UNIVERSITÉ NICE - SOPHIA ANTIPOLIS

ÉCOLE DOCTORALE STIC
SCIENCES ET TECHNOLOGIES DE L'INFORMATION
ET DE LA COMMUNICATION

T H È S E

pour l'obtention du grade de

Docteur en Sciences

de l'Université Nice - Sophia Antipolis
Mention : Informatique

Présentée et soutenue par
Thi Hoa Hue Nguyen

La vérification de patrons de workflow métier basés sur les flux de contrôle

Une approche utilisant les systèmes à base de connaissances

Thèse dirigée par Nhan LE-THANH

soutenue le 23 June 2015

Jury :

Rapporteurs :	Parisa GHODOUS	Université de Lyon I
	Ladjel BELLATRECHE	ISAE-ENSMA
Directeur :	Nhan LE-THANH	Université Nice Sophia Antipolis
Examinateur :	Peter SANDER	Université Nice Sophia Antipolis

Acknowledgments

First of all, I would like to express my sincerest and deepest gratitude to Prof. Nhan LE-THANH. Thank you so much for introducing me to the field of research and taking me on as a PhD candidate. I will always be greatly indebted to you for your support and guidance.

I would like to express my gratefulness to Prof. Parisa GHODOUS and Prof. Ladjel BELLATRECHE for agreeing to review this thesis. I want to say a sincere thank to them for their time and thoughtful comments. And also, I would like to express my gratefulness to the members of jury for agreeing to participate to the presentation.

I would also like to express my sincere regards to Prof. Peter SANDER who looked closely at the final version of the thesis for English style and grammar, correcting both and offering suggestions for improvement. I would also like to extend my thanks to Dr. Alain GIBOIN. You gave me encouragement and very valuable advice for this thesis.

I warmly thank the members of WIMMICS team for their valuable advice on my research methodology and contributions, especially Dr. Fabien GANDON, Dr. Olivier CORBY and Dr. Catherine FARON-ZUCKER.

I am indebted to all my friends who have supported me over the last few years: Oumy SEYE, Alexander SPETKO, Amosse EDOUARD, Amel BEN OTHMANE, Tien Thinh NGUYEN, Jodi SCHNEIDER, Duc Phu CHAU, Imen TAYARI, Anh-Tuan NGHIEM and DUONG Thi Quynh Anh. I would also like to thank my Vietnamese friends in Nice and Sophia Antipolis. With them, I have a feeling like to live in a big family.

Last but not least my family: Words cannot express how grateful I am to them. My parents, my husband and my son, you have always believed in me. You are always only and only supportive without questioning. I would like to thank you for all of the sacrifices that you have made on my behalf. I am very lucky to have a family so great and so unique.

Résumé : Cette thèse traite le problème de la modélisation des patrons de workflow semantiquement riche et propose un processus pour développer des patrons de workflow. L'objectif est de transformer un processus métier en un patron de workflow métier basé sur les flux de contrôle qui garantit la vérification syntaxique et sémantique. Les défis majeurs sont : (i) de définir un formalisme permettant de représenter les processus métiers; (ii) d'établir des mécanismes de contrôle automatiques pour assurer la conformité des patrons de workflow métier basés sur un modèle formel et un ensemble de contraintes sémantiques; et (iii) d'organiser la base de patrons de workflow métier pour le développement de patrons de workflow.

Nous proposons un formalisme qui combine les flux de contrôle (basés sur les Réseaux de Petri Colorés (CPNs)) avec des contraintes sémantiques pour représenter les processus métiers. L'avantage de ce formalisme est qu'il permet de vérifier non seulement la conformité syntaxique basée sur le modèle de CPNs mais aussi la conformité sémantique basée sur les technologies du Web sémantique.

Nous commençons par une phase de conception d'une ontologie OWL appelée l'ontologie CPN pour représenter les concepts de patrons de workflow métier basés sur CPN. La phase de conception est suivie par une étude approfondie des propriétés de ces patrons pour les transformer en un ensemble d'axiomes pour l'ontologie. Ainsi, dans ce formalisme, un processus métier est syntaxiquement transformé en une instance de l'ontologie. La vérification syntaxique d'un processus métier devient simplement une vérification par inférence, par concepts et par axiomes de l'ontologie sur l'instance correspondante. Nous introduisons aussi la définition formelle de contraintes sémantiques, qui exprime les dépendances entre les activités d'un processus métier. Nous présentons un algorithme pour la vérification des contraintes sémantiques redondantes et conflictuelles. Un ensemble de contraintes sémantiques vérifiées est transformé en une instance de l'ontologie de processus métier appelée BP-ontology. Un patron de workflow métier est ensuite développé en créant des correspondances entre l'ontologie BP et l'ontologie CPN. Il permet les vérifications sémantiques d'un processus métier spécifique.

Nous représentons l'ensemble des axiomes de l'ontologie CPN lié à la conformité syntaxique ainsi que les questions de vérification sémantique liées à la conformité sémantique en utilisant des requêtes SPARQL. Afin de vérifier les patrons de workflow, nous utilisons le moteur sémantique Jena pour l'adaptation d'un graphe RDF représentant un patron de workflow métier de ces requêtes SPARQL. Si un patron de workflow métier est vérifié, il sera stocké dans une base de connaissances.

De plus, dans l'objectif de fournir un soutien supplémentaire pour la définition de règles métiers, nous introduisons des règles sous forme de Condition Action Événement (CEA), qui expriment l'exactitude des processus au niveau métier. Les ensembles de règles CEA sont stockés avec le patron de workflow métier correspondant dans la même base de connaissances. La base est organisée pour faciliter la capacité de partage et de réutilisation des patrons de workflow. Enfin, un prototype est conçu pour démontrer la faisabilité et les avantages de l'approche.

Mots clés : Contrainte Sémantique, Ontologie, Réseaux de Petri colorés, SPARQL, Vérification, Workflow métier

Abstract: This thesis tackles the problem of modelling semantically rich business workflow templates and proposes a process for developing workflow templates. The objective of the thesis is to transform a business process into a control flow-based business workflow template that guarantees syntactic and semantic validity. The main challenges are: (i) to define a formalism for representing business processes; (ii) to establish automatic control mechanisms to ensure the correctness of a business workflow template based on a formal model and a set of semantic constraints; and (iii) to organize the knowledge base of workflow templates for a workflow development process.

We propose a formalism which combines control flow (based on Coloured Petri Nets (CPNs)) with semantic constraints to represent business processes. The advantage of this formalism is that it allows not only syntactic checks based on the model of CPNs, but also semantic checks based on Semantic Web technologies.

We start by designing an OWL ontology called the CPN ontology to represent the concepts of CPN-based business workflow templates. The design phase is followed by a thorough study of the properties of these templates in order to transform them into a set of axioms for the CPN ontology. In this formalism, a business process is syntactically transformed into an instance of the CPN ontology. Therefore, syntactic checking of a business process becomes simply a verification by inference, by concepts and by axioms of the CPN ontology on the corresponding instance.

We also introduce the formal definition of semantic constraints, which express dependencies between the activities of a business process. We present an algorithm to check redundant and conflicting semantic constraints. A set of well-checked semantic constraints is transformed into an instance of a business process ontology called the BP ontology. A business workflow template is then developed by creating correspondences between the BP ontology and the CPN ontology. This enables semantic checks related to a specific business process.

We represent the set of axioms of the CPN ontology related to syntactic checks as well as the semantic verification issues related to semantic checks as SPARQL queries. In order to verify workflow templates, we use the Jena semantic engine to match an RDF graph representing a business workflow template to graph patterns of these SPARQL queries. If there are no matches, i.e., no shortcomings, a workflow template is then stored in a knowledge base.

In addition, to provide additional support for specifying business rules, we introduce Event Condition Action (ECA)-like rules that express business level correctness requirements. The sets of ECA-like rules are stored along with the corresponding business workflow template in the same knowledge base. The knowledge base is organized to facilitate the shareability and reusability of workflow templates. Finally, a prototype is developed to demonstrate the feasibility and benefits of the approach.

Keywords: Business Workflow, CPN, Ontology, Semantic Constraint, SPARQL, Verification

Contents

List of Figures

List of Tables

Chapter 1

General Introduction

Contents

1.1 Introduction

Nowadays, software systems that automate business processes have become more and more available and advanced. According to [omg 2000], process models, which are firstly designed during the build-time phase on the basis of design requirements, are then automated by software systems during run-time. Therefore, grasping the requirements properly and then transforming them without losing any information into a semantically rich specification play an important role in supporting business process management.

So far, various researchers have focused on process specification techniques [Ellis 1993, van der Aalst 1998] and conceptual models of workflow [Barros 1997, Koschmider 2005]. However, the existing practice of modelling business processes is mostly manual and is therefore vulnerable to human error. A workflow designed incorrectly may lead to failed workflow processes, execution errors or may not meet the requirements of customers. Therefore, model quality, correctness and re-usability become very important issues. It is desirable to develop a thorough and rigorous method that automatically supports workflow designers to ensure high quality and semantically rich business processes.

In fact, existing techniques applied to check the correctness of workflows are particularly used in commercial business workflow systems. Most of them assume that a workflow is correct if it complies with the constraints on data and control flow during execution [Lu 2006]. Whether the workflow is in conformity with the design requirements is neither specified nor proved. Consequently, numerous approaches have been developed to ensure workflow correctness at the syntactic level (e.g.,

avoiding deadlocks and infinite cycles, etc.), however, it is usually not sufficient. In fact, at the semantic level errors may still exist.

Let us take an example in a process for the order management activity, when an order is approved, an order confirmation has to be sent to the customer. However, if the order confirmation is sent before the approval of the order, a semantic error occurs.

Recently, there is a little few research that focus on checking the semantic conformance of workflows. Nevertheless, there is an inherent problem regarding the combination of syntactic and semantic checks that needs to be taken into account.

In order to address the above-mentioned problem, we focus on machine-readable knowledge bases. The objective is to support workflow designers in generating semantically rich business workflow templates which allow syntactic and semantic verification. With regard to the former, a set of syntactic constraints is introduced to provide automated support for workflow designers. With regard to the latter, we specify semantic constraints as *domain specific restrictions on a business process which express dependencies between activities and need to be conformed while the process is executed.* We concentrate on the following research questions relating to the verification of a business workflow template:

1. How to model semantically business workflow templates?
2. Can syntactic and semantic checks be supported?
3. How to organize the knowledge base of business workflow templates for a workflow development process?

To better motivate our research, let us consider the following scenario, which can serve as a typical example for better understanding the problem of modelling business processes and reusing them. The scenario will illustrate the problem descriptions that will be used as examples to demonstrate our proposed solution in the next chapters.

1.2 Scenario

In the scenario we will mention:

- A repository, called *CBWTRepository*, contains business workflow templates. The templates stored in *CBWTRepository* are generic and can be used to model specific process models according to the *CBWTRepository* customer's requirements;
- A customer company, named *CompanyA*, has imported workflow templates from *CBWTRepository* to build its own business application.

In the following we describe a set of workflow templates relating to the *fromOrdertoDelivery* (fOtD) process. We also present the requirements of *CompanyA*

concerning its business policy. Customer companies can use the workflow templates to model their own fOtD process in compliance with their requirements. In Section 1.2.1, the templates are mentioned and described in their generic form. In Section 1.2.2, we introduce a *CompanyA* variant of the fOtD process and illustrate an adaptation of the templates used to model the fOtD process for *CompanyA*.

There are a lot of workflow templates used to model the *fromOrdertoDelivery* process, such as templates for dunning, templates for returning purchased goods, templates for claims and templates for notification. However, to make this scenario easier to understand, we just highlight the four main templates as follows:

(i) *Order Processing*

(ii) *Invoicing*

(iii) *Payment*

(iv) *Shipment*

1.2.1 *fromOrdertoDelivery* Process Model

1.2.1.1 Order Processing

The *Order Processing* template (see Figure 1.1[1].) is used to model an order processing process. It is worth noting that a workflow-step can be a sub-workflow in itself. For example, the step *check item availability* contains some workflow-steps, e.g., *check internal item availability*, *check external item availability*, which are not illustrated in the figure for the sake of simplicity.

Table 1.1: *Order processing* template document

Order processing template	
Description	This template covers the time from the creation of an order to the approval of the order. An order can contain one or more requested items and the information concerning clients. Therefore, a checking phase, which may consist of a validation of client's data and validation of the availability of requested items, can be initiated after receiving an order from a client. The result of this phase is then evaluated. Based on the evaluation, a decision whether the order is approved or rejected is made.
Purpose	To represent a set of activities for modelling an order processing process
Related templates	*Invoicing*, *Notification*, *Payment*, *Shipment*, *Inventory*, *Purchased Goods Returning*
Keywords	Approval, Checking, Confirmation, Creation, Items, Order, Submitting, Validation

[1]The templates are described in Section 1.2 based on BPMN [bpm 2011]

1.2.1.2 Invoicing

The *Invoicing* template (see Figure 1.3) is used to model an invoicing process.

Table 1.2: *Invoicing* template document

Invoicing template	
Description	This template is used to model the process that generates new invoices if ordered items have been shipped or if the payment is obligatory to be handled before the shipment/delivery step. An invoice is prepared to send to the client (purchaser, buyer, customer) for each order.
Purpose	To represent a set of activities for invoicing an order
Related templates	*Order Processing*, *Notification*, *Payment*, *Shipment*
Keywords	Invoice, Bill

1.2.1.3 Payment

The *Payment* template (see Figure 1.2) is used to execute a payment process in response to the received invoices.

Table 1.3: *Payment* template document

Payment template	
Description	This template is used to handle the payment process. In this process, a client (purchaser, buyer, customer) has to choose a payment method (through a payment service provider or a bank) to pay the agreed monetary value to a seller. The template also contains activities to process overdue payments and to remind the client about outstanding debts.
Purpose	To represent a set of activities for modelling a payment process
Related templates	*Invoicing*, *Order Processing*, *Notification*, *Shipment*
Keywords	Cash, Credit card, Payment

1.2.1.4 Shipment

The *Shipment* template (see Figure 1.4) is used to model a shipment process.

Table 1.4: *Shipment* template document

Shipment template	
Description	In general, there are two contexts that a shipment process can take place. - A shipment process can be initiated after receiving a request against an order; or - Ordered items can be shipped directly to the client from the supplier when a shipment process is in 'drop shipment'. In both cases, the ordered items are expected to be delivered to the correct address indicated by the client. A shipment process terminates when the ordered items reach the delivery address. Besides, some activities can be involved in the shipment template, such as packing, service delivery or transportation.
Purpose	To represent a set of activities for modelling a shipment process
Related templates	*Order Processing*, *Inventory*, *Invoicing*, *Order Processing*, *Payment*, *Purchased Goods Returning*, *Notification*
Keywords	Delivery, Goods, Items, Packing, Shipment scheduling, Transportation

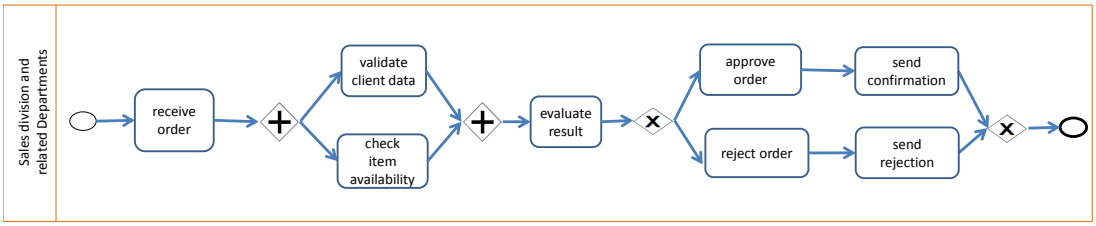

Figure 1.1: *Order processing* template

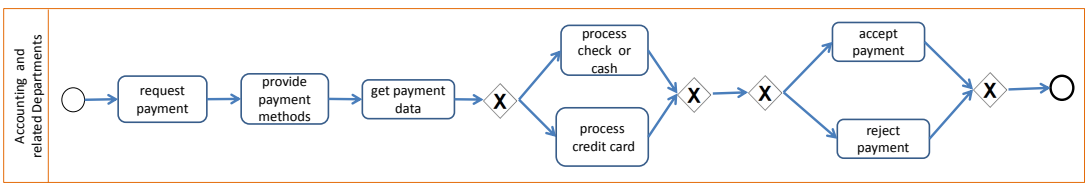

Figure 1.2: *Payment* template

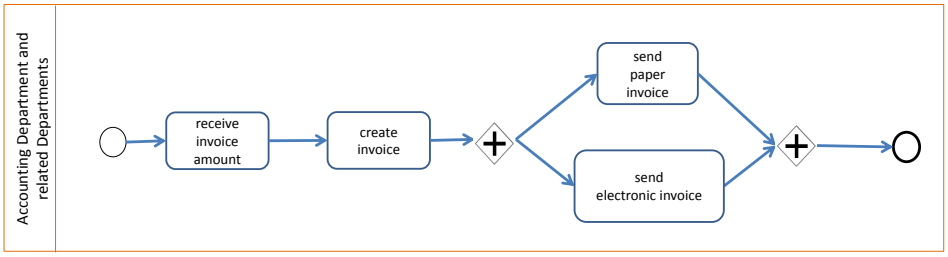

Figure 1.3: *Invoicing* template

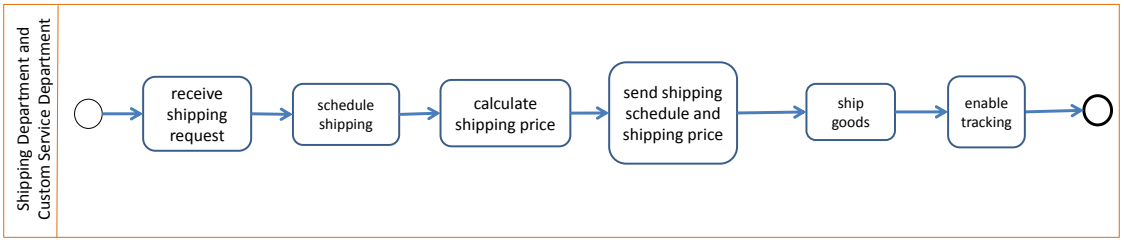

Figure 1.4: *Shipment* template

In the upcoming section, we present the business of a company, namely *companyA* and describe how to apply the above templates to its fOtD process.

1.2.2 Adapting templates stored in *CBWTRepository* to model the *fromOrdertoDelivery* Process for *CompanyA*

CompanyA, based in *France*, plans to create a fromOrdertoDelivery process. Instead of developing the process from scratch, this company has imported workflow templates from *CBWTRepository* to build its own business application.

Let us take a brief look at the company's policy concerning the fromOrdertoDelivery process: *CompanyA* manages an online shopping website selling beauty products. About payment, with regard to online cosmetic orders, all orders must be prepaid. The company accepts credit cards, including *VISA*, *MasterCard*, and *American Express*. For the promotional codes, only one code (if applicable) may be used for one purchase.

An order can be shipped via an indicated shipping service. Back orders are not accepted. Customers are allowed to change their shipping method before completing their online order. Shipping charges are based on the order value and shipping address as follows:

- Within *France*, goods which cost in excess of *EUR 100* per order will be delivered free of charge, conversely, a flat rate delivery charge of *EUR 6.80* will be applied.

- Within the rest of the *European Union* (*EU*), goods which cost in excess of

EUR 150 per order will be delivered free of charge, conversely, a flat rate delivery charge of *EUR 7.50* will be made.

- Shipment to *NON-EU* countries will be free of charge for order values of *EUR 200* or over. If the order value is less than *EUR 200*, a flat rate delivery charge of *EUR 10* will be made. Additional customs duties, taxes and charges may be incurred for delivering to the *NON-EU* countries.

Charges are for each shipment and will be added to the invoice.

An order can be cancelled by calling to the Customer Service Department but only if the shipment has not yet been confirmed.

Customers can return their purchased goods by sending them back to the indicated company's address. Returns must be accompanied by invoice and they can be accepted only within *30* days of purchase. All returned products must be unused, and in saleable condition.

Accepted returns will be re-credited to the corresponding customers. Requests for refunds must be made in writing and will be granted only if no account balance is due.

Figure 1.5 shows an excerption of the fOtD process applied to company *CompanyA*. In this excerption we can see the re-use of two templates, i.e., *Shipment* and *Payment*. Some steps of these templates are modified or deleted. For example, a set of steps, which is used to calculate shipping price, replaces the step *calculate the shipping price* in the *Shipment* template.

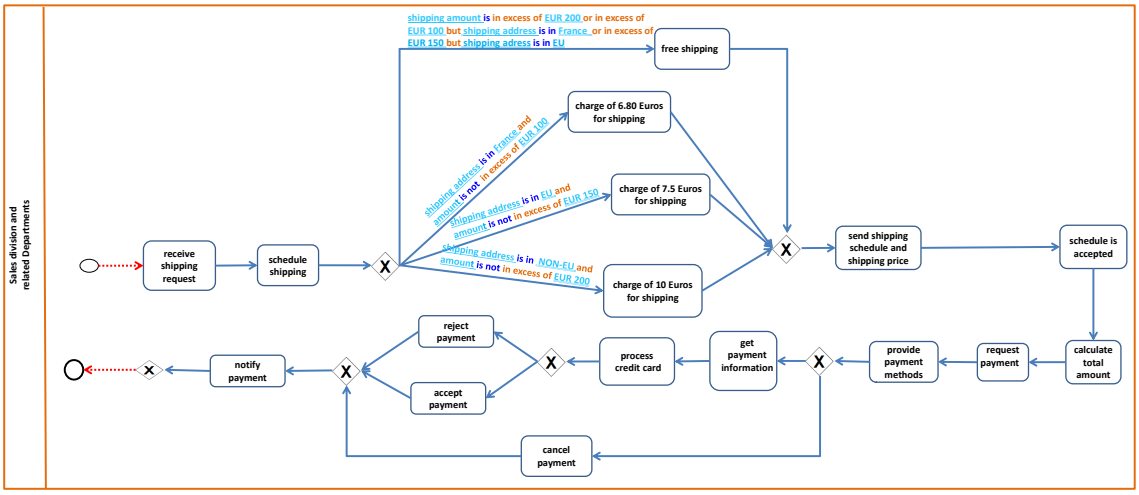

Figure 1.5: *CompanyA* variant of the fOtD process (excerpt)

1.3 Proposal and Main Contributions

To answer the research questions mentioned in Section 1.1, we introduce a formalism to represent control flow-based business workflow templates (CBWTs) in a knowl-

edge base. The formalism is designed to facilitate the shareability and reusability of workflow templates. It combines control-flow (based on Coloured Petri Nets (CPNs)) with semantic constraints of business processes. This combination enables syntactic checks based on the model of CPNs and semantic checks based on Semantic Web technologies. Here are the main contributions of this thesis:

- Modelling semantically rich business workflow templates:

  On the one hand, for the formalization of control-flow in workflow templates, we focus on modelling business processes with CPNs. We first design an OWL ontology, called the CPN ontology, to represent the concepts of control flow-based business workflow templates (i.e., templates of business processes modelled with CPNs). Next, we thoroughly study the properties of the workflow templates in order to transform them into a set of axioms for the ontology. A business process is thus syntactically transformed into an instance of the CPN ontology. As a result, syntactic checks become simply a verification by inference, by concepts and by axioms of the CPN ontology on the corresponding instance.

  On the other hand, a formal definition of semantic constraints is introduced to model semantic business processes. A set of semantic constraints is generally specified with the help of domain experts[2]. However, when defining a set of semantic constraints, it may be redundant or conflicting. Therefore, we introduce an algorithm to validate sets of semantic constraints. A set of well-checked semantic constraints is then automatically transformed into an instance of a business process ontology, called the BP ontology.

  By creating correspondences between the CPN ontology and the BP ontology, a workflow template is developed. Semantic checks related to a specific business process, therefore, are enabled.

- Providing automated support for syntactic and semantic checks related to a workflow template.

  In this thesis, the set of axioms of the CPN ontology related to syntactic checks as well as the semantic verification issues related to semantic checks are represented as SPARQL queries. The Jena semantic engine is then used to match an RDF graph representing a business workflow template to graph patterns of these SPARQL queries. If there are no matches, a workflow template is verified and stored in a knowledge base.

- Expressing business level correctness requirements by using Event Condition Action (ECA)-like rules.

  In order to provide additional support for specifying business rules, we introduce ECA-like rules to represent the business level correctness requirements that semantic constraints cannot capture.

[2] A group of people who are responsible for relevant business processes working at operational departments, where the business processes are intended to be run.

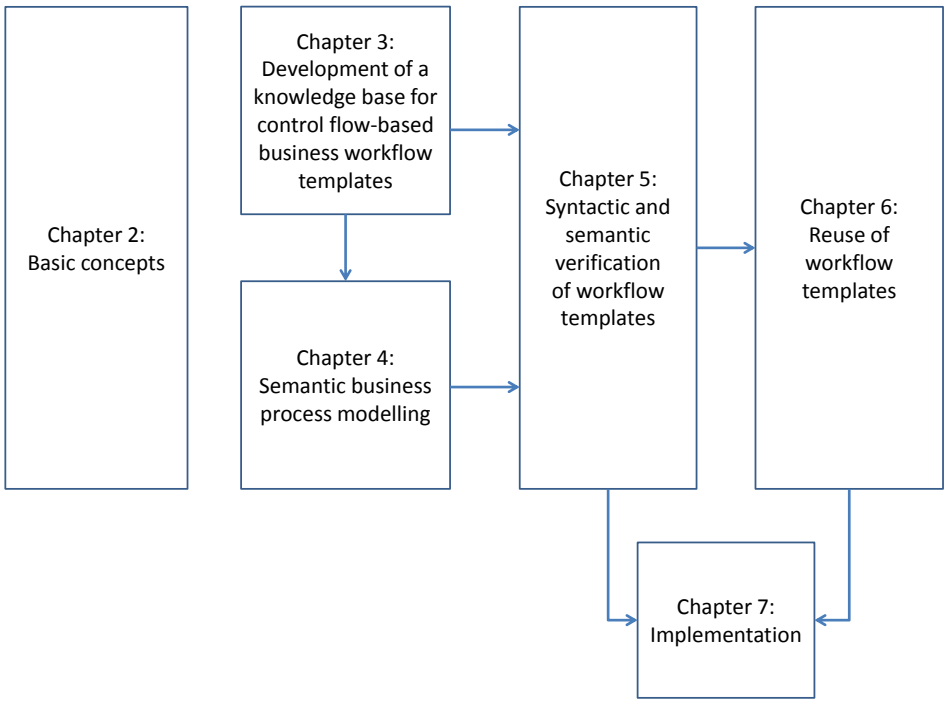

Figure 1.6: Overview of thesis

- Establishing a knowledge base to guide the appropriate workflow templates for the development of a business workflow template.

1.4 Thesis Outline

This thesis is structured as follows (see Figure 1.6):

- Chapter 2 introduces the basic concepts of business workflows and business rules. Another objective of this chapter is to represent the knowledge involved in knowledge bases relying on the Semantic Web models for the verification of a business workflow template.

- Chapter 3 provides a formal definition of CPN-based process models. In addition, the CPN ontology, which is developed to represent the concepts of CPN-based business workflow templates, is also introduced.

- Chapter 4 gives a formal definition of semantic constraints and an algorithm for inferring implicit semantic constraints and detecting shortcomings. A set of well-checked constraints is then used to model a semantic business workflow template. In addition, to integrate domain knowledge, ECA-like rules are also introduced to represent business level correctness requirements.

- Chapter 5 concentrates on the syntactic and semantic verification of a business workflow template. The verification indicates that a template does or does not conform to a set of given constraints.
- Chapter 6 describes a repository that contains business workflow templates and their ECA-like rules. It provides an organizational mechanism for CBWTs to guarantee an effective search of workflow templates. Thereby users can select and modify the workflow templates along with their ECA-like rules for each use case.
- Chapter 7 provides an overview of the CBWT prototype which is implemented to validate the concepts discussed in the previous chapters.
- Chapter 8 concludes this thesis and provides some future research tracks.

We utilise our journal, conference and workshop publications, including [Nguyen 2013, Nguyen 2014a, Nguyen 2014b, Nguyen 2014c, Nguyen 2015, Pham 2015], to complete some parts of this thesis.

Chapter 2

Basic Concepts

Contents

In this chapter, we focus on: (i) briefly comparing business workflows with scientific workflows; (ii) introducing the basic concepts of business workflows and business rules; (iii) the representation of the knowledge involved for the verification of a business workflow template.

2.1 Workflows and Workflow Languages

2.1.1 Business Workflows versus Scientific Workflows

Over the years, workflows have drawn an enormous amount of attention from the research communities. Many workflow products, which are mainly workflow management systems (WfMSs), have become commercially available. Business, scientific calculations and experiments are two main areas that drive and utilize workflows. In this section, we present the similarities and differences between business and scientific workflows based on their objectives from different point of views

In fact, in the business world, the formal concept of workflows has existed for a long time. In [WFMC 1999], the Workflow Management Coalition described a business workflow as *"the automation of a business process*[1]*, in whole or part, during*

[1]WfMC [WFMC 1999] defined a business process as *"a set of one or more linked procedures or activities which collectively realise a business objective or policy goal, normally within the context of an organisational structure defining functional roles and relationships"*

which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules". On the other hand, to help scientists to implement and execute complex analyses, scientific workflows are defined differently, *"these are networks of analytic steps that may involve, e.g., database access and querying steps, data analysis and mining steps, and many other steps including computationally intensive jobs on high performance cluster computers"* [Ludäscher 2006].

For WfMSs that control aspects of a workflow, scientific and business WfMSs offer different sets of features. From the end-user's point of view, as stated in [Yildiz 2009], they both refer to:

(i) model and specify processes with design primitives;

(ii) re-engineer developed processes, like verification and optimization;

(iii) execute automatically processes by scheduling, controlling and monitoring the tasks.

The design of business WfMSs is generally independent from the concrete business area of employing enterprises. Consequently, this workflow technology follows the generic approach. Therefore, IT experts play an important role in implementing business processes of the enterprise and establishing the software infrastructure (see Figure 2.1). It is important to note that business workflows aim to automate and optimize an organization's processes in an administrative context to reduce costs (e.g., human resources) and increase revenue. They often represent the products of enterprises [Sonntag 2010], for example a reservation in a travel agency stands for the product "reservation". Up to now, there are more than a hundred business WfMSs, such as FileNet[2], SAP[3], JBPM[4] and Spiff Workflow[5]. Insurance, banking and health industries, for example, are domains using business workflows.

In contrast to business counterparts, scientific WfMSs are often designed for a specific application domain. Scientific workflow systems focus on supporting scientists in designing and implementing large-scale and complex e-science processes of scientific applications. Figure 2.2 depicts the scientific workflow life cycle. In this context, workflows implement scientific simulations, experiments, and computations often dealing with large amounts of data [Sonntag 2010]. Scientific workflows enable scientists to integrate, structure, and orchestrate heterogeneous and distributed services and applications into scientific processes [Lin 2008]. Obviously, scientists are expert in their own research areas and of course, they are the main users able to model, execute, monitor, and analyse their own workflows without requiring deep knowledge as professional software developers. Besides, in order to implement a

[2] http://www-01.ibm.com/software/ecm/filenet/

[3] http://help.sap.com/saphelp_46c/helpdata/en/c5/e4a930453d11d189430000e829fbbd/content.htm

[4] http://www.jbpm.org/

[5] https://pypi.python.org/pypi/SpiffWorkflow

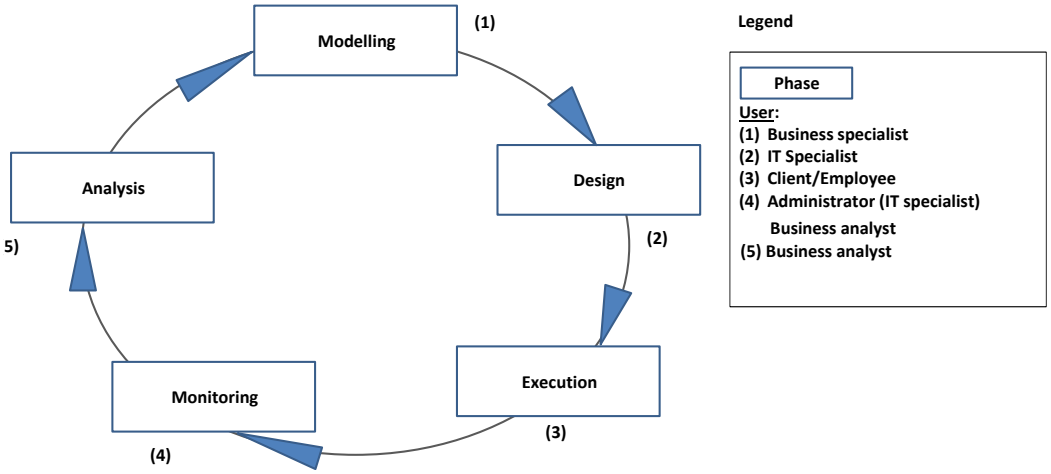

Figure 2.1: Business Process Management life cycle [Sonntag 2010]

workflow design, workflow engineers commonly are involved because of several complex computing units and some technical skills. Furthermore, scientific workflows are usually executed in an evolving environment, therefore, the goal of scientific workflows is not only to reduce both human and computing cost, but also speed up the transfer of large amounts of bits and bytes into knowledge and discovery. A number of scientific WfMSs have been designed and developed such as Pegasus[6], Taverna[7], Triana[8] and Kepler[9].

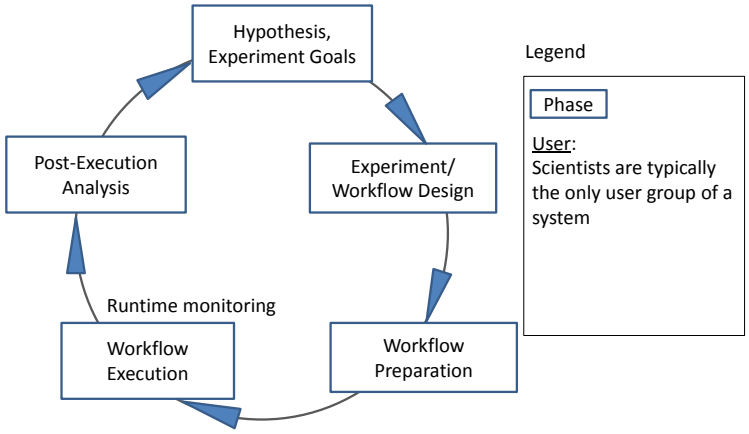

Figure 2.2: Scientific Workflow life cycle [Ludäscher 2009]

With regard to conceptual modelling and workflow design, a set of features and primitives is often provided to process designers. Both scientific and business

[6] http://pegasus.isi.edu/
[7] http://www.taverna.org.uk/
[8] http://www.trianacode.org/
[9] https://kepler-project.org/

WfMS use primitives to represent different tasks, dependencies, decisions and models of computational structures. The primary difference between them is that business workflows focus on modelling control-flow oriented business processes while scientific workflows, which aim to model large-scale data-intensive and compute-intensive scientific processes, tend to be dataflow oriented.

As depicted in Figure 2.3, an edge $A \rightarrow B$ in a business workflow naturally means that B is executed after A and they are only executed once, i.e., the edge represents control-flow. Furthermore, dataflow, which is implicit or modelled separately, is often the secondary issue in business workflows. Conversely, in a scientific workflow, $A \rightarrow B$ typically represents dataflow, i.e., actor B consumes the output of actor A. In dataflow modelling, no precise execution order between tasks is mentioned. Therefore, in contrast to business workflows where only tasks not on the same paths can be executed concurrently, scientific workflows can execute simultaneously a number of tasks on the same dataflow path as illustrated in Figure 2.3. Consequently, dataflow and control-flow are normally married in scientific workflows. The advantage of the marriage is that the resulting model is often simpler and allows stream-based, pipeline-parallel execution [Ludäscher 2009]. The disadvantage is that it is not easy to model certain workflow patterns (for conditional execution, for example) via dataflow.

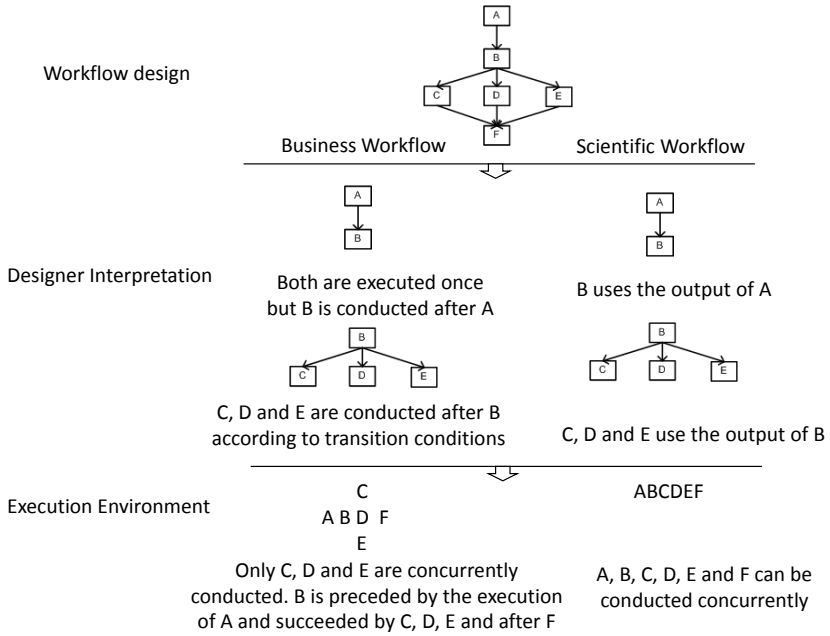

Figure 2.3: Brief comparison of Business Workflows and Scientific Workflows

Each typical scientific workflow can be seen as a computational experiment. They are exploratory in nature and often conducted in a what-if or a trial-and-error manner. Hence, the outcome of a scientific workflow not only can validate/prove or invalidate a scientific hypothesis, but also can serve some similar experimental

goals. In contrast, the outcome of a business workflow is already known before it starts through business-driven goals. For example, when applying for a bank loan, the proposal can be approved or rejected.

With regard to workflow instances, large numbers of cases and independent workflow instances can be handled by business workflows at any given time. However, truly independent instances are not as common in scientific workflows. A scientific workflow can invoke multiple related and interdependent instances, for example, in the context of parameter studies.

In compliance with our objective, business workflows are chosen for our work. We concentrate on the representation of control flow-based business workflow templates in a knowledge base.

2.1.2 Workflow Charateristics

In this Subsection, we introduce some basic concepts of business workflows and their perspectives based on [van der Aalst 1998, van der Aalst 2002b, van der Aalst 2003a].

According to [van der Aalst 1998], workflows are case-based, i.e., *tasks* are executed for specific *cases*. Some examples of cases are an order, a tax declaration, a wire transfer or a request for a medical examination. Each case has a unique identity and a limited lifetime. For example, in case of a wire transfer, it begins at the moment when the wire transfer is submitted and expires when the processing of the wire transfer has been completed.

Similar cases have the same case type and in principle they can be handled in the same way. A *workflow process* is designed to handle similar cases as efficiently and effectively as possible. The *workflow process definition* specifies which tasks need to be performed and in which order [van der Aalst 2002b]. Workflow process definition can also be regarded as 'procedure', 'flow diagram' or 'routing definition'.

Being a logical unit of work, a task is atomic and thus always executed in full. Checking account information, informing a result, calculating a formula are some examples of tasks. Since the task is done in a specific order, identifying *conditions* which relate to causal dependencies between tasks is necessary. A condition holds or does not hold (true or false) [van der Aalst 2003a]. Each task has pre-conditions and post-conditions which should hold before and after the task is executed, respectively.

A task, which refers to a generic piece of work, is defined for a type of case not for one specific case, i.e., the same task can be performed for many cases. In addition, to avoid confusion between the task itself and its performance relating to a particular case, the terms *work item* and *activity* are used. The former refers to a task which needs to be executed for one specific case. The latter refers to the actual execution of a work item.

