetri nets, and the corresponding iBPMN constructs.

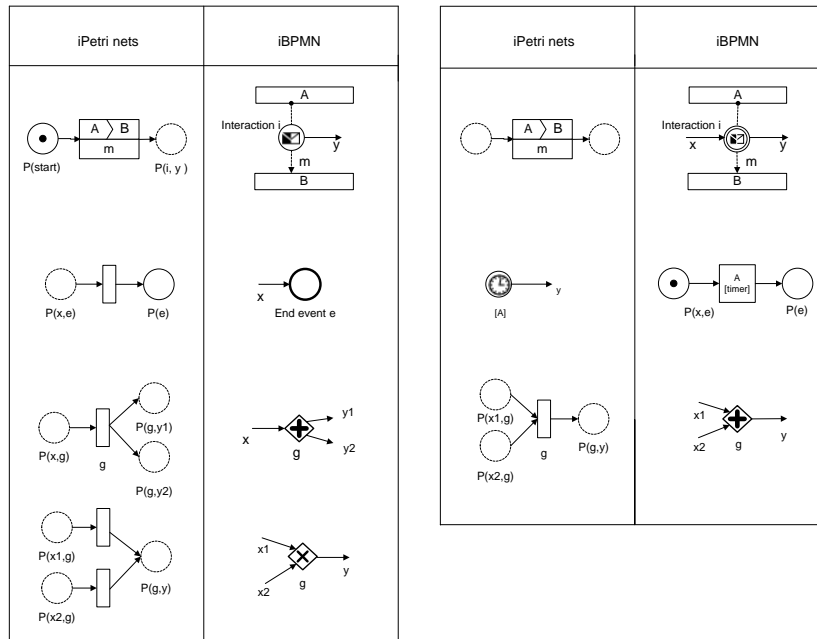


Figure 5.1: Mapping of basic constructs

As shown in Figure 5.1: Mapping of basic constructs, start places are the only places, which have no input arcs and only one output transition. There might be more than one start place. In iBPMN choreography, we use start interaction as the entry of a conversation. And there might be more than one entry node. In iPetri nets, we identify all places, which have no incoming arcs. Then we translate the places without incoming arcs and the following labelled transitions as a whole to start interactions.

In a similar way, end places are the only places, which have no output arcs. There might be more than one end place. We can translate the transitions and their output places without outgoing arcs together to end event.

As introduced in Section 2.4, we knew that the labelled transitions indicate the message exchanges or timer events. Intuitively, we translate the transitions with timer flag directly into timer event in iBPMN. The transitions representing interactions occurrences in iPetri nets contain the information of message sender/receiver and message type. Such transitions will be mapped onto interactions in iBPMN. In iBPMN we do not care about the internal behaviours of process. While in BPMN pools represent different roles, we use brackets to represent the roles involved in a conversation. In iPetri nets we attach the roles to the label of interaction occurrences. Hence, we translate the sender/receiver of message to different roles in iBPMN. Meanwhile, the message flows can be determined according to the sender/receiver information. That means, a message flow will be defined from role of sender to interaction, then to role of receiver in iBPMN.

As introduced in Section 3.1, we know that in iBPMN gateway plays the same role like in BPMN. The difference is only that the ownership of the decision gateway might be several roles. Hence, we can intuitively identify these gateways in iPetri nets, like the mapping from BPMN onto iPetri nets.

### 5.2.2 Normalized iPetri nets

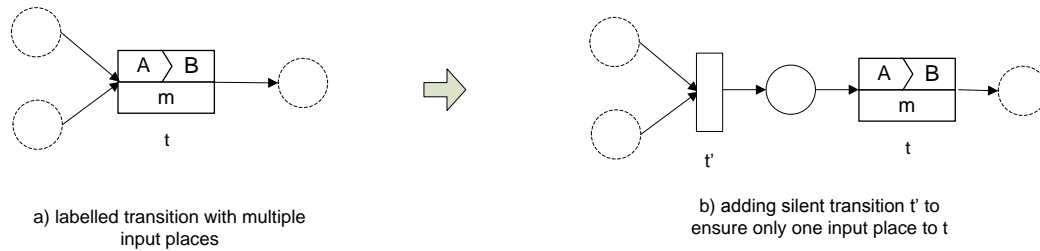
After the mapping from BPMN onto iPetri nets, we get the constructs as shown in Figure 4.19. Note that there might be labelled transitions which have multiple incoming or outgoing flows. These multiple incoming or outgoing flows of labelled transitions generated in the mapping. As shown in Figure 4.5 and Figure 4.6, the send task and the receive task originally are located in two different pools with different incoming and outgoing sequence flows. The algorithm joins the send and the receive task, but keeps the incoming and outgoing sequence flow. Thus, the labeled transitions have two incoming arcs and two outgoing arcs.

