

MODELING CLINICAL PATHWAYS AS BUSINESS PROCESS MODELS  
USING BUSINESS PROCESS MODELING NOTATION

by

Nima Hashemian

Submitted in partial fulfilment of the requirements  
for the degree of Master of Computer Science

at

Dalhousie University  
Halifax, Nova Scotia  
March 2012

© Copyright by Nima Hashemian, 2012

DALHOUSIE UNIVERSITY

FACULTY OF COMPUTER SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled “MODELING CLINICAL PATHWAYS AS BUSINESS PROCESS MODELS USING BUSINESS PROCESS MODELING NOTATION” by Nima Hashemian in partial fulfilment of the requirements for the degree of Master of Computer Science.

Dated: March 5, 2012

Supervisor: \_\_\_\_\_

Readers: \_\_\_\_\_

\_\_\_\_\_

DALHOUSIE UNIVERSITY

DATE: March 5, 2012

AUTHOR: Nima Hashemian

TITLE: Modeling Clinical Pathways as Business Process Models using Business Process Modeling Notation

DEPARTMENT OR SCHOOL: Faculty of Computer Science

DEGREE: M.C.Sc CONVOCATION: May YEAR: 2012

Permission is herewith granted to Dalhousie University to circulate and to have copied for non-commercial purposes, at its discretion, the above title upon the request of individuals or institutions. I understand that my thesis will be electronically available to the public.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

The author attests that permission has been obtained for the use of any copyrighted material appearing in the thesis (other than the brief excerpts requiring only proper acknowledgement in scholarly writing), and that all such use is clearly acknowledged.

---

Signature of Author

# TABLE OF CONTENTS

<b>LIST OF TABLES .....</b>	<b>vii</b>
<b>LIST OF FIGURES .....</b>	<b>viii</b>
<b>ABSTRACT.....</b>	<b>xi</b>
<b>LIST OF ABBREVIATIONS USED.....</b>	<b>xii</b>
<b>ACKNOWLEDGEMENTS .....</b>	<b>xiii</b>
<b>CHAPTER 1 INTRODUCTION .....</b>	<b>1</b>
1.1 Research Motivation and Problem Statement.....	1
1.2 Research Objective and Goals .....	3
1.3 Solution Approach .....	6
<b>CHAPTER 2 WORKFLOWS AND WORKFLOW MODELING.....</b>	<b>10</b>
2.1 Workflow .....	10
2.1.1 Workflow Purposes, Benefits, and Capabilities.....	13
2.1.2 Workflow Elements .....	13
2.1.3 Workflow Execution and Workflow Management System .....	17
2.1.4 Workflow Management System Architecture.....	19
2.1.5 Workflow Modeling Approaches.....	22
2.1.6 Workflow Formalisms .....	24
2.2 Business Process Modeling.....	27
2.2.1 Event-Driver Process Chain (EPC).....	28
2.2.2 Petri-Nets .....	30
2.2.3 Unified Modeling Language (UML).....	32
2.2.4 Business Process Execution Language (BPEL).....	33
2.3 Business Process Modeling Notation (BPMN).....	34
2.4 Comparing BPMN with other Languages.....	42
<b>CHAPTER 3 COMPUTERIZING CP AND CPG.....</b>	<b>46</b>
3.1 PROforma .....	47
3.2 Asbru.....	48
3.3 Gaston .....	50
3.4 SAGE .....	51
3.5 Semantic-based CP Workflow and Variance Management System .....	52
3.6 Comparison of CPG Formalisms .....	53



<b>CHAPTER 4</b>	<b>ONTOLOGIES IN USE: CP AND BPMN ONTOLOGIES.....</b>	<b>57</b>
4.1	Ontology .....	57
4.2	The CP Ontology .....	60
4.3	The BPMN Ontology.....	66
4.4	An Introduction to Ontology Mapping.....	72
4.5	Mapping Techniques.....	76
4.5.1	Terminological Techniques.....	76
4.5.2	Structural Techniques .....	77
4.5.3	Extensional Techniques .....	78
4.5.4	Semantic Techniques .....	78
4.6	Tools and Frameworks.....	79
4.6.1	PROMPT.....	79
4.6.2	GLUE.....	83
4.6.3	MAFRA .....	84
<b>CHAPTER 5</b>	<b>ONTOLOGY MAPPING.....</b>	<b>87</b>
5.1	CP-BPMN Ontology Mapping .....	89
5.2	CP-BPMN Mapping Expressions .....	92
5.2.1	Step 2: Mapping Discovery.....	93
5.2.2	Step 3: Documenting the Mappings.....	99
5.2.2.1	Class-Class Mapping .....	99
5.2.2.2	Property-Property Mapping.....	104
5.2.2.3	Class-Property Mapping .....	109
5.2.2.4	Property-Instance Mapping .....	111
5.2.3	Step 4: Consistency Checking and the Output.....	116
5.3	Extended BPMN Ontology.....	117
<b>CHAPTER 6</b>	<b>MODELING AND EXECUTION OF CP IN LOMBARDI.....</b>	<b>121</b>
6.1	Introduction to Lombardi.....	122
6.2	Lombardi Ontology.....	124
6.3	BPMN-Lombardi Ontology Mapping.....	128
6.4	Modeling CP in Lombardi .....	133
<b>CHAPTER 7</b>	<b>EVALUATION .....</b>	<b>140</b>
7.1	The Encoded CP in the CP Ontology .....	141
7.2	Encoding CP in BPMN and Lombardi Ontologies.....	142

<b>CHAPTER 8 CONCLUSION .....</b>	<b>147</b>
8.1 Enhancing the CP Ontology.....	149
8.2 Limitations and Future Work.....	150
<b>BIBLIOGRAPHY .....</b>	<b>152</b>

## LIST OF TABLES

Table 2.1	Expressiveness of Workflow patterns in Petri nets [90] .....	31
Table 2.2	Modeling constructs of different business process modeling languages [89].....	44
Table 3.1	Comparing computer guideline models [10] .....	54
Table 3.2	Modeling constructs or primitives [10] .....	55
Table 4.1	The mapping between the workflow constructs in CP ontology and BPMN ontology.....	71
Table 5.1	The mapping between the classes of the CP and BPMN ontologies.....	96
Table 5.2	The mapping between the properties of the CP and BPMN ontologies .....	97
Table 6.1	The mappings between the classes of BPMN and Lombardi ontologies ..	130