Work items are executed by *resources*. Resources are human (e.g., workers, employees, etc.) and/or non-human (e.g., machines). Resources are grouped into classes in order to facilitate the allocation of work items to resources. Each *resource class* contains a set of resources with similar characteristics. A resource class based

on the capabilities of its members is call a *role*.

Figure 2.4 depicts three dimensions of a workflow [van der Aalst 1998], including the process (or control flow), the resource and the case dimension.

A work item and an activity are both related to a specific case. Consequently, the process dimension and the resource dimension are generic, not tailored towards any specific case. Individual cases that are concerned with the third dimension are executed in accordance with the process definition by the proper resources.

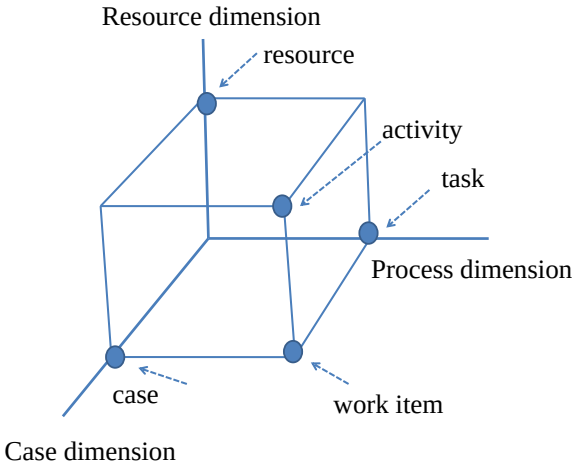

Figure 2.4: A three dimensional view of a workflow [van der Aalst 1998]

In this thesis, we concentrate on developing business workflows which handle cases. Therefore, we focus on the process and case dimension. By using Coloured Petri Nets (CPNs) (see Section 3.1.1) as the workflow language, the routing of cases, which is one of the main issues of the two dimensions, is syntactically represented. We will only present the mechanisms. Therefore, the resource dimension, which relate to human resource aspects, as well as the mapping of resources to work items will not be discussed in detail.

2.1.3 Workflow Languages

Workflow languages can be categorized into several classes according to their underlying methodologies and meta models, such as graph-based, Petri-net based and workflow programming languages [Weske 1998]. The constructs and relationships of workflow models of certain workflow languages are described through a meta model.

Graph-based languages allow the specification of workflows, which consists of workflow activities, their hierarchical relationships and constraints on their data flow and control flow, by using directed graphs. Therefore, to cover the workflow aspects (i.e., the functional, behavioural, informational, operational and flexibility aspect), these graphs need to be enhanced, for instance, using graph notation to specify the functional and behavioural aspects. Graph-based languages support the basic workflow patterns as stated in [van der Aalst 2003b]. These languages also provide workflow modelling constructs, such as iteration and nesting.

The second class of workflow languages is based on **Petri Nets** (PNs) [Petri 1962]. A PN is a directed bipartite graph that comprises three main components:

- *Places*: Holding tokens that represent states. The number of tokens in a place can vary over time;
- *Transitions*: Representing activities or tasks. Transitions may consume and produce tokens;
- *Directed arcs*: Linking transitions and places. An arc can only connect a place with a transition or vice versa.

PNs have been developed as a tool to represent, validate and verify workflow procedures [van der Aalst 1997, van der Aalst 2000]. The authors in [van der Aalst 2002a] stated three good reasons for using PNs as a workflow language, including: "(i) formal semantics despite the graphical nature; (ii) state-based instead of (just) event-based; (iii) abundance of analysis techniques". In addition, PNs have been extended with colour and time, called high-level Petri Nets, to improve expressiveness. High-level Petri net tools, e.g., CPN Tools [The AIS group, Eindhoven University of Technology 2013] or ExSpect [exs 2000], have been developed to incorporate these extensions and support workflow designers in modelling and analysing complex systems. Coloured Petri Nets are chosen as the workflow language in our work. Therefore, we will briefly introduce CPNs in Section 3.1.1.

The last class of workflow languages are **workflow programming** (or **script**) languages. However, since script languages are often used in projects where system development issues play a major role, in this research we will not pay much attention to them.

2.2 Business Rules

The concept of 'business rule' has been widely used in the context of expert systems. According to the Business Rules Group (BRG)[10], a business rule is "*a statement that defines or constrains some aspect of business. It is intended to assert business structure or to control or influence the behaviour of the business.*".

Generally speaking, in a business process, there exist multiple decision points at which a number of criteria (so-called business rules) are evaluated. The behaviour of the business process is then changed based on these business rules. Consequently, business rules play the core drivers role in the business processes of an enterprise.

Business rules represent particular business logic in a specific context. They consist of internal and external business rules. The internal business rules of a

[10]The BRG is an independent organization which comprises experts in the field of systems and business analysis methodology. Website: http://www.businessrulesgroup.org/first_paper/br01c1.htm

company are normally expressed by documents in natural language about operating principles, marketing strategies and pricing policies, etc. However, some business rules may exist as expert knowledge of some particular domain and have never been written down. On the other hand, the external business rules are defined by other instances, such as legal requirements.

It is worth noting that business rules tend to be changed often. Therefore, it is difficult and costly to change and maintain business rules if a business process embeds its rules inside itself. According to [van Eijndhoven 2008], the solution to avoid this issue is to separate business processes from business rules. Business Rules Management Systems (BRMSs) [Resch 2010], tools for business rules management coming from expert systems, are used for separating logic from the application code.

In literature, a number of approaches focus on describing business rules, such as Semantics Of Business Vocabulary And Rules (SBVR) [sbv 2013], Production Rule Representation (PRR) [prr 2009], Object Constraint Language (OCL) [ocl 2014]. Being adopted as a standard of the Object Management Group (OMG), SBVR is proposed to formalize complex compliance rules, such as operational rules for an enterprise, standard compliance or regulatory compliance rules. The objectives of SBVR are to define:

- the vocabulary and rules for documenting the semantics of business vocabularies, business facts, and business rules in a certain business domain. Consequently, SBVR rules capture "what" business rules are, rather than "how" they can be executed;
- an XML representation for the interchange of business vocabularies and business rules among organizations and between software tools.

In SBVR, each rule builds on at least one fact and facts build on concepts as expressed by terms. For example, "It is necessary that each customer has at least one bank account" is a business rule[11]

Business rules can generally be seen as independent business knowledge units which relate to some forms of reasoning. They are categorized based on certain characteristics in order to easily handle the set of business rules. According to [sbv 2013], business rules are divided into three types, i.e., *static constraints*, *dynamic constraints* and *derivation rules*. The authors in [Hay 2000] introduce a similar classification which includes *structural assertions*, *action assertions* and *derivations*. Three same types can be found in [Romanenko 2006] named as *structural rules*, *dynamic*

[11] There are four font styles used in the SBVR-based Structured English:

- term: The 'term' font is used for designations for object types, noun concepts (other than individual noun concepts).
- Name: The 'name' font is used for designations of individual noun concepts — names.
- verb: The 'verb' font is used for designations for verbs, prepositions, or combination thereof.
- keyword: The 'keyword' font is used for linguistic symbols used to construct statements - the words that can be combined with terms, Names and verb to create business rules.

rules and *derivation rules* respectively. Eijndhoven et al. in [van Eijndhoven 2008] identify two main categories, *rules* (consisting of derivation rules and action rules) and *constraints* (enforcing certain limitations to the structure, behaviour or information of an organisation or system). G. Wagner in [Wagner 2002] also defines *constraint*, *derivation* and *reaction* rules. Among different classifications just mentioned above, we will follow the classification of SBVR. We use the denotations: **structural rules**, **action rules** and **derivation rules** (see Appendix A).

In addition, business rules are often represented as Condition-Action rules (so-called production rules) or Event-Condition-Action (ECA) rules to enforce business rules directly by an automated system:

- A production rule, which is expressed in the format of "**IF** condition **THEN** action", specifies that one or more concrete actions are executed in the case that its conditions are fulfilled. Usually, users or an application can invoke production rules but then a rule engine will process them automatically.

- An ECA rule specifies that after an event (E) takes place, a clause condition (C) is checked and if it is fulfilled then the action (A) is executed. The general syntax of ECA rules is "**ON** event **IF** condition **DO** actions".
  Besides, ECA rules can be automatically triggered when certain events take place. They can react to events in real time. Furthermore, depending on the rule language is used, a ECA rule can specify a single (atomic) event or a composite event. For example, a temporal composition of events is mentioned in [Boley 2007, Bry 2005, Taveter 2001].

Production rules and ECA rules are widely supported by existing engine rules. They can be regarded as two variants of action rules [van Eijndhoven 2008]. Besides, it is necessary to underline that some structural rules and derivation rules can be also represented in the form of production rules as well as ECA rules. Let us take an example: The following structural rule:
It is obligatory that each rental car is owned by exactly one branch.
can be represented by the following production rule:
If a car is a rental car then it belongs to one branch.
Therefore, production rules and ECA rules are considered as the most convenient way for representing business rules.

2.3 Knowledge Representation in the Semantic Web Models

Our work aims to develop a knowledge base for workflow process templates. Therefore, we base on the Semantic Web models in which the accessibility, interoperability, expressiveness, share and reuse of workflow process templates are guaranteed. This section provides a brief overview of the Semantic Web models and the formalisms that are currently used for knowledge representation: RDF, RDFS, OWL and SPARQL.

2.3.1 Semantic Web Pyramid

Being a proposition of Tim Berners-Lee who invented the World Wide Web, the Semantic Web is characterized by a set of technologies, tools and standards. They are organized into a Semantic Web Stack that is an expression of their interrelationships. Figure 2.5[12] describes different layers of the Semantic Web architecture where each layer uses the capabilities of the layer below. The lower layers provide the syntactic interoperability (URI, Unicode and XML). The upper layers correspond to a standard model for data interchange on the Web (RDF), ontology modelling languages (RDFS and OWL), a query language designed specifically to query RDF databases (SPARQL) and rule languages (RIF and SWRL).However, according to [Bénel 2010], the feasibility of the last three layers (i.e., the Logic, Proof and Trust layers) still seems unclear.

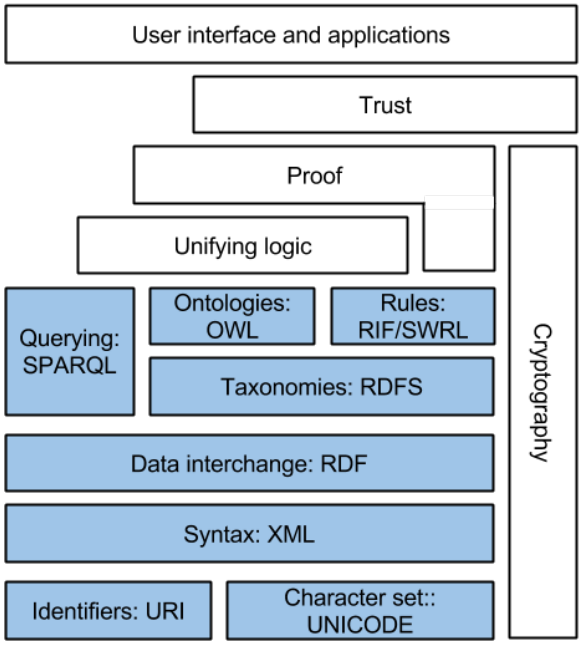

Figure 2.5: Semantic Web layered architecture[13]

2.3.2 An Assertional Language: RDF

RDF (Resource Description Framework) [RDF 2014a] is a framework for representing information. It is a standard model for data interchange on the Web. RDF is based on the idea of identifying things using Uniform Resource Identifiers (URIs) [Berners-Lee 2005] and describing resources in terms of simple properties and property values. A URI is a string of characters used to identify a name of a resource.

[12] The coloured layers (in blue) have been standardized [Bénel 2010]
[13] http://en.wikipedia.org/wiki/Semantic_Web_Stack

The basic structure of RDF is a graph (called "RDF graph"), composed of triplets. An RDF triple contains three components conventionally written in the order {subject, predicate, object}, where:

- the subject is an RDF URI reference or a blank node;
- the predicate is an RDF URI reference;
- the object is an RDF URI reference, a literal or a blank node.

RDF is regarded as the basis of the Semantic Web. The Semantic Web is then seen as a graph whose resources are interconnected via properties whereas the Web connects documents via hyperlinks. RDF has an XML syntax and is recommended by W3C[14].

2.3.3 Ontology Representation Languages: RDFS and OWL

- **RDF Schema** (RDFS) [rdf 2014b], which is a W3C recommendation since February 2004, semantically extends RDF. RDFS defines the vocabulary used in RDF descriptions. In other words, RDFS provides mechanisms for describing groups of related resources and the relationships between these resources. RDFS is written in RDF using its terms and intended to structure RDF resources. It allows users to define resources with classes, properties and values.

  Although RDFS provides simple but powerful modelling primitives for capturing basic semantics of the domain knowledge, it has some limitations. For example, it is not able to express equivalence between properties and does not have the capability of expressing the uniqueness and the cardinality of properties [Cardoso 2007]. Therefore, by representing classes and properties, RDFS is suitable for representing lightweight ontologies.

- **The Web Ontology Language** (OWL), a W3C Recommendation, is a family of knowledge representation languages for authoring ontologies. OWL provides a greater ability to interpret Web content than that supported by XML, RDF, and RDFS. Using RDF/XML syntax, OWL integrates a number of elements of its predecessor RDFS. It provides more vocabulary for describing properties, classes and relations between classes (e.g., $owl : disjointWith$), cardinality (e.g., $owl : someValuesFrom$), characteristics of properties (e.g., $owl : TransitiveProperty$).

  In the first version of OWL (named OWL 1 [owl 2004]), OWL can be categorized into three sub-languages with an increasing degree of expressiveness: OWL Lite, OWL DL, and OWL Full (see Figure 2.6):

  - **OWL Lite** is the syntactically simplest OWL language and corresponds to description logic $\mathcal{SHIF}(\mathcal{D})$. It supports creating simple class hierar-

[14] http://www.w3.org/

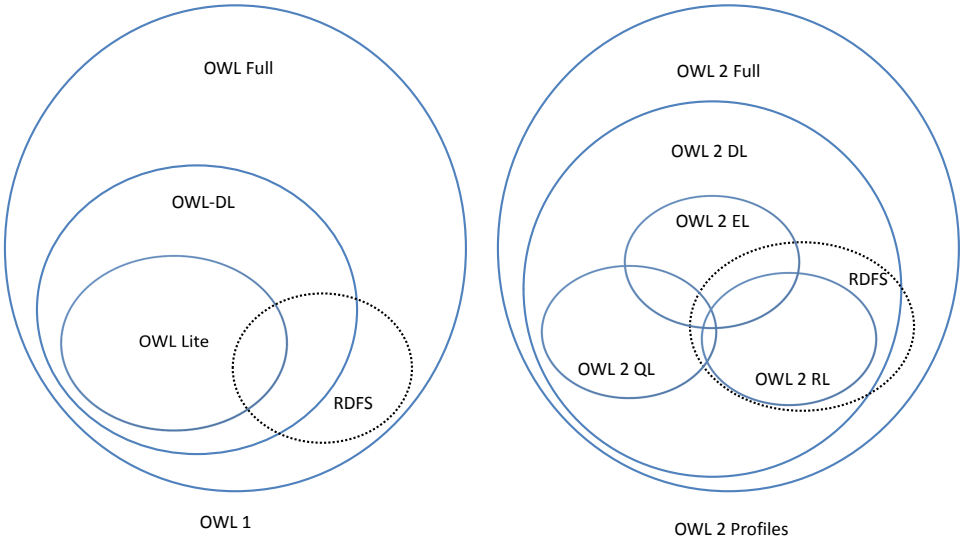

Figure 2.6: Web Ontology Languages OWL

chies and simple constraints (e.g., only cardinality values of 0 or 1 are allowed).

- **OWL DL**, which stands for OWL Description Logic, is equivalent to Description Logic $\mathcal{SHOIN}(\mathcal{D})$. It supports all OWL language constructs with restrictions (e.g, type separation) and provides maximum expressiveness while always keeping computational completeness and decidability.
- **OWL Full** which is the most expressive sub-language of OWL. OWL Full is intended to be used in applications where very high expressiveness is more important than being able to guarantee the decidability or computational completeness of the language. It is thus impossible to perform automated reasoning on OWL-Full ontologies.

The second version of OWL (named OWL 2 [owl 2012]) has a very similar overall structure to OWL 1. Although all OWL 1 Ontologies remain valid in OWL 2 Ontologies, new ontological components are introduced in OWL 2. The axioms of disjoint union of classes, of new properties for expressing qualified cardinality restrictions and of annotation properties; new data types and data ranges; and the concept of property chains are some examples.

In addition to OWL 2 DL and OWL 2 Full, OWL 2 specifies three profiles (see Figure 2.6):

- OWL 2 DL is defined from the set of primitives of OWL 2 under certain conditions of use of these primitives similar to those previously stated for OWL DL. It corresponds to the description logic $\mathcal{SROIQ}(\mathcal{D})$.
- OWL 2 EL, which corresponds to the description logic EL (Existential Language), provides only the existential quantification.

- OWL 2 QL (Query Language) is a specific subset of primitives of OWL 2 for response operations to queries, which can be implemented by rewriting the queries in a relational language like SQL.
- OWL 2 RL (Rule Language) is defined from OWL 2 by imposing certain restrictions, i.e., it does not allow existential quantification to a class, union and disjoint union to class expressions. These restrictions allow OWL 2 RL to be implemented using rule-based technologies such as rule extended DBMSs and Prolog.

2.3.4 Representation of Queries: SPARQL

SPARQL [spa 2013] is a query language, inspired by SQL for querying RDF data. It is adapted to the specific structure of RDF and relies on the triplets that constitute them. SPARQL allows adding, removing, searching and/or modifying data in RDF format. It can also be used to query RDFS or OWL vocabularies (written in RDF).

The SPARQL query language has the four following forms that use the solutions from pattern matching to form result sets or RDF graphs:

- SELECT query is used to extract values, which are all, or a subset of the variables bound in a query pattern match, from a SPARQL endpoint. The variables, which contain the return values, are listed after a SELECT keyword. In the WHERE clause, one or more graph patterns can be specified to describe the desired result;
- CONSTRUCT query is used to return an RDF graph constructed by substituting variables in a set of triple templates;
- ASK query is used to return a boolean indicating whether a query pattern matches or not;
- DESCRIBE query is used to return an RDF graph that describes the resources found.

Here are the reasons why we choose the SPARQL query language for the verification of a workflow template in Chapter 5:

(i) It is an RDF query language;

(ii) It is a W3C Recommendation and is widely accepted in the Semantic Web and also Artificial Intelligence community;

(iii) Its syntax is quite simple which allows for a query to include triple patterns, conjunctions, disjunctions and optional patterns;

(iv) It can be used with any modelling language.

2.4 Conclusion

The key issue in ensuring the syntactic and semantic correctness of business workflow templates during design time is to automate the process of checking whether a workflow template is or is not consistent with a set of predefined constraints. This problem is characterized by a large amount of semantic constraints, which express dependencies between activities of a business process, and a set of syntactic constraints using to model a business workflow template. To effectively maintain this knowledge, it is desirable to first formally represent it.

In this chapter, we have presented some basic concepts of business workflows and business rules. We have also introduced the models of the Semantic Web, which we use to represent the knowledge involved in modelling semantically rich business workflow templates (Chapter 3, Chapter 4 and Chapter 6) and the verification of workflow templates (Chapter 5).

Chapter 3

Development of a Knowledge Base for Control flow-based Business Workflow Templates

Contents

According to [Jørgensen 2008], Coloured Petri Nets (CPNs) have formal semantics and can describe any type of workflow system, behavioral and syntax wise simultaneously. They have been successfully applied in modelling workflows and workflow systems. Therefore, CPNs are chosen as the workflow language in our work.

In this chapter, we introduce an ontological approach to represent Control flow-based Business Workflow Templates (CBWTs) (i.e., templates of business processes modelled with CPNs) in a knowledge base. In detail, we first introduce a formal definition of CPN-based business process models which is used to transform a business process into a control flow-based business workflow template. Next, the CPN ontology is developed to represent Coloured Petri Nets with OWL DL. We then introduce manipulation operations on workflow templates for developing CBWTs.

3.1 Modelling Business Processes with Coloured Petri Nets

In order to help readers easily understand the following definitions, we first provide some syntax used to write the expressions:

- C_{MS} denotes the multiset over set C. The notion of multiset is a generalization of the notion of set in which elements can appear more than once;
- $Type(v)$ denotes the type of variable v;
- $Var(E)$ denotes the set of variables in expression E;
- For each arc a, $a.p$ and $a.t$ denote place p and transition t connected by a;
- $M(p)$ is the value of the token in place p.

3.1.1 Overview of Coloured Petri Nets

CPNs [Kristensen 1998] are extended from Petri Nets with colour, time and expressions attached to arcs and transitions. A CPN is a directed bipartite graph, which consists of *places* (drawn as ellipses) and *transitions* (drawn as rectangles) connected by *directed arcs* (drawn as arrows). Each place holds a set of markers called *tokens*. Each token can carry both a data value called its colour and a timestamp. A token has the same type as its place.

Since transitions may consume and produce tokens, it is necessary to use *arc expressions* to determine the input-output relations. An incoming arc indicates that tokens may be removed by the transition from the corresponding place while an outgoing arc indicates that tokens may be added by the transition. Consequently, tokens are used to simulate control flows in a business workflow. They play a crucial role in providing an instrument to check the syntactic correctness of the workflow.

We next present a definition of CPNs, which is close to the one introduced in [Kristensen 1998]. This provides the foundation for the definitions introduced in the following section.

Definition 1 (Coloured Petri Nets)**.** A Coloured Petri net is formally defined as a 9-tuple $CPN = (\sum, P, T, A, N, C, G, E, I)$, where:

- $\sum$ is a finite set of non-empty types, called colour sets.
- P is a finite set of places.
- T is a finite set of transitions.
- A is a finite set of directed arcs such that: $P \cap T = P \cap A = T \cap A = \emptyset$.
- $N : A \rightarrow P \times T \cup T \times P$ is a node function. It is defined from A into $P \times T \cup T \times P$.

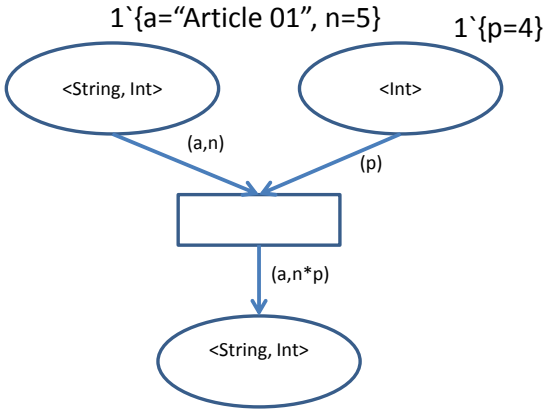

Figure 3.1: Example of a CPN

- $C : P \rightarrow \sum$ is a colour function. It is defined from P into $\sum$.
- $G : T \rightarrow expression$ is a guard function. It is defined from T into expressions such that:

$$\forall t \in T : [Type(G(t)) = Bool \wedge Type(Var(G(t))) \subseteq \sum]$$

- $E : A \rightarrow expression$ is an arc expression function. It is defined from A into expressions such that:

$$\forall a \in A : \left[Type(E(a)) = C(p(a))_{MS} \wedge Type(Var(E(a))) \subseteq \sum\right]$$

  where $p(a)$ is the place of $N(a)$.

- $I : P \rightarrow expression$ is an initialization function. It is defined from P into closed expressions such that:

$$\forall p \in P : [Type(I(p)) = C(p)_{MS}]$$

Figure 3.1 depicts an example of a CPN. This CPN has three places and one transition. Two places have a type $String \times Int$ and one has a type Int. When the transition fires, it consumes two tokens from its input places and produces one token to its output place.

Why is CPN chosen for our work?

There are many benefits to using CPNs as a workflow language, such as:

- CPNs have very well-defined semantics. They have been developed into a full-fledged language for the design, specification, simulation, validation and implementation of large-scale software systems;

- CPNs have a graphical representation. Their notation is similar to existing workflow languages;
- Since CPNs support different types of data (i.e., colours) and the use of global variables, it is easy to adapt CPNs to define Object-Oriented languages;
- The expressiveness of state and also behavioural changes are allowed in CPNs simultaneously.
- CPNs provide hierarchical descriptions. They offer interactive simulations where the CPN diagram can present directly the results;
- CPNs are executable and allow for different types of analysis, such as state-space analysis and invariants [Pesic 2007];
- CPNs have computer tools, named CPN Tools [The AIS group, Eindhoven University of Technology 2013], which support their drawing, simulation and formal analysis.

3.1.2 Coloured Petri Net-based Process Models

To take advantage of using CPNs, we introduce here a formal definition of CPN-based process models used to transform a business process into a control flow-based business workflow template.

Definition 2 (CPN-based process model)**.** A CPN-based process model, $\mathcal{PM}$, is formally defined as a 8-tuple $\mathcal{PM} = (\sum, P, T, A, C, G, E, I)$, where:

- $\sum$ is a finite set of non-empty types.
- $P = P_{in} \cup P_{out}$ is a non-empty finite set of places. P_{in} and P_{out} denote the input and output states of the activity nodes in a process model, respectively.
  - Place $s \in P_{in}$ is the start point in a process model. It is the input place of transition $t_{start} \in T_{act}$ and has no entering arc. In a process model, there is only one start point.
  - Place $e \in P_{out}$ is the end point in a process model. It is the output place of transition $t_{end} \in T_{act}$ and has no leaving arc. In a process model, there is only one end point.
  - Place $p \in P \backslash \{s, e\}$ has one leaving arc and one entering arc.

  The number of tokens in place p: $\forall p \in P : [w(p) = 0] or [w(p) = 1]$.
- $T = T_{act} \cup T_{ctrl}$ is a non-empty finite set of transitions.

  - T_{act} is a non-empty finite set of activity nodes. Each activity node has one entering arc and one leaving arc.
  - T_{ctrl} is a finite set of control nodes. A control node connects the output states of activity nodes with the input states of other activity nodes.

- $A \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs connecting input places to transitions or transitions to output places.
- $C : P \rightarrow \sum$ is a colour function. It is defined from P into $\sum$.
- $G : T \rightarrow expression$ is a guard function associating an operation with a transition.
- $E : A \rightarrow expression$ is an arc expression function. It is defined from A into $expression$ such that:

$$\forall a \in A : \left[Type(E(a)) = C(a.p) \wedge Type(Var(E(a))) \subseteq \sum\right]$$

- $I : P \rightarrow expression$ is an initialization function. It is defined from P into closed expressions such that:

$$\forall p \in P : [Type(I(p)) = C(p)]$$

A CPN-based process model is null if it has no places, activity nodes or arcs.

Business process models generally contain standard building blocks, including $Sequence$, $And-split$, $And-join$, $Xor-split$ and $Xor-join$ as shown in Figure 3.2. It is worth noting that the two building blocks, $Or-split$ and $Or-join$, are not used in the workflow modelling standards [van der Aalst 1998] nor in our work (called control nodes). The reason is that an OR (i.e., $Or-split$ and $Or-join$) can be simulated by a combination of the two other building blocks (i.e., AND and XOR) although that makes workflows become more bulky.

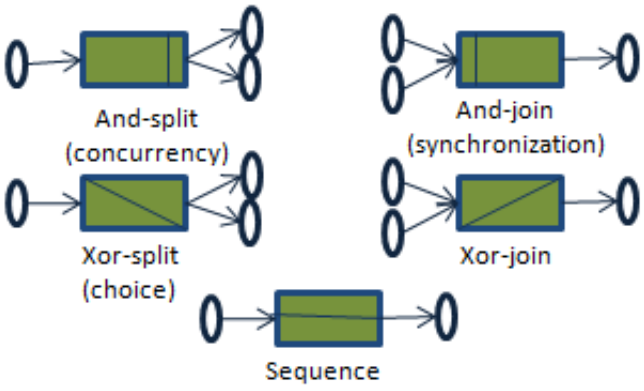

Figure 3.2: Five building blocks for modelling routing compositions

The five building blocks are used to model sequential, parallel, conditional and iterative routing.

- *Sequential*: The activities can be executed sequentially if the execution of one activity is followed by the next activity. The control node *Sequence* is thus necessary for this case.

- *Parallel*: Several activities can be executed at the same time or in any order. The two control nodes $And-split$ and $And-join$ are required to model this composition.

- *Conditional*: It means that there is a choice between two or more alternatives. The two control nodes $Xor-split$ and $Xor-join$ are used to model the choice.

- *Iterative*: A composition in which several activities are executed iteratively until a given condition is satisfied.

A routing composition is defined by a mapping between the outputs and the inputs of activity nodes via control nodes. Consequently, each composition comprises at least two activity nodes, one control node, three places and six directed arcs in total. We can decompose every business process model into exactly one set of routing compositions. Subsequently, we present the definitions of the components involved in routing compositions.

Definition 3 (AF (Activity Function))**.** *AF* describes an operation in an activity node and is defined as a 8-tuple:
$NF = (\sum, P, T, A, C, G, E, I)$ where:

- $\sum$ is a finite set of non-empty types.

- $P = P_{in} \cup P_{out}$ is a finite set of places defining the input and output states of the AF.

  P_{in} and P_{out} are the set of input and output places respectively where: $P = P_{in} \cup P_{out}$; $P_{in} \cap P_{out} = \emptyset$; $P_{in} = \{p_{in}\}$; $P_{out} = \{p_{out}\}$.

- T is a finite set of transitions denoted the behaviour of the AF.

  $T = \{t\}$ where transition t is an activity node containing the operation to be executed.

- $A \subseteq (P \times \{t\}) \cup (\{t\} \times P)$ is a set of directed arcs connecting input places to transitions or transitions to output places.

- $C : P \to \sum$ is a colour function associating a type to each place. It is defined from P into $\sum$.

- $G : \{t\} \to expression$ is a guard function associating an operation to transition t. It is defined from G into *expression* where:

$$Type(G(t)) = Type(Var(G(t))) \wedge C(p_{out}) \subseteq \sum$$

- $E : A \rightarrow expression$ is an arc expression function. It is defined from A into $expression$ where:

$$\forall a \in A : E(a) = \begin{cases} M(a.p) & \text{if } a.p \in P_{in} \\ G(a.t) & \text{otherwise} \end{cases}$$

- $I : \{p_{in}\} \rightarrow expression$ is an initialization function associating initial values to the input place.

Definition 4 (Sequence operator)**.** Sequence operator maps the output place of an AF to the input place of another AF. It is defined as 8-tuple: $SequenceO = (\sum, P, T, A, C, G, E, I)$, where:

- $\sum$ is a finite set of non-empty types.
- P is set of places defining the input and output states of the sequence operator.

  $P = P_{in} \cup P_{out}$; $P_{in} \cap P_{out} = \emptyset$ where $P_{in} = \{p_{in}\}$ and $P_{out} = \{p_{out}\}$.
- T is a finite set of transitions.

  $T = \{t\}$ where transition t is a control node containing the sequence operator.
- $A = (\{p_{in}\} \times \{t\}) \cup (\{t\} \times \{p_{out}\}) = \{a_{in}, a_{out}\}$ is a set of directed arcs connecting input places to transitions or transitions to output places.
- $C : P \rightarrow \sum$ is a colour function associating a type to each place where: $C(p_{in}) = C(p_{out})$.
- $G : \{t\} \rightarrow expression$ is a guard function associating an operation to transition t where: $Type(G(t)) = C(p_{out})$
- $E : A \rightarrow expression$ is an arc expression function. It is defined from A into $expression$ where:

$$\forall a \in A : E(a) = \begin{cases} M(a.p) & \text{if } a.p = p_{in} \\ G(a.t) & \text{otherwise} \end{cases}$$

- $I : \{p_{out}\} \rightarrow expression$ is an initialization function associating initial values to p_{out}.

Definition 5 (And-split operator)**.** And-split operator indicates that multiple threads are generated. These threads can be executed in parallel or in any order. It is defined as a 8-tuple:
$AndsplitO = (\sum, P, T, A, C, G, E, I)$ where:

- $\sum$ is a finite set of non-empty types.
- P is a finite set of places defining the input and output states of the And-split operator.

  $P = P_{in} \cup P_{out}$; $P_{in} \cap P_{out} = \emptyset$ where $P_{in} = \{p_{in}\}$ and $P_{out} = \{p_{out1}, p_{out2}, \ldots, p_{outM}\}$.

- T is a finite set of transitions.

  $T = \{t\}$ where transition t is a control node containing the And-split operator.

- $A = (\{p_{in}\} \times \{t\}) \cup (\{t\} \times P_{out}) = \{a_{in}, a_{out1}, a_{out2}, \ldots, a_{outM}\}$

  is a set of directed arcs connecting input places to transitions or transitions to output places.

- $C : P \to \sum$ is a colour function associating a type to each place where:

$$C(p_{in}) = C(p_{out1}) \wedge C(p_{out2}) \wedge \ldots \wedge C(p_{outM})$$

- $G : \{t\} \to expression$ is a guard function associating an operation to transition t where: $Type(G(t)) = C(p_{in})$.

- $E : A \to expression$ is an arc expression function where $Expr$ is a set of expressions. It is defined from A into $expression$ where:

$$\forall a \in A : E(a) = \begin{cases} M(a.p) & \text{if } a.p = p_{in} \\ G(a.t) & \text{otherwise} \end{cases}$$

- $I : P_{out} \to expression$ is an initialization function associating initial values to output places.

Definition 6 (And-join operator)**.** And-join operator indicates that there is a convergence with synchronization of multiple parallel threads. It is defined as a 8-tuple: $AndjoinO = (\sum, P, T, A, C, G, E, I)$, where:

- $\sum$ is a finite set of non-empty types.

- P is a finite set of places defining the input and output states of the And-join operator.

  $P = P_{in} \cup P_{out}$; $P_{in} \cap P_{out} = \emptyset$ where $P_{in} = \{p_{in1}, p_{in2}, \ldots, p_{inN}\}$ and $P_{out} = \{p_{out}\}$.

- T is a finite set of transitions.

  $T = \{t\}$ where transition t is a control node containing the And-join operator.

- $A = (P_{in} \times \{t\}) \cup (\{t\} \times \{p_{out}\}) = \{a_{in1}, a_{in2}, \ldots, a_{inN}, a_{out}\}$

  is a set of directed arcs connecting input places to transitions or transitions to output places.

- $C : P \to \sum$ is a colour function associating a type to each place where:

$$C(p_{out}) = C(p_{in1}) \wedge C(p_{in2}) \wedge \ldots \wedge C(p_{inN})$$

- $G : \{t\} \to expression$ is a guard function associating an operation to transition t where: $Type(G(t)) = C(p_{out}) \subseteq \sum$

- $E : A \rightarrow expression$ is an arc expression function where $Expr$ is a set of expressions. It is defined from A into $Expr$ where:

$$\forall a \in A : E(a) = \begin{cases} G(a.t) & \text{if } a.p = p_{out} \\ M(a.p) & \text{otherwise} \end{cases}$$

- $I : \{p_{out}\} \rightarrow expression$ is an initialization function associating initial values to the output place.

Definition 7 (Xor-split operator)**.** Xor-split operator indicates that only one of multiple threads is to be executed. It is defined as a 8-tuple:

$$XorsplitO = (\sum, P, T, A, C, G, E, I)$$

The Xor-split operator is defined similarly to the And-split operator except for the two functions G and E. We define these functions for $XorsplitO$ as follows:

- $G : \{t\} \rightarrow expression$ is a guard function where:

$$Type(G(t)) = Bool \wedge Type(Var(G(t))) \wedge C(p_{in}) \subseteq \sum$$

- $E : A \rightarrow expression$ is an arc expression function where:

  $\forall a \in A$: if $a.p = p_{in}$: $E(a) = M(a.p)$ else: Either $E(a) = G(a.t)$ or $E(a)$ is empty.

Definition 8 (Xor-join operator)**.** Xor-join operator indicates that whenever any one of multiple activities is executed, it causes the following activity to be executed. The operator is defined as a 8-tuple:

$$XorjoinO = (\sum, P, T, A, C, G, E, I)$$

The Xor-join operator is defined similarly to the And-join operator except for the two functions G and E. We define these functions for $XorjoinO$ as follows:

- $G : \{t\} \rightarrow expression$ is a guard function where:

$$Type(G(t)) = Bool \wedge Type(Var(G(t))) \wedge C(p_{out}) \subseteq \sum$$

- $E : A \rightarrow expression$ is an arc expression function where:

  $\forall a \in A$: if $a.p = p_{out}$: $E(a) = G(a.t)$ else: Either $E(a) = M(a.p)$ or $E(a)$ is empty.

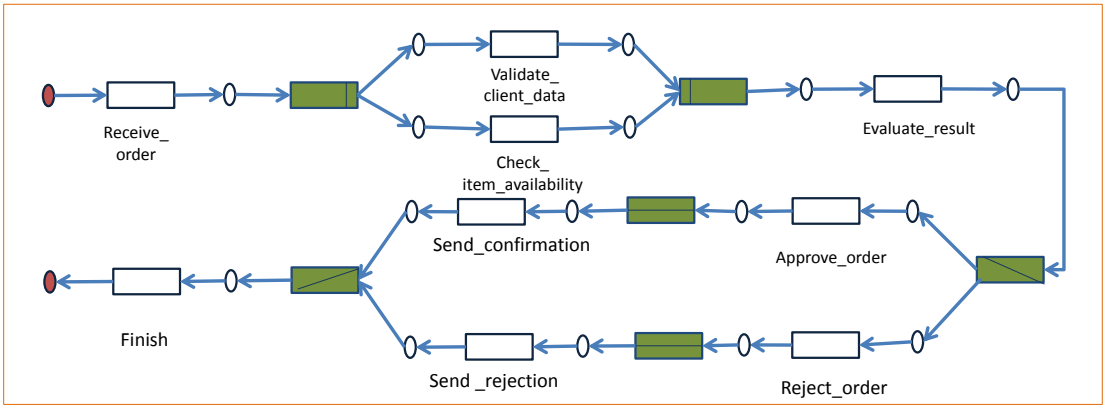

Figure 3.3: Order processing template modelled with CPNs

3.1.3 A simple Order Process Example

Example 3.1.1. In Figure 3.3, we represent the *Order processing* template, which is introduced in Section 1.2.1.1. To connect two activity nodes, we use one control node. $And-split$ and $And-join$ are used to connect a group of tasks executed in parallel, for example $Validate\_client\_data$ and $Check\_item\_availability$. $Xor-split$ and $Xor-join$ are used to connect a group of alternative tasks. And control nodes are used to connect tasks executed in sequence.

Although CPNs have been widely studied and successfully applied in modelling workflows and workflow systems, the lack of semantic representation of CPN components can make business processes modelled with CPNs (i.e., business workflows) difficult to interoperate, share and reuse. Besides, an ontology with its components, which provides machine-readable definitions of concepts, can play a pivotal role in representing semantically rich workflow definitions. Once workflow definitions are stored as semantically enriched workflow templates, IT experts can easily develop their appropriate software systems from the workflow templates. In the upcoming section, we will present the definition of semantic metadata for business workflow templates. The main purpose is to facilitate business workflow templates to be shared and reused among process-implementing software components.

3.2 An Ontology for Coloured Petri Nets-based Business Workflow Templates

3.2.1 Representation of Coloured Petri Net with OWL DL Ontology

Our CPN ontology developed to represent Coloured Petri Nets with OWL DL, is first proposed in [Nguyen 2014c]. Each element of CPNs is translated concisely into a corresponding OWL concept. Figure 3.4 depicts the core concepts of our CPN

ontology. The CPN ontology is described based on DL syntax (summarized in Table 3.1) and the axioms (summarized in Table 3.2) supported by OWL.