Using reduction rules, we can cut down the size of large iPetri nets. Meanwhile we reduce the redundant places. Thereby, the multiple incoming or outgoing flows will be removed as shown in Figure 4.15 or Figure 4.16 or Figure 4.17. When the multiple incoming or outgoing places of labelled transition are not redundant, the

multiple incoming or outgoing flows are kept in iPetri nets model. In the mapping concept, we will use the silent transitions with multiple incoming or outgoing places to represent the gateways. Labelled transitions represent the interactions in iBPMN. Thereby multiple incoming/outgoing flows lead to the problem that we cannot distinguish the branching constructs and interactions. Hence, we need to find an approach to solve this problem.

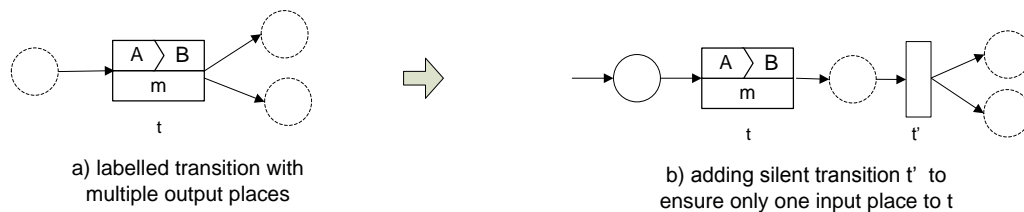
**Definition 10. (Normalized iPetri nets)** Let  $iPN$  be a iPetri nets, if all  $t \in T_i \cup T_c$ ,  $|\bullet t| = |t \bullet| = 1$ , we call  $iPN$  normalized.

To normalize the labelled transitions as nodes with single entry and single exit, we can add a silent transition between the multiple incoming/outgoing places and labelled transition.



**Figure 5.2: Adding silent transition before labelled transition**

We can prove that adding silent transition will not change the firing of labelled transitions. In Figure 5.2 b) the added silent transition  $t_s$  is in a sequence with labelled transition. The firing of added silent transition  $t_s'$  in Figure 5.2 b) is same as the labelled transition  $t$  in Figure 5.2 a). As shown in Figure 5.2 b), if the added silent transition  $t_s'$  can be fired, the labelled transition  $t$  can also be fired.



**Figure 5.3: Adding silent transition after labelled transition**

The same to the silent transition added before labelled transition, adding a silent transition after the labelled transition will not change the firing sequence. As shown in Figure 5.3, if labelled transition  $t$  is fired, so is the added silent transition  $t'$ .

Adding silent transitions helps us to distinguish the interactions and gateways in iPetri nets. It simplifies the identification of them. We can identify a silent

transition with one input place and multiple output places as parallel fork gateway. If this transition is fired, it produces tokens for every output place. All the following transitions of these places are fired. It captures the property of parallel fork gateways, i.e., all tasks following parallel gateways are parallel executed.

In a similar way, we can identify the transition with one output place and multiple input places as parallel join gateway. That means that the transition can be fired if every incoming place possesses a token. It captures the property of parallel join gateway, i.e. outgoing task of parallel join gateway can be executed until every incoming task has been finished.

In iPetri nets, multiple transitions merging to one place represents XOR merge gateway in BPMN. As defined in Chapter2, we know that the semantic and syntax of XOR merge gateways in iBPMN is same as in BPMN. Hence, in the mapping from iPetri nets onto iBPMN, we identify such construct and translate directly to XOR merge gateway.

The mapping of data-based gateway and event-based gateway is challenging. In the following subsection we discuss the mapping of data-based gateway and event-based gateway.

### **5.2.3 Mapping of Data-based and Event-based gateway**

In this thesis we are interested only in the direction from BPMN to iBPMN, i.e., the mapping from BPMN onto iBPMN. In the mapping from BPMN onto iPetri nets, we ignore some anti-patterns which can lead to modelling errors. These anti-patterns are already introduced in Section 2.2.3 and Section 2.3.2. We assume that the input models in BPMN do not contain such anti-patterns (D2, D3, O1,O3 ) but D1. D1 in a source model can be mapped eventually into correct constructs in iBPMN. As introduced in following, we summarize a few types of structure of iPetri nets constructs. We note that D1 might occur in type1, and it can be correctly mapped into iBPMN constructs.

In the mapping from BPMN onto iPetri nets, we translate a data-based decision gateway to one input place followed by multiple output transitions. Such transitions represent the structure of the data-based decision. They shared places with the transitions which represent the activities following the gateway. We translate the event-based gateway to one input place with multiple transitions which represent the activities following the gateway. We can see the difference between the structures of two gateways in Figure 4.1. After the reductions they are structurally same. The semantic of such two gateways, however, is different. We will observe the possible decision gateway constructs in iPetri nets and discuss the possible XOR decision gateway patterns in BPMN, the corresponding mapping rules from iPetri nets onto iBPMN from the point of structural view, and resulting models in iBPMN. In the following we first introduce the possible observed decision gateway constructs in iPetri nets.