## LIST OF FIGURES

Figure 1.1	Thesis structure .....	9
Figure 2.1	Workflow glossary [104] .....	12
Figure 2.2	The standard workflow elements [106] .....	14
Figure 2.3	A business workflow with the BPMN elements [57] .....	15
Figure 2.4	A fragment of a scientific workflow [120] .....	15
Figure 2.5	Comparison of scientific workflows and business workflows at different levels [120] .....	16
Figure 2.6	A clinical workflow with the standard workflow elements [106] .....	17
Figure 2.7	The main components of a workflow management system [111] .....	20
Figure 2.8	Different frameworks for modeling a workflow [113] .....	23
Figure 2.9	Transformation rule for a decision and activity state to Petri-net [115] .....	25
Figure 2.10	EPC diagram [116] .....	30
Figure 2.11	The basic elements of UML activity diagram [118] .....	33
Figure 2.12	An example of a private business process .....	35
Figure 2.13	An example of an abstract business process .....	36
Figure 2.14	An example of a collaboration business process .....	36
Figure 2.15	Flow objects [24] .....	37
Figure 2.16	The complete list of BPMN event types [24] .....	37
Figure 2.17	Activities [24] .....	38
Figure 2.18	The complete list of BPMN gateway types [24] .....	39
Figure 2.19	Connecting objects [31] .....	39
Figure 2.20	A pool with two lanes [31] .....	40
Figure 2.21	Artifacts [31] .....	40
Figure 2.22	Attaching data object to the sequence flow .....	41

Figure 2.23	Association lines between data objects and a task.....	41
Figure 4.1	The CP ontology [56] .....	61
Figure 4.2	CP ontology - The subclasses of the Guideline_Step class [56] .....	64
Figure 4.3	Physic patient evaluation workflow [98].....	65
Figure 4.4	Ontology mapping steps [67] .....	72
Figure 4.5	The snapshot of the ontology merging process in Prompt [77].....	81
Figure 4.6	The flow of Prompt algorithm [77] .....	81
Figure 4.7	Concept and Attribute Bridge in MAFRA [79].....	86
Figure 5.1	The overall ontology-mapping framework.....	88
Figure 5.2	The actual class-class mapping for the Admission_Step to the instance of the User_Task class .....	101
Figure 5.3	The actual class-class mapping for the provider and system decision steps to the event and data based exclusive gateways .....	102
Figure 5.4	The Event_Based_Gateway construct.....	103
Figure 5.5	The actual property-property mapping for the author property to the has_business_process_diagram_author property.....	105
Figure 5.6	The actual property-property mapping for the properties of the Date_Time class of the CP ontology .....	106
Figure 5.7	The actual property-property mapping for the domain and range of the condition_to_go_forward .....	108
Figure 5.8	Mapping the next_step property to the Sequence_Flow class.....	109
Figure 5.9	The actual property-class mapping for the next_step property to the Sequence_Flow class.....	110
Figure 5.10	Mapping the branching_steps property to the Parallel_Gateway class .....	112
Figure 5.11	The actual property-instance mapping for the Branching_Step to the Parallel_Gateway class.....	113
Figure 5.12	The actual property-instance mapping for the acceptable_duration_of_results to the instances of the User_Task and the Time_Date_Expression class of the BPMN ontology .....	115

Figure 5.13	Copying the subclasses of the Action_Steps to the subclasses of the User_Task class (only some of the subclasses are shown here).....	118
Figure 5.14	The actual class-class mapping for the subclasses of the Action_Steps to the subclasses of the User_Task class .....	119
Figure 6.1	The architecture of IBM WebSphere Lombardi [7] .....	122
Figure 6.2	The constructs of Lombardi [7] .....	124
Figure 6.3	Lombardi ontology .....	127
Figure 6.4	Creating separate lanes for each of the subclasses of the User_Task class .....	132
Figure 6.5	The actual Class-Class mapping for the Admission_Step class to the instance of the Human_Service class .....	132
Figure 6.6	PMRT CP [56].....	133
Figure 6.7	PMRT CP in Lombardi.....	134
Figure 6.8	Inclusion Criteria interface, designed by a coach component .....	137
Figure 6.9	Rule service component.....	138
Figure 6.10	Decision options in a gateway .....	138
Figure 6.11	Attaching a message or timer intermediate event to an activity .....	139
Figure 7.1	The instantiations of the CP ontology .....	142
Figure 7.2	The initial part of PMRT encoding in the BPMN ontology .....	143
Figure 7.3	The initial part of AOM encoding in the Lombardi ontology .....	145
Figure 8.1	Class hierarchy of the new constructs for the domain ontology.....	150

## **ABSTRACT**

We take a healthcare knowledge management approach to represent the Clinical Pathway (CP) as workflows. We have developed a semantic representation of CP in terms of a CP ontology that outlines the different clinical processes, their properties, constraints and relationships, and is able to computerize a range of CP. To model business workflows we use the graphical Business Process Modeling Notation (BPMN) modeling language that generates a BPMN ontology. To represent a CP as a BPMN workflow, we have developed a semantic interoperability (mapping ontology) framework between the CP ontology and the BPMN ontology. The mapping ontology allows the alignment of relations between two ontologies and ensures that a clinical process defined in the CP ontology is mapped to a standard BPMN workflow element. We execute our BPMN-based CP in the Lombardi workflow engine, whereby users can view the execution of the CP and make the necessary adjustments.

## LIST OF ABBREVIATIONS USED

BPD	Business Process Diagram
BPEL	Business Process Execution Language
BPEL4WS	Business Process Execution Language for Web Service
BPML	Business Process Modeling Language
BPMN	Business Process Modeling Notation
BPMNO	Business Processes Modeling Notation Ontology
CP	Clinical Pathway
CPG	Clinical Practice Guidelines
CPO	Clinical Pathway Ontology
DL	Description Logic
EBM	Evidence Based Medicine
EPC	Event-driven Process Chain
FOL	First Order Logic
GEL	Guideline Expression Language
HTML	HyperText Markup Language
IRIS	Integrated Rule Inference System
LP	Logic Programming
MAFRA	Mapping FRAmework
NDF-RT	National Drug File-Reference Terminology
NLP	Natural Language Processing
OKBC	Open Knowledge Base Connectivity
OMG	Object Management Group
OWL	Web Ontology Language
PSM	Problem Solving Method
RDF	Resource Description Format
SAGE	Standards-Based Active Guideline Environment
SBO	Semantic Bridge Ontology
SWRL	Semantic Web Rule Language
TNM	Task-Network Models
TURTLE	Terse RDF Triple Language
UML	Unified Modeling Language
UML AD	Unified Modeling Language Activity Diagram
UNA	Unique Name Assumption
VMR	Virtual Medical Record
URI	Uniform Resource Identifier
VMR	Virtual Medical Record
W3C	World Wide Web Consortium
WfMS	Workflow Management System
WSDL	Web Service Description Language
WSML	Web Service Management Layer
XML	Extensible Markup Language
YAWL	Yet Another Workflow Language



## **ACKNOWLEDGEMENTS**

First and foremost I would like to express my gratitude to Dr. Raza Abidi, for his supervision, encouragement, support and guidance from the initial to the final stages of this research. I would not have accomplished the objectives of this thesis without his supervision.