Table 3.1: OWL constructors

Constructor	DL syntax
$intersectionOf$	$C_1 \sqcap \ldots \sqcap C_n$
$unionOf$	$C_1 \sqcup \ldots \sqcup C_n$
$complementOf$	$\neg C$
$oneOf$	$\{x_1 \ldots x_n\}$
$allValuesFrom$	$\forall P.C$
$someValuesFrom$	$\exists r.C$
$hasValue$	$\exists r.\{x\}$
$minCardinality$	$(\geqslant nr)$
$maxCardinality$	$(\leqslant nr)$
$inverseOf$	r^-

Table 3.2: OWL axioms

Axiom	DL syntax
$subClassOf$	$C_1 \sqsubseteq C_2$
$equivalentClass$	$C_1 \equiv C_2$
$subPropertyOf$	$P_1 \sqsubseteq P_2$
$equivalentProperty$	$P_1 \equiv P_2$
$disjointWith$	$C_1 \sqsubseteq \neg C_2$
$sameAs$	$\{x_1\} \equiv \{x_2\}$
$differentFrom$	$\{x_1\} \sqsubseteq \neg\{x_2\}$
$TransitiveProperty$	P transitive role
$FunctionalProperty$	$\top \sqsubseteq (\leqslant 1P)$
$InverseFunctionalProperty$	$\top \sqsubseteq (\leqslant 1P^-)$
$SymmetricProperty$	$P \equiv P^-$

The meaning of the main elements in the CPN ontology is described as follows:

- The concept $CPNOnt$ is defined for all possible PMs (cf. Definition 2). This concept can be glossed as 'The class $CPNOnt$ is defined as the intersection of: (i) any class having at least one property $hasPlace$ whose value is restricted to the class $Place$ and; (ii) any class having at least one property $hasTransition$

$$
\begin{aligned}
&CPNOnt \equiv \geq 1 hasTrans.Transition \sqcap \geq 1 hasPlace.Place \\
&\quad \sqcap \geq 1 hasArc.(InputArc \sqcup OutputArc) \\
&Place \equiv connectsTrans.Transition \sqcap \leq 1 hasMarking.Token \\
&Transition \equiv connectsPlace.Place \sqcap = 1 hasGuardFunction.GuardFunction \\
&InputArc \equiv \geq 1 hasExpresion.Delete \sqcap \exists hasPlace.Place \\
&OutputArc \equiv \geq 1 hasExpression.Insert \sqcap \exists hasTrans.Transition \\
&Delete \equiv \forall hasAttribute.Attribute \\
&Insert \equiv \exists hasAttribute.Attribute \\
&GuardFunction \equiv \geq 1 hasAttribute.Attribute \sqcap = 1 hasActivity.ActNode \\
&\quad \sqcup = 1 hasControl.CtrlNode \\
&Token \equiv \geq 1 hasAttribute.Attribute \\
&Attribute \equiv \leq 1 valueAtt.Value \\
&CtrlNode \equiv \leq 1 valueAtt.Value \\
&ActNode \equiv = 1 valueAtt.Value \\
&Value \equiv valueRef.Value
\end{aligned}
$$

Figure 3.4: CPN ontology expressed in a description logic

whose value is restricted to the class *Transition* and; (iii) any class having at least one property *hasArc* whose value is either restricted to the class *InputArc* or the class *OutputArc*'.

- The concept *Place* is defined for all places of P. We consider the case in which one place contains at most one token at one time. Therefore, this concept can be glossed as 'The class *Place* is defined as the intersection of: (i) any class having at least one property *connectsTrans* whose value is equal to the class *Transition* and; (ii) any class having at most one property *hasMarking* whose value is restricted to the class *Token*'.

- The concept *Transition* is defined for all transitions of T. This concept can be glossed as 'The class *Transition* is defined as the intersection of: (i) any class having at least one property *connectsPlace* whose value is equal to the class *Place* and; (ii) any class having one property *hasGuardFunction* whose value is restricted to the class *GuardFunction*'.

- The concept *InputArc* is defined for all directed arcs from places to transitions in A. This concept can be glossed as 'The class *InputArc* is defined as the intersection of: (i) any class having at least one property *hasExpression* whose value is restricted to the class *Delete* and; (ii) any class having at least one property *hasPlace* whose value is restricted to the class *Place*'.

- The concept *OutputArc* is defined for all directed arcs from transitions to places in A. This concept can be glossed as 'The class *OutputArc* is defined as the intersection of: (i) any class having at least one property *hasExpression*

whose value is restricted to the class *Insert* and; (ii) any class having at least one property *hasTrans* whose value is restricted to the class *Transition*'.

- The concept *Delete* is defined for all expressions in input arcs. This concept can be glossed as 'The class *Delete* is defined as any class having all of properties *hasAttribute* whose values are equal to the class *Attribute*'.
- The concept *Insert* is defined for all expressions in output arcs. This concept can be glossed as 'The class *Insert* is defined as any class having at least one property *hasAttribute* whose value is restricted to the class *Attribute*'.
- The concept *GuardFunction* is defined for all transition expressions. This concept can be glossed as 'The class *GuardFunction* is defined as the intersection of: (i) any class having at least one property *hasAttribute* whose value is restricted to the class *Token* and; either any class having one property *hasActivity* whose value is restricted to the class *ActNode* or any class having one property *hasControl* whose value is restricted to the class *CrtNode*'.
- The concept *Token* is defined for all tokens in places. This concept can be glossed as 'The class *Token* is defined as any class having at least one property *hasAttribute* whose value is restricted to the class *Attribute*'.
- The concept *Attribute* is defined for all attributes defined for the individuals. This concept can be glossed as 'The class *Attribute* is defined as any class having at least one property *value* whose value is restricted to the class $Value$'.
- The concept *AtcNode* is defined for occurrence operation in activity nodes. This concept can be glossed as 'The class *AtcNode* is defined as any class having one property *value* whose value is restricted to the class $Value$'.
- The concepts *CtrNode* is defined for the occurrence condition in control nodes. This concept can be glossed as 'The class *CtrlNode* is defined as any class having at most one property *value* whose value is restricted to the class $Value$'.
- The concept $Value$ is defined for all subsets of $I_1 \times \times I_2 \times \ldots \times I_n$ where I_i is a set of individuals. This concept can be glossed as 'The class $Value$ is defined as any class having at least one property $valueRef$ whose value is equal to the class $Value$'.

3.2.2 Realization

We rely on OWL DL and use Protégé[1], an OWL editor, to develop the CPN ontology.

First of all, it is necessary to note that two OWL class identifiers, named $owl : Thing$ and $owl : Nothing$, are particularly predefined. The class extension of $owl : Thing$ is the set of all OWL individuals. The class extension of $owl : Nothing$ is the empty set. As a result, each user-defined class is absolutely a subclass of $owl : Thing$.

[1] http://protege.stanford.edu/

Besides, the following types of properties in the Web Ontology Language (OWL) are used to build an ontology:

- Object properties used to link an individual to another individual;
- Data properties used to link an individual to an RDF literal or an XML Schema data type;
- Domains and ranges indicate that properties link individuals from one domain to individuals from another domain;
- Datatypes: There are three types of data range specifications in OWL, including a RDF datatype, the RDFS class $rdfs : Literal$ and an enumerated datatype;
- Restriction types: They are divided into three main categories, such as Quantifier Restrictions ($allValuesFrom$, $someValuesFrom$), Cardinality Restrictions ($minCardinality$, $maxCardinality$, $cardinality$, $minQualifiedCardinality$, $maxQualifiedCardinality$, $qualifiedCardinality$) and $hasValue$ Restrictions. These types are used to specify the restriction of individuals that belong to a class.

In the following we describe some axioms created for the CPN ontology. The full description of the CPN ontology can be found in Appendix B.

With regard to classes, we start by creating the class axiom for the class $CPNOnt$ containing the properties $hasPlace$, $hasTrans$ and $hasArc$. OWL provides the syntactic form $EquivalentClasses(C_1 \dots C_n)$ to express synonyms. Therefore, the class axiom is created as follows:

$EquivalentClasses(CPNOnt\ intersectionOf(restriction(hasPlace\ allValuesFrom(Place)\ minQualifiedCardinality(1))\ restriction(hasTrans\ allValuesFrom(Transition)\ minQualifiedCardinality(1))\ restriction(hasArc\ allValuesFrom(unionOf(InputArc\ OutputArc))\ minQualifiedCardinality\ (1))))$;

In order to define a class as a subclass of another one, an axiom written in the syntactic form $SubClassOf(C_1, C_2)$ is used. For example, the class $Place$ is a sub-class of the class $CPNOnt$, the class axiom is created as follows:

$SubClassOf(Place\ CPNOnt)$;

If two classes are disjoint, an individual cannot be an instance of more than one of the two classes. For example, the class $Place$ and the class $Transition$ are mutually disjoint. This disjointness can be expressed using the syntactic form $DisjointClasses(C_1\ C_2)$ as follows:

$DisjointClasses(Place\ Transition)$;

With regard to properties, let us consider the property $connectsTrans$ (as depicted in Figure 3.4): The domain of this property is a union of the class $Place$ with the class $InputArc$. The range of this property is a union of the class $Transition$ with the class $OutputArc$. Therefore, we use the syntactic form

Figure 3.5: Property *connectsTrans* and property *connectsPlace*

$SubClassOf(C_1, C_2)$ to express this coherence. We create the property axiom for *connecsTrans* (Figure 3.5) as follows:

$ObjectProperty(connectsTrans\ domain(unionOf(Place\ InputArc))$
$range(unionOf(Transition\ OutputArc)))$;

We next introduce the modelling of *Individuals*, which are the third OWL element besides *Classes* and *Properties*. It is important to underline that individuals or instances are chosen by the modeller and depend on the modelling objective. For example, Figure 3.6 shows the mapping of the transition *Receive_order*, which is depicted in Figure 3.3, to the classes and properties of the CPN ontology.

```
<Transition rdf:ID="Receive_order">
 <connectsPlace rdf:resource="#Receive_order_out"/>
 <hasGuardFunction>
    <GuardFunction rdf:ID="Receive_order_guard">
       <hasAttribute rdf:resource="#Receive_order_attribute_1"/>
       ...
       <hasActivity ActNode rdf:resource="Receive_order_activity"/>
    </GuardFunction>
 </hasGuardFunction>
</Transition>
```

Figure 3.6: Mapping Individuals to Classes and Properties of the CPN ontology

We have introduced the CPN ontology represented in OWL DL. For the development of CBWTs (i.e., business processes modelled with CPNs), in the next section, we will introduce manipulation operations on their elements. We will also present the corresponding manipulation statements written in the SPARQL language used to store concrete CBWTs in RDF format.

3.3 Manipulation of Business Workflow Templates

In order to develop a business workflow template, the following basic types of operations on its elements are required:

(i) Inserting new elements (i.e., places, transitions or arcs, etc.) into a workflow template;

(ii) Deleting existing elements from a workflow template;

(iii) Updating existing elements for adapting to a workflow template;

(iv) Editing the order of existing elements in a workflow template.

More complex operations can then be developed based upon these basic operations. For example, two separate CBWTs, which represent two business workflow templates, can be merged into a single CBWT by inserting all places, transitions and arcs from one template to the other. A new arc is also inserted in order to link these CBWTs.

We next define the operations by the corresponding pseudo codes. We also introduce the SPARQL statements being suitable to the operations, which enable CBWTs to be stored in RDF format.

(i) Inserting new elements into a workflow template.

$INSERT\ ELEMENT\ \{e_1, e_2, \ldots, e_n\}\ INTO\ PROCESS\ wf$

$[WHERE\ cond_1, cond_2, \ldots, cond_m];\ (n \geq 1, m \geq 1)$

This statement means that 'elements $e_1, e_2, \ldots, e_n$, each of which has been created, are inserted into a workflow template named wf. The conditions $cond_1, cond_2, \ldots, cond_m$ in the $WHERE$ clause (if any) specify how to insert these new elements into the workflow template wf'.

The INSERT DATA statement or the INSERT WHERE statement in the SPARQL query language can be used to insert new elements on workflow templates into RDF files. As an example, Figure 3.7 illustrates a new place, which contains a token and is connected to a transition, being inserted into a workflow template[2].

```
INSERT DATA{
 k:NameOfPlace a h:Place;
    h:hasMarking k:NameOfToken.
 k:NameOfWF h:hasPlace k:NameOfPlace.}
```

Figure 3.7: An example of the INSERT DATA statement

(ii) Deleting existing elements from a workflow template.

$DELETE\ ELEMENT\ \{e_1, e_2, \ldots, e_n\}\ FROM\ PROCESS\ wf;\ (n \geq 1)$

This statement means that existing elements $e_1, e_2, \ldots, e_n$ are completely deleted from a workflow template named wf.

The DELETE DATA statement or the DELETE WHERE statement in the SPARQL query language can be used to delete existing elements from the

[2]Two prefixes are assumed as:
$PREFIX h :< http://www.semanticweb.org/CPNWF\# >$
$PREFIX\ k :\ < http://WFTemplate\# >$

RDF file format. As an example, Figure 3.8 illustrates an existing place being deleted from a workflow template.

```
DELETE  WHERE{
 k:NameOfPlace ?pr1 ?o.
 ?s ?pr2 k:NameOfPlace. }
```

Figure 3.8: An example of the DELETE WHERE statement

(iii) Updating existing elements for adapting to a workflow template.

$UPDATE\ ELEMENT\ \{e_1, e_2, \ldots, e_n\}\ ON\ PROCESS\ wf$

$[WHERE\ cond_1, cond_2, \ldots, cond_m];\ (n \geq 1, m \geq 1)$

This statement means that elements $e_1, e_2, \ldots, e_n$ in a workflow template named wf, each of which has been created, are updated. The conditions $cond_1, cond_2, \ldots,\ cond_m$ in the $WHERE$ clause (if any) specify how to update these elements in the template wf.

In this case, some statements in the SPARQL query language can be used, such as the INSERT DATA statement, the INSERT WHERE statement or the DELETE INSERT WHERE statement. As an example, in Figure 3.9, an existing place in a workflow template changes its token.

```
DELETE
 {k:NameOfPlace h:hasMarking k:NameToken1}
INSERT
 {k:NameOfPlace h:hasMarking k:NameOfToken2}
WHERE{
 k:NameOfWF h:hasPlace k:NameOfPlace }
```

Figure 3.9: An example of the DELETE INSERT WHERE statement

(iv) Editing the order of existing elements in a workflow template.

$MODIFY\ PROCESS\ wf$

$WHERE\ cond_1, cond_2, \ldots, cond_n$

$REPLACE\ condR_1, condR_2, \ldots, condR_m;\ (n \geq 1, m \geq 1)$

This statement is used to edit ordering relationships in a workflow template. No element inserted, deleted or updated in the template.

The DELETE INSERT DATA statement is used to edit the order of existing elements in the RDF file format. As an example, in Figure 3.10, an existing place in a workflow template changes its connection from a transition to another transition.

```
DELETE
 {k:NameOfPlace h:connectsTrans k:NameOfTrans1}
INSERT
 {k:NameOfPlace h:connectsTrans k:NameOfTrans2}
WHERE{
 k:NameOfWF h:hasTrans k:NameOfTransition1;
               h:hasTrans k:NameOfTransition2;
               h:hasPlace k:NameOfPlace. }
```

Figure 3.10: An example of editing ordering relationships

3.4 Related Work

To the best of our knowledge, the ontology-based approach for modelling workflow templates is not a new idea. There has been some work to build workflow ontologies, such as [Greco 2004, Koschmider 2005, Gasevic 2006, Sebastian 2008, Zhang 2011] to support (semi-)automatic system collaboration and provide machine-readable definitions of concepts and interpretable format. Section 3.4.1 describes approaches focusing on combining workflows with ontologies while approaches focusing on combining Petri Nets with ontologies are described in Section 3.4.2.

3.4.1 On Combining Workflows with Ontologies

By analysing workflows for several active projects, the authors of [Sebastian 2008] describe a set of workflow properties. On this basis, they introduce an ontology to represent different aspects of workflows for collaborative ontology development. The ontology becomes a key component of the customizable workflow support in Protégé. However, this work refers to no existing process modelling languages. Therefore, to work with a workflow execution engine, it is necessary to map the top level of the ontology to the process-modelling language required by the workflow execution engine. In contrast to this work, we develop the CPN ontology to represent CPNs, a modelling language, with OWL DL.

O. Thomas and M. Fellmann in [Thomas 2009a] address a problem of semantic process modelling. They introduce an extension of process modelling languages to represent the semantics of process model element labels. As shown in Figure 3.11, the labels formulated in natural language can be represented by terms from a formal ontology. The benefits of this formalization of model element-related semantics are that it eliminates the scope of interpretation related to the use of natural language and it supports semantic validation. Furthermore, this work provides a very useful inspiration for our work, but it does not discuss how to formulate semantic constraints and also does not mention the control-flow perspective in process models as does our approach.

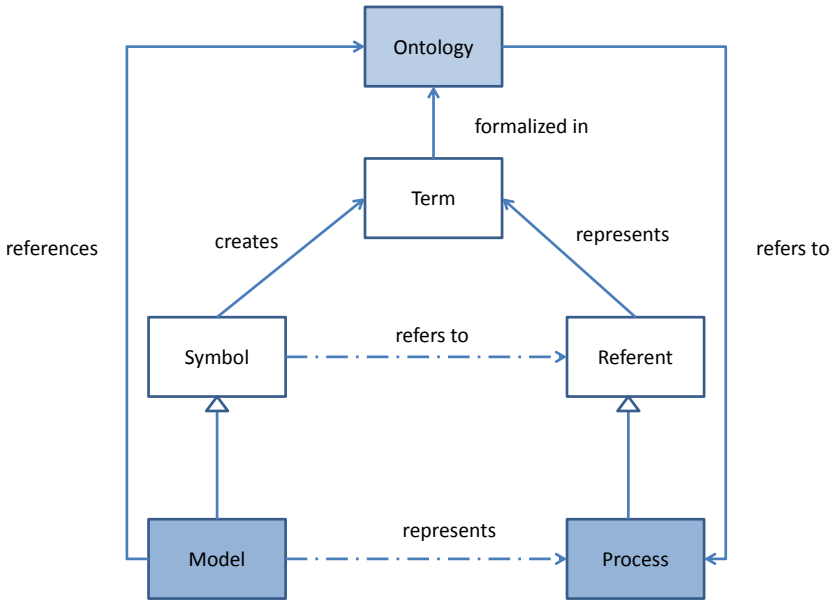

Figure 3.11: Extended semiotic triangle "model", "ontology" and "process" for the semantic process modelling [Thomas 2009a]

3.4.2 On Combining Petri Nets/High-Level Petri Nets with Ontologies

The authors of [Gasevic 2006] propose a Petri Net ontology[3], which is defined for the semantic description of PN concepts and their relationships. The purpose of this work is to enable sharing PNs on the Semantic Web and transform a specific XML-based PN format into OWL. A PN UML model is used as the starting point to implement the ontology. The resulting PN models are then represented using Semantic Web languages, RDF(S) and OWL. So far, the Petri net ontology is also extended for: P/T nets, Time Petri nets[4] [Murata 1989], and Upgraded Petri nets[5] [Strbac 2013]. With the development of the CPN ontology, our work aims to provide the shareability and reusability of CPN-based business workflow templates not only for the Semantic Web, but also for business workflow systems.

[Koschmider 2005] also introduces an ontology to describe business processes modelled with Petri Nets (PNs). The ontology is aimed to facilitate the semantic interconnectivity of semantic business processes that enables semantic information exchange. Furthermore, the translation of traditional PNs into OWL is used to semantically align business process models (see [Brockmans 2006]) and automatically compute similarities between business process models (see [Ehrig 2007]) to support (semi-)automatic interconnectivity of business processes.

[3] http://www.sfu.ca/~dgasevic/projects/PNO/RDFS/PNO.rdfs
[4] http://www.sfu.ca/~dgasevic/projects/PNO/RDFS/TimePNO.rdfs
[5] http://www.sfu.ca/~dgasevic/projects/PNO/RDFS/UpgradedPNO.rdfs

Our CPN ontology, a representation of Coloured Petri Nets with OWL DL ontology, is very close to the one proposed by [Koschmider 2005], however, there are some differences. We focus on representing business workflow templates developed based on the ontology in a knowledge base, which is defined in order to share and reuse them.

3.5 Discussion and Conclusion

This chapter focused on representing control flow-based business workflow templates. We first presented a formal definition of CPN-based business models. We then defined the CPN ontology to represent CPNs with OWL DL. Each element of CPNs has been translated into a corresponding OWL concept. In addition, some of the axioms created for *Classes* and *Properties* in the CPN ontology have been presented. *Individuals*, the third OWL element, have been also considered. As a result, the combination of CPNs and ontologies provides not only semantically rich business process definitions but also machine-processable ones. Moreover, in order to model business processes, the basic types of manipulation operations on the elements of process models have been presented. Besides, the SPARQL statements, which correspond to the operations, have been indicated to develop or modify CBWTs encoded in RDF format. The results of this work were published in [Nguyen 2013, Nguyen 2015, Nguyen 2014c].

We know that the specification of a real-world business process is mostly manual and is thus prone to human error, resulting in a considerable number of failed projects. Therefore, to ensure the correctness of concrete CBWTs, we will implement SPARQL queries to detect shortcomings in concrete workflow templates at design phase in Chapter 5.

Chapter 4

Semantic Business Process Modelling

Contents

The verification of a business workflow generally covers the following aspects:

1. To check the syntactic correctness of a workflow based on the general properties.

2. To check that a workflow complies with a set of properties given by a formula.

In the previous chapter, a formal definition of CPN-based process models has been introduced. It is intended to support the syntactic verification of a business workflow template (Section 5.1). Therefore, in this chapter, we focus on a solution to modelling semantic business processes, which aims to support the semantic verification related to the above-mentioned second aspect (Section 5.2).

The main purpose of this chapter is to formally describe a semantic business workflow template by identifying a set of semantic constraints. We first give a formal definition of semantic constraints in form of a set of attributes. We then introduce an algorithm to check redundant and conflicting constraints. A formalized repository is thus constructed on the top of the set of well-checked[1] semantic constraints, from which a semantic business workflow template is developed. In addition, we introduce ECA-like rules to represent business level requirements. This allows for integrating requirements into a workflow template.

[1] A well-checked set of semantic constraints means that there are neither redundant constraints nor conflicting constraints

4.1 Formal Definition of Semantic Constraints

As mentioned previously, the verification of business workflows is an important step before executable workflow templates are deployed. The soundness verification concerning the control-flow perspective of process models is a necessary but not sufficient condition for correctness checks regarding the individual workflow activities and their semantics[2]. Hence, to ensure that a business workflow works as intended, their individual activities are needed to take into account - What are the meaning and relations between activities in a workflow? What do they actually execute during their performance? In fact, no information about this can be found in traditional workflows[3] except the naming of the activities. For simple applications in closed domains where the behavior of activities are understood in detail by involved persons and/or not overly complicated, naming of model activities may be sufficient. However, for more complicated applications, there is a strong demand for a powerful method to describe semantically rich activities and the relations between them. It is also useful to avoid issues which limit the use of workflows as a medium for communication or by different agents in a heterogeneous and distributed environment. For example, an activity in a workflow is referred to as "goods" whereas in another workflow, a further activity is referred to as "merchandise" and of course both of these activities represent the same object.

Indeed, the two questions stated above motivate us to design a semantic constraint specification language which allows modellers to construct semantic business process models. Semantic constraints are here used to represent various dependencies between activities of a business process, such as ordering relations and existence dependencies. Consequently, semantic constraints tackled in this thesis can be regarded as a subset of business rules.

Based on the analysis of the state-of-the-art concerning the division of semantic constraints, we classify semantic constraints into four basic types as follows:

1. **Mutual exclusion constraints** (*mExclusion*) express that the presence of an activity imposes the exclusion on another activity and therefore, the execution order between these activities is not specified;

2. **Choice constraints** (*choice*) express that only one of two activities must be executed and therefore, the execution order between these activities is not specified;

[2]Semantics refers to the study of meaning in language, which focuses on the relations between words, phrase, signs and symbols, what they represent and denote. Linguistic semantics is the study of meaning employed for comprehending human expression through language. In scientific disciplines, the scientific meaning often refers to the conception of linguistics. "In this discipline, semantics refers to the branch that deals with the meaning and significance of language resp. linguistic signs. In other words: the teaching of the meaning and the relations of signs for a certain object. If this is transferred to process modeling languages, the semantics of a process model can be understood as the relationship between the elements of a model (sign) and an existing or future operational business process (universe of discourse)" [Fellmann 2011].

[3]In general, traditional workflows focus on syntactical relationships between activities and their black box character.

3. **Dependency constraints** (*dependency*) express the presence of one activity (called the source activity) imposes that the other activity (called the target activity) must be included, but not conversely. These activities are executed dependently (i.e., the source activity is executed before or after the target activity).

4. **Coexistence constraints** (*coexistence*) express that two activities must be both executed or both excluded. These activities are executed concurrently or dependently (i.e., one is executed before or after the other).

Definition 9 (Semantic Constraint)**.** Let τ be a set of relevant activities[4] in the context of a specific business process. A 6-tuple $SC = (constraintType, appliedActivity, relatedActivity, order, description, [Equivalence])$ is called the semantic constraint definition, in which:

- $constraintType \in \{mExclusion, choice, dependency, coexistence\}$;
- $appliedActivity \in \tau$;
- $relatedActivity \in \tau$;
- $order \in \{before, after, concurrence, notSpecified\}$;
- *description* is used to describe a constraint;
- *Equivalence* is a set of activities which are equivalent to activity *appliedActivity*, $Equivalence \subset \tau$.

The first parameter *constraintType* denotes the type of a semantic constraint, it is *mExclusion* or *choice* or *dependency* or *coexistence*. Each value of *constraintType* refers to the relationship between the executions of the source activity expressed by the second parameter *appliedActivity* and the target activity expressed by the third parameter *relatedActivity*. The parameter *order* specifies the execution order between the source and target activity. The default value *notSpecified* is assigned to the constraints of the type *mExecution* or *choice*. The first four parameters are very important and obligatory when defining a semantic constraint. The parameter *description* is used to describe the constraint in a natural language[5]. And the last parameter *Equivalence*[6] is optional, which contains a set of activities (if any) equivalent to the source activity.

Example 4.1.1. Let us continue the example of the fOtD process described in Section 1.2. Consider the template *Payment*, which is presented in Section 1.2.1.3, a set of relevant semantic constraints is created as follows:

[4]The issue relative to naming activities will be discussed in Appendix C.

[5]In our case, English is used to describe semantic constraints.

[6]In general, in a constraint, each value in the set *Equivalence* is equivalent to the value of the parameter *appliedActivity*. With the implicit requirement relating to the naming of activities of a workflow template, if a name has been used for an activity in the parameter *appliedActivity* or the parameter *relatedActivity* will not appear as a value in the parameter *Equivalence* and vice versa to avoid confusion

```
sc1=(dependency, Get_payment_data, Provide_payment_methods, after,
‘‘after choosing one of provided payment methods, user must enter
payment data’’, {Get_payment_information})
sc2=(dependency, Process_check_or_cash, Get_payment_data, after,
‘‘paying by check or cash has to be checked and validated’’)
sc3=(dependency, Process_check_or_cash, Provide_payment_methods,
after, ‘‘processing check or cash is only executed after choosing a
payment method’’)
sc4=(dependency, Process_credit_card, Get_payment_data, after,
‘‘paying par credit card must be checked and validated’’)
sc5=(dependency, Process_check_or_cash, Get_payment_data, after,
‘‘paying by check or cash must be checked and valided’’)
sc6=(choice, Process_credit_card, Process_check_or_cash, notSpecified,
‘‘customers can only pay by credit card or check or cash’’)
```

4.2 Implicit, Redundant and Conflicting Semantic Constraints

4.2.1 Algebraic Properties of Semantic Constraints

Through the definition of semantic constraints, information about how to use activities and about the relations between those activities is captured. However, when defining a set of semantic constraints, it may occur implicit, redundant or conflicting semantic constraints. Two constraints can be combined together to constitute new constraints. This is demonstrated by the parameters *constraintType* and *order* in the definition of semantic constraints. As stated previously, the parameter *constraintType* expresses the semantic constraint's type and the parameter *order* indicates the execution order of a source activity and a target activity. In this section, we present the properties related to these properties in Table 4.1 and Table 4.2. The properties are used to infer implicit constraints (see Section 4.2.2) and create business workflow templates (see Section 4.3.2).

We use the notation: $activity_1$ ***order_value*** $activity_2$ to denote that $activity_1$ and $activity_2$ are involved in a (inferred) semantic constraint like $(constraintType, activity_1, activity_2, \textbf{\textit{order\_value}}, description, [Equivalent])$; and the notation: $activity_1$ ***constraint_type*** $activity_2$ to denote that $activity_1$ and $activity_2$ are involved in a (inferred) semantic constraint like $(\textbf{\textit{constraint\_type}}, activity_1, activity_2, order, description, [Equivalent])$.

In Table 4.1, we present the associative, transitive and commutative properties identified based on the parameter *constraintType* where a_1, a_2 and a_3 are activities. It is important to note that for each associative property in Table 4.1, the value of the parameter *order* in the dependency constraints must be the same.

Table 4.1: Algebraic properties identified based on the parameter *constraintType*

Name		Expression
Association	(1)	a_1 *dependency* a_3, a_2 *dependency* a_3, a_1 *coexistence* $a_2 \rightarrow$ (a_1 *coexistence* a_2) *dependency* a_3
	(2)	a_1 *dependency* a_3, a_2 *dependency* a_3, a_1 *mExclusion* $a_2 \rightarrow$ (a_1 *mExclusion* a_2) *dependency* a_3
	(3)	a_1 *dependency* a_3, a_2 *dependency* a_3, a_1 *choice* $a_2 \rightarrow$ (a_1 *choice* a_2) *dependency* a_3
	(4)	a_1 *dependency* a_2, a_1 *dependency* a_3, a_2 *coexistence* $a_3 \rightarrow$ a_1 *dependency* (a_2 *coexistence* a_3)
	(5)	a_1 *dependency* a_2, a_1 *coexistence* a_3, a_2 *coexistence* $a_3 \rightarrow$ (a_1 *dependency* a_2) *coexistence* a_3
	(6)	a_1 *dependency* a_2, a_1 *mExclusion* a_3, a_2 *mExclusion* $a_3 \rightarrow$ (a_1 *dependency* a_2) *mExclusion* a_3
	(7)	a_1 *dependency* a_2, a_1 *choice* a_3, a_2 *choice* $a_3 \rightarrow$ (a_1 *dependency* a_2) *choice* a_3
	(8)	a_1 *coexistence* a_2, a_1 *coexistence* a_3, a_2 *dependency* $a_3 \rightarrow$ a_1 *coexistence* (a_2 *dependency* a_3)
	(9)	a_1 *mExclusion* a_2, a_1 *mExclusion* a_3, a_2 *dependency* $a_3 \rightarrow$ a_1 *mExclusion* (a_2 *dependency* a_3)
	(10)	a_1 *choice* a_2, a_1 *choice* a_3, a_2 *dependency* $a_3 \rightarrow$ a_1 *choice* (a_2 *dependency* a_3)
Transitivity	(1)	a_1 *coexistence* a_2, a_2 *choice* $a_3 \rightarrow a_1$ *choice* a_3
	(2)	a_1 *coexistence* a_2, a_2 *mExclusion* $a_3 \rightarrow$ a_1 *mExclusion* a_3
Commutativity	(1)	a_1 *coexistence* $a_2 \Leftrightarrow a_1$ *coexistence* a_2
	(2)	a_1 *choice* $a_2 \Leftrightarrow a_1$ *choice* a_2
	(3)	a_1 *mExclusion* $a_2 \Leftrightarrow a_1$ *mExclusion* a_2

In order to easily prove the algebraic properties presented in Table 4.1, we express the execution of an activity as an integer programming formulation. Using function $exe(a_i)$ to indicate whether activity $a_i \in \tau$ must be executed or not. Each value of function $exe(a_i)$ is considered as a propositional variable that ranges over domain $D = \{0, 1\}$:

(i) $exe(a_i) = 0$ indicates that activity a_i must not be executed.

(ii) $exe(a_i) = 1$ indicates that activity a_i must be executed.

(iii) $exe(a_i) \leq exe(a_j)$ indicates that if activity a_i is executed, activity a_j must be executed, but not conversely. It corresponds to a semantic constraint of the type *denpendency*.

(iv) $exe(a_i) = exe(a_j)$ indicates that two activities a_i and a_j must both be executed or neither is executed. It corresponds to a semantic constraint of the type *coexistence*.

(v) $exe(a_i) + exe(a_j) \leq 1$ indicates that either the execution of two activities a_i and a_j are mutually exclusive or these activities are not executed at all. It corresponds to a semantic constraint of the type $mExclusion$.

(vi) $exe(a_i) + exe(a_j) = 1$ indicates that only one of two activities a_i and a_j is executed. It corresponds to a semantic constraint of the type *choice*.

Based on this expression, the proofs of the algebraic properties related to the parameter *constraitType* are given below.

4.2.1.1 Associative Property of the Parameter *constraintType*

(i) Proof of the associative property (1): Consider the following semantic constraints sc_1, sc_2 and sc_3 where:

- $sc_1 = (dependency, a_1, a_3, order_1, description_1, [activities\_are\_equivalent\_to\_Activity\_a_1])$
- $sc_2 = (dependency, a_2, a_3, order_1, description_2, [activities\_are\_equivalent\_to\_Activity\_a_2])$
- $sc_3 = (coexistence, a_1, a_2, order_3, description_3, [activities\_are\_equivalent\_to\_Activity\_a_1])$

In order to prove the associative property (1) of the parameter *constraitType* (i.e., (a_1 *coexistence* a_2) *dependency* a_3), we have to prove that $exe(a_1) = exe(a_2) \leq exe(a_3)$.

Proof. By using our expression of the execution of an activity and Definition 9, we get:

$$a_1 \ dependency \ a_3 \Rightarrow exe(a_1) \leq exe(a_3). \tag{4.1}$$

$$a_2 \ dependency \ a_3 \Rightarrow exe(a_2) \leq exe(a_3). \tag{4.2}$$

$$a_1 \ coexistence \ a_2 \Rightarrow exe(a_1) = exe(a_2). \tag{4.3}$$

By combining (4.1), (4.2) and (4.3), we get: $exe(a_1) = exe(a_2) \leq exe(a_3)$ □

(ii) Proof of the associative property (2): Consider the following semantic constraints sc_1, sc_2 and sc_3 where:

- $sc_1 = (dependency, a_1, a_3, order_1, description_1, [activities\_are\_equivalent\_to\_Activity\_a_1])$
- $sc_2 = (dependency, a_2, a_3, order_1, description_2, [activities\_are\_equivalent\_to\_Activity\_a_2])$
- $sc_3 = (mExclusion, a_1, a_2, order_3, description_3, [activities\_are\_equivalent\_to\_Activity\_a_1])$

In order to prove the associative property (1) of the parameter *constraitType* (i.e., (a_1 *mExclusion* a_2) *dependency* a_3), we have to prove that $exe(a_1) + exe(a_2) \leq exe(a_3)$.

Proof. By using our expression of the execution of an activity and Definition 9, we get:

$$a_1 \text{ dependency } a_3 \Rightarrow exe(a_1) \leq exe(a_3) \Rightarrow \left[\begin{array}{llllll} exe(a_1) & = & 0, & exe(a_3) & = & 0 \\ exe(a_1) & = & 0, & exe(a_3) & = & 1 \\ exe(a_1) & = & 1, & exe(a_3) & = & 1 \end{array}\right. \tag{4.4}$$

$$a_2 \text{ dependency } a_3 \Rightarrow exe(a_2) \leq exe(a_3) \Rightarrow \left[\begin{array}{llllll} exe(a_2) & = & 0, & exe(a_3) & = & 0 \\ exe(a_2) & = & 0, & exe(a_3) & = & 1 \\ exe(a_2) & = & 1, & exe(a_3) & = & 1 \end{array}\right. \tag{4.5}$$

$$a_1 \text{ mExclusion } a_2 \Rightarrow exe(a_1) + exe(a_2) \leq 1 \Rightarrow \left[\begin{array}{llllll} exe(a_1) & = & 0, & exe(a_2) & = & 0 \\ exe(a_1) & = & 0, & exe(a_2) & = & 1 \\ exe(a_1) & = & 1, & exe(a_2) & = & 0 \end{array}\right. \tag{4.6}$$

By combining (4.4), (4.5) and (4.6), we get:

$$\left[\begin{array}{llllll} exe(a_1) & = & 0, & exe(a_2) & = & 0, \quad exe(a_3) = 0 \\ exe(a_1) & = & 0, & exe(a_2) & = & 0, \quad exe(a_3) = 1 \\ exe(a_1) & = & 0, & exe(a_2) & = & 1, \quad exe(a_3) = 1 \\ exe(a_1) & = & 1, & exe(a_2) & = & 0, \quad exe(a_3) = 1 \end{array}\right. \Rightarrow exe(a_1) + exe(a_2) \leq exe(a_3) \tag{4.7}$$

□

(iii) Proof of the associative property (4): Consider the following semantic constraints sc_1, sc_2 and sc_3 where:

- $sc_1 = (dependency, a_1, a_2, order_1, description_1, [activities\_are\_equivalent\_to\_Activity\_a_1])$
- $sc_2 = (dependency, a_1, a_3, order_1, description_2, [activities\_are\_equivalent\_to\_Activity\_a_1])$
- $sc_3 = (coexistence, a_2, a_3, order_3, description_3, [activities\_are\_equivalent\_to\_Activity\_a_2])$

In order to prove the associative property (4) of the parameter *constraitType* (i.e., a_1 *dependency* (a_2 *coexistence* a_3)), we have to prove that $exe(a_1) \leq exe(a_2) = exe(a_3)$.

Proof. By using our expression of the execution of an activity and Definition 9, we get:

$$a_1 \; dependency \; a_2 \Rightarrow exe(a_1) \leq exe(a_2) \tag{4.8}$$

$$a_1 \; dependency \; a_3 \Rightarrow exe(a_1) \leq exe(a_3) \tag{4.9}$$

$$a_2 \; coexistence \; a_3 \Rightarrow exe(a_2) = exe(a_3) \tag{4.10}$$

By combining (4.8), (4.9) and (4.10), we get: $exe(a_1) \leq exe(a_2) = exe(a_3)$. □

The rest of associative properties can be proven in the similar way.

4.2.1.2 Transitive Property of the Parameter *constraintType*

(i) Proof of the transitive property (1):

Consider the following semantic constraints sc_1 and sc_2 where:

- $sc_1 = (coexistence, a_1, a_2, order_1, description_1, [activities\_are\_equivalent\_to\_Activity\_a_1])$
- $sc_2 = (choice, a_2, a_3, order_2, description_2, [activities\_are\_equivalent\_to\_Activity\_a_2])$

In order to prove the transitive property (2) of the parameter *constraitType* (i.e., a_1 *choice* a_3), we have to prove that $exe(a_1) + exe(a_3) = 1$.

Proof. By using our expression of the execution of an activity and Definition 9, we get:

$$a_1 \; coexistence \; a_2 \Rightarrow exe(a_1) = exe(a_2) \tag{4.11}$$

$$a_2 \; choice \; a_3 \Rightarrow exe(a_2) + exe(a_3) = 1 \tag{4.12}$$

By combining (4.11) and (4.12), we get: $exe(a_1) + exe(a_3) = 1$. □

(ii) Proof of the transitive property (2):

Consider the following semantic constraints sc_1 and sc_2 where:

- $sc_1 = (coexistence, a_1, a_2, order_1, description_1, [activities\_are\_equivalent\_to\_Activity\_a_1])$
- $sc_2 = (mExclusion, a_2, a_3, order_2, description_2, [activities\_are\_equivalent\_to\_Activity\_a_2])$

In order to prove the transitive property (2) of the parameter *constraitType* (i.e., a_1 *mExclusion* a_3), we have to prove that $exe(a_1) + exe(a_3) \leq 1$.