### Type 1

In the Type 1 construct there are the same sender and the same receiver in every transition following the only one input place. The message types are different however. Figure 5.4 a) presents an example: A is the sender, B is the receiver. Every interaction transition has a distinct message type. From the semantic we can understand that role A sends different messages on different conditions. We can derive the original BPMN model: in pool of role A there must have been a decision gateway followed by different send tasks; and this decision gateway must be data-based decision gateway, because the event-based gateway cannot be syntactically followed by send tasks; in pool of role B there must have been an event-based gateway followed by receive tasks or intermediate message events. We know the sender A is responsible to the decision gateway, i.e. A is controlling role, who determine which message should be sent to B. We can translate this construct to a data-based decision gateway with controlling role A in iBPMN. The resulted construct in iBPMN is shown as in Figure 5.4 b).

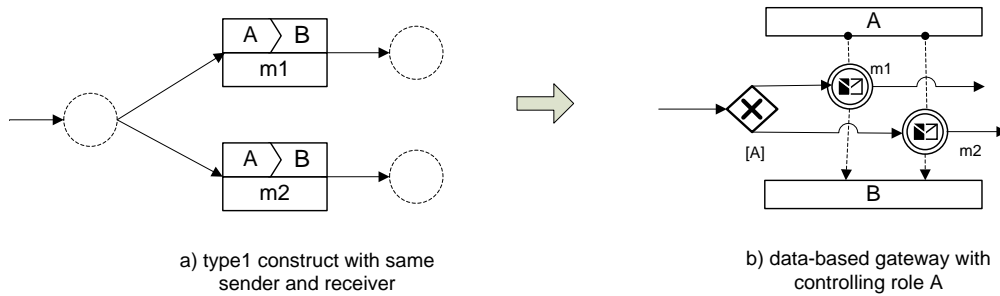
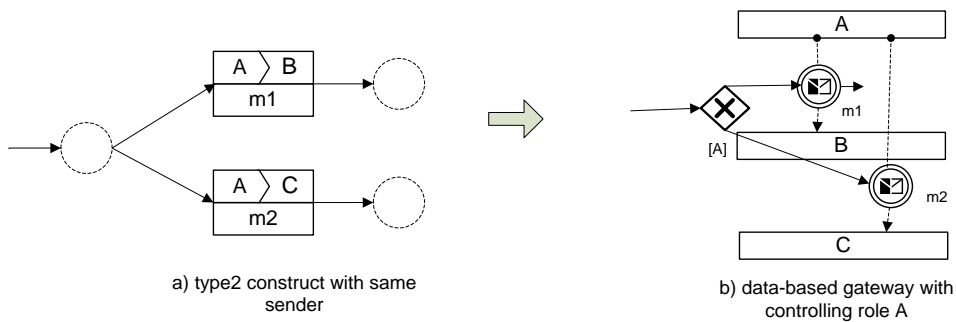


Figure 5.4: Type 1 decisions gateway in iPetri nets and iBPMN

### Type 2

In Type 2, as shown in Figure 5.5, all sender roles of labelled transitions are the same, the receivers are different. In BPMN, the decision which message is sent to who will be internally made in Pool of sender role. We translate this construct to a data-based decision gateway.



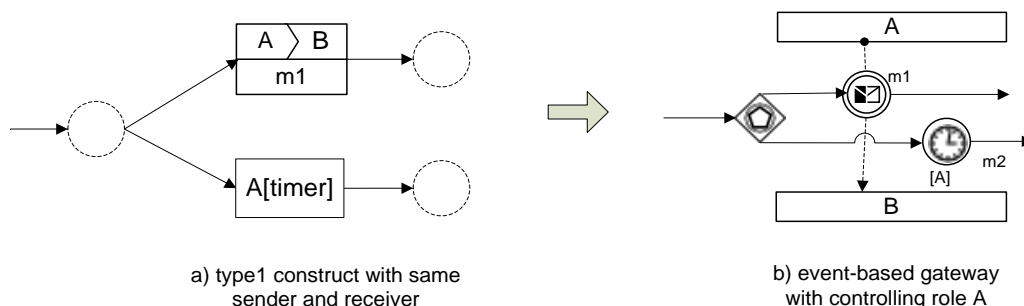
**Figure 5.5: Type2 decision gateway in iPetri nets and iBPMN**

### Type 3

In Type3, as shown in Figure 5.6, there is labelled transition with timer flag. This transition represent timer event.

In the resulted iPetri nets, we can get a construct, in which there is one input places followed by several labelled transitions and a controlled silent transition with a timer flag. We always translate such construct to event-based gateway independent of the other transitions following the input place. One of the events following the gateway is timer event. This construct implies race conditions. The decision which branch is chosen will not be based on the internal data of any process but on the event occurring first. Thus, we cannot determine the type of decision gateway according to the semantic, we can only deduce from the structure of the construct in iPetri nets. Hence, we map all the constructs followed by transition with timer flag onto event-based gateway in iBPMN. We show the example in Figure 5.6. Figure 5.6 a) shows the corresponding construct in iPetri nets, and Figure 5.6 b) represents the resulted construct in iBPMN.