I am also thankful to Dr. Michael Shepherd and Dr. Grace Paterson for being readers in the examining committee, and providing their valuable feedback.

I am grateful to Dalhousie University, and the faculty of Computer Science for providing excellent facilities and a supportive environment.

Last but not least, I am tremendously grateful to my family and friends who have supported and encouraged me in different ways during this journey.

**NIMA  
MARCH 2012**

## **CHAPTER 1 INTRODUCTION**

Clinical Pathways (CP) can be created from Clinical Practice Guidelines (CPG). Clinical pathways and clinical practice guidelines provide standardized and structured health care to physicians and patients. They improve the quality of health care and provide recommendations for the treatment of diseases to physicians [1].

According to the journal of nursing management [2], *“A clinical pathway is a method for the patient-care management of a well-defined group of patients during a well-defined period of time. A clinical pathway explicitly states the goals and key elements of care based on evidence-based-medicine guidelines, best practice and patient expectations by facilitating the communication, coordinating roles and sequencing the activities of the multidisciplinary care team, patients and their relatives; by documenting, monitoring and evaluating variances; and by providing the necessary resources and outcomes. The aim of a clinical pathway is to improve the quality of care, reduce risks, increase patient satisfaction and increase the efficiency in the use of resources”*.

Clinical practice guidelines are broadly defined as: *“systematically developed statements to assist practitioner and patient decisions about appropriate health care for specific clinical circumstances”* [3].

### **1.1 RESEARCH MOTIVATION AND PROBLEM STATEMENT**

Clinical pathways can be computerized and executed by providing CP ontology, which can be done by a domain expert. A logic-based execution engine can execute an instantiation of CP from CP ontology in order to provide specific recommendations for a

patient [92]. For instance, the SAGE execution engine, processes the guidelines encoded using the SAGE guideline model. The execution engine interprets the actions and decisions in a guideline, executes workflows based on the decision logic, and interacts with clinical information systems [13].

However, the execution of a computerized CP could be challenging, model specific, non-formal, non-standard (reusability, interoperability, analysis) and not connected to resources.

A possible solution approach is to represent CP as the workflows modeling languages, since CP contain workflows; CP describe the medical domain knowledge (i.e. diagnosis and treatment of diseases) and functional knowledge, which is represented as a combination of tasks, plans, decisions, involved users and resources—the combination of these elements provides a workflow structure [1]. CP encapsulates the workflow about how to conduct a specific healthcare procedure for a specific disease/outcome in a specific healthcare setting.

Representing CP as workflows and applying business process modeling principles in the design of CP will ensure data interoperability, resource management and task prioritization. This will yield executable CP that clearly articulate (a) roles and responsibilities of care providers; (b) decision points and care options; (c) well-identified clinical/business rules; (d) handling of operational constraints; (e) task scheduling; and (f) temporal constraints [1].

However, there are some barriers to represent CP as a workflow modeling language. Related barriers are as follows:

- 1. Standard formalisms:** The use of workflow modeling concepts in the design and optimization of CP is not yet well established, and as such there are no standard formalisms for the representation of CP in general, and CP as workflow models in particular.
- 2. Semantic interoperability:** Workflow modeling notation is not based on healthcare processes, therefore no semantic interoperability exists between the workflow modeling notation and the CP ontology.
- 3. Execution Engine:** BPMN modeling language is chosen for the purpose of this thesis. BPMN is a graphical language, and it is easy to understand by both domain experts and business process modelers [94]. However, no execution engine exists for BPMN.
- 4. Expressivity of a tool as an engine for BPMN:** By providing a tool as an execution engine for our BPMN-based CP (*developing an execution engine for BPMN is out of the scope of this study*), we may lose the expressivity of BPMN specification, since there is no tool that can provide the same level of workflow expressiveness and abstraction as the BPMN specification. BPMN is a much richer workflow representation formalism.

## **1.2 RESEARCH OBJECTIVE AND GOALS**

There is a case to explore the potential of business process modeling principles and workflow modeling formalisms—such as Business Process Modeling Notation (BPMN), Business Process Execution Language (BPEL), Unified Modeling Language (UML),

etc.—to design standardized CP that can be executed through workflow execution engines.

Our research objective is to model a number of existing CP to a BPMN based workflow-modeling language—The semantic description of the CP tasks ensures that the transformation of a CP to a BPMN workflow maintains the clinical pragmatics of the CP. The graphical notation of the CP enables rapid user feedback and adjustments to optimize performance metrics. We pursue a number of goals to meet our objective, as follows:

- 1. Development of a semantic interoperability framework:** We have the semantic representation of CP in terms of a clinically oriented CP ontology that outlines the different clinical processes, their properties, constraints and relationships, and is able to computerize a range of CP. To model CP as business workflows we use a workflow-oriented BPMN ontology that contains a semantic description of BPMN constructs.

Our main goal is to develop a semantic interoperability (or ontology mapping) between the CP ontology and the BPMN ontology, since BPMN notation is not based on healthcare processes. The ontology mapping allows the alignment of semantic relations between the clinical and workflow ontologies and thus ensures that a clinical process defined in the CP ontology is mapped to a standard BPMN workflow element.

There is much literature that provides semantic interoperability at the data level, for instance, the semantic interoperability between the health informatics standards, such as HL7 (for messaging), openEHR (patient records) and SNOMED (standard terminology), can be provided to prompt interoperability at

the data level [124]. However, our objective is not to unify ontologies or data, the main objective is to exploit the computerized CP modeled in terms of the CP ontology, to develop a workflow model of the CP.

By developing this mapping ontology, we will establish semantic mappings between the elements of our BPMN and CP ontologies. We will get our BPMN-based CP model by encoding different CP in our BPMN ontology by help of this mapping ontology.

2. **Execution of CP in Lombardi workflow engine:** We need to execute our BPMN-based CP model in a workflow execution engine. Lombardi, a tool from IBM offers a workflow execution environment, which is very useful for analyzing and executing our BPMN-based CP model. It enables users to view the execution of CP and make necessary adjustments to optimize it.
3. **Richer specification for Lombardi:** Lombardi is a tool and not a workflow language. We cannot model and execute our CP in it directly, since it is not based on a workflow language. It only provides a few constructs for modeling and executing business process. In addition, it does not provide the same level of workflow expressiveness and abstraction as our BPMN ontology.  
  
The objective is to model and execute our BPMN-based CP model in Lombardi, and to provide a richer specification for its constructs, since it is not based on the BPMN sepecification.
4. **Enhancing the CP ontology:** We propose some extensions to enhance our CP ontology in order to capture more complex workflow structure of the clinical

pathways. We propose these extensions by studying the constructs of the BPMN ontology.