Proof. By using our expression of the execution of an activity and Definition 9, we get:

$$a_1 \; coexistence \; a_2 \Rightarrow exe(a_1) = exe(a_2) \tag{4.13}$$

$$a_2 \; mExclusion \; a_3 \Rightarrow exe(a_2) + exe(a_3) \leq 1 \tag{4.14}$$

By combining (4.13) and (4.14), we get: $exe(a_1) + exe(a_3) \leq 1$. □

4.2.1.3 Commutative Property of the Parameter *constraintType*

Consider the following semantic constraints sc_1 and sc_2 where:

- $sc_1 = (constraintType_1, a_1, a_2, order_1, description_1, [activities\_are\_equivalent\_to\_Activity\_a_1])$
- $sc_2 = (constraintType_1, a_2, a_1, order_2, description_2, [activities\_are\_equivalent\_to\_Activity\_a_2])$
- $constraintType_1 = \begin{bmatrix} coexistence \\ choice \\ mExclusion \end{bmatrix}$

In order to prove the commutative property (1-3) of the parameter *constraitType*, we have to prove that:

$$\begin{bmatrix} exe(a_1) = exe(a_2) & \Leftrightarrow & exe(a_2) = exe(a_1) \\ exe(a_1) + exe(a_2) = 1 & \Leftrightarrow & exe(a_2) + exe(a_1) = 1 \\ exe(a_1) + exe(a_2) \leq 1 & \Leftrightarrow & exe(a_2) + exe(a_1) \leq 1 \end{bmatrix} \quad (4.15)$$

Proof. They are obviously true. □

In Table 4.2, we present the symmetric, transitive and commutative properties identified based on the parameter *order* where a_1, a_2 and a_3 are activities.

Table 4.2: Algebraic properties identified based on the parameter *order*

Name		Expression
Symmetrization	(1)	$a_1\ before\ a_2 \Leftrightarrow a_2\ after\ a_1$
Transitivity	(1)	$a_1\ before\ a_2, a_2\ before\ a_3 \rightarrow a_1\ before\ a_3$
	(2)	$a_1\ after\ a_2, a_2\ after\ a_3 \rightarrow a_1\ after\ a_3$
	(3)	$a_1\ concurrence\ a_2, a_2\ concurrence\ a_3 \rightarrow a_1\ concurrence\ a_3$
	(4)	$a_1\ concurrence\ a_2, a_2\ before\ a_3 \rightarrow a_1\ before\ a_3$
	(5)	$a_1\ concurrence\ a_2, a_2\ after\ a_3 \rightarrow a_1\ after\ a_3$
Commutativity	(1)	$a_1\ concurrence\ a_2 \Leftrightarrow a_2\ concurrence\ a_1$
	(2)	$a_1\ notSpecified\ a_2 \Leftrightarrow a_2\ notSpecified\ a_1$

In order to easily prove the algebraic properties presented in Table 4.2, we express the time when an activity is executed in a process by a real function. Using function $time(a_i), a_i \in \tau$ to indicate the time, which is calculated from the start point of a process, when an activity is executed. Function $time(a_i)$ returns a non-negative number.

(i) $time(a_i) > 0$ indicates that activity a_i is executed.

(ii) $time(a_i) = 0$ indicates that activity a_i is not executed.

(iii) $time(a_i) \leq time(a_j)$ indicates that activity a_i is executed before activity a_j.

(iv) $time(a_i) \geq time(a_j)$ indicates that activity a_i is executed after activity a_j.

(v) $time(a_i) = time(a_j)$ indicates that activity a_i and activity a_j are executed at the same time.

(vi) $time(a_i) + time(a_j) \geq 0$ and $time(a_i) * time(a_j) = 0$ indicates that either only one of two activities a_i and a_j is executed or both of them are not executed.

Based on this expression, the proofs of the algebraic properties related to the parameter *order* are given below.

4.2.1.4 Symmetric Property of the Parameter *order*

Consider the following semantic constraints sc_1 and sc_2 where:

- $sc_1 = (constraintType_1, a_1, a_2, before, description_1, [activities\_are\_equivalent\_to\_Activity\_a_1])$
- $sc_2 = (constraintType_2, a_2, a_1, after, description_2, [activities\_are\_equivalent\_to\_Activity\_a_2])$

In order to prove the symmetric property (1) of the parameter *order*, we have to prove that $a_1\ before\ a_2 \Leftrightarrow a_2\ after\ a_1$.

Proof. By using our expression of the execution order of two activities and Definition 9, we get:

$$a_1\ before\ a_2 \Rightarrow time(a_1) \leq time(a_2) \Leftrightarrow time(a_2) \geq time(a_1) \Rightarrow a_2\ after\ a_1 \tag{4.16}$$

□

4.2.1.5 Transitive Property of the Parameter *order*

(i) Proof of the transitive property (1):

Consider the following semantic constraints sc_1 and sc_2 where:

- $sc_1 = (constraintType_1, a_1, a_2, before, description_1, [activities\_are\_equivalent\_to\_Activity\_a_1])$
- $sc_2 = (constraintType_2, a_2, a_3, before, description_2, [activities\_are\_equivalent\_to\_Activity\_a_2])$

In order to prove the transitive property (1) of the parameter *order* (i.e., $a_1\ before\ a_3$), we have to prove that $time(a_1) \leq time(a_3)$.

Proof. By using our expression of the execution order of two activities and Definition 9, we get:

$$a_1 \; before \; a_2 \Rightarrow time(a_1) \leq time(a_2) \tag{4.17}$$

$$a_2 \; before \; a_3 \Rightarrow time(a_2) \leq time(a_3) \tag{4.18}$$

By combining (4.17) and (4.18), we get: $time(a_1) \leq time(a_3)$. □

(ii) Proof of the transitive property (3):

Consider the following semantic constraints sc_1 and sc_2 where:

- $sc_1 = (constraintType_1, a_1, a_2, concurrence, description_1, [activities\_are\_equivalent\_to\_Activity\_a_1])$
- $sc_2 = (constraintType_2, a_2, a_3, concurrence, description_2, [activities\_are\_equivalent\_to\_Activity\_a_2])$

In order to prove the transitive property (3) of the parameter *order* (i.e., a_1 *concurrence* a_3), we have to prove that $time(a_1) = time(a_3)$.

Proof. By using our expression of the execution order of two activities and Definition 9, we get:

$$a_1 \; concurrence \; a_2 \Rightarrow time(a_1) = time(a_2) \tag{4.19}$$

$$a_2 \; concurrence \; a_3 \Rightarrow time(a_2) = time(a_3) \tag{4.20}$$

By combining (4.19) and (4.20), we get: $time(a_1) = time(a_3)$. □

(iii) Proof of the transitive property (4):

Consider the following semantic constraints scC_1 and sc_2 where:

- $sc_1 = (constraintType_1, a_1, a_2, concurrence, description_1, [activities\_are\_equivalent\_to\_Activity\_a_1])$
- $sc_2 = (constraintType_2, a_2, a_3, before, description_2, [activities\_are\_equivalent\_to\_Activity\_a_2])$

In order to prove the transitive property (4) of the parameter *order* (i.e., a_1 *before* a_3), we have to prove that $time(a_1) \leq time(a_3)$.

Proof. By using our expression of the execution order of two activities and Definition 9, we get:

$$a_1 \; concurrence \; a_2 \Rightarrow time(a_1) = time(a_2) \tag{4.21}$$

$$a_2 \; before \; a_3 \Rightarrow time(a_2) \leq time(a_3) \tag{4.22}$$

By combining (4.21) and (4.22), we get: $time(a_1) \leq time(a_3)$. □

The rest of transitive properties in Table 4.2 can be proven in the similar way,.

4.2.1.6 Commutative Property of the Parameter *order*

(i) Proof of the commutative property (1):

Consider the following semantic constraints sc_1 and sc_2 where:

- $sc_1 = (constraintType_1, a_1, a_2, concurrence, description_1, [activities\_are\_equivalent\_to\_Activity\_a_1])$
- $sc_2 = (constraintType_2, a_2, a_1, concurrence, description_2, [activities\_are\_equivalent\_to\_Activity\_a_2])$

In order to prove the commutative property (1) of the parameter *order*, we have to prove that $time(a_1) = time(a_2) \Leftrightarrow time(a_2) = time(a_1)$.

Proof. It is obviously true. □

(ii) Proof of the commutative property (2):

Consider the following semantic constraints sc_1 and sc_2 where:

- $sc_1 = (constraintType_1, a_1, a_2, notSpecified, description_1, [activities\_are\_equivalent\_to\_Activity\_a_1])$
- $sc_2 = (constraintType_2, a_2, a_1, notSpecified, description_2, [activities\_are\_equivalent\_to\_Activity\_a_2])$

In order to prove the commutative property (2) of the parameter *order*, we have to prove that:

$$\begin{cases} time(a_1) + time(a_2) & > 0 \\ time(a_1) * time(a_2) & = 0 \end{cases} \Leftrightarrow \begin{cases} time(a_2) + time(a_1) & > 0 \\ time(a_2) * time(a_1) & = 0 \end{cases}.$$

Proof. It is obviously true. □

In order to describe a semantic business process, a set of semantic constraints is defined with the help of domain experts. Consequently, implicit, redundant and conflicting constraints may exist. Moreover, conflicting constraints may lead to undesirable results. Hence, it is necessary to resolve conflicting constraints before a set of semantic constraints can be used. In the upcoming section, we will present our algorithm to validate a set of semantic constraints.

4.2.2 Algorithm for Validating a Set of Semantic Constraints

We use the properties presented in Table 4.1 and Table 4.2 to infer implicit semantic constraints. The detection of them can help to eliminate redundant constraints and to detect conflicting ones.

Given a set of semantic constraints, C, in the context of a specific business process, we have the following notations:

- Let C' be the set of all semantic constraints stemming from the semantic constraints in C.
- Let C^* be the set of all possible constraints: $C^* = C \cup C'$.

By using these notations, we next introduce the two definitions of redundant and conflicting semantic constraints.

Definition 10 (Redundant semantic constraints)**.** Constraint $sc_i \in C$: $sc_i = (constraintType_1, a_1, a_2, order_1, description_1, [activities\_are\_equivalent\_to\_Activity\_a_1]))$ is called a redundant constraint if and only if: $\exists sc_j \in C^*$ where:

- $sc_j = (constraintType_1, a_1, a_2, order_1, description_2, [activities\_are\_equivalent\_to\_Activity\_a_1])$; or
- $sc_j = (constraintType_1, a_2, a_1, oder_1, description_2, [activities\_are\_equivalent\_to\_Activity\_a_2])$ and $constraintType_1 \in \{choice, mExclusion\}$ and $order_1 = notSpecified$; or
- $sc_j = (constraintType_1, a_2, a_1, oder_1, description_2, [activities\_are\_equivalent\_to\_Activity\_a_2])$ and $constraintType_1 = coexistence$ and $oder_1 = concurrence$;
- $sc_j = (constraintType_1, a_2, a_1, oder_2, description_2, [activities\_are\_equivalent\_to\_Activity\_a_2])$ and $constraintType_1 = coexistence$ and $order_1$, $order_2$ are symmetric.

Definition 11 (Conflicting semantic constraints)**.** Constraint $sc_i \in C$: $sc_i = (constraintType_1, a_1, a_2, order_1, description_1, [activities\_are\_equivalent\_to\_Activity\_a_1]))$ is called a conflicting constraint if and only if: $\exists sc_j \in C^*$ where:

- $sc_j = (constraintType_1, a_2, a_1, order_1, description_2, [activities\_are\_equivalent\_to\_Activity\_a_2])$ and $constraintType_1 = coexistence$ and $order_1 \neq concurrence$; or
- $sc_j = (constraintType_1, a_2, a_1, order_1, description_2, [activities\_are\_equivalent\_to\_Activity\_a_2])$ and $constraintType_1 \neq coexistence$ and $order_1 = concurrence$; or
- $sc_j = (constraintType_1, a_2, a_1, order_1, description_2, [activities\_are\_equivalent\_to\_Activity\_a_2])$ and $constraintType_1 \notin \{choice, mExclusion\}$ and $order_1 = notSpecified$; or
- $sc_j = (constraintType_1, a_2, a_1, order_1, description_2, [activities\_are\_equivalent\_to\_Activity\_a_2])$ and $constraintType_1 \in \{choice, mExclusion\}$ and $order_1 \neq notSpecified$; or
- $sc_j = (constraintType_2, a_1, a_2, order_1, description_2, [activities\_are\_equivalent\_to\_Activity\_a_1])$; or

- $sc_j = (constraintType_1, a_1, a_2, order_2, description_2, [activities\_are\_equivalent\_to\_Activity\_a_1])$; or

- $sc_j = (constraintType_2, a_2, a_1, order_2, description_2, [activities\_are\_equivalent\_to\_Activity\_a_2])$ and $order_1$, $order_2$ are symmetric; or

- $sc_j = (constraintType_1, a_2, a_1, order_2, description_2, [activities\_are\_equivalent\_to\_Activity\_a_2])$ and $order_1$, $order_2$ are symmetric and $constraintType_1 = dependency$.

Example 4.2.1. Let us consider the three constraints, $sc1, sc2$ and $sc3$, expressed in Example 4.1.1. According to the properties, Transitivity (4) in Table 4.1, Symmetrization (1) and Transitivity (1) in Table 4.2, a new constraint, namely $sc1\_2$, can be inferred from the constraints $sc1$ and $sc2$ as follows:

```
sc1_2=(dependency, Process_check_or_cash, Provide_payment_methods,
after,‘‘after choosing one of provided payment methods, user must
enter payment data; paying by check or cash has to be checked and
validated’’)
```

Since the first four attributes of $sc1\_2$ and of $sc3$ are the same, the constraint $sc3$ is redundant according to Definition 10. Therefore, the constraint $sc3$ must be removed.

When a set of constraints is large, we need an algorithm to resolve issues related to redundancy and conflicting semantic constraints. In the following we present our algorithm used to remove the redundancies and detect conflicts.

As shown in Algorithm 1, the procedure to validate the set of constraints will stop as soon as it detects two conflicting constraints or a constraint that conflicts with the implicit constraint inferred from two other constraints and a message is generated to notify the users (line 5, line 11). Regarding redundancy checks, if two constraints are redundant, one of them is removed (line 14). The boolean function $conflict$ is used to check the conflict between two constraints, i.e., it returns $true$ if they are conflicting, otherwise, it returns $false$. The function $infer$ is used to infer implicit constraints. The time complexity of the algorithm is $O(n^3)$ where n is the number of semantic constraints.

Algorithm 1 Validation of the semantic constraint set

sCValidation (sc)
Input: Initial semantic set vector sc
Output: Well-checked semantic constraint set vector sc

```
1:  n = sc.size
2:  for i = 1 to n − 1 do
3:    for j = i + 1 to n do
4:      if conflict(sc[i], sc[j]) then
5:        print The constraint sc[i] conflicts with the constraint sc[j]
6:        break
7:      else if isEmpty(infer(sc[i], sc[j]))=false then
8:        scij = infer(sc[i], sc[j])   # existing an implicit constraint
9:        for k = j + 1 to n do
10:         if conflict(scij, sc[k]) then
11:           print The implicit constraint inferred from sc[i] and sc[j] conflicts
              with sc[k]
12:           break
13:         else if compare(scij, sc[k]) then
14:           C.remove(sc[k])        # removing the redundant constraint sc[k]
15:         end if
16:       end for
17:     end if
18:   end for
19: end for
```

Since there are no redundant and no conflicting constraints, the set of constraints is **well-checked**. In the next section, we describe an approach to construct a business process ontology, on which a semantic business workflow template is developed.

4.3 Organization of the Knowledge Base of Semantic Constraints

4.3.1 Development of a Business Process Ontology

To provide the representation of semantic constraints related to process elements, we propose an ontological approach to construct a formalized repository built on top of a set of well-checked semantic constrains. We focus on formalizing the concepts and relations corresponding to the knowledge required by process elements.

Let us consider the semantic constraint definition (cf. Definition 9): $SC = (constraintType, appliedActivity, relatedActivity, order, description, [Equivalence])$. The main keystones of our approach to constructing a business process ontology, namely the BP ontology, relied on the set of well-checked semantic constraints as follows:

- SC is mapped to an instance of $owl : Class$. The $rdfs : subClassOf$ property is used to state that this class is a subclass of the class $SemanticConstraints$;

- $appliedActivity$ and $relatedActivity$ are mapped to two instances of $owl : Class$. The $rdfs : subClassOf$ property is used to state that these classes are a subclass of the class SC;

- $mExclusion$, $choice$, $dependency$, $coexistence$, $before$, $after$, $concurrence$ and $notSpecified$ are defined as instances of the built-in OWL class $owl : ObjectProperty$;

- $description$ is defined as an instance of the built-in OWL class $owl : Datatype\ Property$;

- The built-in OWL property $owl : sameAs$, which is used to link an individual to an individual, states that the individuals have the same "identity". This property is used to describe each value of the parameter $Equivalence$ is equivalent to the value of the parameter $appliedActivity$ in a semantic constraint.

Figure 4.1: Extract of the ontology building on top of a set of semantic constraints

Example 4.3.1. Consider Example 4.1.1, Figure 4.2 shows the definition of the Individual $Provide\_Payment\_Methods$ since the redundant constraint $SC3$ has been removed.

```
<owl:NamedIndividual rdf:about="#Provide_Payment_Methods">
 <rdf:type rdf:resource="#AppliedActivity"/>
 <rdf:type rdf:resource="#RelatedActivity"/>
 <dependency rdf:resource="#Request_Payment"/>
 <after rdf:resource="#Request_Payment"/>
</owl:NamedIndividual>
```

Figure 4.2: Definition of the Individual $Provide\_Payment\_Methods$ in the $Payment$ template

The results of this work are used to model semantic business processes with CPNs in a knowledge base, which tends to guarantee semantic and syntactic checks at design phase.

4.3.2 Creation of Correspondences between Ontologies

In this section, we concentrate on creating correspondences to match semantics between the BP ontology (presented in Section 4.3.1) and the CPN ontology (presented in Section 3.2). In our case, the articulation of two ontologies are used not only to create semantically workflow templates, but also to verify their correctness (see Chapter 5).

We determine our use of the term "mapping" as follows: We consider two ontologies, O_1 and O_2. Mapping of an ontology with another one is defined as bringing ontologies into mutual agreement in order to make them consistent and coherent. It means that for a concept or a relation in the ontology O_1, we find the same intended meaning in the ontology O_2. For an instance in the ontology O_1, we map it into an instance with the same name in the ontology O_2.

In the following, we present some algorithms used to map the BP ontology, which is developed based on a set of well-checked semantic constraints, namely C, and the CPN ontology. We skip the descriptions of the other algorithms, which are developed in the same way with the ones presented below, to keep the presentation in this thesis short.

Algorithm 2 is firstly applied to map the instances representing the activities related to a set of constraints.

Algorithm 2 Mapping the instances representing the activities between the ontologies

mappingActivities(*bpOnt*)
Input: Given the BP ontology
Output: A set of instances in the CPN ontology represents the set of activities
Programmed Activities

1: $setOfActity$ = ReadAppliedAct(*bpOnt*)∪ ReadRelatedAct(*bpOnt*) # Read all the instances of the class *AppliedActivity* and the class *RelatedActivity* in the BP ontology *bpOnt*
2: **for all** $t \in setOfActity$ **do**
3: createActivity(t) # Create the instances: t of the class *Transition* (expressed as an activity node); pIn_t and $pOut\_t$ of the class *Place*; a_in_t and a_out_t of the classes *InputArc* and *OutputArc*, respectively; $delete\_t$ and $insert\_t$ of the classes *Delete* and *Insert*, respectively in the CPN ontology
4: **end for**

After applying Algorithm 2 to map the instances of the classes *AppliedActivity* and *RelatedActivity* into the CPN ontology, the relations between these instances

need to be considered. Among them, the relations of the instances representing a set of dependency constraints are considered first. In the following, we present Algorithm 3. This algorithm is used to create correspondences in the CPN ontology to represent the relations between the activities related to a set of dependency constraints SCD_{dep} (i.e., $SCD_{dep} \in C$), where:
$\forall sc_i \in SCD_{dep}$: $sc_i = (dependency, a, b_i, order_i, description_i, [activities\_are\_equivalent\_to\_a])$, $order_i \in \{before, after\}$; and
$\forall b_k, b_l : \nexists sc_{kl} \in C, sc_{kl} = (constraintType_{kl}, b_k, b_l, order_{kl}, description_{kl}, [activities\_are\_equivalent\_to\_a])$, $1 \leq i, k, l \leq n, k \neq l$.

The relations between the instances related to sets of choice, mutual exclusion and coexistence constraints must be considered after those related to sets of dependency constraints. Therefore, it is necessary to develop algorithms applied to these relations.

Given a set of choice constraints $SCC_{multi} \in C$ which represents the dependencies between the set of n activities, Act, where $\forall a_i, a_j \in Act$: $\exists sc_{ij} \in SCC_{multi} : sc_{ij} = (choice, a_i, a_j, notSpecified, description_{ij}, [activities\_are\_equivalent\_to\_a_i])$ or $\exists sc_{ji} \in SCC_{multi} : sc_{ji} = (choice, a_j, a_i, notSpecified, description_{ji}, [activities\_are\_equivalent\_to\_a_j])$, $1 \leq i \neq j \leq n$. Algorithm 4 is used to create instances for the set SCC_{multi}.

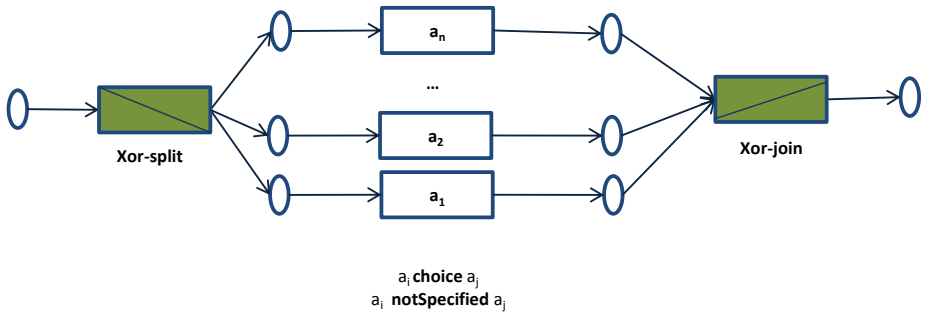

Figure 4.4: Representation of the set of semantic constraints SCC_{multi} in CPNs (Algorithm 4)

The algorithms, which are applied to map the instances representing activities related to different semantic constraints of the different types, are developed based on the properties introduced in Table 4.1 and Table 4.2. Algorithm 5, for example, is developed based on the associative property (1) and the commutative property (1) in Table 4.1.

Given a set SCD_O containing three constraints sc_1, sc_2 and sc_3 where: $sc_1 = (dependency, a_1, a_3, order_1, description_1, [activities\_are\_equivalent\_to\_a_1])$; $sc_2 = (dependency, a_2, a_3, order_1, description_2, [activities\_are\_equivalent\_to\_a_2])$, $order_1 \in \{before, after\}$; and $sc_3 = (coexistence, a_1, a_2, order_3, description_3, [activities\_are\_equivalent\_to\_a_1])$, $order_3 \in$

Algorithm 3 Mapping between the ontologies for the dependencies between the activities related to the set SCD_{dep} of the type *dependency*

mapping_dep_appliedAct($bpOnt$,SCD_{dep})
Input: Given the BP ontology, $bpOnt$ and the set of n instancesSCD_{dep} representing a set of n dependency constraints SCD_{dep}
Output: A set of correspondences in the CPN ontology represents the dependencies between the activities related to the set SCD_{dep}
Programmed Activities

1: $setOfSCD$ = ReadInstanceSC(SCD_{dep})
2: n=$setOfSCD.size$
3: **if** n>=1 **then**
4: a=$setOfSCD$[1].$appliedAct$
5: **if** n=1 **then**
6: **if** isConnected(a,$setOfSCD$[n].$relatedAct$)=**false then**
7: ctr_a=createInstanceControl($Transition$)
Create an instance, namely ctr_a of the class $Transition$ (expressed as one control node *Sequence*) in the CPN ontology
8: **if** order(setOfSC[n]) = “before" **then**
9: connectSequence(a,$setOfSCD$[n].$relatedAct$,ctr_a)
10: **else**
11: connectSequence(a,$setOfSCD$[n].$relatedAct$,ctr_a)
12: **end if**
13: isConnected(a,$setOfSCD$[n].$relatedAct$)=**true**
14: **end if**
15: **else**
16: $andSplit_a$=createInstanceControl($And-split$)
17: $andJoin_a$=createInstanceControl($And-join$)
18: **for** i = 1 **to** n **do**
19: **if** isConnected(a,$setOfSCD$[i].$relatedAct$)=**false then**
20: **if** order(a,$setOfSCD$[i].$relatedAct$) = “before" and **then**
21: connectSequence(a,$setOfSCD$[i].$relatedAct$,$andSplit_a$)
22: **else**
23: connectSequence($setOfSCD$[i].$relatedAct$,a,$andJoin_a$)
24: **end if**
25: isConnected(a,$setOfSCD$[i].$relatedAct$)=**true**
26: **end if**
27: **end for**
28: **if** isUsedToConnect($andSplit_a$) =**false then**
29: delete($andSplit_a$)
30: **end if**
31: **if** isUsedToConnect($andJoint_a$) =**false then**
32: delete($andJoin_a$)
33: **end if**
34: **end if**
35: **end if**

Algorithm 4 Mapping between the ontologies for the dependencies between the activities related to the set SCC_{multi} of the type *choice*

mapping_choice_multi($bpOnt$,SCC_{multi}**)**
Input: Given the BP ontology, $bpOnt$ and the set of n instances SCC_{multi} representing a set of n choice constraints SCC_{multi}
Output: A set of correspondences in the CPN ontology represents the dependencies between the activities related to the set SCC_{multi}
Programmed Activities

1: $setOfSCC$ = ReadInstanceSC(SCC_{multi})
2: n=$setOfSCC.size$
3: createInstanceControl($xorSplit\_scc\_multi$, $Xor-split$)
4: createInstanceControl($xorJoin\_scc\_multi$, $Xor-join$)
5: **for** i=1 **to** n **do**
6: **if** isConnected($setOfSCC[i].appliedAct$,$setOfSCC[i].relatedAct$)=**false then**
7: connectXorSplit($setOfSCC[i].appliedAct$,$setOfSCC[i].relatedAct$, $xorSplit\_scc\_multi$)
8: connectXorJoint($setOfSCC[i].appliedAct$,$setOfSCC[i].relatedAct$, $xorJoin\_scc\_multi$) # $setOfSCC[i].appliedAct$,$setOfSCC[i].relatedAct$ are connected together via $xorSplit\_scc\_multi$ and $xorJoin\_scc\_multi$
9: isConnected($setOfSCC[i].appliedAct$,$setOfSCC[i].relatedAct$)=**true**
10: **end if**
11: **end for**
12: **if** isUsedToConnect($xorSplit\_scc\_multi$) =**false then**
13: delete($xorSplit\_scc\_multi$)
14: **end if**
15: **if** isUsedToConnect($xorJoin\_scc\_multi$)=**false then**
16: delete($xorJoin\_scc\_multi$)
17: **end if**

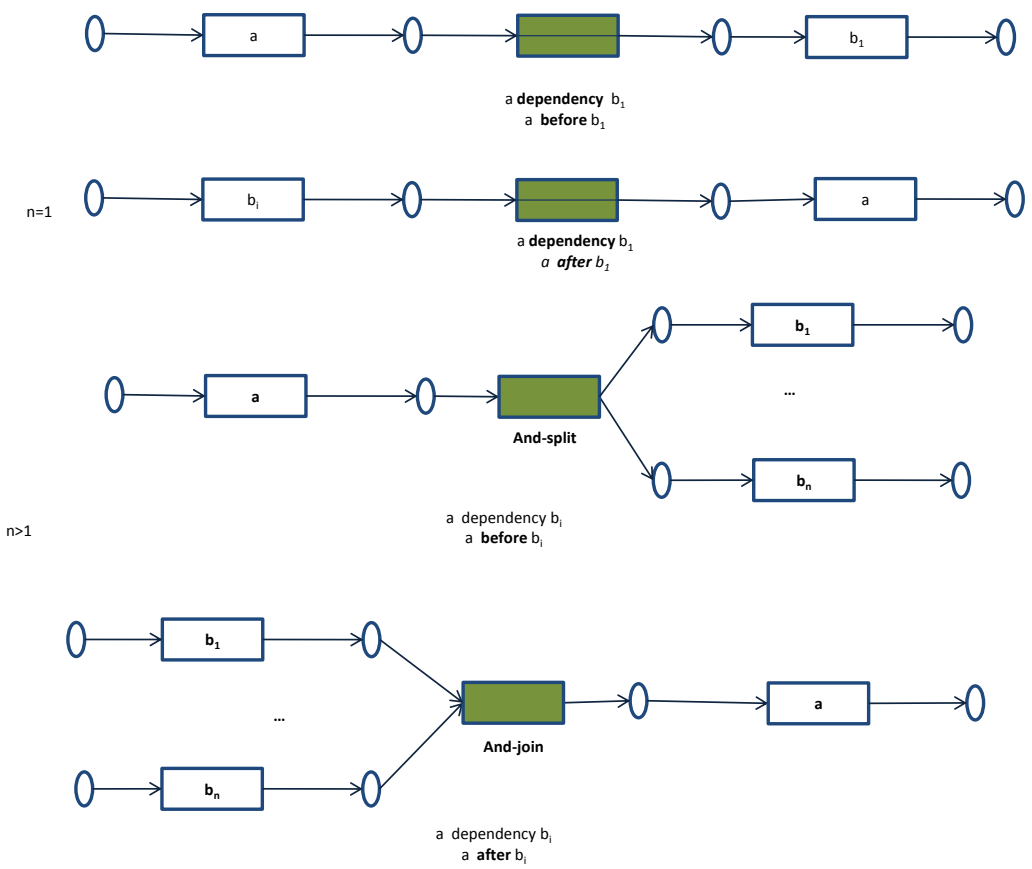

Figure 4.3: Representation of the set of semantic constraints SCD_{dep} in CPNs (Algorithm 3)

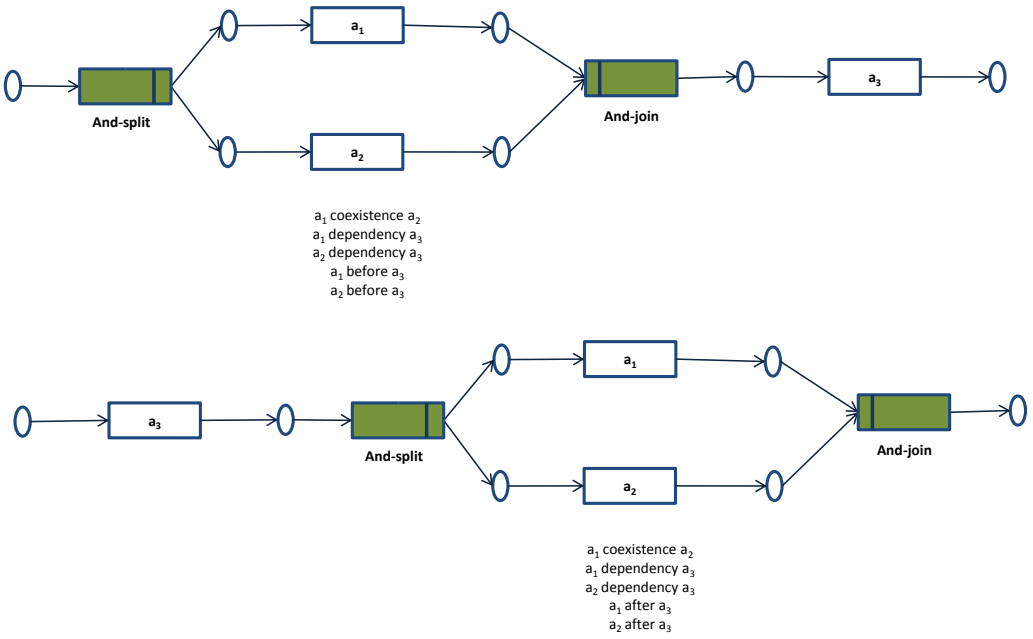

Figure 4.5: Representation of two semantic constraints of the type *coexistence* and one constraint of the type *choice* in CPNs (Algorithm 5)

$\{before, after, concurrence\}$. Algorithm 5 is used to create instances for these constraints.

Example 4.3.2. Considering Examples 4.1.1 and 4.2.1, Figure 4.6 shows the mapping of some instances between the two onotologies, the CPN ontology and the BP Ontology.

We have introduced the formal definition of semantic constraints and illustrated how to model a workflow template with CPNs based on specified semantic constraints. In the next section we are going to show how to integrate business level correctness requirements into semantic business workflows.

4.4 Integration of Event-Condition-Action Rules

In order to ensure the semantic correctness of business processes, it is necessary to integrate (semantic) domain knowledge (for example, a condition in which an activity must be performed) into workflow management systems. It is clear that the combination of workflow templates and ontologies enables the semantic representation of workflow templates. The definitions in the BP ontology (formalized in OWL) can be used not only to standardize the terminologies, but also to semantically verify workflow templates. However, the terms and relations expressed in this ontology only focus on representing the dependencies between activities of a

Algorithm 5 Mapping between the two ontologies for the dependencies between one semantic constraint of the type *coexistencey* and two constraints of the type *dependency*

mapping_dependency_coexistence($bpOnt$,SCD_O)
Input: Given the BP ontology and the set SCD_O
Output: A set of correspondences in the CPN ontology represents the relations between the activities related to the constraints sc_1, sc_2 and sc_3
Programmed activities

1: $setOfSCDO$ = ReadInstanceSC(SCD_O)
2: $SCO_{multi} = \emptyset$
3: m=$setOfSCC.size$
4: **for** $i = 1$ **to** m **do**
5: **if** $setOfSCDO[i].constraintType$ = "$coexistence$" **then**
6: SCO_{multi}=$SCO_{multi} \cup \{setOfSCDO[i]\}$
7: **if** isConnected($setOfSCDO[i].appliedAct$,$setOfSCDO[i].relatedAct$)=**false** **then**
8: $mapping\_coexistence\_multi(bpOnt,SCO_{multi})$
9: isConnected($setOfSCDO[i].appliedAct$,$setOfSCDO[i].relatedAct$)=**true**
10: **end if**
11: **end if**
12: **end for**
13: **for** i=1 **to** m **do**
14: **if** isExistAnd(SCO_{multi}) **then**
15: **if** $setOfSCDO[i].constraintType$ = "$dependency$" **then**
16: **if** isConnected($setOfSCDO[i].appliedAct$,$setOfSCDO[i].relatedAct$)=**false** **then**
17: **if** $setOfSCDO[i].order$ = "$before$" **then**
18: connectSequence($setOfSCDO[i].appliedAct$,$setOfSCDO[i].relatedAct$, getAndJoin(SCO_{multi}))
19: **else**
20: connectSequence($setOfSCDO[i].appliedAct$,$setOfSCDO[i].relatedAct$, getAndSplit(SCO_{multi}))
21: **end if**
22: isConnected($setOfSCDO[i].appliedAct$,$setOfSCDO[i].relatedAct$)=**true**
23: **end if**
24: **end if**
25: **end if**
26: **end for**

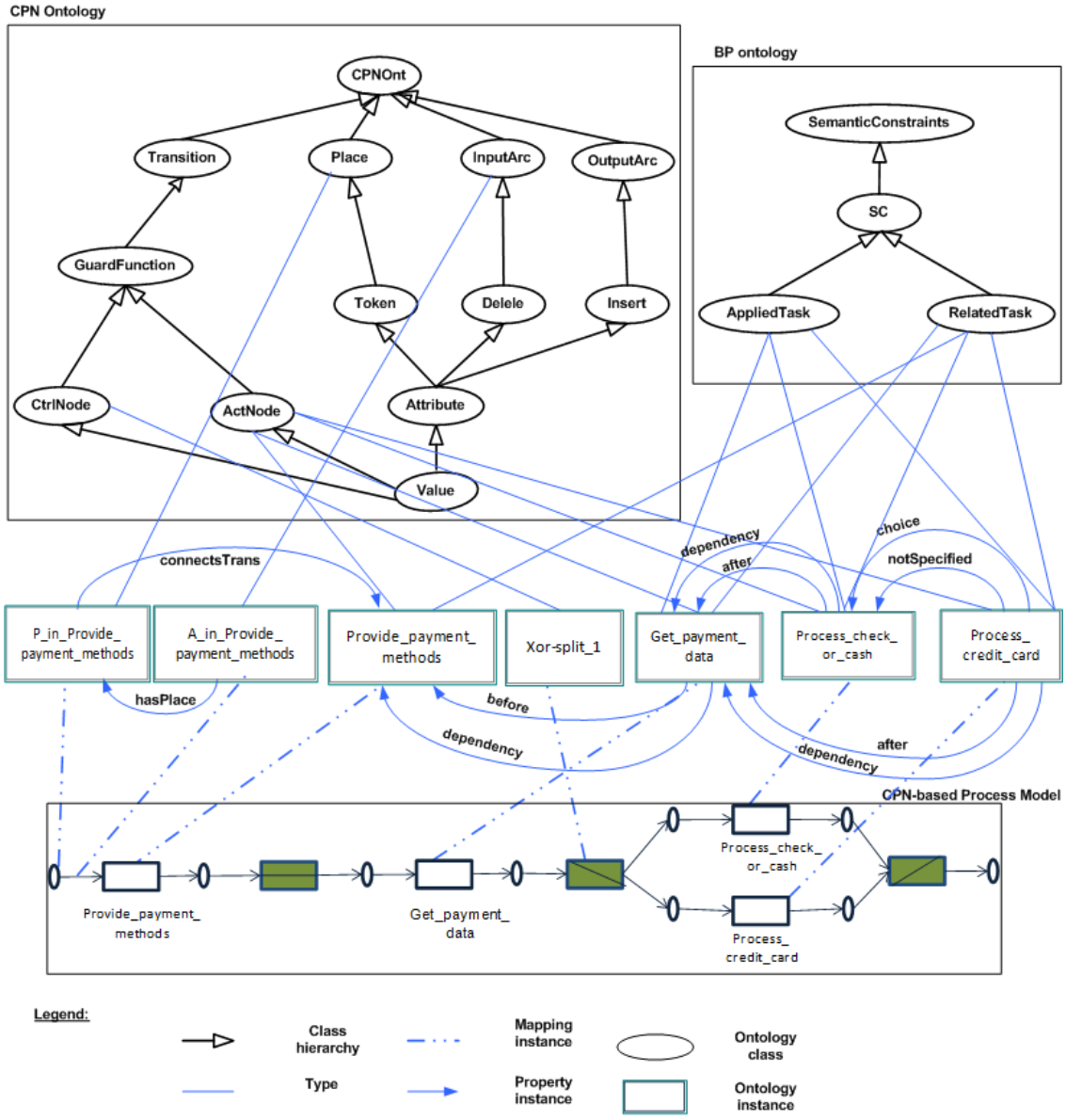

Figure 4.6: An example of ontology mapping (excerpt)

business process. They cannot capture business level correctness requirements, for example, a constraint which specifies that a certain user task has to be performed in a certain activity of a business process, or through which activities have to be enabled after the execution of a certain activity of a business process. Therefore, an extension to the use of rules is needed especially for the representation of business level correctness requirements.

As stated in Section 2.2, ECA rules can be automatically triggered when certain events take place. Therefore, we decide to use Event-Condition-Action (ECA)-like rules to express business level correctness requirements. By taking into account expert knowledge, requirements are represented in a structured way as follows:

```
ON  transition
IF  condition
DO  [action] [RAISE other_transition(s)]
```

A business level correctness requirement is expressed in the vocabulary given by the two ontologies, the CPN ontology and the BP ontology. It can be developed by using the Add ECA Rule editor as shown in Figure 4.7.