It is important to note that we cannot get mixed choices in iBPMN, since mixed choices are not supported in BPMN [DBKL08].



**Figure 5.6: Type 3 decision gateway in iPetri nets and iBPMN**

### Type 4

By parity of reasoning we should have such construct ( shown in : one input place with multiple labelled transitions. Some of labelled transitions present that A sends message to B. Some of labelled transitions present the opposite direction, i.e. B sends message to A. It can be mapped onto event-based gateway followed interactions in iBPMN. And it is the anti-pattern that is introduced in Section 2.2.3 and Section 2.3.2. We know that this pattern is syntactically not supported in BPMN. Hence, we discuss this patter not in detail in this thesis.

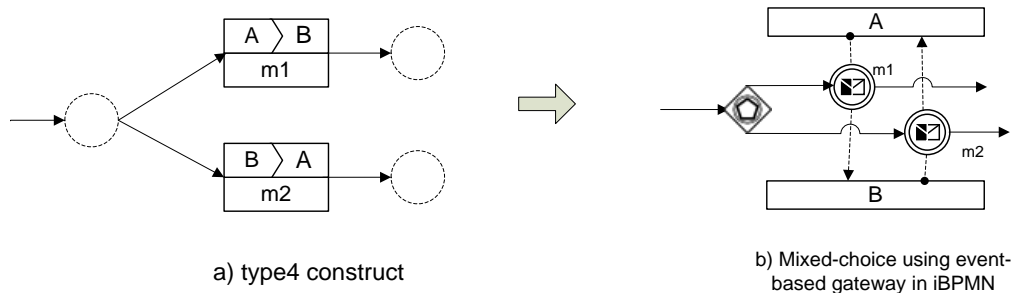


Figure 5.7: Type 4 mixed choices

### 5.2.4 Mapping of looping constructs

Like in BPMN, there are two types of representation for loops in iBPMN. One is using a shortcut notation with a loop marker to represent loops. Another is using XOR gateways. Hence, we proposal two approaches for the mapping of loops from iPetri nets onto iBPMN.

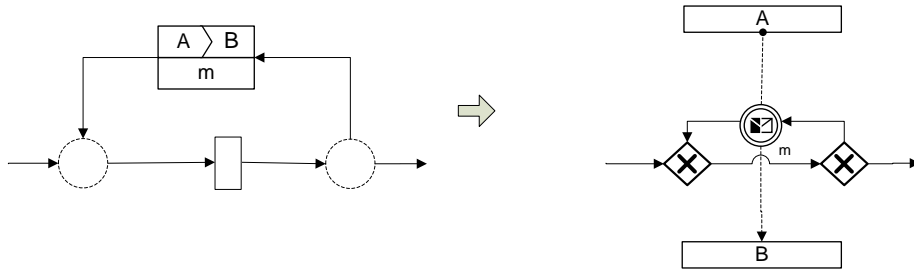
#### *Looping constructs with XOR gateways*

In Section 4.5 we introduced the mapping of loops from BPMN onto iPetri nets. The two forms of loops, “while-do” and “do-until”, can be mapped intuitively using XOR gateways. After the reduction, we usually get constructs shown in left side of Figure 5.8 which represent the two forms of loops in iPetri nets. We can directly identify the constructs which represent the XOR gateways. Note that we use data-based XOR gateways to represent the looping constructs in the mapping, because the decision making is according to the process data at each participant, not external trigger. The Figure 5.8 shows on the right side the looping constructs represented by XOR gateways in iBPMN.

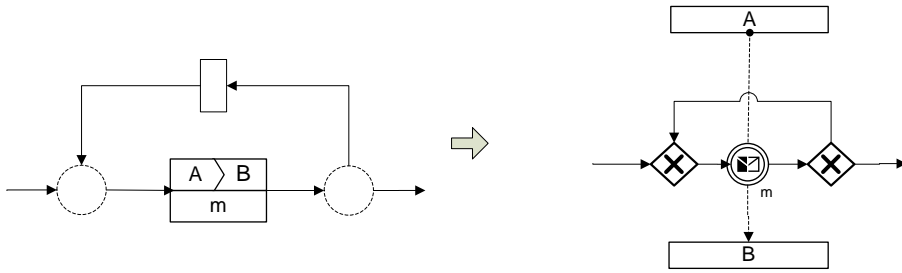
#### *Looping constructs with marked task*

Using the shortcut notation to represent loops covers both two forms of loops, i.e. both of two forms are represented by a rectangle with a loop marker in iBPMN. The two forms of loos will be distinguished by attribute “TestTime” (befor|after). Hence, the looping constructs will be mapped in a whole to the shortcut notation.