### **1.3 SOLUTION APPROACH**

Our solution approach is to define a mapping expression language that allows the alignment of relations between two ontologies—the relations are represented in terms of *mapping expressions*. The mapping expressions, therefore allow the mapping of a CP to a business workflow represented using the BPMN modeling language. The mapping expressions are represented in a *mapping ontology*—the mapping ontology basically establishes semantic mappings between the CP and BPMN ontologies, such that the concepts in the mapping expressions will have their domain and ranges defined as concepts in the CP and BPMN ontologies.

Lombardi is a tool and not a workflow language. It is not based on the BPMN specification; it only provides a few constructs for modeling and executing business process. In order to model and execute our BPMN-based CP model in Lombardi, and to provide richer specification for its constructus in terms of BPMN specification: (a) we create an ontology for Lombardi to formalize the structure of the Lombardi constructs; (b) we establish a mapping ontology between the BPMN and Lombardi ontologies. The mapping ontology establishes semantic mappings between the elements of these two ontologies, and it can represent the Lombardi constructs in terms of our BPMN ontology. The mapping expressions in the mapping ontology enable us to model our BPMN-based CP model by the Lombardi constructs. After modeling our BPMN-based CP model in Lombardi, we can execute our model and it results to the CP execution.

The result is that we have a *semantic interoperability* framework whereby clinical processes/pathways can be conveniently mapped to business process notations thus enabling CP to be executed and simulated for adjusting various cost functions. It is achieved by providing a mapping ontology that establishes a high-level semantic mapping between our ontologies. Our mapping framework allows healthcare professionals to model a CP using modeling constructs that they are familiar with, and then we transform their CP model to a business process model. The use of ontologies, at both representation and mapping levels, allow for the semantic description of concepts and their relations, with provisions for semantic classification of healthcare concepts to ensure the right level of conceptual granularity in the representation scheme.

This thesis is organized into eight chapters, and its structure is represented as a business process model in Figure 1.1.

**Chapter 2-Workflows and Workflow Modeling:** We review a number of modeling languages and notations for representing business processes and workflows. In addition, we gather state-of-the-art work on BPMN. We provide a high-level overview of BPMN, which is a modeling language for modeling business process.

**Chapter 3-Computerizing Clinical Pathways and Clinical Practice Guidelines:** We review a number of approaches and frameworks for computerizing clinical pathways and clinical practice guidelines.

**Chapter 4-Ontologies in use: CP and BPMN Ontologies:** In this chapter we provide an introduction to ontology, and then we discuss our CP and BPMN ontologies. We provide a literature review on ontology mapping techniques as well.



**Chapter 5-Ontology Mapping:** In this chapter, we discuss our ontology-mapping framework, and we provide examples for different types of our mapping expressions.

**Chapter 6-IBM WebSphere Lombardi:** In this chapter, first we provide an introduction to IBM WebSphere Lombardi v7.1 [7], which is a platform for modeling business process. We explain the created ontology for Lombardi, and then we discuss our ontology mapping between the BPMN and Lombardi ontologies. In addition, we explain how to model and execute clinical pathways in the Lombardi environment as well.

**Chapter 7-Evaluation** In this chapter, we evaluate our proposed semantic interoperability framework by encoding six different clinical pathways in our ontologies.

**Chapter 8-Conclusion** This chapter summarizes the findings and purposes future research. In particular, we discuss the implications of this thesis for a process modeling of clinical pathways in BPMN and their execution in IBM WebSphere Lombardi. We also propose to enhance our CP domain ontology and to make it more expressive by adding constructs from the BPMN ontology.