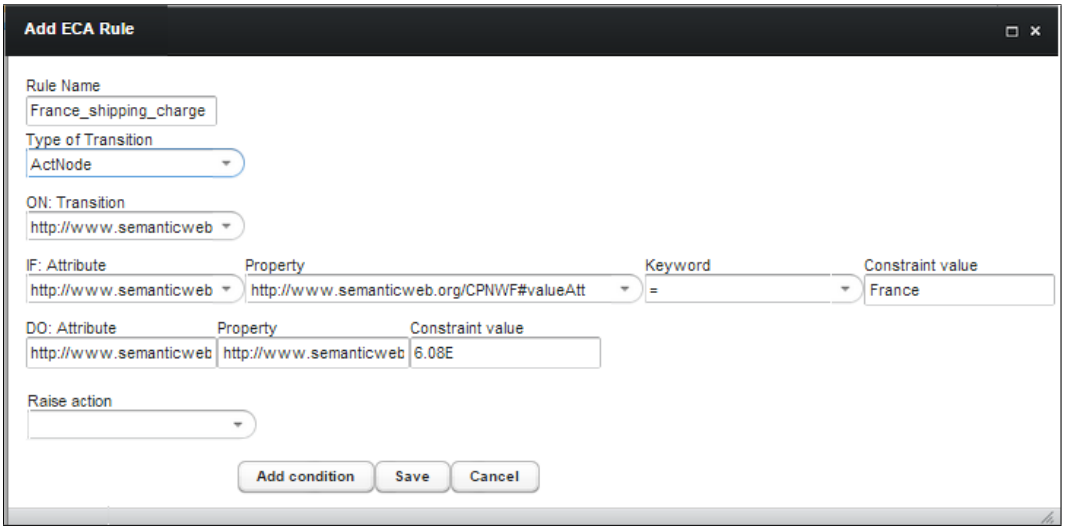

Figure 4.7: Add correctness requirement dialog

For each business workflow template, a requirement can be defined on a transition and a transition can have several requirements. Therefore, the combo box *Transition* offers all of the available transitions in a given workflow template.

Regarding the *IF condition* statement, if the guard function of a chosen transition contains attributes that their values satisfy the given conditions, then:

- an action is performed in case the transition is an activity node. Otherwise,
- at least one transition is raised.

```
<rdf:Description rdf:about="http://ECARule/Shipment_0001/#Rule_0003">
    <rule:on rdf:resource="http://WFTemplate#Calculate_
    shipping_price"/>
    <rule:if rdf:parseType="Resource">
      <rule:attribute rdf:resource="http://WFTemplate
      #Calculate_shipping_price _Attribute_Country"/>
      <rule:property rdf:resource="http://www.semanticweb.org/CPNWF
      #valueAtt"/>
      <rule:keyword>=</rule:keyword>
      <rule:value>France</rule:value>
    </rule:if>
    <rule:if rdf:parseType="Resource">
      <rule:attribute rdf:resource="http://WFTemplate
      #Calculate_shipping_price _Attribute_ Amount"/>
      <rule:property rdf:resource="http://www.semanticweb.org/CPNWF
      #valueAtt"/>
      <rule:keyword>&lt;</rule:keyword>
      <rule:value>100 E</rule:value>
    </rule:if>
    <rule:do rdf:parseType="Resource">
      <rule:attribute rdf:resource="http://WFTemplate
      #Calculate_shipping_price _Attribute_Charge"/>
      <rule:property rdf:resource="http://www.semanticweb.org/CPNWF
      #valueAtt"/>
      <rule:value>6.80 E</rule:value>
    </rule:do>
    <rule:raise></rule:raise>
  </rdf:Description>
```

Figure 4.8: Extract of a set of ECA-like rules defined for the *fOtD* process of *CompanyA*

The statements *IF condition* and *DO* [*action*] are thus expressed in terms of literals. Each literal of a *condition* or an *action* consists of a binary predicate and a set of terms. Each binary predicate, also called a property, has exactly two terms and a keyword which is offered in a combo box. The two terms relating to every property are also called domain and range. Figure 4.7 illustrates the domain, the property, and the range of a business level correctness requirement defined for shipping charges presented in Section 1.2.2. With regard to our Add ECA Rule editor, the domain of a literal is always a variable, whereas the range depends on the property. More specifically, if the property is an object property, the range is a variable. If the property is a data property, the range is a string value.

Although a set of business level correctness requirements is concerned in a cer-

tain workflow template, it should be maintained outside of the current technical representatives of the workflow template. More precisely, it is necessary to separate them from the actual technical representatives of the workflow template to ensure their persistence, even if this workflow template is redesigned or removed or even deleted. Therefore, in our work, each set of correctness requirements defined for a specific workflow template is stored in RDF format (see Figure 4.8).

4.5 Related Work

In many application domains, processes must comply with business rules and policies which are derived from domain specific requirements (e.g., standards, legal regulations). For example, in the construction industry, technical guides [Bouzidi 2012] can be considered as examples of domain requirements. As previously stated in Section 1.1, our work focuses on domain specific requirements imposing constraints on the relations of the execution of activities in a process instance. In retrospect, each process instance can be described by a sequence of events related to the activities, which are executed in the process. To date, many approaches addressing the issue of business process specification based on rules/constraints have been proposed in the literature.

M. B. Dwyer et al. [Dwyer 1999] collect and analyze over 500 examples of property specifications from different domains. They indicate that most of these examples are conformed to eight property patterns within five basic kinds of scopes. A scope (depicted in Figure 4.9) is determined by the specification of a starting and an end state/event for each pattern. Most of them are self-explanatory, for example, **Before** indicates that the execution up to a given state/event. According to [Dwyer 1999], the property patterns are organized into two major groups, **Occurrence** and **Order** (see Figure 4.10) consisting of:

- **Absence** requires that the defined scope is free from a given state/event;
- **Existence** requires that a given state/event must occur within the scope;
- **Bounded existence** requires that a given state/event must occur at most a specific number of times within the scope;
- **Universality** requires that a given state/event is true throughout the scope;
- **Precedence** requires that the occurrence of a given state/event prior to the occurrence of another state/event in the scope.
- **Response** requires the occurrence of a given state/event must always be followed by the occurrence of another state/event (i.e., cause-effect relationships);
- **Chain Precedence** requires that a given sequence of states/events must always be preceded by a sequence of other states/events in the scope;

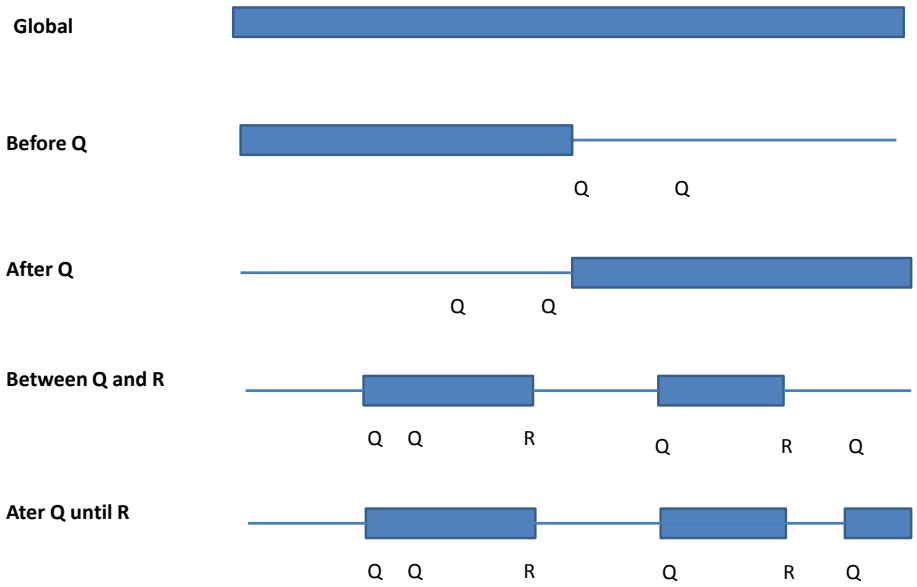

Figure 4.9: Scopes for property specification patterns

- **Chain Response** requires that a given sequence of states/events must always occur as response to the occurrence of a sequence of other states/events in the scope.

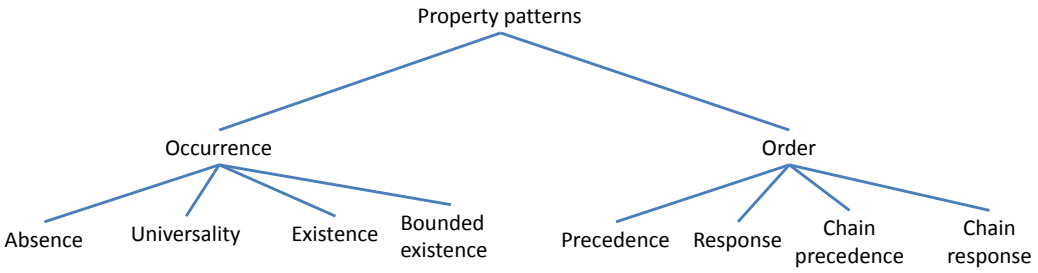

Figure 4.10: Property specification patterns introduced in [Dwyer 1999]

Indeed, although their patterns express formal requirements related to the occurrence and order of states/events during system execution, they can be used as fundamental for compliance rule specification as we can see in the approaches briefly introduced in the following.

The authors in [Sadiq 2005] describe an approach for specifying and validating process constraints for flexible workflows. According to them, the key issue in flexible workflows is the specification of subprocesses, from which a full workflow specification may be derived at runtime. They use different types of constraints (i.e., serial,

order, fork, inclusion and exclusion constraints) that express dependencies between activities to restrict composition possibilities. A subprocess has to be validated against the set of constraints before it is executed. By enabling the definition of process models ranging from completely modelled to mainly constraint-based, this approach provides an appropriate balance between flexibility and control. Another formal specification of semantic constraints is introduced in [Kumar 2010]. An integer programming formulation is used to express semantic constraints and also to detect and handle constraint violations. They focus on the occurrence of each activity in a process. An activity must be either executed (one or several times) or not executed. In the relationship with other activities, they can be executed in choice, in exclusion or in dependency or together. However, this method does not mention the execution order between two activities.

In [Ly 2008], two fundamental kinds of semantic constraints, i.e., mutual exclusion constraints and dependency constraints, are introduced. The former expresses that two activities are incompatible and should not be performed together. The later expresses that an activity is dependent on the other activity and they have to take place together in the process. Practically speaking, there exist other kinds of constraints, for example, constraints can express that two relevant activities must be both included or be both excluded, or only one of two relevant activities must be executed. Consequently, a precise classification of semantic constraints is required. We focus on both occurrence and ordering constraints on sequences of events. We are able to represent the patterns in [Dwyer 1999] by using different types of constraints and different execution orders between activities.

In the matter of correctness requirements related to business rules at design time, [Namiri 2007, Namiri 2008] represents compliance requirements as production rules according to the terms and concepts, which are defined in a formal ontology. However, due to the emphasis they put on events of ECA rules, they are better suitable for modelling the variable parts of a process flow and for distributed applications [van Eijndhoven 2008, Berstel 2007]. Therefore, to develop semantically rich control flow-base workflow templates, in our work, we use ECA-like rules to express business level correctness requirements.

4.6 Discussion and Conclusion

This chapter has presented a formal method for describing semantic constraints used for generating semantic workflow templates. We first proposed a formal definition of semantic constraints. We then introduced an algorithm for detecting redundant and conflicting semantic constraints. A set of well-checked semantic constraints is transformed into an instance of a business process ontology, called the BP ontology. To develop workflow templates, we have also presented a set of algorithms to create correspondences between the BP ontology and the CPN ontology (cf. Chapter 3). The results of this work were published in [Nguyen 2014b, Pham 2015].

In the following chapter, we show that the SPARQL query language is able to

check the syntactic and semantic correctness of concrete workflow templates represented in RDF format.

Chapter 5

Verification of Workflow Templates

Contents

Providing a high-level specification of business processes is the objective of process modelling. This makes process models independent of the target workflow management system. Arguably, high quality workflow definitions play an important role in the organization. A workflow defined incorrectly may lead to unintended consequences, for instance, a waste of time and effort, loss of trust in users. That is why a workflow definition should be analyzed and verified[1] before it is put into use.

In this chapter, we introduce a solution to verify workflow templates at the design phase. We focus on checking the syntactic and semantic correctness of business workflow templates as depicted in Figure 5.1.

[1] According to the IEEE 1012-2012 definition [iee 2012], "verification" means to evaluate whether or not a product, service, or system conforms to a set of given requirements. Hence, it relates to the internal constitution of a model. In contrast, "validation" implies the appropriateness of a model with regard to the needs of the customer and other identified stakeholders. This means the criteria involve something outside the model.

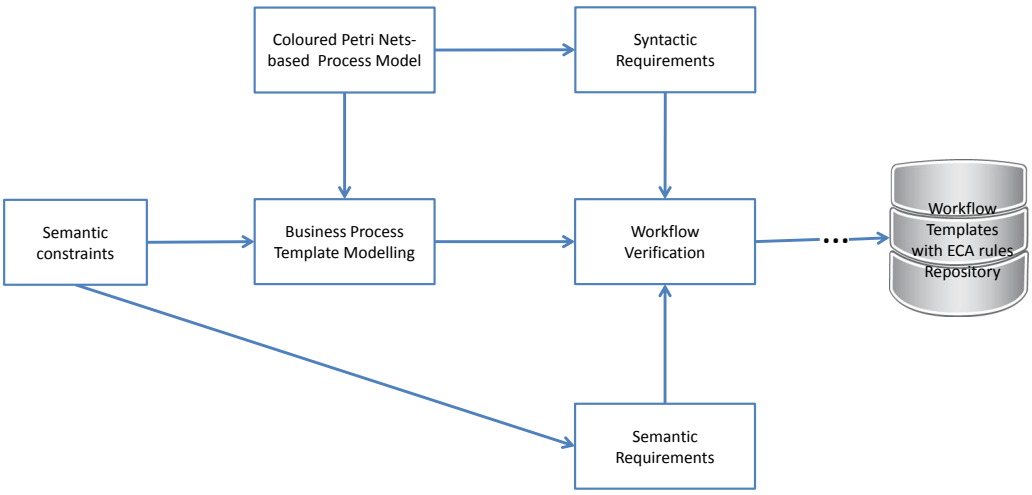

Figure 5.1: Verification of business workflow templates

5.1 Syntactic Verification Issues

To provide automated support for workflow designers in establishing the correctness of ontology-based workflow representations, the syntactic constraints are categorized into two groups. Axioms related to these constraints are also defined using a DL as $\mathcal{SHOIN}(\mathcal{D})$ to complete the CPN ontology.

First of all, let us define some properties for CPN-based process models.

Definition 12 (Reachability). A CPN-based process model $PM = (\sum, P, T, A, F, C, G, E, I)$ and an initial state M_0 where start place s contains one token. We say that transition t makes state M_1 reachable from state M_0 if in state M_0, t is enabled and firing it results in state M_1, written $M_0 \xrightarrow{t} M_1$.

A state M_n is called reachable from state M_0 iff there is a firing sequence[2] $t_1 t_2 \dots t_j$ such that $M_0 \xrightarrow{t_1} M_2 \xrightarrow{t_2} \dots \xrightarrow{t_{j-1}} M_j$ and written $M_0 \xrightarrow{*} M_j$.

Definition 13 (Connected). A CPN-based process model $PM = (\sum, P, T, A, F, C, G, E, I)$ is connected iff for every pair of places (one input place and one output place) u and v, there exists a directed path either from u to v or from v to u. Formally:

(i) $\forall u \in P_{in}, v \in P_{out}, \exists p_1, t_1, \dots, p_k, t_k, p_{k+1}, p_i \in P, t_i \in T, u = p_1, v = p_{k+1} : p_i t_i \in A, t_i p_{i+1} \in A, \forall i \in 1, \dots, k$ or

(ii) $\forall u \in P_{in}, v \in P_{out}, \exists p_1, t_1, \dots, p_k, t_k, p_{k+1}, p_i \in P, t_i \in T, v = p_1, u = p_{k+1} : p_i t_i \in A, t_i p_{i+1} \in A, \forall i \in 1, \dots, k$

Where $p_i t_i$ is the directed arc from place p_i to transition t_i, $t_i p_{i+1}$ is the directed arc from transition t_i to place p_{i+1}.

[2] Relying on the firing rule in [van der Aalst 1997]

Definition 14 (Well-formed). A CPN-based process model $PM = (\sum, P, T, A, F, C, G, E, I)$ is well-formed iff:

(i) Every element $x \in P \cup T$ is on a path from start point s to end point e.

(ii) For every state M' which is reachable from state Start M_0 and every transition $t \in T$, there exists a state M'' reachable from state M' which activates transition t.

The following definition is defined as the soundness property, which is very close to the one proposed in [van der Aalst 1997].

Definition 15 (Sound). A CPN-based process model $PM = (\sum, P, T, A, F, C, G, E, I)$ is sound iff:

(i) PM is connected.

(ii) PM is well-formed.

(iii) For every state M_j reachable from state Start M_0, there also exists another firing sequence starting from state M_j to state End M_e. Formally:

$$\forall M_j : (M_0 \xrightarrow{*} M_j) \Rightarrow (M_j \xrightarrow{*} M_e)$$

(iv) State End M_e is the only state which is reachable from state Start M_0 with one token in place e.

(v) There is no deadlock, no infinite cycle and no missing synchronization in PM.

As mentioned earlier, we aim at representing CBWTs in a knowledge base. Therefore, the soundness property (Definition 15) is used as the criterion to check the correctness of workflow templates at the syntactic level.

5.1.1 Syntactic Constraints related to the Definition of Process Model

- **Constraints related to places.**

  Constraint 1. For every place $p \in P$, p connects and/or is connected with transitions via arcs.

  We create the axiom corresponding to Constraint 1 as follows:

  $hasPlace^{-}.CPNOnt \sqcap \neg(\exists connectsTrans.hasTrans^{-}.CPNOnt \sqcup \exists connectsPlace^{-}.hasTrans^{-}.CPNOnt) \sqsubseteq \bot$

  Constraint 2. There is one and only one start point in a process model.

  We create the axiom corresponding to Constraint 2 as follows:

  $CPNOnt \sqcap \neg(= 1\ hasPlace.(connectsTrans.hasGuardFunction.hasActivity.$

  $ActNoce \sqcap \neg(\exists\ connectsPlace^{-}.hasTrans^{-}.CPNOnt))) \sqsubseteq \bot$

Constraint 3. There is one and only one end point in a process model.

We create the axiom corresponding to Constraint 3 as follows:

$CPNOnt \sqcap \neg(= 1\ hasPlace.(connectsPlace^{-}.hasGuardFunction.hasActivity.$

$ActNode\ \sqcap\ \neg(\exists\ connectsTrans.hasTrans^{-}.CPNOnt))) \sqsubseteq \bot$

Constraint 4. A place has no more than one leaving arc. If a place is connected to a transition, there exists only one directed arc from the place to the transition.

We create the axiom corresponding to Constraint 4 as follows:

$Place\ \sqcap\ \neg(\leq 1\ hasPlace^{-}.InputArc) \sqsubseteq \bot$

Constraint 5. A place has no more than one entering arc. If a transition is connected to a place, there exists only one directed arc from the transition to the place.

We create the axioms corresponding to Constraint 5 as follows:

$Place\ \sqcap\ \neg(\leq 1\ connectsPlace^{-}.(= 1 hasTrans^{-}.OutputArc)) \sqsubseteq \bot$

Constraint 6. There are no pairs of activity nodes connected via a place.

We create the axiom corresponding to Constraint 6 as follows:

$Place\ \sqcap\ \exists connectsTrans.hasGuardFunction.hasActivity.ActNode\ \sqcap$

$\exists connectsPlace^{-}.hasGuardFunction.hasActivity.ActNode\ \sqsubseteq\ \bot$

Constraint 7. There are no pairs of control nodes connected via a place.

We create the axiom corresponding to Constraint 7 as follows:

$Place\ \sqcap\ \exists connectsTrans.hasGuardFunction.hasControl.CtrlNode\ \sqcap$

$\exists connectsPlace^{-}.hasGuardFunction.hasControl.CtrlNode\ \sqsubseteq\ \bot$

- **Constraints related to transitions.**

  Constraint 8. A transition is on the path from the start point to the end point of a process model.

  - If a transition has no input place, it will never be enabled.
  - If a transition has no output place, it will not lead to the end.

  Consequently, each transition in a workflow must have at least one entering arc and at least one leaving arc.

  We create the axiom corresponding to Constraint 8 as follows:

  $Transition\ \sqsubseteq\ \geq\ 1\ connectsPlace.Place\ \sqcap\ \geq\ 1\ connectsTrans^{-}.Place$

  Constraint 9. An activity node has only one entering arc and one leaving arc.

  We create the axiom corresponding to the Constraint 9 as follows:

$hasGuardFunction.hasActivity.ActNode \sqsubseteq \ = 1\ connectsPlace.Place \sqcap \ = 1\ connectsTrans^{-}.Place$

Constraint 10. According to Definitions 4-8, a control node does not have both multi-leaving arcs and multi-entering arcs.

We create the axiom corresponding to the Constraint 10 as follows:

$\geq 2\ connectsPlace.Place \sqcap \ \geq 2\ connectsTrans^{-}.Place \sqcap$

$hasGuardFunction.hasControl.CtrlNode \sqsubseteq \bot$

- **Constraints related to directed arcs.**

  Constraint 11. Directed arcs connect places to transitions or vice versa.

  We create the axioms corresponding to the Constraint 11 as follows:

  $hasPlace^{-}.InputArc \equiv connectsTrans.hasTrans^{-}.CPNOnt$

  $hasTrans^{-}.OutputArc \equiv connectsPlace.hasPlace^{-}.CPNOnt$

5.1.2 Syntactic Constraints Related to Uses of Control Nodes

A poorly designed workflow due to improper uses of control nodes can result in deadlock, infinite cycle or missing synchronization. However, these errors can be detected when designing a workflow template and therefore, we can get rid of them. To do that, we next introduce Constraint 12 and the symptoms related to deadlock, infinite cycle or missing synchronization.

Constraint 12. There is no deadlock, no infinite cycle and no missing synchronization.

- **Deadlock**: A deadlock is a situation in which a process instance falls into a stalemate such that no more activity can be enabled to execute [Verbeek 2001].

  Accoding to [Bi 2004], there are two types of deadlock (deterministic and non-deterministic deadlock) which relate to the combination of the building blocks, i.e., $Xor - split$ and $And - join$, $And - join$ and $Xor - split$, $And - join$ and $And - split$.

  It is necessary to note that the building blocks $Or - split$ and $Or - join$ are not used in our work. One of the reasons is that the execution of an OR (i.e., $Or - split$ and $Or - join$) is non-deterministic. If a transition $Or - split$ fires, it produces one token for at least one of its output places. Therefore, by not using these building blocks, we can avoid the second type of deadlock. Figure 5.2 shows three simple deadlock simulations.

- **Infinite cycle**: An infinite cycle is derived from structural errors where some activities are repeatedly executed indefinitely.

  Starting with an entrance *Xor-join* and ending with an exit *And-split*, a cycle is infinite. A simple infinite simulation is depicted in Figure 5.3.

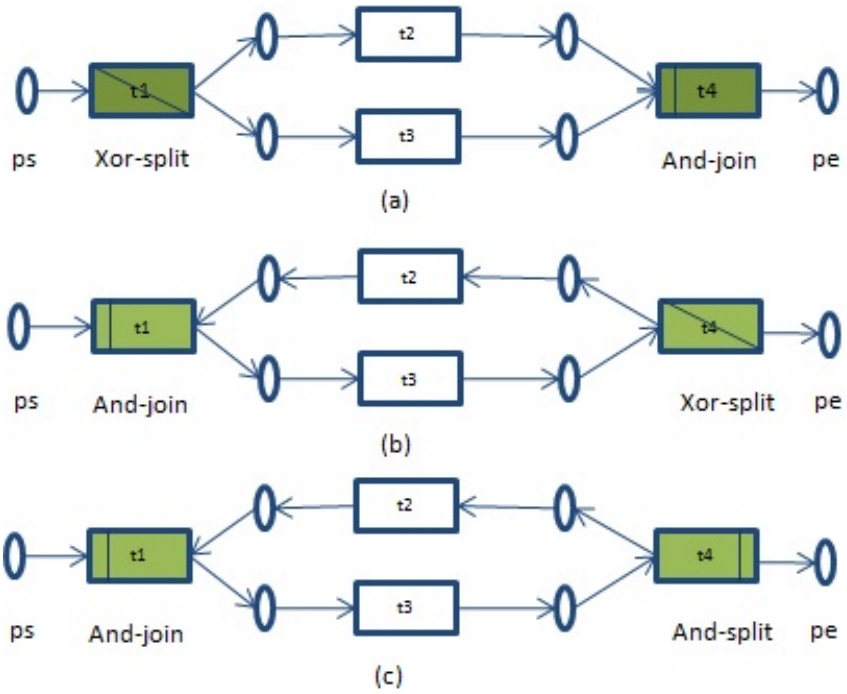

Figure 5.2: Deadlock simulations

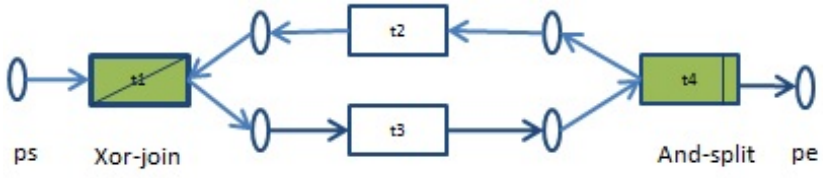

Figure 5.3: Infinite cycle simulation

- **Missing synchronization**: Missing synchronization is a situation in which the mismatch between the building blocks leads to neither deadlock nor infinite cycle, but results in unplanned executions. The mismatch is established by an entrance *And-split* and an exit *Xor-join*. Figure 5.4 shows a simple simulation of missing synchronization.

Therefore, we next create the axioms related to the control nodes, including $And-split$, $And-join$, $Xor-split$ and $Xor-join$ used to detect deadlock, infinite cycle or missing synchronization.

- **And-split**

  This transition is connected to at least two output places. Every output place contains one token. We create the axiom corresponding to the transition *And-split* as follows:

  $AndSplit \sqsubseteq Transition \sqcap connectsPlace.hasMarking.Token \sqcap$

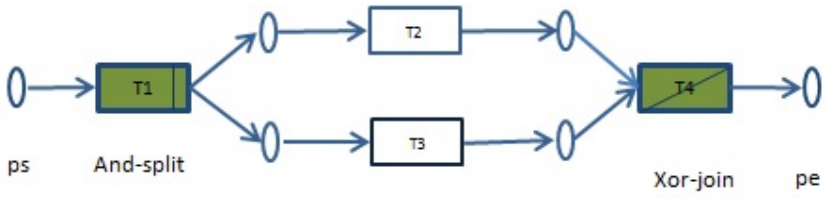

Figure 5.4: Missing synchronization simulation

$connectsTrans^{-}.hasMarking.Token \sqcap hasGuardFunction.hasControl.$

$CtrlNode \sqcap = 1\ connectsTrans^{-}.Place \sqcap \geq 2\ connectsPlace.Place$

- **And-join**

  There are at least two input places connected to the transition *And-join*. In order to activate the transition *And-join*, every input place has to contain one token. We create the axiom corresponding to the transition *And-join* as follows:

  $AndJoin \sqsubseteq Transition \sqcap connectsPlace.hasMarking.Place \sqcap$

  $connectsTrans^{-}.hasMarking.Token \sqcap hasGuardFunction.hasControl.$

  $CtrlNode \sqcap \geq 2 connectsTrans^{-}.Place \sqcap = 1\ connectsPlace.Place$

- **Xor-split**

  This transition is connected to at least two output places. Unlike the transition *And-split*, at any time, one and only one output place of the transition *Xor-join* can contain a token. We create the axiom corresponding to the transition *Xor-split* as follows:

  $XorSplit \sqsubseteq Transition \sqcap \neg AndSplit \sqcap hasGuardFunction.hasControl.$

  $CtrlNode \sqcap = 1\ connectsTrans^{-}.Place \sqcup \geq 2\ connectsPlace.Place \sqcup$

  $connectsTrans^{-}.hasMarking.Token$

- **Xor-join**

  There are at least two input places connected to the transition *Xor-join*. Unlike the transition *And-join*, the transition *Xor-join* is activated if one and only one input place contains a token. We create the axiom corresponding to the transition *Xor-join* as follows:

  $XorJoin \sqsubseteq Transition \sqcap \neg AndJoin \sqcap connectsPlace.hasMarking.Token$

  $\sqcap \geq 2\ connectsTrans^{-}.Place. \sqcap hasGuardFunction.hasControl.CtrlNode$

  $\sqcap = 1\ connectsPlace.Place$

We have introduced the axioms defined to support designers in verifying CPN-based workflow templates at the syntactic level. In the next section, we will show how to use the SPARQL query language to detect syntactic errors of workflow templates.

5.1.3 Compliance Checking of Workflow Templates at the Syntactic Level

In order to verify a workflow template, we initiatively query the workflow template to verify whether it contains syntactic errors or not. Two query forms are used in our work, including ASK and SELECT. The following SPARQL verification queries are created based on the syntactic constraints.

- **Query 1** is created relating to Constraint 1 to list all places not connected to any arcs. They are not on any path from the start point to the end point of a process model.

```
SELECT ?p WHERE
{ ?cp rdf:type h:CPNOnt
  ?cp h:hasPlace ?p
  FILTER NOT EXISTS {
    ?p h:connectsTrans|^h:connectsPlace _:b}
}
```

- **Queries 2.1** and **2.2**: With regard to Constraint 2, two queries are created:

  Query 2.1 is used to ask if the number of start points of the workflow template is not equal to 1.

```
ASK {
  { SELECT (COUNT(distinct ?p) AS ?c)
    WHERE
    { ?i rdf:type h:InputArc
      ?i h:hasPlace ?p
      ?p h:connectsTrans/h:hasGuardFunction/h:hasActivity _:b
      MINUS {
       ?cp rdf:type h:CPNOnt
       ?cp h:hasTrans ?t
       ?t h:connectsPlace ?p }
    }
  } FILTER (?c!=1)
}
```

  Query 2.2 is a SELECT query, which comprises the same WHERE condition with Query 2.1 and is executed to list all places designed as start points.

```
SELECT distinct ?p WHERE
  { ?i rdf:type h:InputArc
    ?i h:hasPlace ?p
    ?p h:connectsTrans/h:hasGuardFunction/h:hasActivity _:b
    MINUS {
```

```
    ?cp rdf:type h:CPNOnt
    ?cp h:hasTrans ?t
    ?t h:connectsPlace ?p }
 }
```

For the sake of simplicity, ASK queries relating to the rest of Constraints are omitted if there are SELECT queries containing the same WHERE condition with them.

- **Query 3** is created relating to Constraint 3.

  Two queries (i.e., one ASK query and one SELECT query) are created. In the following, we present the SELECT query created to list all places designed as end points.

```
 SELECT (COUNT(distinct ?p) AS ?c)
        WHERE {
          ?t h:hasGuardFunction/h:hasActivity _:b1
          ?t h:connectsPlace ?p
          FILTER NOT EXISTS
               {
               ?p h:connectsTrans _:b2
               }
        }
 } FILTER (?c>=2)
```

- **Query 4** is created relating to Constraint 4 to list all places having more than one leaving arc.

```
SELECT distinct ?p WHERE
 {
  ?i1 rdf:type h:InputArc
  ?i2 rdf:type h:InputArc
  ?i1 h:hasPlace ?p
  ?i2 h:hasPlace ?p
  FILTER (?i1!=?i2)
 }
```

- **Query 5** is created relating to Constraint 5 to list all places having more than one entering arc.

```
SELECT ?p ?c WHERE {
  {SELECT ?p (COUNT(?p) AS ?c)
        WHERE {
                ?p rdf:type h:Place
                _:b h:connectsPlace ?p
```

```
                } GROUP BY ?p
  }
  FILTER (xsd:integer(?c)>=2)
}
```

- **Query 6** is created relating to Constraint 6 to list all pairs of activity nodes which are connected via a place.

```
SELECT ?p WHERE
 {
  ?t1 h:hasGuardFunction/h:hasActivity _:b1
  ?t2 h:hasGuardFunction/h:hasActivity _:b2
  ?t1 h:connectsPlace ?p
  ?p h:connectsTrans ?t2
 }
```

- **Query 7** is created relating to Constraint 7 to list all pairs of control nodes which are connected via a place.

```
SELECT ?p WHERE {
  ?t1 h:hasGuardFunction/h:hasControl _:b1
  ?t2 h:hasGuardFunction/h:hasControl _:b2
  ?t1 h:connectsPlace ?p
  ?p h:connectsTrans ?t2
 }
```

- **Queries 8.1** and **8.2**:
  With regard to Constraint 8, two queries are created. The former is used to find all transitions not having any input arcs while the latter is used to find all transitions not having any output arcs.

```
SELECT distinct ?t WHERE {
  ?cp rdf:type h:CPNOnt
  ?cp h:hasTrans ?t
  FILTER NOT EXISTS {_:b h:connectsTrans ?t}
 }
```

```
SELECT ?t WHERE {
  ?cp rdf:type h:CPNOnt
  ?cp h:hasTrans ?t
  FILTER NOT EXISTS { ?t h:connectsPlace _:b}
 }
```

- **Queries 9.1** and **9.2**:
  With regard to Constraint 9, in order to find activity nodes which have neither input arcs nor output arcs, queries 8.1 and 8.2 are used. Therefore, we here focus on how to find activity nodes which have at least two input arcs (Query 9.1):

```
SELECT distinct ?t WHERE
 {
  ?p1 rdf:type h:Place
  ?p1 h:connectsTrans ?t
  ?p2 rdf:type h:Place
  ?p2 h:connectsTrans ?t
  ?t h:hasGuardFunction/h:hasActivity ?a
  FILTER (?p1!=?p2)
}
```

  or at least two output arcs (Query 9.2):

```
SELECT distinct ?t WHERE {
  ?o1 rdf:type h:OutputArc
  ?o1 h:hasTrans ?t
  ?o2 rdf:type h:OutputArc
  ?o2 h:hasTrans ?t
  ?t h:hasGuardFunction/h:hasActivity ?a
  FILTER (?o1!=?o2)
}
```

- **Query 10** is created relating to Constraint 10 to list all control nodes which have at least two leaving arcs and at least two multi-entering arcs.

```
SELECT distinct ?t WHERE
 {
  ?t h:hasGuardFunction/h:hasControl _:b
  ?p1 rdf:type h:Place
  ?p2 rdf:type h:Place
  ?t h:connectsPlace ?p1
  ?t h:connectsPlace ?p2
  ?p3 rdf:type h:Place
  ?p4 rdf:type h:Place
  ?p3 h:connectsTrans ?t
  ?p4 h:connectsTrans ?t
  FILTER (?p1!=?p2 && ?p3!=?p4)
}
```

- **Queries 11.1** and **11.2** are created relating to Constraint 11 to list all places and transitions that do not satisfy this constraint, respectively. This means

that there may exist directed arcs that are dangling (i.e., the absence of one part or both relevant parts).

```
SELECT  distinct ?p  WHERE
{
 { ?p h:connectsTrans _:b1
        MINUS {  ?i rdf:type h:InputArc
                 ?i h:hasPlace ?p }
 }
 UNION
 {  ?i rdf:type h:InputArc
        ?i h:hasPlace ?p
        MINUS { ?p h:connectsTrans _:b2}
 }
 UNION
 {  ?p h:connectsTrans ?t
      MINUS {   ?cpn rdf:type h:CPNOnt
                ?cpn h:hasTrans ?t  }
 }}
```

```
SELECT  distinct ?t  WHERE
{
 {  ?t h:connectsPlace _:b1
         MINUS {  ?o rdf:type h:OutputArc
                  ?o h:hasTrans ?t }
 }
 UNION
 {  ?o rdf:type h:OutputArc
    ?o h:hasTrans ?t
        MINUS {  ?t h:connectsPlace _:b2 }
 }
 UNION
 {  ?t h:connectsPlace ?p
    MINUS {  ?cpn rdf:type h:CPNOnt
             ?cpn h:hasPlace ?p }
 }
}
```

- **Queries 12.1** and **12.2**: We continue to check whether errors exist or not, related to the improper uses of control nodes. However, for workflow templates that contain certain overlapping routing transitions, we cannot check Constraint 12 by only using the SPARQL query language. In order to detect deadlock, infinite cycle and missing synchronization, the reduction algorithm in [Esparza 1994] must be applied. The algorithm is used to transform a

workflow template into a simple form. We then can use the SPARQL query language to query the simple forms of workflow templates.

The following query, Query 12.1, is used for detecting if there exist any deadlocks caused by the combination of control nodes $Xor-split$ and $And-join$. The query will return pairs of control nodes which make deadlocks happen.

```
SELECT distinct ?xorsplit ?andjoin WHERE
 {
  ?xorsplit  rdf:type h:Xor-split
  ?andjoin rdf:type h:And-join
  ?t1 h:hasGuardFunction/h:hasActivity _:b1
  ?t2 h:hasGuardFunction/h:hasActivity _:b2
  ?xorsplit h:connectsPlace/h:connectsTrans ?t1
  ?xorsplit h:connectsPlace/h:connectsTrans ?t2
  ?t1 h:connectsPlace/h:connectsTrans ?andjoin
  ?t2 h:connectsPlace/h:connectsTrans ?andjoin
  FILTER(?t1!=?t2)
 }
```

In order to detect infinite cycles caused by the other combinations of control nodes (shown in Figure 5.4), we create Query 12.2 as follows:

```
SELECT distinct ?xorjoin ?andsplit WHERE
 {
  ?xorjoin  rdf:type h:Xor-join
  ?andsplit rdf:type h:And-split
  ?t1 h:hasGuardFunction/h:hasActivity _:b1
  ?t2 h:hasGuardFunction/h:hasActivity _:b2
  ?xorjoin h:connectsPlace/h:connectsTrans ?t2
  ?t2 h:connectsPlace/h:connectsTrans ?andsplit
  ?andsplit h:connectsPlace/h:connectsTrans ?t1
  ?t1 h:connectsPlace/h:connectsTrans ?xorjoin
  FILTER(?t1!=?t2)
 }
```

The queries used to list all pairs of control nodes causing deadlock (depicted in Figure 5.2) (b) and (c) are created similar to Query 12.2.

5.2 Semantic Verification Issues

5.2.1 Semantic Verification Tasks

We hereinafter pay attention to the research question relating to semantic verification: Is the behavior of the individual activities satisfied and does it conform to the

control flow? To answer this question, we address the following semantic verification issues:

(1) Are there activities whose occurrences are alternative choices or in mutual exclusion, but these activities may be executed in parallel or in sequence?

(2) Are there activities whose executions are interdependent, but these activities may be executed as alternative choices or in mutual exclusion or in parallel?

(3) Are there activities whose occurrences are coexistent, but these activities may be executed as alternative choices or in mutual exclusion?

(4) Are there any couples of activities whose order executions are defined as one before the other, but these activities may be executed in the opposite order?

(5) Are there any couples of activities whose order executions are defined as one after the other, but these activities may be executed in the opposite order?

5.2.2 Compliance Checking of Workflow templates at the Semantic Level

In order to answer the above-mentioned semantic verification issues, we continue using the SPARQL query language. The following SELECT queries are created for semantic checks:

- **Queries 13.1** and **13.2** are created relating to the first semantic verification issue.
  Query 13.1 is used to query if the model contains any pairs of activity nodes whose occurrences are alternative **choices**, but that may be executed in **parallel**. It is necessary to note that the properties $k : choice$ and $k : notSpecified$, which are defined in the BP ontology, indicate the semantic constraint between activities $?t1$ and $?t2$. The rest of the properties, which are defined in the CPN ontology, represent these activities restricted to the control flow perspective.

```
SELECT ?t1 ?t2 WHERE
 {
  ?t1 rdf:type h:Transition;
      h:hasGuardFunction/h:hasActivity _:b1.
  ?t2 rdf:type h:Transition;
      h:hasGuardFunction/h:hasActivity _:b2.
  ?t1 k:choice ?t2;
      k:notSpecified ?t2.
      ?andsplit rdf:type h:And-split
  ?andjoin rdf:type h:And-join
  ?andsplit h:connectsPlace/h:connectsTrans ?t1;
            h:connectsPlace/h:connectsTrans ?t2.
```

```
  ?t1 h:connectsPlace/h:connectsTrans ?andjoin
  ?t2 h:connectsPlace/h:connectsTrans ?andjoin
  FILTER (?t1<?t2)
 }
```

Query 13.2 is used to query any pairs of activity nodes whose occurrences are alternative **choices**, but that may be executed in **sequence**.

```
SELECT ?t1 ?t2 WHERE
 {
  ?t1 rdf:type h:Transition;
      h:hasGuardFunction/h:hasActivity _:b1.
  ?t2 rdf:type h:Transition;
      h:hasGuardFunction/h:hasActivity _:b2.
  ?t1 k:choice ?t2;
      k:notSpecified ?t2.
  ?t3 rdf:type h:Transition;
      h:hasGuardFunction/h:hasControl _:b3.
  ?t1 h:connectsPlace/h:connectsTrans ?t3
  ?t3 h:connectsPlace/h:connectsTrans ?t2
  FILTER (?t1<?t2)
 }
```

Queries, which are used to query any pairs of activity nodes whose occurrences are in **mutual exclusion**, but they may be executed in **parallel** or in **sequence**, are created similar to Queries 13.1 and 13.2, respectively. In addition, SPARQL queries are also created similar to queries 13.1 or 13.2 in order to resolve the second and the third semantic issues.