To map the whole looping construct, we first identify the construct in iPetri nets. For the identification of looping constructs we use component concept. We can regard a looping construct as a component. In the following, we first introduce the definition of component. The idea is based on the work of [ODH+07], where BPMN is mapped to BPEL. Details are presented in Section4 in [ODH+07].



a) "while-do" iPetri nets looping construct and mapped by XOR gateways in iBPMN



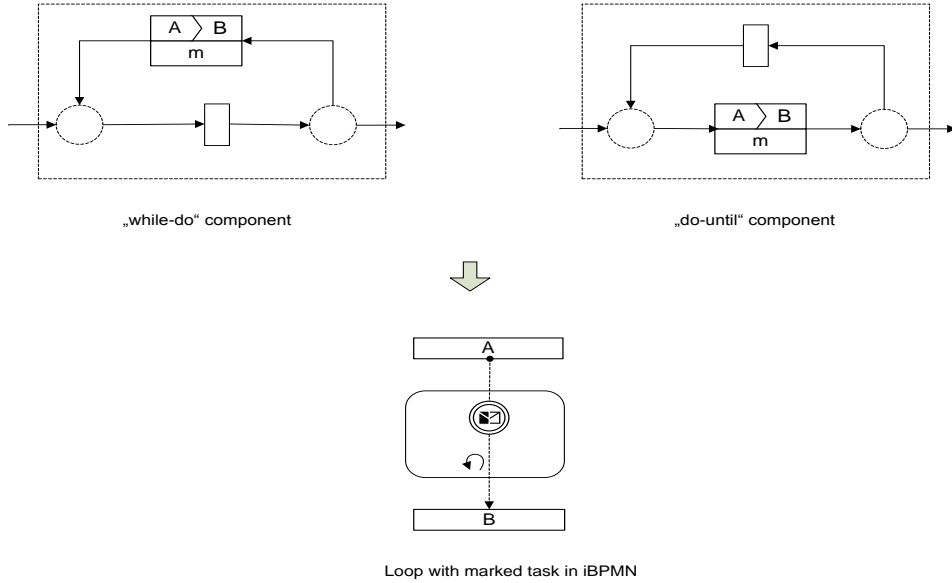
b) "do-until" iPetri nets looping construct and mapped by XOR gateways in iBPMN

**Figure 5.8: looping constructs representations with XOR gateway**

**Definition 11. (Component)** Let  $iPN = (P, T, F, R, s, r, timer, MT, t, M_0)$  be a iPetri nets,  $C$  is a component of iPetri nets if only if

- $C \subseteq P \cup T$ , a component consists of places and transitions,
- $|\bigcup_{x \in C} \{ \bullet x \} \setminus C| = 1$ , there is only one entry node out of a component, denote as  $entry(C)$ ,
- $|\bigcup_{x \in C} \{ x \bullet \} \setminus C| = 1$ , there is only one exit node out of a component, denote as  $exit(C)$ ,
- there exists a unique source node and a unique sink node  $i_C, o_C \in C$  such that,  $i_C \neq o_C, (i_C, o_C) \notin F$ , and  $entry(C) \in \{ \bullet i_C \}, exit(C) \in \{ o_C \bullet \}$ .

The definition declares that a component contains at least a place and a transition. Note that the source and sink nodes of a component in the definition may be either places, or both transitions, or respectively place and transition. Note that the source models in Figure 5.9 are the “place bordered”, i.e.  $i_C, o_C \in P$ . Hence, we only consider the components whose source and sink nodes both are places.



**Figure 5.9: looping constructs representation with marked task**

**Definition 12. (Loop Patterns)** Let  $C$  be a component of an iPN.  $i_C$  is the source node of  $C$  and  $o_C$  is the sink node of  $C$ , and  $i_C \in P \wedge o_C \in P$ .  $C$  is identified as a looping pattern iff

- i.  $C$  is identified as a while-do pattern iff
  - $|\{i_C \bullet\}|=1 \wedge |\{\bullet o_C\}|=1$ , i.e. the source node has only one output flow, and the sink node has only one input flow,
  - $i_C \bullet = \bullet o_C$ , i.e., the transition following the entry place is the same as the input transition of the exit place,
  - $\exists x_1, \dots, x_n \in C$ , such that,  $\{(o_C, x_1), (x_1, x_2), \dots, (x_n, i_C)\}$  subset of edges, i.e. there is a path from  $o_C$  to  $i_C$ .
- ii.  $C$  is identified as a do-until pattern iff
  - $|\{i_C \bullet\}|=1 \wedge |\{\bullet o_C\}|=1$ , i.e. the source node has only one output flow, and the sink node has only one input flow,
  - $\{\bullet i_C\} \setminus \text{entry}(C) = \{o_C \bullet\} \setminus \text{exit}(C)$ , i.e. the input transition of the source place in component is the same as the output transition of the sink place in component,
  - $\exists x_1, \dots, x_n \in C$ , such that,  $\{(i_C, x_1), (x_1, x_2), \dots, (x_n, o_C)\}$ , there is a path from  $i_C$  to  $o_C$ .