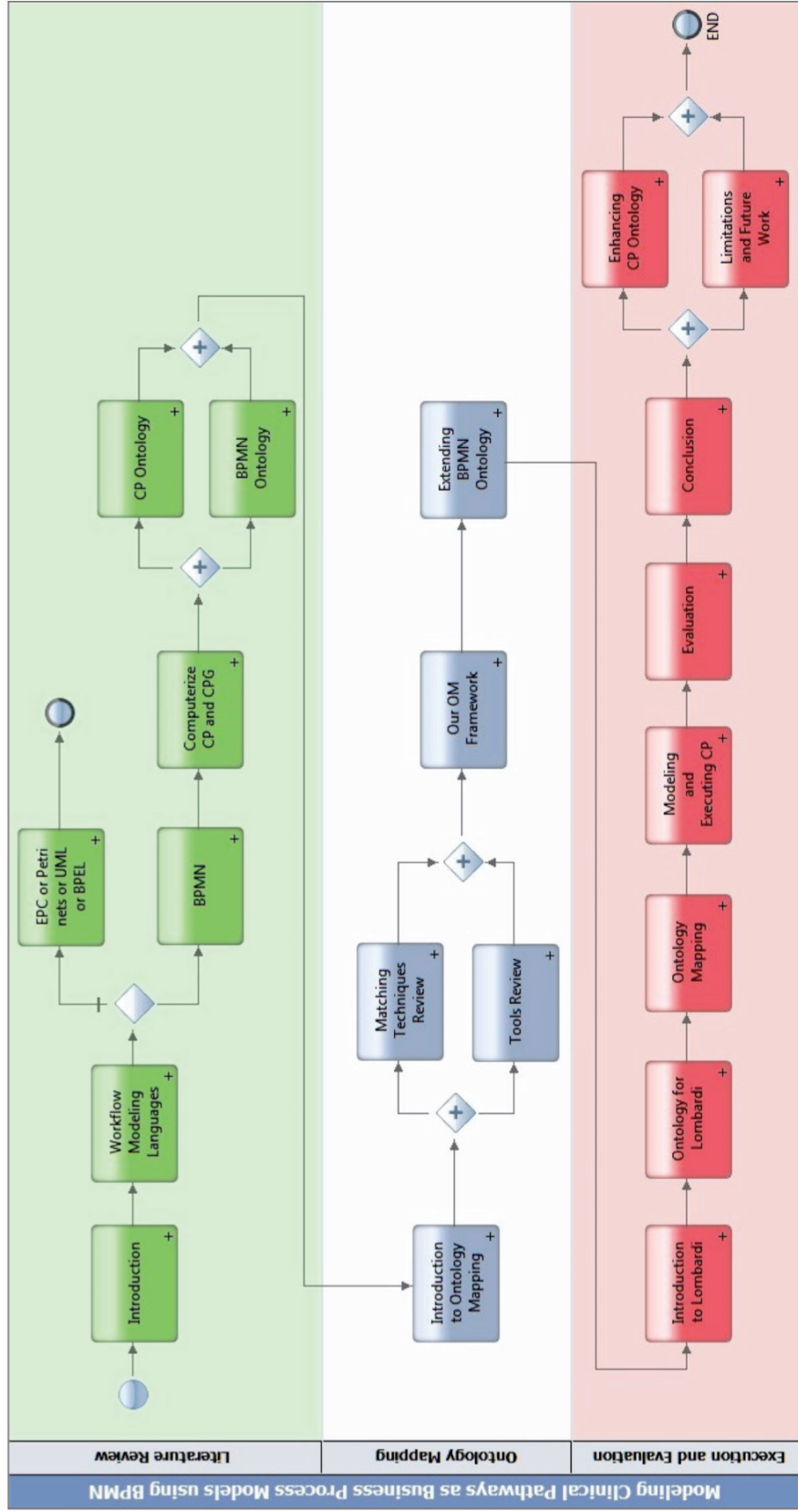


Figure 1.1 Structure of thesis

## CHAPTER 2      WORKFLOWS AND WORKFLOW MODELING

In this chapter, we provide an introduction to workflow, its purposes and benefits and the main components of workflows. Clinical workflows are included in CP and CPG; they model step-by-step procedures for medical treatment and decision-making. Clinical workflows demonstrate the ordering, control, and data flow among the tasks in a CP [1]. We also provide an overview of modeling languages for representing business processes and workflows.

### 2.1    WORKFLOW

The term workflow can be best understood as *“any work process that must go through certain steps and be handled by more than one person on its way to completion. Workflow automation relieves people of some of these tasks. Inherent in workflow are concepts of teamwork, request and approval, routing and tracking of documents, filling out forms and doing things either in series or parallel”* [93].

According to [102], a workflow definition describes a process that includes a sequence of tasks, activities and steps. It describes when and what activity has to be done and by whom. Four types of ordering relationships exist between activities, which are sequence, parallelism, choice and iteration [103].

Workflows have two important dimensions [103]:

- The process-logic dimension: It specifies the order of that tasks that have to be done in a period of time.

- The resource dimension: It concerns the organizational structure to specify who is responsible for the assigned task.

There are two types of activities [103] in a workflow process, which are the compound activity and the atomic activity. The compound activity contains a set of ordered activities that are combined together, and the atomic activity does not contain any other activities within itself. The compound activities can be used in other workflow definition.

The Workflow model (workflow specification) [107] provides the definition of a workflow, which a set of concepts to describe processes, tasks, the required roles to perform the tasks and the relations between them. It provides constructs to model decisions, branching, loop, synchronization, etc.

A workflow model can provide excellent business process models by providing workflow patterns. According to [5] the workflow patterns can be grouped into the four perspectives:

- The **control flow perspective** describes activities, and different constructors describe the execution ordering that allows the flow of execution control, e.g. sequence.
- The **data perspective** layers processing and business data on the control flow perspective.
- The **resource perspective** describes the devices and human roles that are responsible for executing activities in a workflow.
- The **operation perspective** describes the actions that are performed by activities.

There are different concepts and terminologies for a workflow. The relationships between these terminologies are illustrated in Figure 2.1 [104].

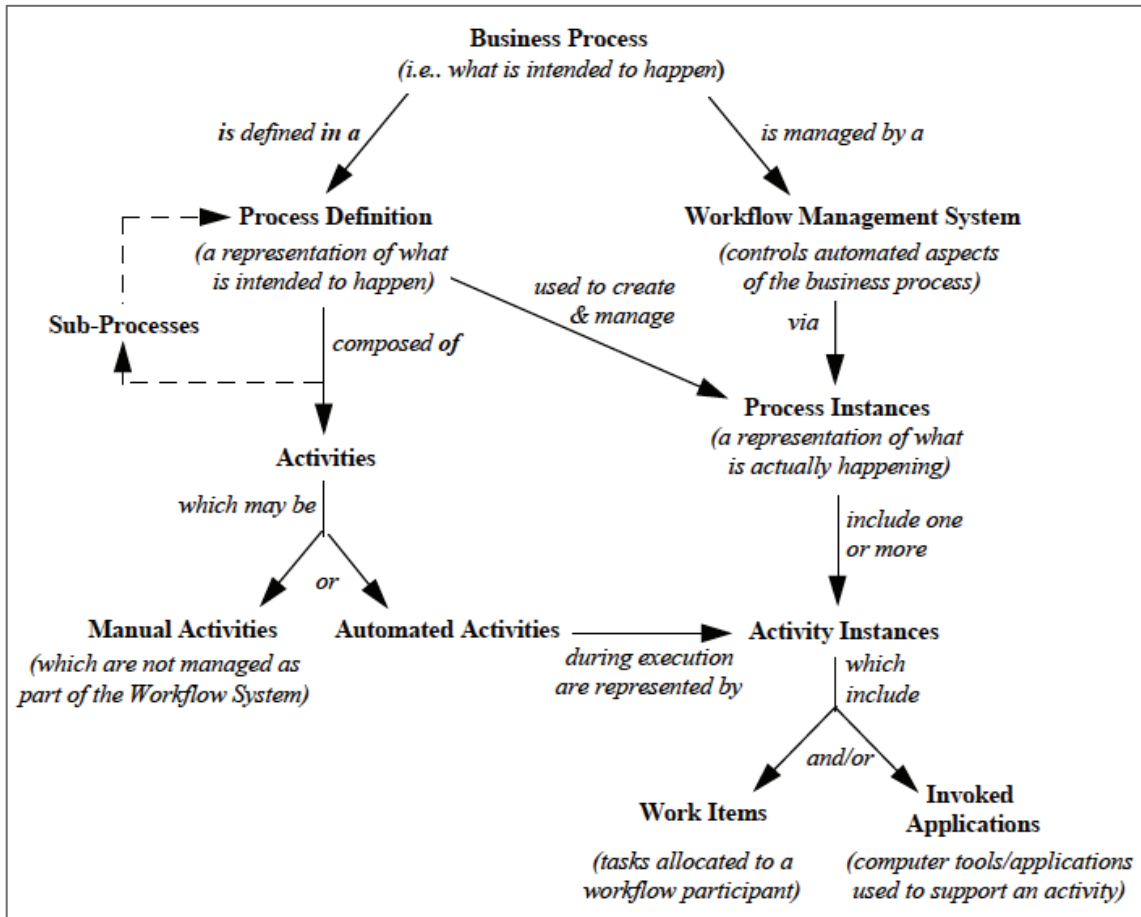


Figure 2.1 Workflow glossary [104]

### 2.1.1 Workflow Purposes, Benefits, and Capabilities

According to [105], a workflow has the following purposes:

- To manage a movement of a task from start to finish.
- To direct a task to the right person, by providing the right instructions.
- To ensure individuals accomplish the assigned tasks in the assigned time.
- To monitor and control the status of each task and process.

According to IBM workflow guide [105], a workflow provides the following benefits and capabilities:

- A task can be directed manually or automatically into a workflow process.
- A user is able to make a decision for an assigned task.
- A time limit can be set for completing a task.
- An instance of a process can be terminated in the workflow.
- A workflow process may contain multiple sub processes.