- **Query 14** is created relating to the fourth semantic verification issue. Query 14 returns all pairs of activities whose occurrences are in **dependency** and whose order executions are defined as one **before** the other, but they may be executed in the opposite order.

```
SELECT ?t1 ?t2 WHERE
 {
  ?t1 rdf:type h:Transition;
      h:hasGuardFunction/h:hasActivity _:b1.
  ?t2 rdf:type h:Transition;
      h:hasGuardFunction/h:hasActivity _:b2.
  ?t1 k:dependency ?t2;
      k:before ?t2.
  ?t3 rdf:type h:Transition;
      h:hasGuardFunction/h:hasControl _:b3.
```

```
  ?t2 h:connectsPlace/h:connectsTrans ?t3
  ?t3 h:connectsPlace/h:connectsTrans ?t1
  FILTER (?t1!=?t2)
}
```

The SPAQRL queries used to solve the other cases of the fourth and the fifth issue can be created similarly to query 14.

5.3 A Wrong Workflow Example

Example 5.3.1. Let us continue *CompanyA* variant of the fOtD process presented in Section 1.2.2. Figure 5.5 illustrates an extraction of a wrongly designed CPN-based business workflow. The example workflow contains not only syntactic errors (e.g., a deadlock is caused by the combination of a $Xor-split$ and an $And-join$), but also semantic errors (e.g., the execution order between $Schedule\_shipping$ and $Receive\_shipping\_request$).

We assume that the input place of the transition $Xor-split$ contains a token that enables this transition. If the transition $Xor-split$ fires, it consumes the token from its input place and then produces one token for only one of the output places. Consequently, only one transition, i.e., $Free$ or $Charge\_6.80\_Euros$ or $Charge\_7.50\_Euros$ or $Charge\_10.00\_Euros$, can be activated. Because only one transition can fire, not all input places of the transition $And-join$ can get its token. Since the transition $And-join$ will never be enabled to fire, a deadlock occurs.

In addition, the tasks $Receive\_shipping\_request$ and $Schedule\_shipping$ are defined by the semantic constraint sc_i where:

```
sc_i=(dependency, Receive_shipping_request, Schedule_shipping,after,
     ''after receiving a shipping request, a shipment is scheduled'')
```

However, as shown in Figure 5.5, the execution of $Receive\_shipping\_request$ may happen after that of $Schedule\_shipping$. Consequently, a semantic error is found.

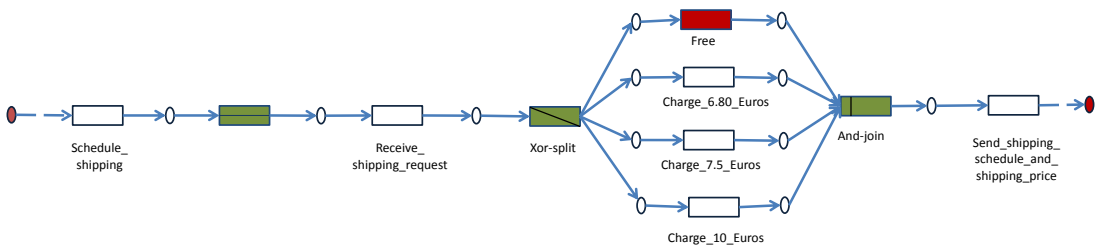

Figure 5.5: A wrongly designed workflow model for the fOtD process (excerpt)

As a result of the execution of each SPARQL query introduced in Section 5.1.3 and Section 5.2.2, we obtain an XML file which results in nodes consisting of required information (e.g., the name) and causes shortcomings. For example, Figure 5.6 shows the result of the execution of Query 12.1 applied to check whether the workflow, depicted in Figure 5.5, contains deadlocks or not.

```
<?xml version="1.0"?>
<sparql xmlns='http://www.w3.org/2005/sparql-results#'>
  ...
  <result>
    <binding name='xorsplit'>
      <uri>
              http://WFTemplate/Shipment#Receive_shipping_request_
              Xor-split
      </uri>
    </binding>
    <binding name='andjoin'>
      <uri>
              http://WFTemplate/Shipment#Send_shipping_schedule_and_
              shipping_price_And-join
      </uri>
    </binding>
  </result>
  ...

</sparql>
```

Figure 5.6: Checking deadlocks caused of the two control nodes $Xor - split$ and $And - join$

5.4 Related Work

In this section, we provide an overview of existing approaches with respect to workflow verification.

5.4.1 Approaches focusing on the Syntactic Level

Checking the correctness by verifying process models against structural requirements is a strategy mentioned in a number of related approaches. In the following, these approaches are classified based on the techniques used for conformity verification.

- **Petri Nets-based Approaches**

Petri Nets (PNs) is a class of modelling tools originated by Petri [Petri 1962]. PNs and their extensions have proven to be useful for the modelling and analysis of business processes. The existing research on PN-based workflows is referred to a concept called *Workflow nets* (WF-nets), which is a subclass of PNs. A Petri net $PN = (P, T, F)$ is a WF-net if and only if:

- It has two special places, a source place i and a sink place o; and
- If a transition t connects the place i with the place o, the resulting PN is strongly connected.

The verification of WF-nets concentrates on the so-called *soundness property*. The property involves a certain number of issues, such as liveness, boundedness, safeness, livelock, deadlock and dead activity [van der Aalst 1997, van der Aalst 2000]. Furthermore, a sound WF-net always terminates properly, i.e., at the moment the WF-net terminates, the place o contains one token and there are no tokens anywhere else.

Using the PN formalism brings significant advantages, such as a formal theory base, the representation of workflow states is based on tokens and its tools for analysing and verifying business workflows (e.g., Woflan [Verbeek 2001]). However, with regard to the soundness verification, only the control flow perspective of workflows is covered. It is essential to note that soundness is a necessary but insufficient condition to verify workflows. Therefore, the issues related to the semantic correctness of workflows need to be taken into account.

- **Model Checking Approaches**

  Model checking is well-researched and therefore many languages, techniques and tools are provided. It provides techniques for verifying a system specification (i.e., a model) against certain particular properties [Clarke 2001]. As depicted in the Figure 5.7, the formal model defined in a language suitable for the model checker's input language and the system property which needs to be checked, are given as inputs to the model checker. The model checker after that is invoked. In case the property could not apply, the model checker typically generates a counterexample.

  In order to specify properties, there are many different languages available, such as, temporal logics (e.g., the Linear Time Logic (LTL) or the Computation Tree Logic (CTL)) or automatons. Both these logics are well-researched and can be seen as decidable notational variants of "modal" fragments of first-order logic [Hustadt 2004]. However, the weakness of temporal logics is that due to their complexity [Dwyer 1999], it is not easy for practitioners who are non-experts to specify system properties.

  A variety of approaches adopt model checking for business process model verification, such as [Förster 2007, Knuplesch 2010, Khaluf 2011, Feja 2011]. Förster et al. in [Förster 2007] introduce an approach which allows the verification of certain constraints like domain specific or quality management

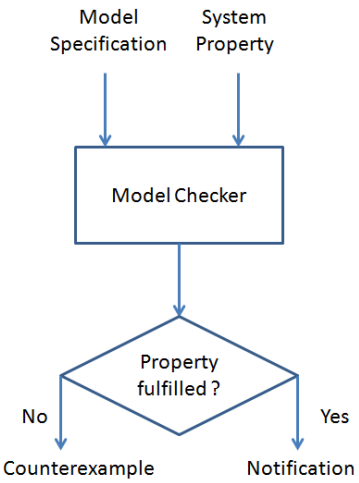

Figure 5.7: A typical model checking workflow

requirements, so-called quality constraints. The process pattern definition language (PPSL), an extension to UML Actitivies, is used for the specification of these constraints. The PPSL patterns, in turn, can be transformed into specifications in linear temporal logic (LTL) while the business process, which is modelled by using UML Actitivies, is transformed into a transition system. This technique allows formal verification of process constraints in business processes. Although model checking provides techniques for the verification of a given model against a certain specification property, it has not been concerned with ontologies, i.e., ontology axioms play a role as part of the model to be checked.

- **Graph Reduction** Graph reduction [Kovalyov 1990, Esparza 1994, Sadiq 2000] was developed to detect structural shortcomings like deadlock or missing synchronization while specifying large and complex business processes. After eliminating the structures which never cause anomalies, the workflow model is reduced.

  [Sadiq 2000] introduces the five reduction rules iteratively applied to retain vertices in a model. These rules are terminal reduction, sequential reduction, adjacent reduction, closed reduction and overlapping reduction. By reducing the graph repeatedly, computational efficiency is improved. If a model contains any deadlocks or missing synchronizations, it is impossible to completely reduce to an empty graph. The time complexity of their main graph reduction algorithm in the worst case is $O(n^2)$. However, this graph reduction technique is not applicable to process models containing cycles. Furthermore, although special overlapping structures can be verified by applying these graph reduction rules, it is hard or even impossible to handle general overlapping structures.

With regard to PNs, the authors in [Esparza 1994] explore the reduction and synthesis techniques for analysis of well-formed PNs[3]. They introduce the complete kits of reduction rules, including abstraction and linear dependency rules, for the analysis of well-formed PNs. A free-choice[4] net is transformed into a simpler one by a reduction rule while maintaining well-formedness. This means that the original net is well-formed if and only if the reduced net is well-formed. This reduction algorithm runs in polynomial time on the size of the system. It can be easily transformed into an algorithm to check liveness and boundedness of free-choice systems. More importantly, the algorithm can be reversed to create a synthesis algorithm, which is used for the stepwise construction of large systems. We use the reduction algorithm presented in [Esparza 1994] to transform a workflow template into a simple one to detect deadlock, infinite cycle and missing synchronization.

5.4.2 Approaches focusing on the Semantic Level

The verification of process models has been studied mostly from the control flow perspective. However, as mentioned previously, in order to ensure that a business model is built correctly, issues beyond pure control-flow verification also need to be taken into account.

- **Correctness beyond Formal Semantics**

  Recently, some research has gone beyond the syntactic and formal semantics, especially in the context of compliance. Most approaches in this area focus on detecting compliance violations related to the model structure or execution semantics [Goedertier 2006, Lu 2008, Awad 2008]. [Lu 2008] introduces a very interesting approach to support process designers in quantitatively measuring the compliance degree between a given process model and a set of control objectives. The calculation of the ideal and sub-optimal compliance degree starts with the extraction of the set of ideal and sub-optimal execution sequences for each control rule. The degree of support for these sequences in the process model is then calculated. This allows process designers to measure how well a given process model represents the ideal and sub-optimal situations in control rules as well as to be better informed on the cost of non-compliance. Some approaches also consider running processes, such as [Ly 2008, Kumar 2010, Ly 2012]. [Ly 2008, Ly 2012] introduce techniques to ensure semantic correctness for single and concurrent changes at process instance level. Their approach checks a notion of semantic correctness based on annotations for tasks. A process is semantically correct if it complies with the annotations. Semantic constraints, which are defined over processes, are

[3] Well-formed PNs are a restrictions of the high-level nets. The main advantage of well-formed PNs is the notion of symbolic reachability graph that is composed of symbolic states [Chiola 1995]

[4] According to [van der Aalst 1997], a Petri Net is a free-choice Petri Net if and only if, for every two places p_1 and p_2 either $p_1 \bullet \cap p_2 \bullet = \emptyset$; or $p_1 \bullet = p_2 \bullet$ where $p \bullet$ denotes the set of transitions sharing p as an input place

used to detect semantic conflicts caused by violation only of dependency and mutual exclusion constraints.

Although these approaches concentrate on aspects of semantic correctness, in contrast to our work they do not mention about the use of a standard ontology language such as OWL.

- **Ontology-based Correctness Checking**

  With regard to ontological approaches, aspects of semantic correctness are considered in some research, such as [Thomas 2009b, Weber 2010, Fellmann 2011].

  The approach of [Weber 2010] focuses on annotated business processes to capture what the process activities actually do when executing them. The individual activities in process models are annotated with logical preconditions and effects, specified relative to an ontology. Therefore, both the annotation of preconditions and effects are required to verify the overall process behavior which stems from the interaction between control-flow and behavior of individual activities. Although this approach combines syntax for control flow and also semantic annotation but the ontology is not built on a formal representation of the semantics of individual activities.

  In [Thomas 2009b, Fellmann 2011], individual model elements are annotated with concepts of a formal ontology. And the SPARQL query language is thus used to check the semantic correctness of ontology-based process representations. Constraints are characterized in four basic types (i.e., element flow, element occurrence, resource usage and resource occurrence). They are formalized as SPARQL queries which are executed against the ontology-based process representation. Furthermore, the work in [Fellmann 2011] provides a very useful inspiration for our work, but it does not cover aspects related to the grammar of the modelling language used and checking the absence of deadlocks and livelocks.

5.5 Discussion and Conclusion

In this chapter, we focused on verifying business process templates at the syntactic and semantic level. At the syntactic level, we have described two groups of constraints that ensure the soundness of workflow templates. We have concentrated on defining the axioms corresponding to the syntactic constraints and the axioms involving the use of control nodes. At the semantic level, we have introduced the five semantic verification issues related to a workflow template.

We have also introduced the SPARQL queries, which are related to the syntactic constraints and the semantic verification issues, to check the correctness of concrete CBWTs. By relying on Jena[5], which is a free and open source Java framework to build Semantic Web and Linked Data applications, we have demonstrated not only

[5] https://jena.apache.org/index.html

the usage of the SPARQL query language for syntactic and semantic checks, but also the usage of terminological background knowledge provided by the CPN ontology and the BP ontology. The results of this work were published in [Nguyen 2014a, Nguyen 2014b, Pham 2015]

Chapter 6

Reuse of Workflow Templates

Contents

Nowadays, business process models have been used in a wide area of enterprise applications. Along with their popularity, interest is growing in how to create them correctly in terms of semantics and syntax while boosting the efficiency of reusing suitable parts of existing models are growing.

Let us consider the following scenario. A person plans to create an ordering process for his own purpose. He has either some experience in working on it or none at all. The question is how he can create his process model in the most effective way without developing it from scratch.

In fact, the different existing workflow templates extracted from a set of process models can support modellers to create new workflows or process models by providing the knowledge about potential and suitable workflow activities. Therefore, in this chapter, we focus on the reuse of workflow templates.

We are interested in the organization of the knowledge base which guides the search for suitable workflow templates in order to reuse them. Users can adapt the resulting workflow templates as well as their ECA-like rules for each specific use case. This is the knowledge on how to model a business process reusing control flow-based business workflow templates (CBWTs). Hence, the annotation and storage of workflow templates play a very important role in the success of reusable CBWTs.

This chapter describes the main ideas about the organization of the knowledge base of workflow templates in order to guarantee an effective search for modelling a business process.

6.1 Organization of the Knowledge Base of Control Flow-based Workflow Templates

In literature, the main goals of workflow reuse are to improve workflow template quality and to increase its development productivity [Kradolfer 2000]. In other

words, the more workflow templates are available, the more difficult they are to be suitable in a specific reuse case. It is also important to note that the reuse of workflow templates is only beneficial if the cost to find and adapt an existing workflow template is smaller than the cost needed to develop a new one from scratch.

After finding suitable workflow templates, it is important for users to understand what the workflow templates actually do. Thus, there is a strong need that the knowledge base of workflow templates could provide enough information for modellers to be able to determine which template is suitable for the reuse case at hand.

It is important to note that the development of a workflow template relies on a set of semantic constraints and the structure of CPNs (cf. Chapters 3 and 4). The workflow template is formalized by an RDF graph in which the dependencies between its activities are expressed. Besides, to provide adequate support for specifying business rules of a workflow template that the set of semantic constraints cannot capture, a set of ECA-like rules stored in RDF format is proposed.

We propose a method to semantically annotate workflow templates. Their retrieval through meta-workflow templates will model expert knowledge and guide the use of existing workflow templates. The idea of using content which characterizes workflow templates is not original. Indeed, it seems reasonable to use explicit information to find suitable templates to build a business workflow. This is particularly important for workflow modellers to be able to deal with the great number of workflow templates.

Based on the analysis of the state-of-the-art concerning the organization and reuse of workflow templates, we annotate workflow templates by the following properties:

- *templateName*: Description of the main task being enacted by the template.
- *description*: Description of the template.
- *keywords*: List of words that characterizes the template. It also includes the words that name the template.
- *listOfActivityLabels*: The labels are extracted from activity labels in the template.
- *creationDate*: The date when the template is created.
- *modificationDate*: The date the template is last modified.
- *relatedTemplates*: List of related templates (if any). The related templates can be predecessors and successors of the template.
- *listOfECARuleFiles*: List of the rule files defined for the workflow template.
- *bpOnt*: Indicating the business process ontology used to develop the template.

The properties *templateName*, *description*, *keywords* and *relatedTemplates* are determined by using expert knowledge. In contrast, the values of the properties *creationDate* and *modificationData* are automatically captured at the moment of storing the template. Depending on all the activity labels in the template, the value of the property *listOfActivityLabels*[1] is automatically retrieved. For example, to get all activity labels of the template *http* : *//WFTemplate#Payment_Processing*, the following SPARQL query is first executed to get all IDs of its transitions:

```
SELECT distinct ?trans WHERE
{
        k:Payment_Processing h:hasTrans ?trans
}
```

Then the labels of these transitions are cut from their IDs and added into the list of activity labels. The properties *listOfECARuleFiles* and *bpOnt* capture the names (or URLs) of the rule files defined for the workflow template and the business process ontology file, respectively. These properties lead us to the representation of additional knowledge that facilitates modellers to search for suitable templates, which can be used to design a new one.

As a result, we propose a semantic annotation of workflow templates which expresses knowledge relative to their properties. The expert knowledge is captured as RDF annotations to conduct users to model new business processes. Figure 6.1 illustrates a simplified example of such semantic annotation.

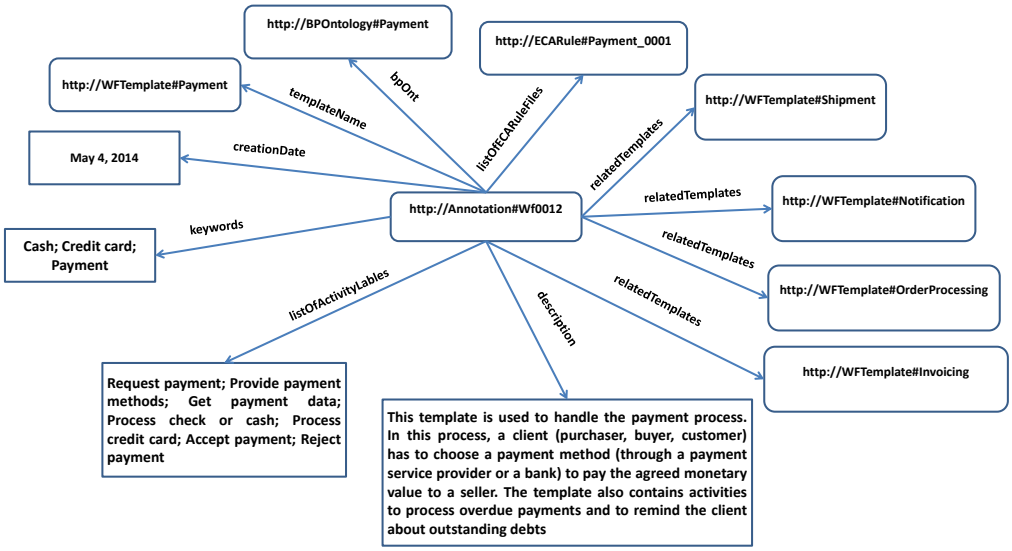

Figure 6.1: Example of the semantic annotation of a workflow template

We also develop an ontology to annotate workflow templates. The ontology

[1]The problem of labelling workflow activities is introduced in Appendix C

describes the main classes and properties for RDF annotations of workflow templates (see Figure 6.2).

In fact, the semantic annotations of workflow templates have been inspired by this idea: the knowledge added into these annotations will be helpful for the (re-)use of workflow templates along with their ECA-like rules. Those meta-workflow templates allow retrieving a list of workflow templates (and also a list of ECA-like rules) that correspond to different criteria. For example, to acquire all existing workflow templates relating to payment by credit card, two criteria are used: (i) one keyword of such a template is *credit card*; (ii) description of such template contains *payment procecss*. This can be performed by the SPARQL[2] query as follows:

```
SELECT * WHERE
{
 ?workflow anno:keywords ?keyword
 FILTER (?keyword ~ "credit_card"^^xsd:string)

 ?workflow anno:description ?descr
 FILTER (?descr ~ "payment_process"^^xsd:string)
 ?workflow      anno:templateName ?name;
                anno:listOfActivityLabels ?actLabel;
                anno:relatedTemplates ?relatedTemp;
                anno:listOfECARuleFiles ?ecaRule;
                anno:creationDate ?crtDate;
                anno:modificationDate ?modDate
}
```

It is important to emphasize that those meta-workflow templates allow retrieving workflow templates, which are annotated with additional expert knowledge formalized with the help of the CPN ontology, the BP ontology and also the sets of ECA-like rules. In the following we introduce an excerpt of the RDF annotation related to the workflow template $http://WFTemplate\#Payment$ depicted in Figure 6.1.

```
<rdf:RDF
xmlns ="http://ontWFTemplateAnnotationsURI.owl#"
xmlns:wf="http://WFTemplate#"
xmlns:rule="http://ECARule#"
... >
  <TemplateAnnotation rdf:ID="wf0012">
   <templateName rdf:resource="http://WFTemplate#Payment"/>
    <keywords>Cash;Credit card; Payment; Payment processing
    </keywords>
    <listOfActivityLables>Request payment; Provide payment methods;
     Get payment data; Process check or cash; Process credit card;
```

[2] $PREFIX\ anno :< http://ontWFTemplateAnnotationsURI.owl\# >$

```
     Accept payment; Reject payment
    </listOfActivityLables>
    <description>Template payment processing is used to handle
     the payment process...
    </description>
    <relatedTemplates rdf:resource="http://WFTemplate#Invoicing"/>
    <relatedTemplates rdf:resource="http://WFTemplate#
     OrderProcessing"/>
      ...
     <listOfECARuleFiles rdf:resource="http://ECARule#
     Payment_0001"/>
     <bpOnt rdf:resource="http://BPOntology#
     Payment"/>
  ...
  </TemplateAnnotation>
</rdf:RDF>
```

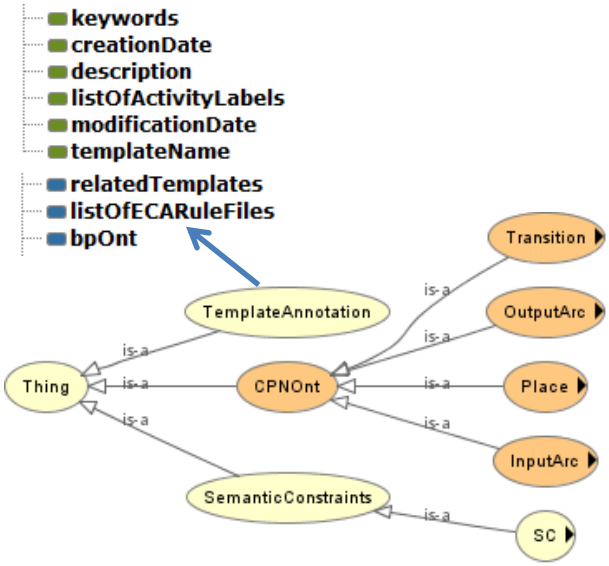

Figure 6.2: Extract of the annotation ontology used to annotate workflow templates

c

6.2 Process for Developing Workflow Templates

In this section, we introduce a process for developing workflow templates, which is regarded as part of the process for developing an encompassing workflow application.

The process consists of the main following phases (see Figure 6.3):

1. **Search for reusable workflow templates**: An analysis of the process(es) is performed before implementing it. This results in a set of requirement descriptions as well as a business process model. The information is then used to start the process for developing workflow templates which may involve the search for reusable workflow templates.

2. **Understand and select potential, suitable templates**: In this phase, modellers have to carefully consider the found workflow templates. They try to understand them to decide which ones are (partly or fully) reused for their application.

3. **Modify selected templates**: If the selected templates do not comply with all the requirements, they have to be modified accordingly. For example, some new activities can be added into a selected template.

4. **Create new sub-workflow templates**: Besides reusing part or all of the existing templates, modellers might have to create new sub-workflow templates to meet all the requirements. However, the creation of a new sub-workflow template is only necessary if no existing templates can be reused instead for the same purpose.

5. **Complete workflow templates**: The last phase is to complete a new workflow template. The existing unmodified, modified and new sub-workflow templates are integrated into a new workflow template for a specific use case. Each of these workflow templates is considered as a sub-workflow of the new workflow template. It is then verified at the syntactic and semantic level. In case of errors, the errors have to be solved. The new workflow template is stored in the CBWT repository if and only if: there exist no syntactic errors nor semantic errors; and at least one set of ECA-like rules is defined for the new workflow template.

To find suitable workflow templates, users can define their criteria by keyword, by description or by activity label. If the search process returns only one template, users can easily make their decision that the template is selected or not selected. Otherwise, the value of the property *RelatedTemplates* can be used to provide more information for users to make their decision.

To sum up, the semantic annotations of workflow templates integrating expert domain knowledge formalized by an RDF graph are used to organize and retrieve workflow templates, their business process ontologies and their sets of ECA-like rules. The resulting templates and their rules can be used in a process for implementing software components or in a process for developing workflow templates.

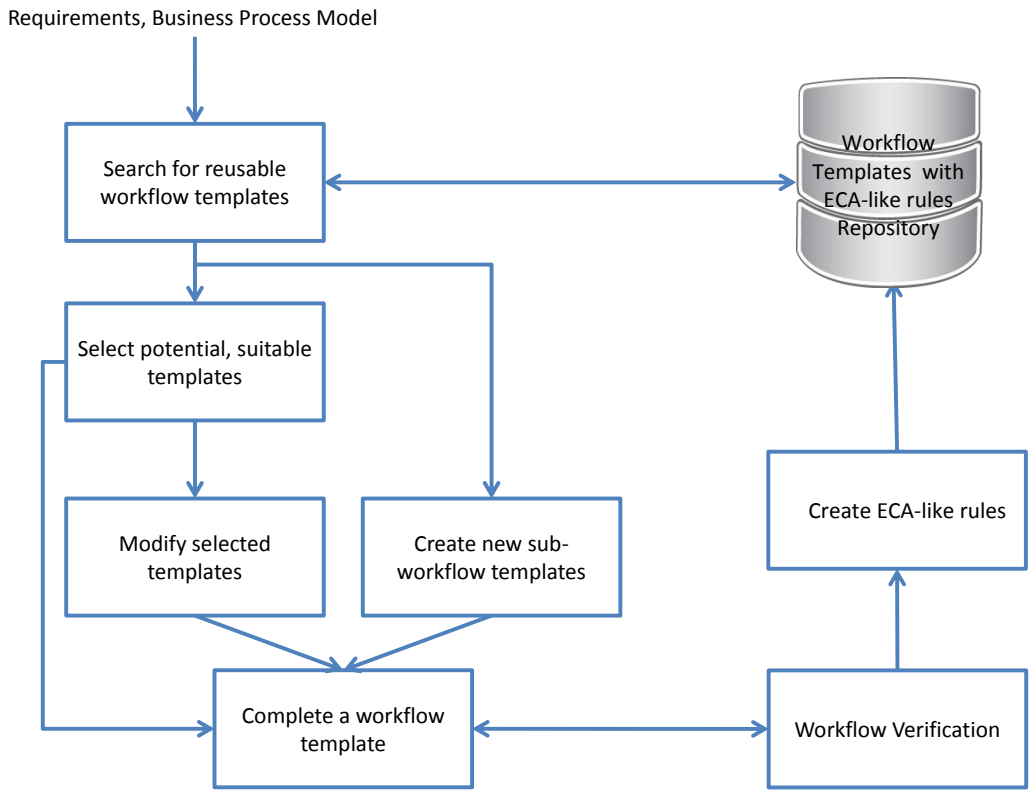

Figure 6.3: Development of reuse-based workflow template

6.3 Related Work

Up to now, the problem of reusing process models or workflows is mentioned in some existing approaches. In general, workflows can be reused manually or semi-automatically [Markovic 2008, Lu 2009, Koschmider 2015]. Moreover, modellers can partly or fully reuse a workflow [Mendling 2006, Eshuis 2008, Koschmider 2011, Koschmider 2015].

The authors in [Mendling 2006] specify a method for business process design by view integration which takes two process views as input. At first, semantic relationships between elements of different process models are formalized. On this basis, the integrated process model applying the merge operator is calculated. [Eshuis 2008] also presents a formal approach for constructing customized process views on structured process models to improve effective cross-organizational collaborations. Each customized process is constructed by hiding and/or omitting activities not requested by the process consumer. However, neither of them considers content-based reuse. In order to overcome this issue, the authors in [Koschmider 2015] introduce a set of Domain Process Patterns (DPPs) that capture process model parts. A DPP represents a specific business function of a process model part in a modelling domain. DPPs facilitate reuse from a content perspective by focusing on domain-centered

reuse of process model content. Nevertheless, DPPs do not provide any syntactic needs for modelling business processes. However, by capturing process model parts with a particular structure, DPPs do not support syntactic checks which are supported in our approach.

6.4 Discussion and Conclusion

The concepts, which have been introduced in the previous chapters, provide useful support for the development and modification of workflow templates, whereas the tasks of searching and understanding workflow templates have not been mentioned. Therefore, in this chapter, we have presented a process for developing workflow templates, which specially emphasizes the different phases of workflow template reuse comprising the tasks of searching, understanding and modifying workflow templates.

Moreover, in order to better support the search for suitable workflow templates, the annotation ontology has been developed to annotate workflow templates. The ontology provides adequate information about the workflow templates and their ECA-like rules for workflow modellers to determine whether a workflow template is able to be reused.

Chapter 7

Prototype

Contents

In this chapter, we present an overview of the CBWT prototype that is implemented to validate the concepts presented in the previous chapters. It is necessary to underline that the prototype is not developed to become a full-fledged workflow template management system. It is just a proof of concept which supports modellers in developing a new workflow template from a set of semantic constraints and/or by reusing some existing workflow templates.

The rest of this chapter is divided into four sections: Section 7.1 introduces an overview of the functionality of the prototype. In Section 7.2, details of the implementation are presented. In Section 7.3, we discuss the evaluation of the prototype. Finally, a conclusion of the chapter is given in Section 7.4.

7.1 Introduction

In order to validate our approach for representing semantically Control flow-based Business Workflow Templates (CBWTs) in a knowledge base, we implement the CBWT prototype allowing us to develop, verify and reuse workflow templates. The conceptual architecture of the CBWT prototype is depicted in Figure 7.1. The functionality of the CBWT prototype corresponds to the main components of our process for developing workflow templates.

At the upper part of Figure 7.1(a), a use case model[1] shows the functionality of the CBWT prototype. There are two types of actors consisting of expert user (i.e., the workflow modeller) and end-user, who interact with the prototype. The communication associations between actors and use cases are represented by arrows. The direction of each arrow is used to indicate the entity (either actor or use case) initiating the communication.

The current version of the CBWT prototype focuses on supporting expert users in the workflow template development process. Therefore, in the following we describe six use cases intended for expert users (i.e., modellers), which are provided by the CBWT prototype:

- **Search for workflow (WF) templates**: Users can search for potential, suitable workflow templates through search criteria as keywords, description and even activity labels.

- **Browse workflow templates and ECA-like rules**: Users can browse a workflow template via an interface illustrated in Figure 7.2. On the left side of the form (see area (1)), a list of workflow templates is shown. By clicking on a workflow template, the information concerning the workflow template is shown in the right part of the form (see area (2)). To browse the detail of a set of ECA-like rules of the workflow template, users can click on its name (see area (3)) and all the rules in that selected set will be displayed in the other form (Figure 7.3). The lower part of the form displays the set of semantic constraints used for developing the template (see area (4)). There are also the set of buttons used to modify the template (see area (5)).

- **Download workflow templates along with their ECA-like rules**: Users can download couples of a workflow template and a set of ECA-like rules defined for the workflow template in RDF format. It is done by selecting the name of the workflow template and the name of ECA-like rules and then clicking on the button **Download Workflow Template**.

- **Modify workflow templates and ECA-like rules**: A couple of a workflow template and a set of ECA-like rules can be modified and updated by modellers (expert user) by applying the manipulation operations (see Chapter 3). A modified workflow template is not stored in the repository if there exist syntactic or semantic errors. With regard to modifying the set of ECA-like rules, if a modification operation would violate one of the ECA-like rules, it is not performed and modellers are informed by a notification. For example, when modellers try to define a duplicate rule for an activity in a workflow template, an error message is sent out.

[1] A use case model describes the proposed functionality of a new system. Two main constructs of a use case model are actors and use cases. An actor, which can be a human or an external system or time, represents a role played by an entity that interacts with the system. A use case represents what is done by the system [Booch 2005].

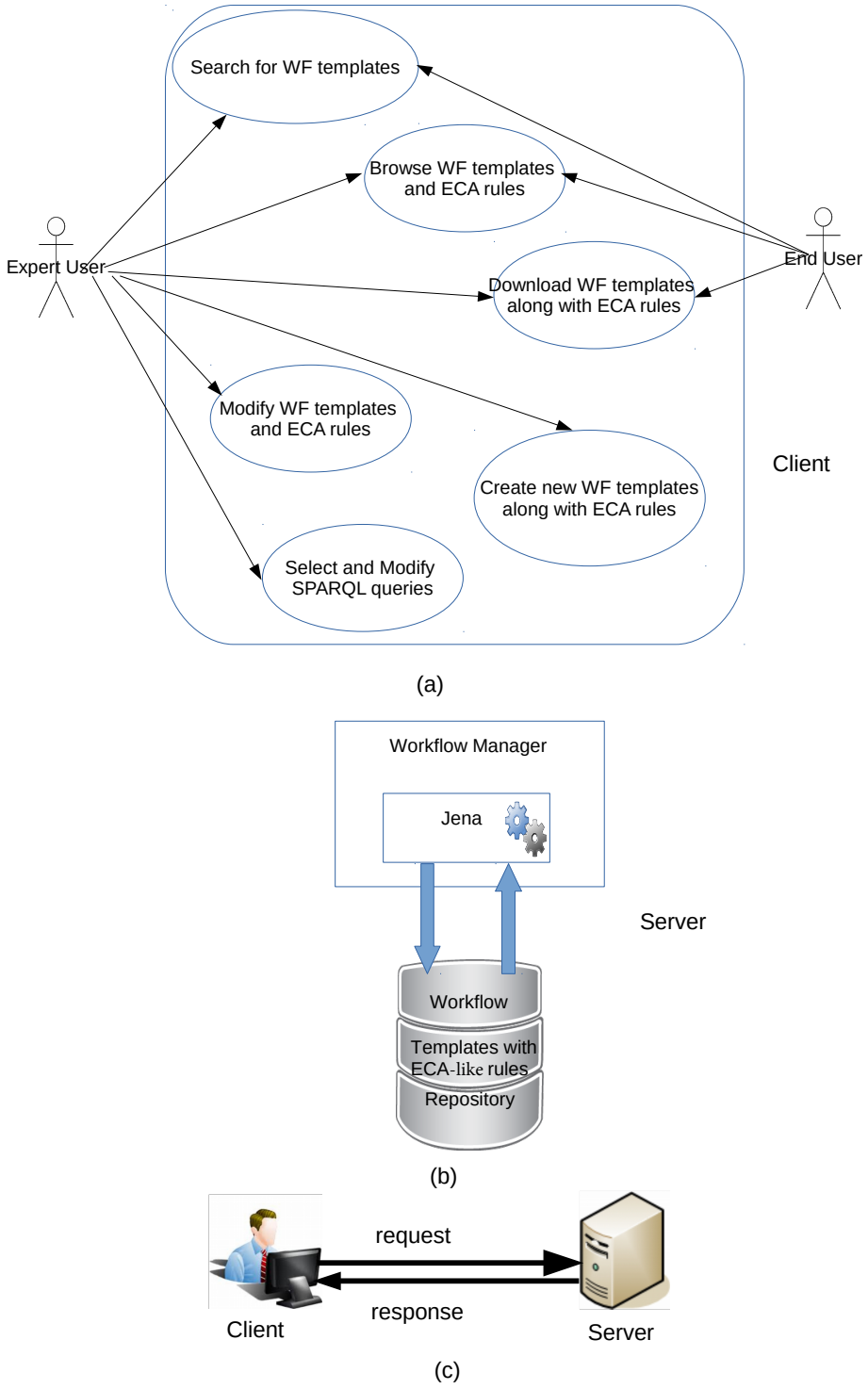

Figure 7.1: The conceptual architecture overview of the CBWT prototype

- **Create new workflow templates along with their ECA-like rules**: A new workflow template can be developed from scratch or by reusing the existing unmodified or modified workflow templates.
- **Select and modify SPARQL queries**: Modellers can choose a level (semantic or syntactic or both) to verify a workflow template. There is a set of SPARQL queries corresponding to the set of constraints presented in Chapter 5 that modellers can select for workflow verification. Modellers can also modify the existing queries or define new queries. When the workflow template is well-verified (there are no errors in the workflow template,), modellers can define a set of ECA-like rules for the workflow template. The workflow template and its set of ECA-like rules are then saved in the repository.