If such components can be identified in the source iPetri nets model, we can directly translate them to the shortcuts notation with loop marker.

### 5.2.5 Algorithm

In this section we introduce a generalized algorithm, which maps the iPetri nets model onto iBPMN choreography model. As introduced above, our approach identifies the patterns in iPetri model, and then translates them into corresponding patterns in iBPMN.

Before we introduce the generalized algorithm, we present an algorithm (shown in Figure 5.10) for adding silent transitions in the reduced iPetri nets model. The algorithm implements the transformation in Figure 5.10. Lines 2 to 20 loop through all silent controlled transitions. Line 3 checks, whether the current transition has more than one incoming edge. If yes, a new transition and a new place are added. Lines 12 checks whether the current transition has more than one outgoing edge. If yes, a new transition and corresponding place are added.

```

1: AddTran(  $iPN=(P,T,F,R,s,r,timer,MT,t,M_0)$ 
2:   for each  $t \in T_c$ 
3:     if  $|in(t)| > 1$ , then
4:        $t_{new} := new()$ ;
5:        $p_{new} := new()$ ;
6:        $T := T \cup \{t_{new}\}$ ;
7:        $P := P \cup \{p_{new}\}$ ;
8:        $F := F \cup \{(t_{new}, p_{new}), (p_{new}, t)\}$ ;
9:       foreach  $x \in in(t)$ 
10:          $F := F \setminus \{(x, t)\}$ ;
11:          $F := F \cup \{(x, t_{new})\}$ ;
12:     if  $|out(t)| > 1$ , then
13:        $t_{new} := new()$ ;
14:        $p_{new} := new()$ ;
15:        $T := T \cup \{t_{new}\}$ ;
16:        $P := P \cup \{p_{new}\}$ ;
17:        $F := F \cup \{(t, p_{new}), (p_{new}, t_{new})\}$ ;
18:       foreach  $x \in out(t)$ 
19:          $F := F \setminus \{(t, x)\}$ ;
20:          $F := F \cup \{(t_{new}, x)\}$ ;
21:   return ( $iPN$ );

```

Figure 5.10: Algorithm for adding silent transitions

And then we present the algorithm for the mapping from reduced iPetri nets onto iBPMN. Adding silent transitions helps us to distinguish the gateway constructs and interactions. As discussed in Section 4.2, we use silent transitions to represent the constructs of gateways. The labeled transitions represent the interactions and timer events. We firstly identify the interactions. Then we can determine the flow relation in iBPMN according to the flow relation in iPetri nets. The gateways will be mapped from the corresponding patterns identified in iPetri nets. As introduced in Section 5.2.3 we propose certain rules to identify the event-based and data-based gateways according to the iPetri nets structure. Note that, we introduce two approaches to represent looping constructs in iBPMN. To simplify the mapping, we use the first approach, i.e. using gateways to represent looping constructs. In the following we introduce the algorithm.

```

1: Input:  $iPN$  Output: iBPMN C
2:  $E^s := \emptyset; E^e := \emptyset;$ 
3:  $I := \emptyset; F_c := \emptyset; F^m := \emptyset;$ 
4:  $G^f := \emptyset; G^j := \emptyset; G^d := \emptyset;$ 
5:  $G^v := \emptyset; G^m := \emptyset;$ 
6:  $R_c := R;$ 
7: foreach  $t \in T_i \cup T_c$ 
8:   if  $\nexists x \in T, (x, in(t)) \in F$ , then //start event
9:     if  $t \in T_i$ , then
10:       $e^s := t.mtype;$ 
11:       $F^m := F^m \cup \{(t.sender, e^s, t.receiver)\};$ 
12:     if  $t \in T_c$ , then
13:       $e^s := t.timer;$ 
14:       $E^s := E^s \cup \{e^s\};$ 
15:     else, then //intermediate event
16:       if  $t \in T_i$ , then
17:         $i := t.mtype;$ 
18:         $F^m := F^m \cup \{(t.sender, i, t.receiver)\}$ 
19:       if  $t \in T_c$ , then
20:         $i := t.timer;$ 
21:         $I := I \cup \{i\};$ 
22:     if  $\nexists y \in T, (out(t), y) \in F$ , then
23:       $e^e := new();$ 
24:       $E^e := E^e \cup \{e^e\};$ 
25:     if  $x, y \in T_i \cup T_c \wedge x = in(t) \wedge y = out(t)$ 
26:       $F_c := F_c \cup \{(i_{[x]}, j_{[y]})\};$ 
27: forall  $t \in T_u$ ,
28:   if  $in(t) = 1 \wedge out(t) > 1$ 
29:      $G^f := G^f \cup g_{[t]};$  //parallel fork gateway
30:   if  $in(t) > 1 \wedge out(t) = 1$ 
31:      $G^j := G^j \cup g_{[t]};$  //parallel join gateway
32:   if  $x, y \in T_i \cup T_c \wedge x = in(t) \wedge y = out(t)$ 