### 2.1.2 Workflow Elements

The workflow has some standard elements. We provide a snapshot of these elements in Figure 2.2 [106].

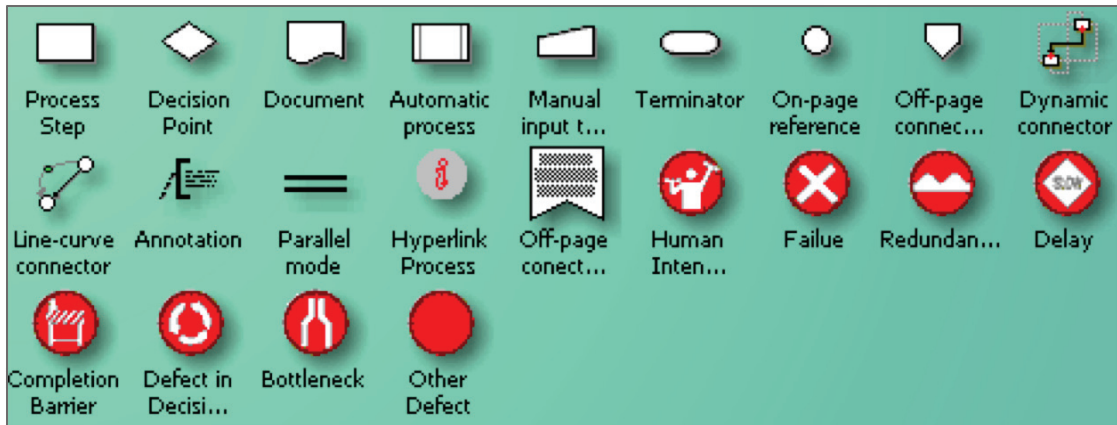


Figure 2.2 The standard workflow elements [106]

- Process step: It is a general or a manual step in a process.
- Decision point: Branching of a process, it controls the divergence and convergence of a flow.
- Document: A task that uses a paper document.
- Automatic process: The step that does not require a direct human interaction.
- Manual input task: A manual input or interaction with the system.
- Terminator/Start: The start or end of a process.
- Dynamic connector: It connects two steps.
- Parallel mode: It is a mode that two steps are simultaneous but independent.

There are different types of workflows:

- *Business workflows* [119,120] demonstrate how organizations achieve their objectives and goals by providing of a set of activities. Business workflows are control flow oriented, they form control flow to describe the flow of the execution from one task another task. Figure 2.3 [57] illustrates a business workflow, with the BPMN elements.

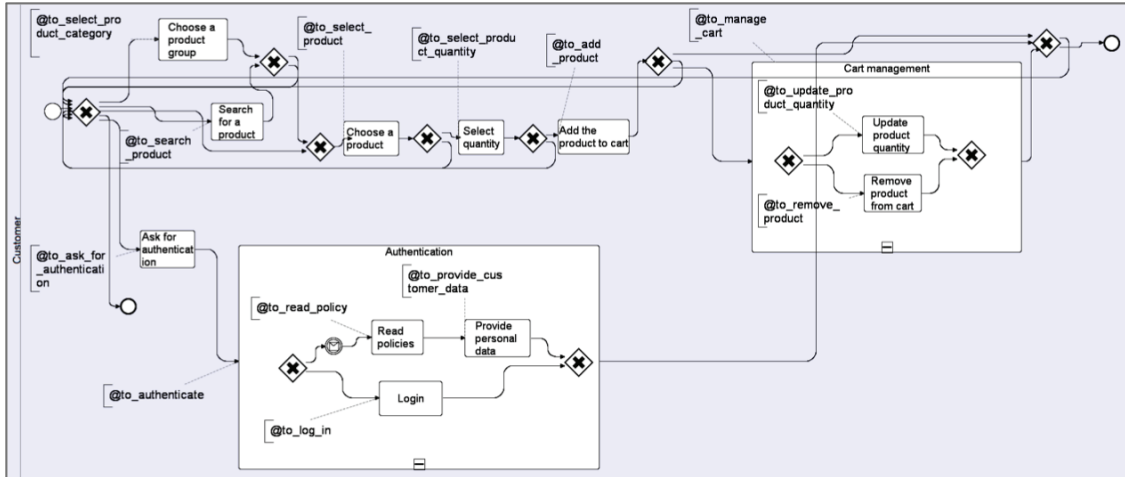


Figure 2.3 A business workflow with the BPMN elements [57]

- *Scientific workflows* [119,120] demonstrate the specification of scientific processes. They automate dataset selection, computation and visualization. They provide a set of constructs with different semantics than the traditional workflows for process modeling and execution. Figure 2.4 [120] illustrates a fragment of a scientific workflow.

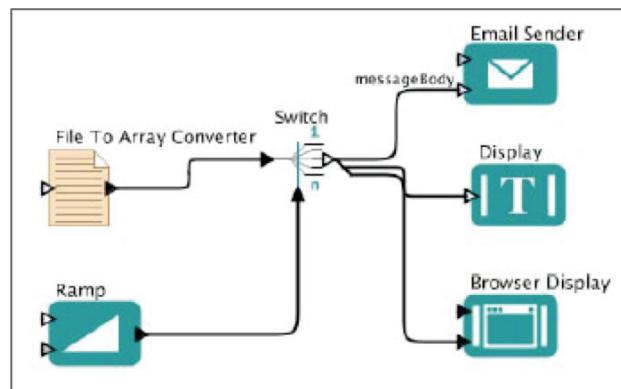


Figure 2.4 A fragment of a scientific workflow [120]

According to [120], scientific workflows are data flow oriented. They describe how input data are provided for the data analysis steps, in order to create workflow data products. Figure 2.5 [120] provides a comparison between



scientific workflows and business workflows at two different levels, which are the designer interpretation and execution environment level. As it is shown in Figure 2.5, at the designer interpretation level, the three tasks (“B”, “C”, “D”) use the output of “A” in the scientific workflow. However, in the business workflow, these three tasks are executed after the termination of “A”. At the execution level, all of the tasks (“A”, “B”, “C”, “D”, “E”, “F”) can be active within the same instance in the scientific workflow, but in the business workflow only the three tasks (“B”, “C”, “D”) can be active within the same instance.