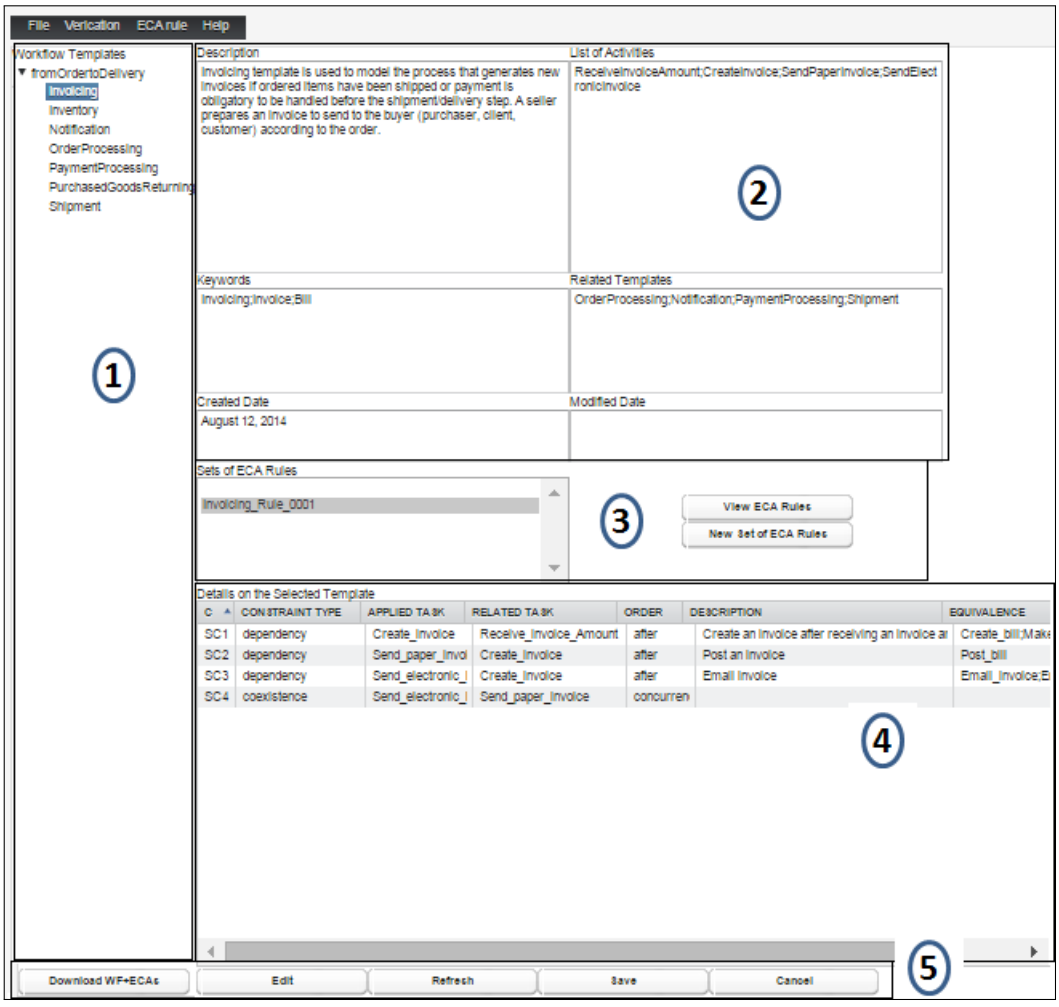

Figure 7.2: Interface used to browse and update workflow templates

The serve-side contains two components as follows:

- **Workflow manager** maintains workflow templates and provides the manipu-

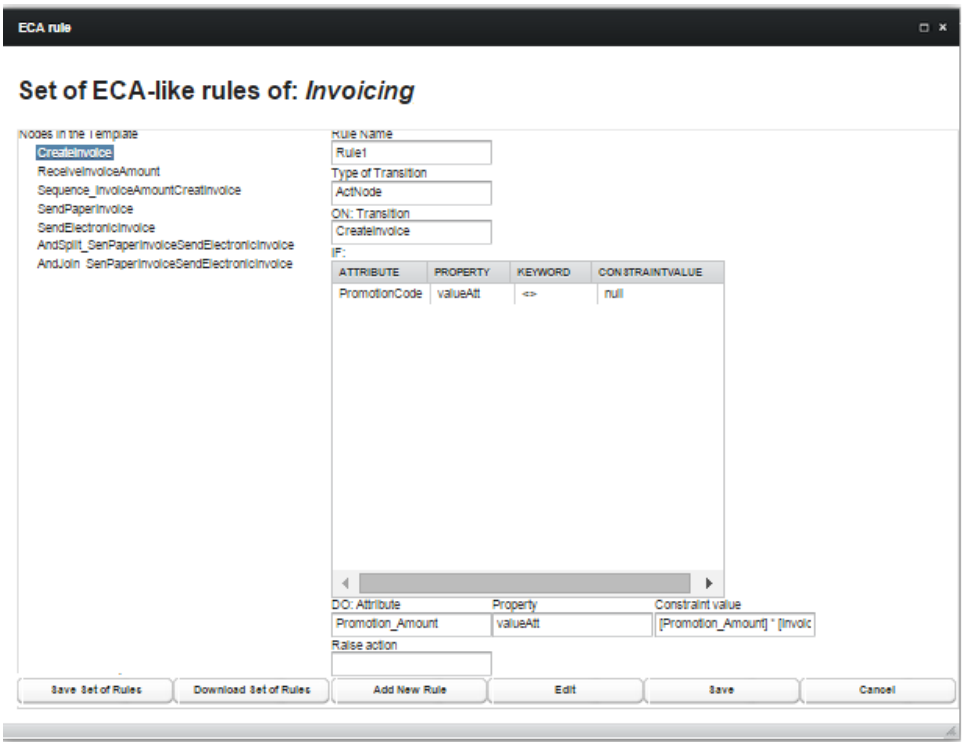

Figure 7.3: Interface used to browse and update ECA-like rules

lation operations to modify and update the workflow templates and ECA-like rules. It supports the modification of workflow templates to avoid incorrect results.

- **Workflow Templates with ECA-like rules Repository** contains the high quality workflow templates, verifying the syntactic and semantic correctness, their business process ontologies and ECA-like rules.

7.2 Technical Implementation of the CBWT Prototype

This section describes the implementation of the CBWT prototype. First, we briefly introduce the Web technologies and software tools which are used to develop our prototype and make it work. Second, we describe technical details of the CBWT prototype as a simple Web application.

7.2.1 Web Technologies and Software Tools

We hereinafter briefly introduce the standard technologies used to implement our workflow template development process model. For the verification of workflow templates, the CBWT prototype relies on Jena, an open source Semantic Web framework for Java, for querying RDF data sources.

The prototype has been encoded with the following technologies:

- Java programming language[2] was originally developed and firstly released as Java 1.0 in 1995 by Sun Microsystems[3]. The development of the prototype is done with the Eclipse[4] 4.3.2 Platform (Kepler), which is an integrated development environment (IDE) comprising extensible application frameworks, tools, and a runtime library for software development and management.

- Vaadin Framework[5] is a Java web application development framework that enables creation and maintenance of high quality web-based user interfaces. It supports different programming models, including server-side and client-side. Programming with Vaadin helps programmers to forget the web and to just program user interfaces. It looks like they are programming a desktop application with conventional Java toolkits such as AWT, Swing or SWT but easier. We use Vaadin 7[6] to develop the prototype.

- Jena[7] is a free and open source Java framework for building Semantic Web and Linked Data applications, originally developed by researchers in HP Labs[8] in UK in 2000. Jena provides extensive Java libraries for developing code that handles RDF, RDFS, RDFa, OWL and SPARQL in accordance with published W3C recommendations. It contains a rule-based inference engine, which can perform reasoning based on OWL and RDFS ontologies. It also contains a number of storage strategies to store RDF triples in memory or on disk. The prototype is developed with the version apache-jena-2.12.0[9].

In the following, we introduce some interfaces used for the development process of workflow templates.

7.2.2 Definition of User's Scope of Interest to Search for Relevant Workflow Templates

The functionality shown in Figure 7.4 allows users to define criteria to search for workflow templates, which are appropriate to a business process model they want to develop. Criteria can be defined as keywords, description and activity labels.

- **Input**: The criteria provided by a user to start a search for relevant templates.

- **Process**: Matching the desired values with the values of the corresponding attributes in the annotation ontology.

- **Output**: A set of relevant templates contains a series of the potential, suitable activities along with their ECA rule files.

[2] http://www.oracle.com/technetwork/java/index.html
[3] Oracle acquired Sun Microsystems in 2010
[4] http://www.eclipse.org/
[5] https://vaadin.com/home
[6] https://vaadin.com/download
[7] https://jena.apache.org/index.html
[8] http://www.hpl.hp.com/
[9] https://jena.apache.org/download/index.cgi

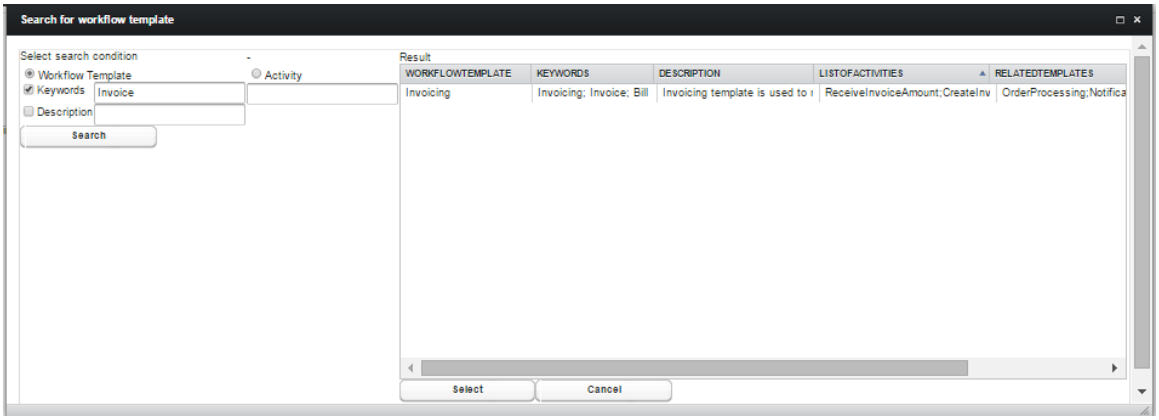

Figure 7.4: Interface of the definition of criteria for searching templates

7.2.3 Creation of a new Semantic Constraint

In order to develop a high quality workflow template, a set of elements (e.g., activities, control nodes, business rules) of the workflow template as well as the relationship between them must be defined.

The functionality shown in Figure 7.5 allows users to input all the necessary information to specify a semantic constraint for a given model.

- **Input**: The information provided by a user to create a new semantic constraint. It is named automatically or manually.

- **Process**: A set of necessary information that is filled to create a new semantic constraint. The new constraint is then checked with the set of existing ones to avoid duplication. The new constraint is regarded as a duplicate of the other ones when their four parameters consisting of *constraintType*, *appliedTask*, *relatedTask* and *order* are the same.

- **Output**: A new semantic constraint is stored if it does not duplicate any existing ones in the set of defined constraints semantics. Otherwise, a notification will be sent out.

7.2.4 Creation of a new Workflow Template

The functionality shown in Figure 7.6 allows users to complete the preparation of a new workflow template.

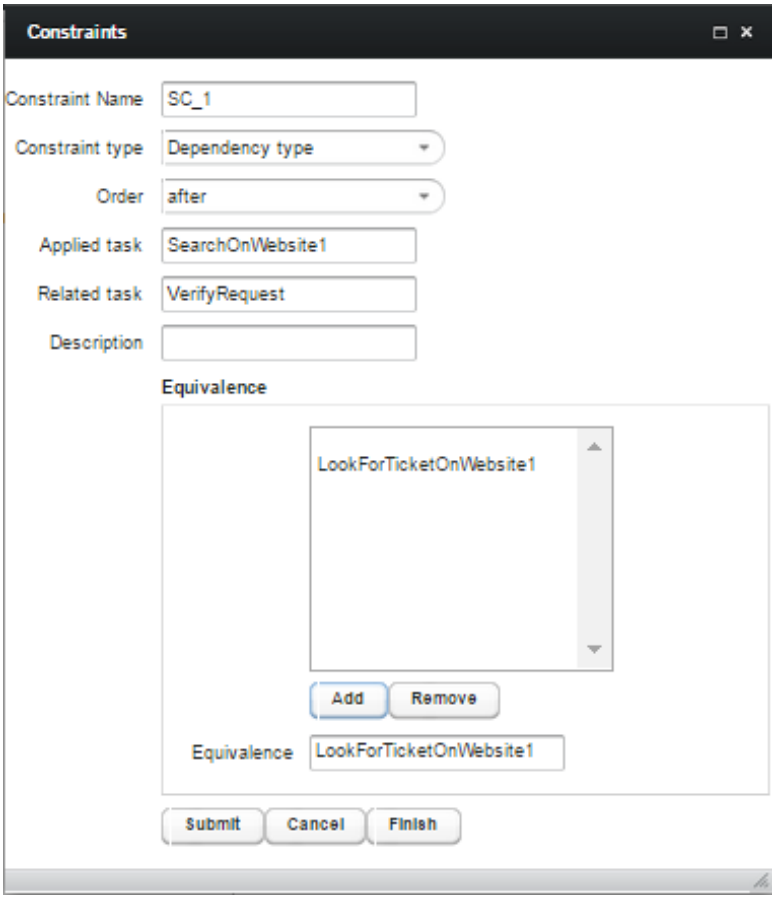

Figure 7.5: Interface of the creation of a semantic constraint

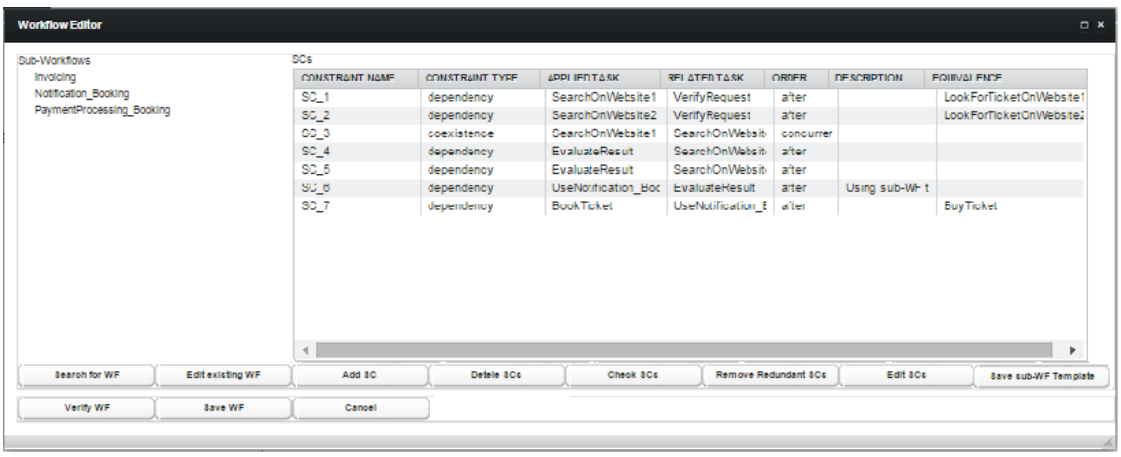

Figure 7.6: Interface of the development of a new template

- **Input**:

  (i) Information provided by the workflow designer to create a workflow tem-

plate;

(ii) The CPN ontology.

- **Process**: A workflow template can be developed by integrating the existing unmodified, modified and new workflow templates. Therefore, the first step of the process for developing workflow templates is to locate reusable workflow templates. The second step is to understand and select potential workflow templates. The third step is to modify the chosen templates if needed. And the last step is to add a set of semantic constraints which is used to complete the new workflow template.
- **Output**: The preparation of a new workflow template in RDF format.

7.2.5 Checking Redundant and Conflicting Semantic Constraints

The functionality shown in Figure 7.7 allows users to valid the set of semantic constraints.

- **Input**: A set of semantic constraints.
- **Process**: The set of semantic constraints have to be checked whether they contain redundant, conflicting constraints or not before it is used to develop a new workflow template. The check is done by applying Algorithm 1. The variable *checkRedundance* (shown in Figure 7.7 as the column *REDUNDANCE*) consists of two possible values: *True* if a semantic constraint is redundant and *False*, otherwise. The variable *checkConflict* (shown in Figure 7.7 as the column *CONFLICT*) has integer values: value -1 means that the constraint does not conflict with the other constraints; a positive value means that the constraint conflicts directly with the other constraint; a negative value (except value -1) means that the constraint conflicts with an inferred constraint.
- **Output**: The result of checking redundant and conflicting semantic constraints.

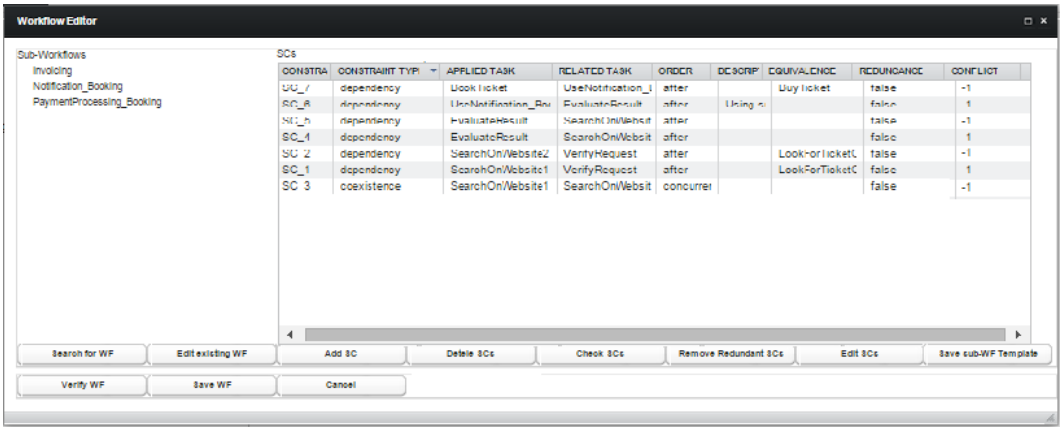

Figure 7.7: Interface for checking redundant and conflicting constraints

7.2.6 Workflow Template Verification

The functionality shown in Figures 7.8 and 7.9 allows users to check the correctness of workflow templates at the syntactic and semantic level.

- **Input**: A workflow template.
- **Process**: Workflow verification is executed by matching an RDF graph representing a workflow template to graph patterns of SPARQL queries concerning the syntactic and semantic constraints (see Chapter 5). If there is at least one match, an XML file is returned to indicate why the errors occur (e.g., see Figure 5.6). The workflow template has to be repaired until it is well-verified and thereafter the workflow template is added to the workflow templates repository.
- **Output**: An XML file which results nodes comprising required information and causes errors.

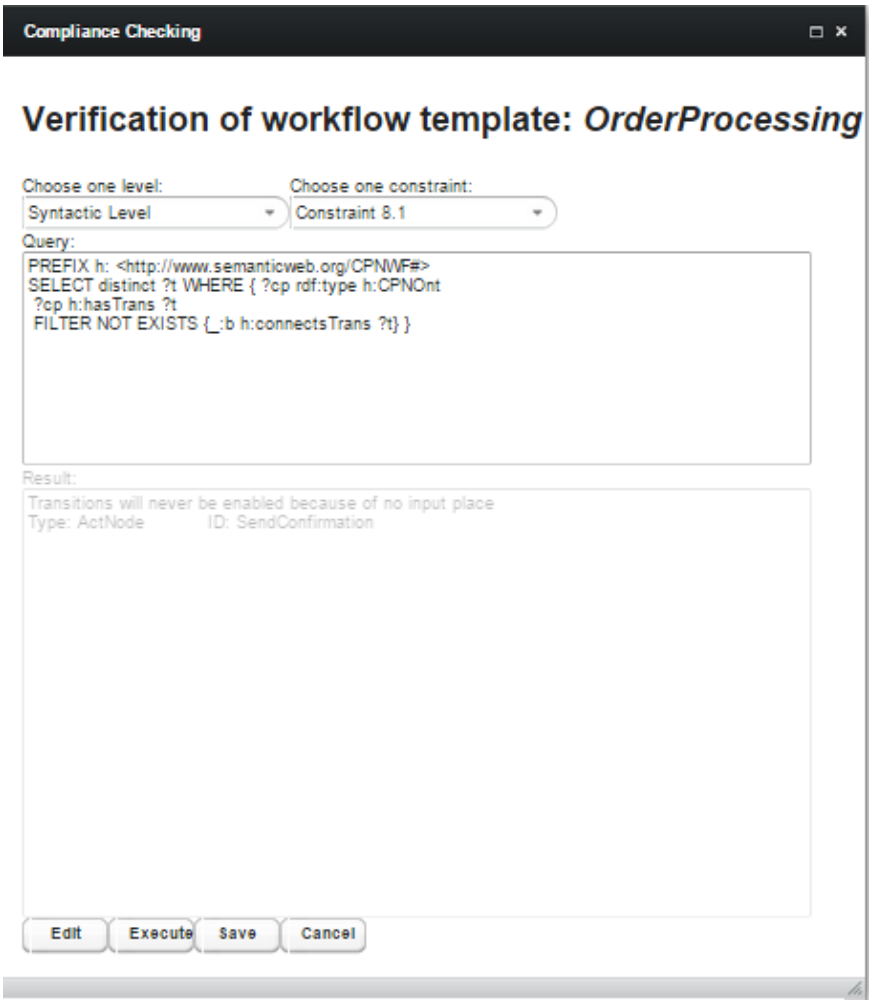

Figure 7.8: Verifying and reporting non-compliance results at the semantic level

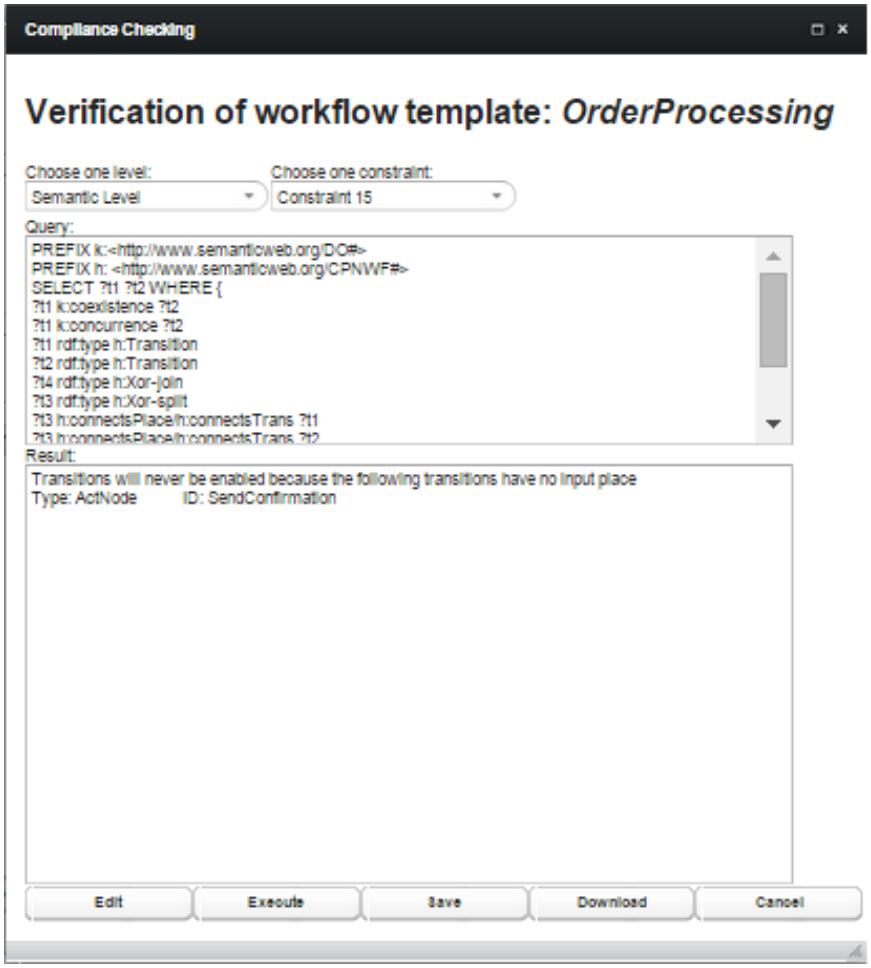

Figure 7.9: Choosing a workflow template to be verified

7.2.7 Creation of a Set of Event-Condition-Action Rules

In each use case of a business workflow template, a set of requirements may be changed. Therefore, as mentioned in Section 4.4, the requirements must be defined and maintained outside of the current technical representatives of the business process. The functionality shown in Figures 7.3 and 4.7 allows users to define a set of Event-Condition-Action rules.

- **Input**:
  - (i) Information provided by the workflow modeller to define a set of ECA-like rules;
  - (ii) An existing workflow template.
- **Process**: An existing workflow template contains a set of activities and control nodes. A requirement can be defined for each node according to the syntax mentioned in Section 4.4.

- **Output**: The set of business rules in RDF format.

7.3 Evaluation

In this section, we discuss the evaluation of the proposed approach by performing an experiment with 6 participants consisting of:

- 4 PhD students in Computer Science (2) and Business Studies (2);
- 2 participants have graduated in Information Systems (1) and Business Management Studies (1).

The participants were divided into two groups to analyse the *Procure to pay business* process. We interviewed 4 economics experts in two companies in Sophia Antipolis, France.

The process was split into two main sub-processes due to its complexity and the different roles involved:

1. **Requisition to Receipt Process** (RRP): This sub process starts by the creation and management of purchase requisitions and corresponding purchase orders to the moment the warehouse staff receives the merchandise.
2. **Supplier Invoice to Payment** (SIP): This sub process continues the previous one by registering the supplier invoices and closes it by paying supplier invoices.

Each group focused on one sub process. They determined activities and dependencies between these activities to model the sub process. We received 38 activities for the first sub process and 26 activities for the second one. The activities were then divided into sets based on their function. There were four sets (i.e., *Purchase requisition processing*, *Checking*, *Contact* and *Inventory*) and two sets (i.e., *Invoicing* and *Payment*) of activities from the first and the second sub process, respectively. We decided to reuse the *Payment* template presented in Section 1.2.1 to model the *Payment* set.

By taking these results, each group created necessary sets of semantic constraints to model the sub processes. The sets were checked for redundant and conflicting constraints (a) by manual search with the tool Microsoft Excel 2013[10] or (b) by using the proposed prototype.

In order to evaluate the effectiveness of our algorithm for checking redundant and conflicting constraints, we measured the time required for finding the shortcomings as shown in Figure 7.10. This figure indicates that using our prototype is faster.

The sets of semantic constraints were then modified (if required) to be validated. Based on these sets, the workflows were developed. To measure the correctness of the mapping method between the BP ontology and the CPN ontology, we checked

[10] https://products.office.com/en-us/home-and-student

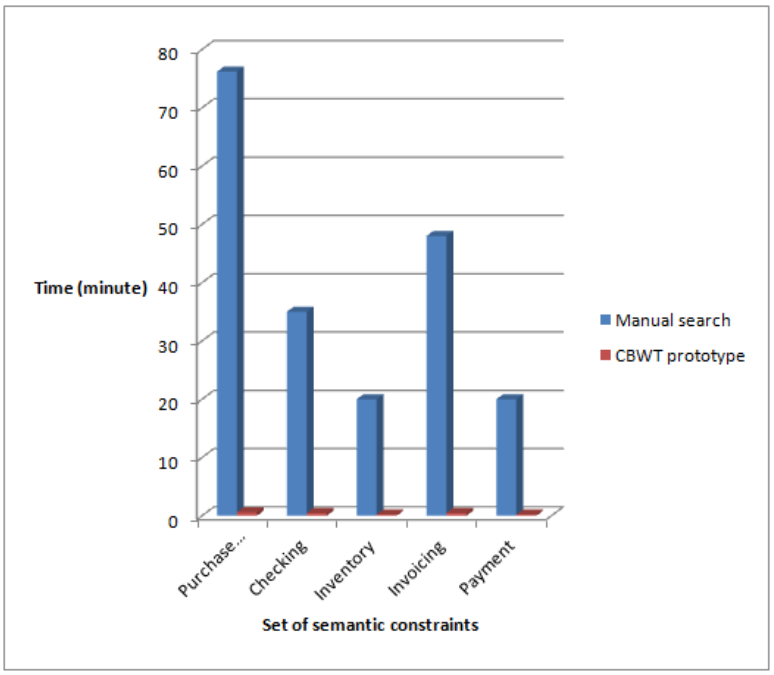

Figure 7.10: Time needed to check redundant and conflicting constraints

the workflows manually for any redundant instances. We found 1 redundant instance of the class Transition (expressed as a control node *Sequence*) in the purchase requisition workflow which is developed from 52 semantic constraints with 12 activities. Consequently, the prototypes have been improved to avoid similar redundancies.

In order to evaluate the effectiveness of SPARQL queries for syntactic and semantic checks we counted the correct, incorrect and missing answers to determine the quality of results. We compared the results obtained by using the SPARQL query language and the ones obtained by manual search. Figure 7.11 shows the number of syntactic and semantic errors of these workflows detected by the SPARQL language and by manual. These results indicate that using SPARQL queries to verify complex workflows is better regarding the accuracy of the results.

7.4 Conclusion

As mentioned at the beginning of this chapter, the CBWT prototype is developed for the process of workflow template development to validate the concepts that we have introduced in the previous chapters of this thesis. We have concentrated on developing the six use cases for expert users who are workflow modellers.

By developing the prototype, we have received useful feedback to improve the concepts relating to model semantically rich workflow templates. For example, in the first version of our prototype, sub-workflows were not supported, which was then considered as a major shortcoming. Therefore, we have expanded the old prototype to allow users to not only create a new workflow template containing one or more

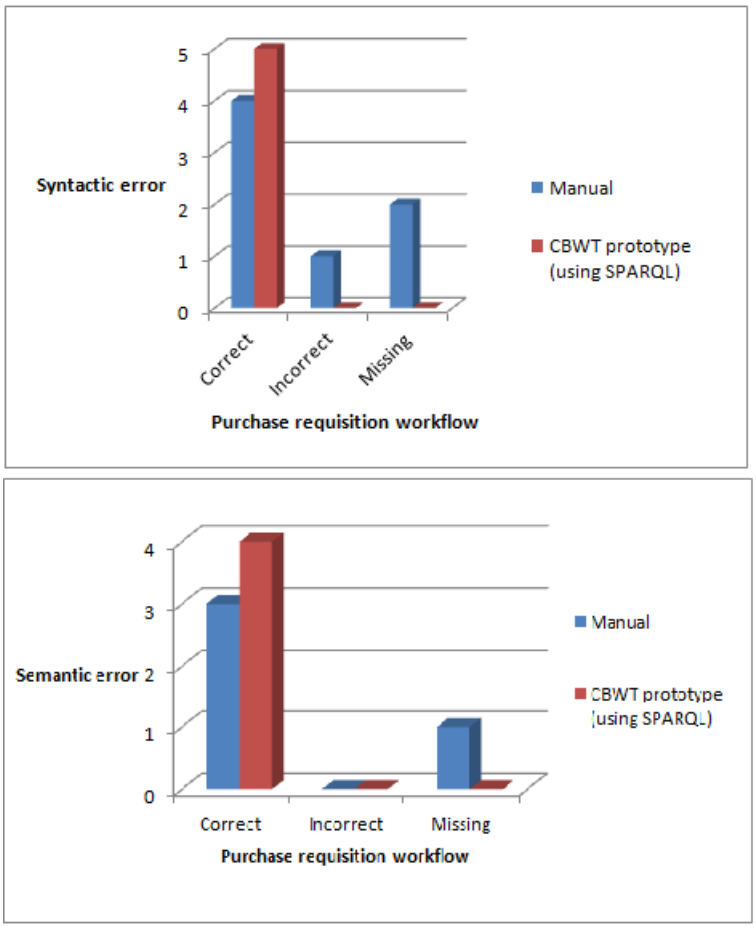

Figure 7.11: Detecting errors by manual searching and querying

sub-workflow templates, but also to check its syntactic and semantic correctness.

Chapter 8

Conclusions and Outlook

Contents

This chapter concludes our doctoral research work by summarizing the main contributions. We also discuss the limitations of the proposed approach for developing workflow templates and the current version of our CBWT prototype. Subsequently, we identify directions for possible future research.

8.1 Summary of Contributions

There are four major contributions of this thesis. Firstly, the CPN ontology has been developed to represent the concepts of CPN-based business workflow templates. Secondly, a formal definition of semantic constraints and a structure of ECA-like rules have been introduced to model semantic business processes. In order to check redundant and conflict constraints, an algorithm has been presented. In addition, to develop a workflow template, a set of algorithms used to create correspondences between the BP ontology (a business process ontology) and the CPN ontology has also been described. Thirdly, the problem of workflow verification has been investigated. A set of syntactic constraints as well as the issues of semantic verification have been determined. They are represented as SPARQL queries used for syntactic and semantic checks related to a specific business process. And lastly, concepts to better support the process for developing workflow templates have been suggested.

In fact, process specification techniques and conceptual models of workflow have been presented in various research papers. However, in most cases, they focused on checking the correctness of a workflow either at the syntactic or at the semantic level only. As the result, this is not sufficient for guaranteeing the correctness of a workflow template at both levels. In contrast, our approach focused on the combination of control flow (based on CPNs) and semantic constraints that enables syntactic and semantic checks related to a workflow template.

To summarize, in comparison with the current approaches, our approach has the following distinguishing features:

- *Representing semantically rich business workflow templates*: The CPN ontology has been developed to represent the concepts of CPN-based business workflow templates. A business process is thus syntactically transformed into an instance of the CPN ontology, which enables syntactic checks based on CPNs. The purpose of the CPN ontology is to semantically enrich workflow templates. Once workflow definitions are stored as semantic enriched workflow templates, IT experts can easily develop their appropriate software systems from the workflow templates.

- *Describing semantically a business process by identifying a set of semantic constraints and ECA-like rules.*

  Semantic constraints are specified as domain specific restrictions on a business process. They express the dependencies between activities, such as existing dependencies and ordering relations. A set of semantic constraints is transformed into an instance of the BP ontology if there is no redundant and conflicting constraints. A business workflow template is then developed by creating a correspondence between the BP ontology and the CPN ontology.

  The definitions in the BP ontology are used not only to standardize the terminologies, but also to check the semantic correctness of workflow templates. However, semantic constraints can not capture some business level correctness requirements, such as the constraint specifying that a certain user task has to be performed in a certain activity of a business process. Therefore, ECA-like rules are proposed to express those requirements. The combination of semantic constraints and ECA-like rules supports workflow modellers in modelling semantic business processes.

- *Correctness criteria for business workflow templates*: The correctness criteria are considered at the two levels, syntactic and semantic.

  Since a business workflow template is developed based on the CPN and the BP ontology, it allows syntactic and semantic checks. The performance of the former relies on the classification of syntactic constraints in modelling business processes. The latter is performed in order to answer the five semantic verification issues of a workflow template.

  Furthermore, since workflow templates are encoded in RDF format, the SPARQL query language is used to check their correctness. Correctness criteria are formalized as SPARQL queries, which can be asked against an RDF graph describing a workflow template.

In order to modify workflow templates and their ECA-like rules, a set of manipulation operations has been proposed that allows modifying and updating the workflow templates and their ECA-like rules. The set includes operations to add, delete, update workflow elements and ECA-like rules as well as to modify the order of the existing workflow elements.

Moreover, the issue of reusing workflow templates is addressed. This contains several phases: searching, understanding, modifying and integrating workflow templates. Each phase provides adequate support to facilitate the reuse of workflow templates. The annotations of each workflow template help users to find and select the most suitable ones. The selected workflow templates along with their ECA-like rules then can be adapted and modified by applying the proposed manipulation operations. The integration of the existing modified, unmodified and new sub-workflow templates (if any) is supported by enabling the composition of sub-workflows.

Finally, the CBWT prototype has been implemented to demonstrate the feasibility of the concepts introduced in this thesis.

8.2 Limitations and Perspectives

In this section, we discuss limitations of our approach as well as provide a brief description of the main perspectives of our research.

The main limitation of our research comes from the complexity of modelling domain knowledge. The research has been particularly oriented to model semantic business processes and business level requirements by expert-users. It is sometimes difficult to specify semantic constraints and to represent all business level requirements as ECA-like rules. For a set of complex semantic constraints, e.g., an activity which may relate to a lot of semantic constraints, the automated approach, which is used to create correspondences between the BP ontology and the CPN ontology to develop a workflow template, may result in redundant control nodes. Therefore, the resulting workflow template in some cases need to be manually optimized.

Another limitation is that at the moment only design time is supported and there is no support for multiple modellers who might be involved in workflow modelling.

In the following, several other directions to extend the results presented in this thesis are identified:

- **Support of multiple workflow modellers** (i.e., expert users): The CBWT prototype has been developed as a simple web application. Thus, it can be further extended to support multiple workflow modellers who might be involved in modelling a business process. To address the problem of concurrency access, the following solutions are mentioned:

  - Locking the template file for writing. It means that once an expert user (modeller) starts to modify an existing workflow template (file) nobody else can commit any change to this file.
  - Using workspaces (or named repositories). A workspace is a named repository on the server. The access right is controlled by the server. It is mandatory to apply some forms of version management and sharing strategy to the repository (i.e., workflow templates can be merged, modified and copied, etc.).

- **Workflow verification at runtime**: To resolve the limitation related to the verification of process instances at runtime, new research has begun in 2014 in the research team Wimmics[1]. Based on the results of our research, this new research action will adopt and extend our process for developing workflow templates. It will focus on business level correctness requirements as well as the verification of process instances.

- **Authorization constraints**: There is usually no expectation that such missions as modifying a workflow template, changing business level requirements, completing a workflow template, etc., can be performed by all expert users. Instead, each expert user should only be permitted to undertake a clearly-defined set of missions. In this case, the definition of authorization constraints should be possible and is enforced by the WFMS. Therefore, the approach introduced in this thesis can be extended to support authorization constraints.

[1] https://wimmics.inria.fr

Appendix A

Classification of Business Rules

In our work, we classify business rules into three groups as follows:

- **Structural rules** detail a specific, static aspect of the business. They express restrictions on business concepts and facts. For example:

  At a time, a customer can rent at most one car.

  It is obligatory that each rental car is owned by exactly one branch.

- **Action rules** that concern some dynamic aspect of the business. They establish when certain activities should take place. For example:

  A car can be handed over to the customer if and only if the deposit has been confirmed.

  If a customer is blacklisted, his/her rental reservation must not be accepted.

- **Derivation rules** are generated by an inference or a mathematical calculation from terms, facts, other derivations or even action rules. Consequently, they are based on one or more business rules. Therefore, it is unnecessary to store them explicitly. For example:

  - Derivation/Inference: Each French is a person who is a citizen of country 'FR'.
  - Derivation/Mathematical calculation: The 'rental amount' in Rental is equal to the 'rental rate' times its 'number of days'.

Note that the most important difference between action and structural rules is that the former is related to a concrete event (e.g., the rejection of a rental reservation related to a customer in a blacklist in the examples above), when the latter does not imply any relevant event (e.g., a customer can always reserve one car at a time, whoever she/he is).