```

```

33:       $F_c := F_c \cup \{(i_{[x]}, g_{[t]}), (g_{[t]}, i_{[y]})\}$ 
34: forall  $p \in P$ 
35:   if  $in(p) = 1 \wedge out(p) > 1$ 
36:     if  $\nexists out(p) \in T_c$ , then
37:        $G^d := G^d \cup g_{[p]}$ ; //data-based gateway
38:     else  $G^v := G^v \cup g_{[p]}$ ; //event-based gateway
39:   if  $in(p) > 1 \wedge out(p) = 1$ 
40:      $G^m := G^m \cup g_{[p]}$ ; //exclusive merge gateway
41:   if  $x, y \in T_i \cup T_c \wedge x = in(t) \wedge y = out(t)$ 
42:      $F_c := F_c \cup \{(i_{[x]}, g_{[t]}), (g_{[t]}, i_{[y]})\}$ 
43: return  $C := (E^s, E^e, F_c, F^m, G^f, G^j, G^d, G^v, G^m, R_c)$ ;

```

**Figure 5.11 presents an algorithm for the mapping from iPetri net onto iBPMN**

Figure 5.11 presents an algorithm for the mapping from iPetri nets onto iBPMN. The input of the algorithm is an iPetri net model. And the output of the algorithm is an iBPMN model. In this algorithm, line 7 loops through all labelled transitions. Lines 8 to 14 look for the start interactions, which are represented by a place without incoming arcs and a labelled transition in iPetri nets. And the message flows will be added in  $F^m$ . Lines 15 to 26 handle the intermediate interactions, and add the relations of these interactions in F, if they are connected by a place in iPetri nets. According to the information of sender and receiver in labelled transitions, the message flows will be added in  $F^m$ . From line 27 to 42 the algorithm handles the gateways, and adds the relations between interactions and gateways in flow relation F.

## 6 Summary and outlook

This thesis proposes an approach to provide a conceptual mapping from interconnection models to interaction models. This mapping is accomplished in two steps. Firstly BPMN models are mapped onto iPetri nets models. Next iPetri nets models are mapped onto iBPMN models. After first step we get a large iPetri net, Hence, the reduction of iPetri nets is an important part of the mapping. The approach provided in this thesis realizes the transformations between the interconnection choreography models and interaction choreography models. Using this approach, we can transform a designed interconnection model for choreography to an interaction model.

In this thesis we propose a method for the mapping of two different structured modelling languages. That is to use an intermediate modelling language. We choose iPetri nets in this thesis. Despite of reuse the notations in BPMN, iBPMN is fundamentally different with BPMN. As a formal representation for interaction models [DeWe08], iPetri nets are used to be a bridge between BPMN and iBPMN.

However there are some restrictions in the mapping in this thesis. Due to the limitations of iPetri nets, we cannot properly map the multiple instances. Hence, in this thesis we ignore these constructs. Because in BPMN we represent the interactions using respectively send and receive tasks/events, we can map them onto unidirectional message flow in iBPMN. However, using bidirectional message flows to represent interactions can be supported in iBPMN. These restrictions may be discussed in future work.

There exists Oryx editor for BPMN, iBPMN and iPetri nets [DOW08]. The mapping from interconnection models onto interaction models should be implemented and integrated into the Oryx editor as well.

## 7 Bibliography

- [Aa98] Van der Aalst W M P. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 1998, 8 (1) :21~66
- [AaLa05a] W.M.P. van der Aalst and K.B. Lassen. Translating Workflow Nets to BPEL. In: BETA Working Paper Series, WP145, Eindhoven University of Technology, 2005-08.
- [AaLa05b] W.M.P. van der Aalst and K.B. Lassen. Translating Workflow Nets to BPEL4WS. BETA Working Paper Series, WP 145, Eindhoven University of Technology, Eindhoven, 2005.
- [AH02] W.M.P. van der Aalst & A.H.M. ter Hofstede. Workflow patterns: On the expressive power of (petri-net-based) workflow languages, 2002.
- [AH04] W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245-275, 2004
- [BDO05] Barros, Alistair; Phillipa, Dumas; Oaks, Phillipa (2005): A Critical Overview of the Web Services Choreography Description Language. In: BPTrends, Jg. 2005, H. March.
- [Ber85] G.Berthelot, "Checking Properties of Nets Using Transformations," G.Rozenberg (ed), *Advances in Petri Nets 1985*, LNCS 222, N.Y., Springer-Verlag, 1987, pp. 19-40.
- [BPEL07] Web Services Business Process Execution Language Version 2.0. OASIS Standard, April 2007.
- [BPMN06] Business Process Modeling Notation (BPMN) Specification. Final Adopted Specification, Object Management Group (OMG), February 2006.
- [DB07] G. Decker and A. Barros. Interaction Modelling using BPMN. In Proceedings 1<sup>st</sup> International Workshop on Collaborative Business Processes (CBP), number 4928 in LNCS, pages 206{217, Brisbane, Australia, September 2007
- [DBKL08] Gero Decker, Alistair Barros, Frank Michael Kraft and Niels Lohmann. *Non-desynchronizable Service Choreographies*. Proceedings of the 6th International Conference on Service Oriented Computing (ICSOC), LNCS 5364, pp. 331-346, Sydney,

Australia, December 2008. Springer Verlag.