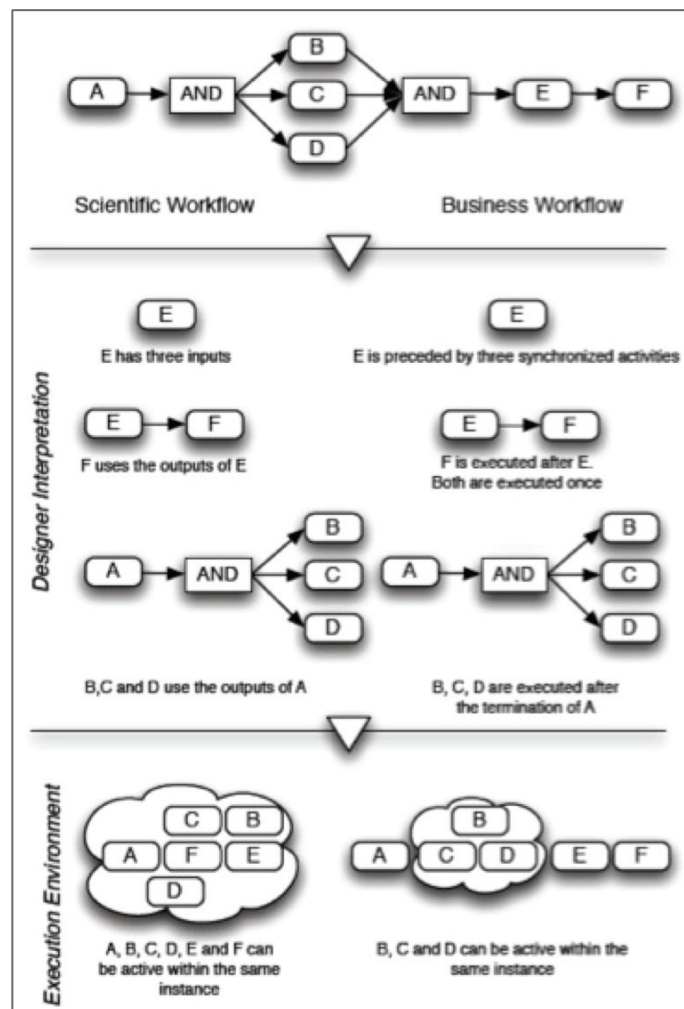


Figure 2.5 Comparison of scientific workflows and business workflows at different levels [120]

- Clinical workflows [1] model step-by-step procedures for medical care and decisions. Clinical workflows describe a series of tasks, how to achieve them and the ordering among the tasks. Figure 2.6 [106] illustrates a clinical workflow with the standard workflow elements.

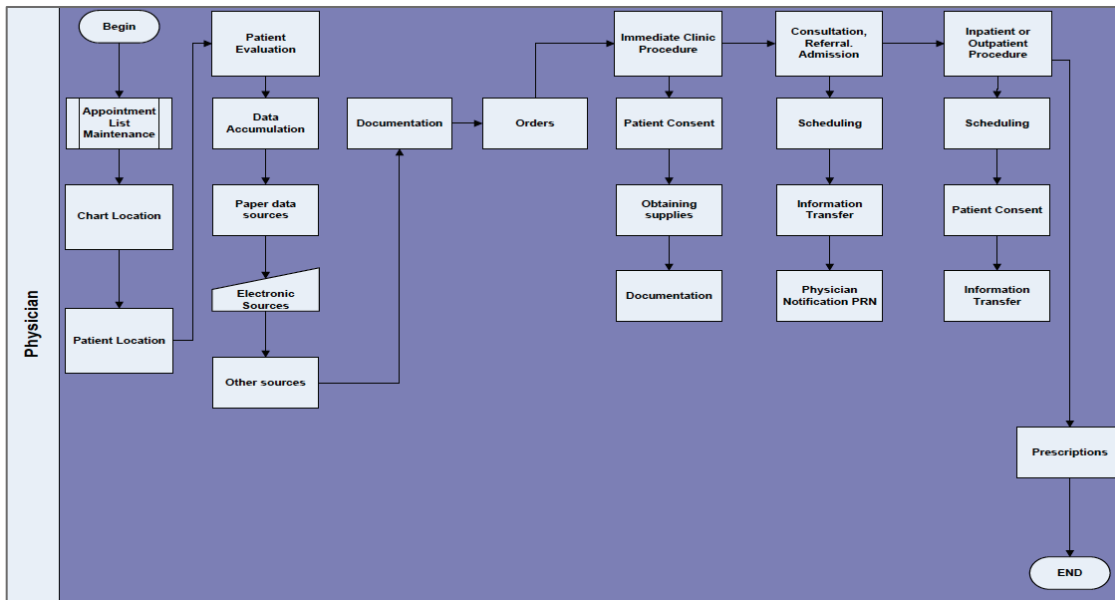


Figure 2.6 A clinical workflow with the standard workflow elements [106]

### 2.1.3 Workflow Execution and Workflow Management System

A computer-based workflow automates the business processes, which are a set of activities to accomplish the business objectives. It passes the data or information from one person to another person, based on a set of rules in a period of time. The automation of a business process is included in the workflow process definition, it provides different process activities and rules for managing the flow of activities [104].

According to [93], the workflow execution entails the traversal of the sequenced tasks leading to the generation and consumption of information/work products, in

accordance with the specified constraints, inputs and user responses, in order to achieve the desired objective.

Computerization of workflows provides the following benefits [104]: it can improve efficiency and reduce costs, improve the business process by providing better controlling of processes, analyzing the workflow that allows to provide better decision supports and tasks can be passed or assigned automatically to the responsible person.

According to [107], a workflow specification captures a process abstraction for modeling a process. A workflow model is required in order to perform the workflow specification. The workflow model provides a set of concepts to describe process, task, roles and resources for performing tasks. The workflow specification language implements the workflow model, and the workflow specification languages (or workflow management systems) provide the required constraints, graphical elements and rules to describe the ordering of tasks in a workflow.

A Workflow Management System (WfMS) automates and implement workflows by supporting workflow design and execution functions. The execution of workflows can be managed by software running on the workflow engine. The workflow engine (a software service) creates workflow instances and manages them during the execution. It interprets the definition of the workflow model, assigns each task to the responsible person and monitors the status of a process [4].

Stoilov [108] states that a WfMS improves the efficiency and business processes in the organizations. Without a workflow management system, an organization cannot monitor and control the status of the processes during the execution. Organizations can model, execute and monitor the processes by providing a WfMS.

According to [109], in order to have an efficient workflow management, the various characteristics of the workflow have to be analyzed. There are tools that provide a complete analyzing of a workflow. These tools analyze workflows in four areas, which are processes, applications, data, or information and organization. They have a framework that captures the knowledge of the processes, describes the obstacles exist in a process and monitor them.

There are two kinds of WfMS [102]:

- Activity-based: The WfMS focuses on activities that have to be completed by a workflow.
- Entity-based: The WfMS focuses on entities (e.g. documents) that have to be processed by a workflow.

#### 2.1.4 Workflow Management System Architecture

The main components of a workflow management system are illustrated in Figure 2.7 [110,111].