Appendix B

The CPN ontology (CpnOnt.owl)

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>
<rdf:RDF xmlns="http://www.semanticweb.org/CPNWF#"
     xml:base="http://www.semanticweb.org/CPNWF"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <owl:ObjectProperty rdf:about="#connectsPlace">
        <rdfs:range rdf:resource="#InputArc"/>
        <rdfs:domain rdf:resource="#OutputArc"/>
        <rdfs:range rdf:resource="#Place"/>
        <rdfs:domain rdf:resource="#Transition"/>
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:about="#connectsTrans">
        <rdfs:domain rdf:resource="#InputArc"/>
        <rdfs:range rdf:resource="#OutputArc"/>
        <rdfs:domain rdf:resource="#Place"/>
        <rdfs:range rdf:resource="#Transition"/>
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:about="#hasActivity">
        <rdfs:range rdf:resource="#ActNode"/>
        <rdfs:domain rdf:resource="#GuardFunction"/>
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:about="#hasArc">
        <rdf:type rdf:resource="&owl;FunctionalProperty"/>
        <rdfs:domain rdf:resource="#CPNOnt"/>
        <rdfs:range>
            <owl:Class>
                <owl:unionOf rdf:parseType="Collection">
                    <rdf:Description rdf:about="#InputArc"/>
```

```
                <rdf:Description rdf:about="#OutputArc"/>
            </owl:unionOf>
        </owl:Class>
    </rdfs:range>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasAttribute">
    <rdfs:range rdf:resource="#Attribute"/>
    <rdfs:domain rdf:resource="#Delete"/>
    <rdfs:domain rdf:resource="#GuardFunction"/>
    <rdfs:domain rdf:resource="#Insert"/>
    <rdfs:domain rdf:resource="#Token"/>
</owl:ObjectProperty>
 <owl:ObjectProperty rdf:about="#hasControl">
    <rdfs:range rdf:resource="#CtrlNode"/>
    <rdfs:domain rdf:resource="#GuardFunction"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasExpression">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:range>
        <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
                <rdf:Description rdf:about="#Delete"/>
                <rdf:Description rdf:about="#Insert"/>
            </owl:unionOf>
        </owl:Class>
    </rdfs:range>
    <rdfs:domain>
        <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
                <rdf:Description rdf:about="#InputArc"/>
                <rdf:Description rdf:about="#OutputArc"/>
            </owl:unionOf>
        </owl:Class>
    </rdfs:domain>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasGFunction">
    <rdfs:range rdf:resource="#GuardFunction"/>
    <rdfs:domain rdf:resource="#Transition"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasMarking">
    <rdfs:domain rdf:resource="#Place"/>
    <rdfs:range rdf:resource="#Token"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasPlace">
```

```
    <rdfs:domain rdf:resource="#CPNOnt"/>
    <rdfs:range rdf:resource="#Place"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasTrans">
    <rdfs:domain rdf:resource="#CPNOnt"/>
    <rdfs:range rdf:resource="#Transition"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#valueAtt">
    <rdfs:domain rdf:resource="#ActNode"/>
    <rdfs:domain rdf:resource="#Attribute"/>
    <rdfs:domain rdf:resource="#CtrlNode"/>
    <rdfs:range rdf:resource="#Value"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#valueRef">
    <rdfs:domain rdf:resource="#Value"/>
    <rdfs:range rdf:resource="#Value"/>
</owl:ObjectProperty>
<owl:Class rdf:about="#ActNode">
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#valueAtt"/>
            <owl:onClass rdf:resource="#Value"/>
            <owl:qualifiedCardinality rdf:datatype=
            "&xsd;nonNegativeInteger">1
            </owl:qualifiedCardinality>
        </owl:Restriction>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="#GuardFunction"/>
</owl:Class>
<owl:Class rdf:about="#Attribute">
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#valueAtt"/>
            <owl:onClass rdf:resource="#Value"/>
            <owl:maxQualifiedCardinality rdf:datatype=
            "&xsd;nonNegativeInteger">1
            </owl:maxQualifiedCardinality>
        </owl:Restriction>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="#Delete"/>
    <rdfs:subClassOf rdf:resource="#Insert"/>
    <rdfs:subClassOf rdf:resource="#Token"/>
</owl:Class>
<owl:Class rdf:about="#CPNOnt">
```

```
        <owl:equivalentClass>
            <owl:Class>
                <owl:intersectionOf rdf:parseType="Collection">
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#hasArc"/>
                        <owl:onClass>
                            <owl:Class>
                                <owl:unionOf rdf:parseType="Collection">
                                    <rdf:Description rdf:about="#InputArc"/>
                                    <rdf:Description rdf:about="#OutputArc"/>
                                </owl:unionOf>
                            </owl:Class>
                        </owl:onClass>
                        <owl:minQualifiedCardinality rdf:datatype=
                        "&xsd;nonNegativeInteger">1
                        </owl:minQualifiedCardinality>
                    </owl:Restriction>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#hasPlace"/>
                        <owl:onClass rdf:resource="#Place"/>
                        <owl:minQualifiedCardinality rdf:datatype=
                        "&xsd;nonNegativeInteger">1
                        </owl:minQualifiedCardinality>
                    </owl:Restriction>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#hasTrans"/>
                        <owl:onClass rdf:resource="#Transition"/>
                        <owl:minQualifiedCardinality rdf:datatype=
                        "&xsd;nonNegativeInteger">1
                        </owl:minQualifiedCardinality>
                    </owl:Restriction>
                </owl:intersectionOf>
            </owl:Class>
        </owl:equivalentClass>
    </owl:Class>
    <owl:Class rdf:about="#CtrlNode">
        <owl:equivalentClass>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#valueAtt"/>
                <owl:onClass rdf:resource="#Value"/>
                <owl:maxQualifiedCardinality rdf:datatype=
                "&xsd;nonNegativeInteger">1
                </owl:maxQualifiedCardinality>
            </owl:Restriction>
```

```
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="#GuardFunction"/>
</owl:Class>
<owl:Class rdf:about="#Delete">
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasAttribute"/>
            <owl:allValuesFrom rdf:resource="#Attribute"/>
        </owl:Restriction>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="#InputArc"/>
</owl:Class>
<owl:Class rdf:about="#GuardFunction">
    <owl:equivalentClass>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <owl:Class>
                    <owl:unionOf rdf:parseType="Collection">
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasActivity"/>
                            <owl:onClass rdf:resource="#ActNode"/>
                            <owl:qualifiedCardinality rdf:datatype=
                            "&xsd;nonNegativeInteger">1
                            </owl:qualifiedCardinality>
                        </owl:Restriction>
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasControl"/>
                            <owl:onClass rdf:resource="#CtrlNode"/>
                            <owl:qualifiedCardinality rdf:datatype=
                            "&xsd;nonNegativeInteger">1
                            </owl:qualifiedCardinality>
                        </owl:Restriction>
                    </owl:unionOf>
                </owl:Class>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#hasAttribute"/>
                    <owl:onClass rdf:resource="#Attribute"/>
                    <owl:minQualifiedCardinality rdf:datatype=
                    "&xsd;nonNegativeInteger">1
                    </owl:minQualifiedCardinality>
                </owl:Restriction>
            </owl:intersectionOf>
        </owl:Class>
    </owl:equivalentClass>
```

```
        <rdfs:subClassOf rdf:resource="#Transition"/>
    </owl:Class>
    <owl:Class rdf:about="#InputArc">
        <owl:equivalentClass>
            <owl:Class>
                <owl:intersectionOf rdf:parseType="Collection">
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#hasPlace"/>
                        <owl:someValuesFrom rdf:resource="#Place"/>
                    </owl:Restriction>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#hasExpression"/>
                        <owl:onClass rdf:resource="#Delete"/>
                        <owl:minQualifiedCardinality
                        "&xsd;nonNegativeInteger">1 rdf:datatype=
                        </owl:minQualifiedCardinality>
                    </owl:Restriction>
                </owl:intersectionOf>
            </owl:Class>
        </owl:equivalentClass>
        <rdfs:subClassOf rdf:resource="#CPNOnt"/>
    </owl:Class>
    <owl:Class rdf:about="#Insert">
        <owl:equivalentClass>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasAttribute"/>
                <owl:someValuesFrom rdf:resource="#Attribute"/>
            </owl:Restriction>
        </owl:equivalentClass>
        <rdfs:subClassOf rdf:resource="#OutputArc"/>
    </owl:Class>
    <owl:Class rdf:about="#OutputArc">
        <owl:equivalentClass>
            <owl:Class>
                <owl:intersectionOf rdf:parseType="Collection">
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#hasTrans"/>
                        <owl:someValuesFrom rdf:resource="#Transition"/>
                    </owl:Restriction>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#hasExpression"/>
                        <owl:onClass rdf:resource="#Insert"/>
                        <owl:minQualifiedCardinality rdf:datatype=
                        "&xsd;nonNegativeInteger">1
```

```
                    </owl:minQualifiedCardinality>
                </owl:Restriction>
            </owl:intersectionOf>
        </owl:Class>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="#CPNOnt"/>
</owl:Class>
<owl:Class rdf:about="#Place">
    <owl:equivalentClass>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#connectsTrans"/>
                    <owl:allValuesFrom rdf:resource="#Transition"/>
                </owl:Restriction>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#hasMarking"/>
                    <owl:onClass rdf:resource="#Token"/>
                    <owl:maxQualifiedCardinality rdf:datatype=
                    "&xsd;nonNegativeInteger">1
                    </owl:maxQualifiedCardinality>
                </owl:Restriction>
            </owl:intersectionOf>
        </owl:Class>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="#CPNOnt"/>
</owl:Class>
<owl:Class rdf:about="#Token">
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasAttribute"/>
            <owl:onClass rdf:resource="#Attribute"/>
            <owl:minQualifiedCardinality rdf:datatype=
            "&xsd;nonNegativeInteger">1
            </owl:minQualifiedCardinality>
        </owl:Restriction>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="#Place"/>
</owl:Class>
<owl:Class rdf:about="#Transition">
    <owl:equivalentClass>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <owl:Restriction>
```

```
                    <owl:onProperty rdf:resource="#connectsPlace"/>
                    <owl:allValuesFrom rdf:resource="#Place"/>
                </owl:Restriction>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#hasGFunction"/>
                    <owl:onClass rdf:resource="#GuardFunction"/>
                    <owl:qualifiedCardinality rdf:datatype=
                    "&xsd;nonNegativeInteger">1
                    </owl:qualifiedCardinality>
                </owl:Restriction>
            </owl:intersectionOf>
        </owl:Class>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="#CPNOnt"/>
</owl:Class>
<owl:Class rdf:about="#Value">
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#valueRef"/>
            <owl:allValuesFrom rdf:resource="#Value"/>
        </owl:Restriction>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="#ActNode"/>
    <rdfs:subClassOf rdf:resource="#Attribute"/>
    <rdfs:subClassOf rdf:resource="#CtrlNode"/>
</owl:Class>
<owl:Class rdf:about="#And-join">
    <rdfs:subClassOf>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#connectsPlace"/>
                    <owl:allValuesFrom>
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasMarking"/>
                            <owl:allValuesFrom rdf:resource="#Token"/>
                        </owl:Restriction>
                    </owl:allValuesFrom>
                </owl:Restriction>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#hasGFunction"/>
                    <owl:allValuesFrom>
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasControl"/>
```

```
                        <owl:allValuesFrom rdf:resource="#CtrlNode"/>
                    </owl:Restriction>
                </owl:allValuesFrom>
            </owl:Restriction>
            <owl:Restriction>
                <owl:onProperty>
                    <rdf:Description>
                        <owl:inverseOf rdf:resource="#connectsTrans"/>
                    </rdf:Description>
                </owl:onProperty>
                <owl:allValuesFrom>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#hasMarking"/>
                        <owl:allValuesFrom rdf:resource="#Token"/>
                    </owl:Restriction>
                </owl:allValuesFrom>
            </owl:Restriction>
            <owl:Restriction>
                <owl:onProperty>
                    <rdf:Description>
                        <owl:inverseOf rdf:resource="#connectsTrans"/>
                    </rdf:Description>
                </owl:onProperty>
                <owl:onClass rdf:resource="#Place"/>
                <owl:minQualifiedCardinality rdf:datatype=
                "&xsd;nonNegativeInteger">2</owl:minQualifiedCardinality>
            </owl:Restriction>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#connectsPlace"/>
                <owl:onClass rdf:resource="#Place"/>
                <owl:qualifiedCardinality rdf:datatype=
                "&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
            </owl:Restriction>
        </owl:intersectionOf>
    </owl:Class>
</rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#And-split">
    <rdfs:subClassOf>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#connectsPlace"/>
                    <owl:allValuesFrom>
```

```
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasMarking"/>
                <owl:allValuesFrom rdf:resource="#Token"/>
            </owl:Restriction>
        </owl:allValuesFrom>
    </owl:Restriction>
    <owl:Restriction>
        <owl:onProperty rdf:resource="#hasGFunction"/>
        <owl:allValuesFrom>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasControl"/>
                <owl:allValuesFrom rdf:resource="#CtrlNode"/>
            </owl:Restriction>
        </owl:allValuesFrom>
    </owl:Restriction>
    <owl:Restriction>
        <owl:onProperty>
            <rdf:Description>
                <owl:inverseOf rdf:resource="#connectsTrans"/>
            </rdf:Description>
        </owl:onProperty>
        <owl:allValuesFrom>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasMarking"/>
                <owl:allValuesFrom rdf:resource="#Token"/>
            </owl:Restriction>
        </owl:allValuesFrom>
    </owl:Restriction>
    <owl:Restriction>
        <owl:onProperty rdf:resource="#connectsPlace"/>
        <owl:onClass rdf:resource="#Place"/>
        <owl:minQualifiedCardinality rdf:datatype=
        "&xsd;nonNegativeInteger">2</owl:minQualifiedCardinality>
    </owl:Restriction>
    <owl:Restriction>
        <owl:onProperty>
            <rdf:Description>
                <owl:inverseOf rdf:resource="#connectsTrans"/>
            </rdf:Description>
        </owl:onProperty>
        <owl:onClass rdf:resource="#Place"/>
        <owl:qualifiedCardinality rdf:datatype=
        "&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
    </owl:Restriction>
```

```
                </owl:intersectionOf>
            </owl:Class>
        </rdfs:subClassOf>
    </owl:Class>

   ...

</rdf:RDF>
<!--Generated by the OWL API(version 3.4.2)http://owlapi.sourceforge.net-->
```

Appendix C

Labelling Workflow Activities

Chapter 4 introduces a formal definition of semantic constraints. We only concentrate on how to represent dependencies between activities in a business process while omitting the discussion of activity labels, which are captured through the definition of semantic constraints. However, activity labels play an essential role in searching for workflow templates in accordance with their intended use. Therefore, for the purpose of organization of the knowledge base of workflow templates discussed in Section 6.1, it is necessary to take a look at labelling workflow activities.

In fact, some classes of activity labels have been found in practice. According to [Mendling 2010, Leopold 2012], they are mainly divided into verb-object labels, action-noun labels and a rest category. A verb–object label contains an action followed by a business object, such as "*Create invoice*". An action-noun label may start with a business object followed by an action (e.g., " *Schedule approval*") or a noun phrase containing a prepositional phrase (e.g., "*Creation of specification*") or a verb in -ing form (e.g., "*Creating version*"). Regarding to a rest category, it consists of descriptive labels, e.g., *Accounting creates invoice*" and no-action labels, e.g., "*Error*", etc. Furthermore, action-noun labels can be automatically refactored to verb-object labels by using the refactoring approach of [Leopold 2012]. Therefore, we recommend workflow modellers to use the verb–object style.

Bibliography

[Awad 2008] Ahmed Awad, Gero Decker and Mathias Weske. *Efficient Compliance Checking Using BPMN-Q and Temporal Logic.* In Business Process Management, 6th International Conference, BPM 2008, Milan, Italy, September 2-4, 2008. Proceedings, pages 326–341, 2008. (Cited on page 94.)

[Barros 1997] Alistair P. Barros, Arthur H. M. ter Hofstede and Henderik Alex Proper. *Essential Principles for Workflow Modelling Effectiveness.* In PACIS, page 15, 1997. (Cited on page 1.)

[Bénel 2010] Aurélien Bénel, Chao Zhou and Jean-Pierre Cahier. *Beyond Web 2.0 ... and Beyond the Semantic Web.* In David Randall and Pascal Salembier, editeurs, From CSCW to Web 2.0: European Developments in Collaborative Design, Computer Supported Cooperative Work, pages 155–171. Springer London, 2010. (Cited on page 20.)

[Berners-Lee 2005] Tim Berners-Lee, Roy Fielding and Larry Masinter. *Request for Comments: 3986: Uniform Resource Identifier (URI): Generic Syntax.* http://tools.ietf.org/html/rfc3986, January 2005. (Cited on page 20.)

[Berstel 2007] Bruno Berstel, Philippe Bonnard, François Bry, Michael Eckert and Paula-Lavinia Pătrânjan. *Reactive Rules on the Web.* In Grigoris Antoniou, Uwe Abmann, Cristina Baroglio, Stefan Decker, Nicola Henze, Paula-Lavinia Pătrânjan and Robert Tolksdorf, editeurs, Reasoning Web, volume 4636 of *Lecture Notes in Computer Science*, pages 183–239. Springer Berlin Heidelberg, 2007. (Cited on page 73.)

[Bi 2004] Henry H. Bi and J. Leon Zhao. *Applying Propositional Logic to Workflow Verification.* Information Technology and Management, vol. 5, no. 3-4, pages 293–318, 2004. (Cited on page 79.)

[Boley 2007] Harold Boley, Michael Kifer, Paula-Lavinia Pătrânjan and Axel Polleres. *Rule Interchange on the Web.* In Grigoris Antoniou, Uwe Abmann, Cristina Baroglio, Stefan Decker, Nicola Henze, Paula-Lavinia Pătrânjan and Robert Tolksdorf, editeurs, Reasoning Web, volume 4636 of *Lecture Notes in Computer Science*, pages 269–309. Springer Berlin Heidelberg, 2007. (Cited on page 19.)

[Booch 2005] Grady Booch, James Rumbaugh and Ivar Jacobson. Unified Modeling Language User Guide, The (2Nd Edition) (Addison-Wesley Object Technology Series). Addison-Wesley Professional, 2005. (Cited on page 106.)

[Bouzidi 2012] Khalil Riad Bouzidi, Bruno Fiés, Catherine Faron-Zucker, Alain Zarli and Nhan Le Thanh. *Semantic Web Approach to Ease Regulation Com-*

pliance Checking in Construction Industry. Future Internet, vol. 4, no. 3, pages 830–851, 2012. (Cited on page 71.)

[bpm 2011] *Business Process Model and Notation, V2.0.* http://www.bpmn.org/, 2011. (Cited on page 3.)

[Brockmans 2006] Saartje Brockmans, Marc Ehrig, Agnes Koschmider, Andreas Oberweis and Rudi Studer. *Semantic Alignment of Business Processes.* In ICEIS 2006 - Proceedings of the Eighth International Conference on Enterprise Information Systems: Databases and Information Systems Integration, Paphos, Cyprus, May 23-27, 2006, pages 191–196, 2006. (Cited on page 43.)

[Bry 2005] François Bry and Massimo Marchiori. *Ten Theses on Logic Languages for the Semantic Web.* In Principles and Practice of Semantic Web Reasoning, Third International Workshop, PPSWR 2005, Dagstuhl Castle, Germany, September 11-16, 2005, Proceedings, pages 42–49, 2005. (Cited on page 19.)

[Cardoso 2007] Jorge Cardoso. The syntactic and the semantic web. IGI Global, 2007. (Cited on page 21.)

[Chiola 1995] G. Chiola. *Characterization of Timed Well-formed Petri Nets Behavior by Means of Occurrence Equations.* In Proceedings of the Sixth International Workshop on Petri Nets and Performance Models, PNPM '95, pages 127–, Washington, DC, USA, 1995. IEEE Computer Society. (Cited on page 94.)

[Clarke 2001] Edmund M. Clarke, Orna Grumberg and Doron Peled. Model checking. MIT Press, 2001. (Cited on page 92.)

[Dwyer 1999] Matthew B. Dwyer, George S. Avrunin and James C. Corbett. *Patterns in Property Specifications for Finite-state Verification.* In Proceedings of the 21st International Conference on Software Engineering, ICSE '99, pages 411–420, New York, NY, USA, 1999. ACM. (Cited on pages 71, 72, 73 and 92.)

[Ehrig 2007] Marc Ehrig, Agnes Koschmider and Andreas Oberweis. *Measuring Similarity between Semantic Business Process Models.* In Conceptual Modelling 2007, Proceedings of the Fourth Asia-Pacific Conference on Conceptual Modelling (APCCM2007), Ballarat, Victoria, Australia, January 30 - February 2, 2007, Proceedings, pages 71–80, 2007. (Cited on page 43.)

[Ellis 1993] Clarence A. Ellis and Gary J. Nutt. *Modeling and Enactment of Workflow Systems.* In Application and Theory of Petri Nets, pages 1–16, 1993. (Cited on page 1.)

[Eshuis 2008] Rik Eshuis and Paul W. P. J. Grefen. *Constructing customized process views.* Data Knowl. Eng., vol. 64, no. 2, pages 419–438, 2008. (Cited on page 103.)

[Esparza 1994] Javier Esparza. *Reduction and Synthesis of Live and Bounded Free Choice Petri Nets.* Inf. Comput., vol. 114, no. 1, pages 50–87, October 1994. (Cited on pages 86, 93 and 94.)

[exs 2000] *ExSpect 6.41.* http://www.exspect.com/, 2000. (Cited on page 17.)

[Feja 2011] S. Feja, S. Witt and A Speck. *BAM: A Requirements Validation and Verification Framework for Business Process Models.* In Quality Software (QSIC), 2011 11th International Conference on, pages 186–191, July 2011. (Cited on page 92.)

[Fellmann 2011] Michael Fellmann, Oliver Thomas and Bastian Busch. *A Query-Driven Approach for Checking the Semantic Correctness of Ontology-Based Process Representations.* In BIS, pages 62–73, 2011. (Cited on pages 46 and 95.)

[Förster 2007] A. Förster, G. Engels, T. Schattkowsky and R. Van Der Straeten. *Verification of Business Process Quality Constraints Based on Visual Process Patterns.* In Theoretical Aspects of Software Engineering, 2007. TASE '07. First Joint IEEE/IFIP Symposium on, pages 197–208, June 2007. (Cited on page 92.)

[Gasevic 2006] Dragan Gasevic and Vladan Devedzic. *Petri net ontology.* Knowl.-Based Syst., vol. 19, no. 4, pages 220–234, 2006. (Cited on pages 42 and 43.)

[Goedertier 2006] Stijn Goedertier and Jan Vanthienen. *Designing Compliant Business Processes with Obligations and Permissions.* In Business Process Management Workshops, BPM 2006 International Workshops, BPD, BPI, ENEI, GPWW, DPM, semantics4ws, Vienna, Austria, September 4-7, 2006, Proceedings, pages 5–14, 2006. (Cited on page 94.)

[Greco 2004] Gianluigi Greco, Antonella Guzzo, Luigi Pontieri and Domenico Saccà. *An Ontology-Driven Process Modeling Framework.* In Fernando Galindo, Makoto Takizawa and Roland Traunmüller, editeurs, Database and Expert Systems Applications, volume 3180 of *Lecture Notes in Computer Science*, pages 13–23. Springer Berlin Heidelberg, 2004. (Cited on page 42.)

[Hay 2000] David Hay, Keri Anderson Healy and et al. *GUIDE Business Rules Project - Final Report.* Rapport technique, The Business Rules Group, July 2000. (Cited on page 18.)

[Hustadt 2004] U. Hustadt, R. A. Schmidt and L. Georgieva. *A Survey of Decidable First-Order Fragments and Description Logics.* Journal of Relational Methods in Computer Science, vol. 1, page 2004, 2004. (Cited on page 92.)

[iee 2012] *1012-2012 - IEEE Standard for System and Software Verification and Validation.* http://standards.ieee.org/findstds/standard/1012-2012.html, 2012. (Cited on page 75.)

[Jørgensen 2008] Jens Bæk Jørgensen, Kristian Bisgaard Lassen and Wil M. P. van der Aalst. *From task descriptions via colored Petri nets towards an implementation of a new electronic patient record workflow system.* STTT, vol. 10, no. 1, pages 15–28, 2008. (Cited on page 25.)

[Khaluf 2011] Lial Khaluf, Christian Gerth and Gregor Engels. *Pattern-Based Modeling and Formalizing of Business Process Quality Constraints.* In Haralambos Mouratidis and Colette Rolland, editeurs, Advanced Information Systems Engineering, volume 6741 of *Lecture Notes in Computer Science*, pages 521–535. Springer Berlin Heidelberg, 2011. (Cited on page 92.)

[Knuplesch 2010] David Knuplesch, Linh Thao Ly, Stefanie Rinderle-Ma, Holger Pfeifer and Peter Dadam. *On Enabling Data-Aware Compliance Checking of Business Process Models.* In Jeffrey Parsons, Motoshi Saeki, Peretz Shoval, Carson Woo and Yair Wand, editeurs, Conceptual Modeling – ER 2010, volume 6412 of *Lecture Notes in Computer Science*, pages 332–346. Springer Berlin Heidelberg, 2010. (Cited on page 92.)

[Koschmider 2005] Agnes Koschmider and Andreas Oberweis. *Ontology Based Business Process Description.* In EMOI-INTEROP, pages 321–333. Springer, 2005. (Cited on pages 1, 42, 43 and 44.)

[Koschmider 2011] Agnes Koschmider, Thomas Hornung and Andreas Oberweis. *Recommendation-based editor for business process modeling.* Data Knowl. Eng., vol. 70, no. 6, pages 483–503, 2011. (Cited on page 103.)

[Koschmider 2015] Agnes Koschmider and Hajo A. Reijers. *Improving the process of process modelling by the use of domain process patterns.* Enterprise IS, vol. 9, no. 1, pages 29–57, 2015. (Cited on page 103.)

[Kovalyov 1990] A. V. Kovalyov. *On Complete Reducibility of Some Classes of Petri Nets.* In Proceedings of the 11th International Conference on Application and Theory of Petri Nets, 1990, Paris, France, pages 352–366, 1990. NewsletterInfo: 36. (Cited on page 93.)

[Kradolfer 2000] Markus Kradolfer. *A workflow metamodel supporting dynamic, reuse-based model evolution.* PhD thesis, 2000. (Cited on page 97.)

[Kristensen 1998] Lars M. Kristensen, Soren Christensen and Kurt Jensen. *The practitioner's guide to coloured Petri nets.* International Journal on Software Tools for Technology Transfer, vol. 2, pages 98–132, 1998. (Cited on page 26.)

[Kumar 2010] Akhil Kumar, Wen Yao, Chao-Hsien Chu and Zang Li. *Ensuring Compliance with Semantic Constraints in Process Adaptation with Rule-Based Event Processing.* In RuleML, pages 50–65, 2010. (Cited on pages 73 and 94.)

[Leopold 2012] Henrik Leopold, Sergey Smirnov and Jan Mendling. *On the refactoring of activity labels in business process models.* Inf. Syst., vol. 37, no. 5, pages 443–459, 2012. (Cited on page 137.)

[Lin 2008] Cui Lin, Shiyong Lu, Zhaoqiang Lai, A. Chebotko, Xubo Fei, Jing Hua and F. Fotouhi. *Service-Oriented Architecture for VIEW: A Visual Scientific Workflow Management System.* In Services Computing, 2008. SCC '08. IEEE International Conference on, volume 1, pages 335–342, July 2008. (Cited on page 12.)

[Lu 2006] Shiyong Lu, Arthur J. Bernstein and Philip M. Lewis. *Automatic workflow verification and generation.* Theor. Comput. Sci., vol. 353, no. 1-3, pages 71–92, 2006. (Cited on page 1.)

[Lu 2008] Ruopeng Lu, Shazia Sadiq and Guido Governatori. *Compliance Aware Business Process Design.* In Arthur ter Hofstede, Boualem Benatallah and Hye-Young Paik, editeurs, Business Process Management Workshops, volume 4928 of *Lecture Notes in Computer Science*, pages 120–131. Springer Berlin Heidelberg, 2008. (Cited on page 94.)

[Lu 2009] Ruopeng Lu, Shazia Sadiq and Guido Governatori. *On Managing Business Processes Variants.* Data Knowl. Eng., vol. 68, no. 7, pages 642–664, July 2009. (Cited on page 103.)

[Ludäscher 2006] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao and Yang Zhao. *Scientific Workflow Management and the Kepler System: Research Articles.* Concurr. Comput. : Pract. Exper., vol. 18, no. 10, pages 1039–1065, August 2006. (Cited on page 12.)

[Ludäscher 2009] Bertram Ludäscher, Mathias Weske, Timothy McPhillips and Shawn Bowers. *Scientific Workflows: Business as Usual?* In Umeshwar Dayal, Johann Eder, Jana Koehler and HajoA. Reijers, editeurs, Business Process Management, volume 5701 of *Lecture Notes in Computer Science*, pages 31–47. Springer Berlin Heidelberg, 2009. (Cited on pages 13 and 14.)

[Ly 2008] Linh Thao Ly, Stefanie Rinderle and Peter Dadam. *Integration and verification of semantic constraints in adaptive process management systems.* Data Knowl. Eng., vol. 64, no. 1, pages 3–23, 2008. (Cited on pages 73 and 94.)

[Ly 2012] Linh Thao Ly, Stefanie Rinderle-Ma, Kevin Göser and Peter Dadam. *On enabling integrated process compliance with semantic constraints in process management systems - Requirements, challenges, solutions.* Information Systems Frontiers, vol. 14, no. 2, pages 195–219, 2012. (Cited on page 94.)

[Markovic 2008] Ivan Markovic and Alessandro Costa Pereira. *Towards a Formal Framework for Reuse in Business Process Modeling.* In Proceedings

of the 2007 International Conference on Business Process Management, BPM'07, pages 484–495, Berlin, Heidelberg, 2008. Springer-Verlag. (Cited on page 103.)

[Mendling 2006] Jan Mendling and Carlo Simon. *Business Process Design by View Integration.* In Business Process Management Workshops, BPM 2006 International Workshops, BPD, BPI, ENEI, GPWW, DPM, semantics4ws, Vienna, Austria, September 4-7, 2006, Proceedings, pages 55–64, 2006. (Cited on page 103.)

[Mendling 2010] J. Mendling, H. A. Reijers and J. Recker. *Activity Labeling in Process Modeling: Empirical Insights and Recommendations.* Inf. Syst., vol. 35, no. 4, pages 467–482, June 2010. (Cited on page 137.)

[Murata 1989] Tadao Murata. *Petri nets: Properties, analysis and applications.* Proceedings of the IEEE, vol. 77, no. 4, pages 541–580, Apr 1989. (Cited on page 43.)

[Namiri 2007] Kioumars Namiri and Nenad Stojanovic. *Pattern-Based Design and Validation of Business Process Compliance.* In On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS, OTM Confederated International Conferences CoopIS, DOA, ODBASE, GADA, and IS 2007, Vilamoura, Portugal, November 25-30, 2007, Proceedings, Part I, pages 59–76, 2007. (Cited on page 73.)

[Namiri 2008] Kioumars Namiri. *Model-Driven Management of Internal Controls for Business Process Compliance.* PhD thesis, Universität Karlsruhe, 2008. (Cited on page 73.)

[Nguyen 2013] Thi-Hoa-Hue Nguyen and Nhan Le-Thanh. *Representation of RDF-Oriented Composition with OWL DL Ontology.* In Web Intelligence/IAT Workshops, pages 147–150, 2013. (Cited on pages 10 and 44.)

[Nguyen 2014a] Thi-Hoa-Hue Nguyen and Nhan Le-Thanh. *Ensuring the Correctness of Workflow Processes at the Syntactic Level: An Ontological Approach.* In The International Cross Domain Conference and Workshop (CD-ARES 2014)(Accepted), pages 1–15, September 2014. (Cited on pages 10 and 95.)

[Nguyen 2014b] Thi-Hoa-Hue Nguyen and Nhan Le-Thanh. *Ensuring the Semantic Correctness of Workflow Processes: An Ontological Approach.* In Grzegorz J. Nalepa and Joachim Baumeister, editeur, Proceedings of 10th Workshop on Knowledge Engineering and Software Engineering (KESE10) co-located with 21st European Conference on Artificial Intelligence (ECAI 2014), volume 1289. CEUR Workshop Proceedings, Prague, Czech Republic, August 2014. (Cited on pages 10, 73 and 95.)

[Nguyen 2014c] Thi-Hoa-Hue Nguyen and Nhan Le-Thanh. *An Ontology-Enabled Approach for Modelling Business Processes.* In Beyond Databases, Architectures, and Structures, volume 424 of *Communications in Computer and Information Science*, pages 139–147. Springer International Publishing, 2014. (Cited on pages 10, 34 and 44.)

[Nguyen 2015] Thi-Hoa-Hue Nguyen and Nhan Le-Thanh. *Coloured Petri Nets-based Approach for Manipulating RDF Data.* Journal of Automation and Control Engineering (JOACE), vol. 3, no. 2, pages 171–177, April 2015. (Cited on pages 10 and 44.)

[ocl 2014] *Object Constraint Language (OCL), Version 2.4.* http://www.omg.org/spec/OCL/2.4/, February 2014. (Cited on page 18.)

[omg 2000] *Workflow Management Facility Specification, V1.2.* http://www.workflowpatterns.com/documentation/documents/00-05-02.pdf, 2000. (Cited on page 1.)

[owl 2004] *OWL Web Ontology Language Overview* . http://http://www.w3.org/TR/owl-features/, February 2004. W3C Recommendation. (Cited on page 21.)

[owl 2012] *OWL 2 Web Ontology Language Document Overview (Second Edition).* http://www.w3.org/TR/owl2-overview/, December 2012. W3C Recommendation. (Cited on page 22.)

[Pesic 2007] Maja Pesic and Wil M. P. van der Aalst. *Modelling work distribution mechanisms using Colored Petri Nets.* STTT, vol. 9, no. 3-4, pages 327–352, 2007. (Cited on page 28.)

[Petri 1962] Carl Adam Petri. *Communication with Automata (Kommunikation mit Automaten, in German).* PhD thesis, University of Bonn, 1962. (Cited on pages 17 and 92.)

[Pham 2015] Tuan Anh Pham, Thi-Hoa-Hue Nguyen and Nhan Le Thanh. *Ontology-based workflow validation.* In The 2015 IEEE RIVF International Conference on Computing & Communication Technologies - Research, Innovation, and Vision for Future, RIVF 2015, Can Tho, Vietnam, January 25-28, 2015, pages 41–46, 2015. (Cited on pages 10, 73 and 95.)

[prr 2009] *Production Rule Representation (PRR) Version 1.0.* http://www.omg.org/spec/PRR/1.0/, December 2009. (Cited on page 18.)

[RDF 2014a] *RDF 1.1 Concepts and Abstract Syntax.* http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/, February 2014. (Cited on page 20.)

[rdf 2014b] *RDF Schema 1.1.* http://www.w3.org/TR/rdf-schema/, February 2014. (Cited on page 21.)

[Resch 2010] Olaf Resch. *Six Views on the Business Rule Management System.* e-Journal of Practical Business Research, vol. 11, page 8, 2010. (Cited on page 18.)

[Romanenko 2006] Inna Romanenko, Aufgabesteller Prof, François Bry, Betreuer Prof, François Bry, Paula lavinia Pătrânjan and Abgabetermin Januar. *Use Cases for Reactivity on the Web: Using ECA Rules for Business Process Modeling.* Rapport technique, Institut für informatik der der Ludwig-Maximilian-Universität München, 2006. (Cited on page 18.)

[Sadiq 2000] Wasim Sadiq and Maria E. Orlowska. *Analyzing process models using graph reduction techniques.* Information Systems, vol. 25, no. 2, pages 117 – 134, 2000. The 11th International Conference on Advanced Information System Engineering. (Cited on page 93.)

[Sadiq 2005] Shazia W. Sadiq, Maria E. Orlowska and Wasim Sadiq. *Specification and validation of process constraints for flexible workflows.* Information Systems, vol. 30, no. 5, pages 349 – 378, 2005. (Cited on page 72.)

[sbv 2013] *Business Vocabulary And Business Rules (SBVR), V1.2.* http://www.omg.org/spec/SBVR/1.2/, November 2013. (Cited on page 18.)

[Sebastian 2008] Abraham Sebastian, Natalya Fridman Noy, Tania Tudorache and Mark A. Musen. *A Generic Ontology for Collaborative Ontology-Development Workflows.* In Knowledge Engineering: Practice and Patterns, 16th International Conference, EKAW 2008, Acitrezza, Italy, September 29 - October 2, 2008. Proceedings, pages 318–328, 2008. (Cited on page 42.)

[Sonntag 2010] Mirko Sonntag, Dimka Karastoyanova and Frank Leymann. *The Missing Features of Workflow Systems for Scientific Computations.* In Software Engineering 2010 - Workshopband (inkl. Doktorandensymposium), Fachtagung des GI-Fachbereichs Softwaretechnik, 22.-26.02.2010, Paderborn, pages 209–216, 2010. (Cited on pages 12 and 13.)

[spa 2013] *SPARQL 1.1 Query Language.* http://www.w3.org/TR/sparql11-query/, March 2013. W3C Recommendation. (Cited on page 23.)

[Strbac 2013] Perica Strbac and Gradimir V. Milovanović. *Upgraded Petri net model and analysis of adaptive and static arithmetic coding.* Mathematical and Computer Modelling, vol. 58, no. 7-8, pages 1548–1562, 2013. (Cited on page 43.)

[Taveter 2001] Kuldar Taveter and Gerd Wagner. *Agent-Oriented Enterprise Modeling Based on Business Rules.* In Proceedings of the 20th International Conference on Conceptual Modeling: Conceptual Modeling, ER '01, pages 527–540, London, UK, UK, 2001. Springer-Verlag. (Cited on page 19.)

[The AIS group, Eindhoven University of Technology 2013] The AIS group, Eindhoven University of Technology. *CPN Tools, version 4.0.0.* http://cpntools.org/, September 2013. (Cited on pages 17 and 28.)

[Thomas 2009a] Oliver Thomas and Michael Fellmann. *Semantic Process Modeling - Design and Implementation of an Ontology-based Representation of Business Processes.* Business & Information Systems Engineering, vol. 1, no. 6, pages 438–451, 2009. (Cited on pages 42 and 43.)

[Thomas 2009b] Oliver Thomas and Michael Fellmann. *Semantic Process Modeling - Design and Implementation of an Ontology-based Representation of Business Processes.* Business & Information Systems Engineering, vol. 1, no. 6, pages 438–451, 2009. (Cited on page 95.)

[van der Aalst 1997] W.M.P. van der Aalst. *Verification of Workflow Nets.* In Proceedings of the 18th International Conference on Balbo Petri Nets, ICATPN '97, pages 407–426, London, UK, 1997. Springer-Verlag. (Cited on pages 17, 76, 77, 92 and 94.)

[van der Aalst 1998] Wil M. P. van der Aalst. *The Application of Petri Nets to Workflow Management.* Journal of Circuits, Systems, and Computers, vol. 8, no. 1, pages 21–66, 1998. (Cited on pages 1, 15, 16 and 29.)

[van der Aalst 2000] W. M. P. van der Aalst and Arthur H. M. ter Hofstede. *Verification Of Workflow Task Structures: A Petri-net-baset Approach.* Inf. Syst., vol. 25, no. 1, pages 43–69, 2000. (Cited on pages 17 and 92.)

[van der Aalst 2002a] Wil M. P. van der Aalst and A. H. M. Ter Hofstede. *Workflow Patterns: On the Expressive Power of (Petri-net-based) Workflow Languages.* In of DAIMI, University of Aarhus, pages 1–20, 2002. (Cited on page 17.)

[van der Aalst 2002b] Wil M. P. van der Aalst and Kees M. van Hee. Workflow management: Models, methods, and systems. MIT Press, 2002. (Cited on page 15.)

[van der Aalst 2003a] Wil M. P. van der Aalst. *Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Management.* In Lectures on Concurrency and Petri Nets, Advances in Petri Nets [This tutorial volume originates from the 4th Advanced Course on Petri Nets, ACPN 2003, held in Eichstätt, Germany in September 2003. In addition to lectures given at ACPN 2003, additional chapters have been commissioned], pages 1–65, 2003. (Cited on page 15.)

[van der Aalst 2003b] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, Bartek Kiepuszewski and Alistair P. Barros. *Workflow Patterns.* Distributed and Parallel Databases, vol. 14, no. 1, pages 5–51, 2003. (Cited on page 16.)

[van Eijndhoven 2008] Tim van Eijndhoven, Maria-Eugenia Iacob and María Laura Ponisio. *Achieving Business Process Flexibility with Business Rules.* In 12th International IEEE Enterprise Distributed Object Computing Conference, ECOC 2008, 15-19 September 2008, Munich, Germany, pages 95–104, 2008. (Cited on pages 18, 19 and 73.)

[Verbeek 2001] H. M. W. (Eric) Verbeek, Twan Basten and Wil M. P. van der Aalst. *Diagnosing Workflow Processes using Woflan.* The Computer Journal, vol. 44, no. 4, pages 246–279, 2001. (Cited on pages 79 and 92.)

[Wagner 2002] Gerd Wagner. *How to Design a General Rule Markup Language?* In XML Technologien FÜR Das Semantic Web - XSW 2002, Proceedings Zum Workshop, pages 19–37, 2002. (Cited on page 19.)

[Weber 2010] Ingo Weber, Jörg Hoffmann and Jan Mendling. *Beyond soundness: on the verification of semantic business process models.* Distributed and Parallel Databases, vol. 27, no. 3, pages 271–343, 2010. (Cited on page 95.)

[Weske 1998] Mathias Weske and Gottfried Vossen. *Workflow Languages.* In Peter Bernus, Kai Mertins and Günter Schmidt, editeurs, Handbook on Architectures of Information Systems, International Handbooks on Information Systems, pages 359–379. Springer Berlin Heidelberg, 1998. (Cited on page 16.)

[WFMC 1999] WFMC. *Workflow Management Coalition Terminology and Glossary (WFMC-TC-1011), Document Number WFMC-TC-1011.* Rapport technique, 1999. (Cited on page 11.)

[Yildiz 2009] U. Yildiz, A. Guabtni and A.H.H. Ngu. *Business versus Scientific Workflows: A Comparative Study.* In Services - I, 2009 World Conference on, pages 340–343, July 2009. (Cited on page 12.)

[Zhang 2011] Fu Zhang, Z. M. Ma and Slobodan Ribaric. *Representation of Petri net with OWL DL ontology.* In Eighth International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2011, 26-28 July 2011, Shanghai, China, pages 1396–1400, 2011. (Cited on page 42.)