- [DDO07a] R.M.Dijkman, M. Dumas, and C. Ouyang. Formal semantics and automated analysis of BPMN process models. Preprint 7115, Queensland University of Technology, Brisbane, Australia, 2007.
- [DDO07b] R.M.Dijkman, M. Dumas, and C. Ouyang. Formal semantics and analysis of BPMN process models using Petri Nets. Technical report, Technical Report 7115, Queensland University of Technology, Brisbane, 2007.
- [Dec08] Gero Decker. “Choreografiemodellierung: Eine Übersicht”. In: Informatik-Spektrum 31.2 (2008), pp. 161–166. ISSN: 0170-6012. DOI: 10.1007/s00287-008-0227-3.
- [Dec09] Gero Decker. Design and Analysis of Process Choreographies. Doctoral Thesis. Business Process Technology Group Hasso Plattner Institute, University of Postdam, Postdam, Germany. June 2009.
- [DeWe07] G. Decker and M. Weske. Local Enforceability in Interaction PetriNets. In Proceedings 5th International Conference on Business Process Management 43 (BPM 2007), number 4714 in LNCS, pages 305{319, Brisbane, Australia, September 2007. Springer Verlag.
- [DeWe08] Decker, Gero; Weske, Mathias: Interaction-centric Modeling of Process Choreographies. Preprint submitted to Information Systems, 2008.
- [DKB08] Decker, Gero; Kopp, Oliver; Barros, Alistair P.: “An Introduction to Service Choreographies (Servicechoreographien - eine Einführung)”. In: it - Information Technology, Jg. 2008, H. 2, S. 122–127.
- [DOW08] Gero Decker, Hagen Overdick and Mathias Weske. *Oryx - An Open Modeling Platform for the BPM Community*. Demo Session of the 6th International Conference on Business Process Management (BPM), LNCS 5240, pp. 382-385, Milan, Italy, September 2008. Springer Verlag.
- [Jen92] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*. EATCS monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1992.

- [KL09] Kopp, Oliver and Leymann, Frank: *Do we need internal behavior in choreography models?* , Vol. 438 CEUR-WS.org (2009) , S. 68-73 .
- [LKL07] Lohmann, Niels; Kopp, Oliver; Leymann, Frank; Reisig, Wolfgang: Analyzing BPEL4Chor: Verification and Participant Synthesis. In: Dumas, Marlon (Hrsg); Heckel, Reiko (Hrsg): *Web Services and Formal Methods, Forth International Workshop, WS-FM 2007 Brisbane, Australia*
- [May81] Ernst W. Mayr, An algorithm for the general Petri net reachability problem, *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, p.238-246, May 11-13, 1981, Milwaukee, Wisconsin, United States [doi>10.1145/800076.802477]
- [SMD96] S.M. Shatz, S. Tu, T. Murata, and S. Duri, "An Application of Petri Net Reduction for Ada Tasking Deadlock Analysis," *IEEE Trans. Parallel and Distributed Systems*, pp. 1,307-1,322, vol. 7, no. 12, Dec. 1996.
- [Mur89] Murata,T.: *Petri nets: Properties, analysis and applications*. Proc.IEEE77(4)(1989) 541–580
- [ODH+07] Ouyang,C.,Dumas,M.,ter Hofstede,A.H.,van der Aalst, W.M.:Pattern-based translation of BPMN process models to BPEL web services. *International Journal of Web Services Research* (2007)
- [PK07] K. Pfitzner. *Choreography Configuration for BPMN*. Diploma Thesis No. 2618, University of Stuttgart, 2007.
- [Reis85] Wolfgang Reisig, *Petri nets: an introduction*, Springer-Verlag New York, Inc., New York, NY, 1985
- [WCL+05] Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey , Donald F.Ferguson, *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More*, Prentice Hall PTR, Upper Saddle River, NJ, 2005
- [WVA+06] M.T. Wynn, H.M.W. Verbeek, W.M.P. van der Aalst, A.H.M. ter Hofstede, D. Edmond, *Reduction rules for Reset Workflow Nets*. Technical report BPM-06-25, BPM Center (bpmcenter.org), 2006.

## Erklärung

Hiermit versichere ich, diese Arbeit selbstständig verfasst und nur die angegebenen Quellen benutzt zu haben.

Unterschrift:

Stuttgart, 14.12.2009