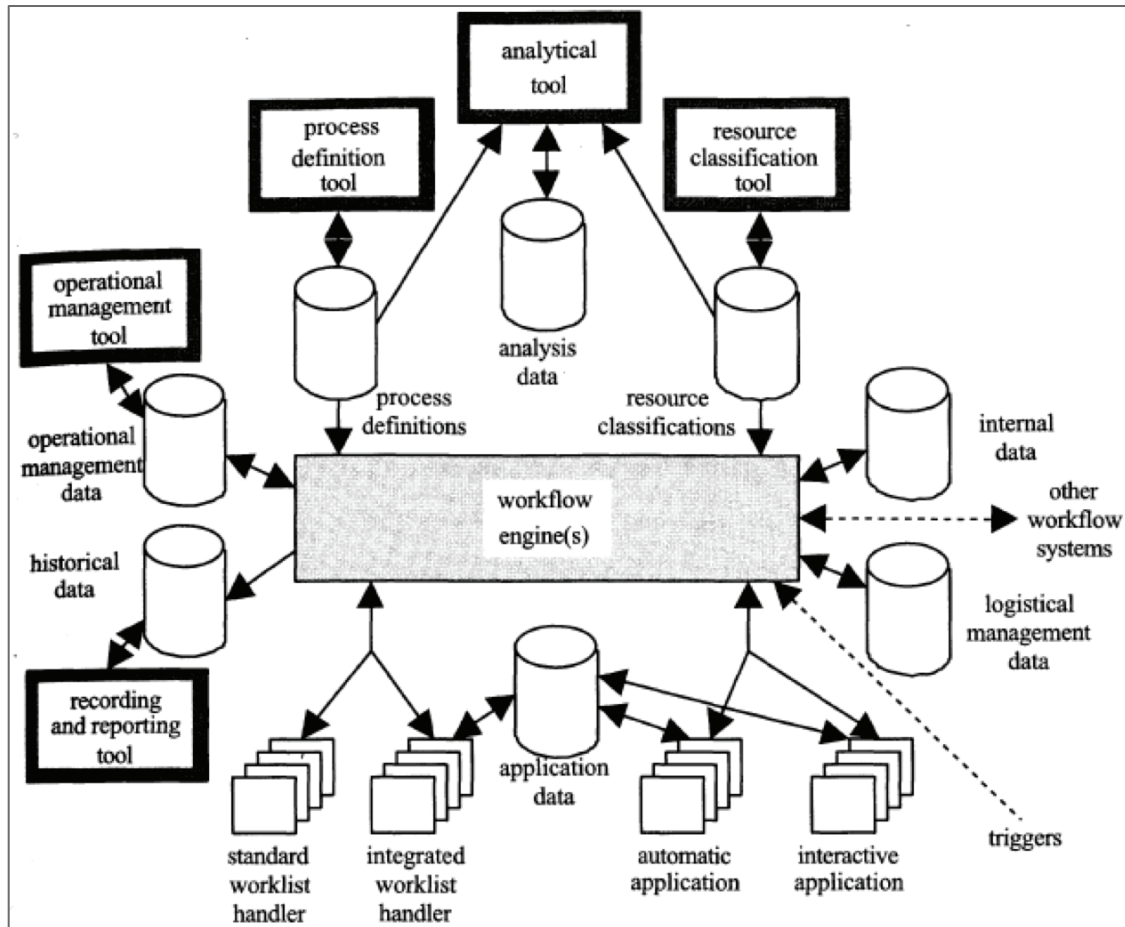


Figure 2.7 The main components of a workflow management system [111]

The main components of a workflow management system [111]:

- *Workflow Enactment Service*: A software service that consists of a workflow engine, it creates and manages the instances of a workflow during the execution.
- *Workflow Engine*: The workflow engine is the core of a workflow management system. It executes instances, and controls the execution of a set of processes. It creates, terminates and maintains the process instances. In addition, it passes the workflow data between users or/and applications.
- *Workflow Application Programming Interface & Interchange*: A set of application programming interfaces and interchanges functions that provides

interaction with other resources, and is supported by a workflow enactment service.

- *Workflow Control/Relevant/Application Data*: The control and relevant data are managed by the workflow management system or engine. The control data identifies the state of individual process and the relevant data identifies the state transition of a process instance. The application data are specific only to the applications, and cannot be accessible by the workflow management system.
- *Workflow Client Applications*: An application that interacts with users, since a human decision is required.
- *Process Definition Tools*: A number of tools may be used to analyze, model or describe a process.
- *The Recording and Reporting Tool*: The historical data can be stored during the execution of a workflow, which later can be used for the reporting purposes.
- *The Operational Management Tool*: It includes all operations belong to the management of a workflow, such as adding or removal of a user.

A lot of workflow management systems are available, for modeling, simulating and executing workflows. Such as Active Webflow (BPEL business workflow standard), jBPM (BPM business workflow standard) and YAWL (XPDL business workflow standard) [108].

### 2.1.5 Workflow Modeling Approaches

There are different frameworks [112,113] to model a workflow, and each framework has one or more formalisms, such as Petri-nets. According to [113,117], the most common frameworks are:

- *Control flow graphs*: Control flow graphs represent the execution dependencies and ordering of the activities in a workflow by modeling the control flow. They represent the initial and the final activity in a workflow. A control flow graph represents all the successor activities for an activity, and whether they have to be executed simultaneously or not.

The control flow graph is a labeled directed graph. A node in a graph represents the task that has to be performed, and an arc represents the control and the flow of data between activities. The arcs are marked with the transition conditions, which are related to the current state of a workflow.

In Figure 2.8, one of the successors of activity “c” must be executed, and then there is a choice of executing “f” or “g”. Arcs can be marked with transition conditions, and the conditions specify the current state of the workflow.

A task can begin, if all the previous tasks have been completed, and all the related transition conditions are evaluated to true.

- *Rule-Based (Triggers)*: Workflows can be stated as sets of triggers (Figure 2.8). These formalisms use logical rules to represent the dependencies between the tasks in a workflow, such as data, structural or resources.

In these approaches, the logic is divided into a set of rules, and each rule belongs to one or more activity. These rules specify the properties of an activity, such as

pre and post conditions of an execution.

There is a rule inference engine that analyzes and controls the data and conditions during the execution. The conditions specify the order of an execution. The Event-Condition-Action (E-C-A) [117] rules can be used for the task execution. It has the following syntax [117]:

ON event IF condition Do action

An event states the triggering process to evaluate a condition or a simple task execution (e.g. purchase order). The condition should be evaluated to true before triggering any other actions (e.g. adding to cart and making a payment). After evaluating the conditions, the next action can be performed (e.g. shipping the order). Some of the rule-based modeling approaches are [117]: AgentWork, ADEPT, PLM<sub>flow</sub> and AgFlow.

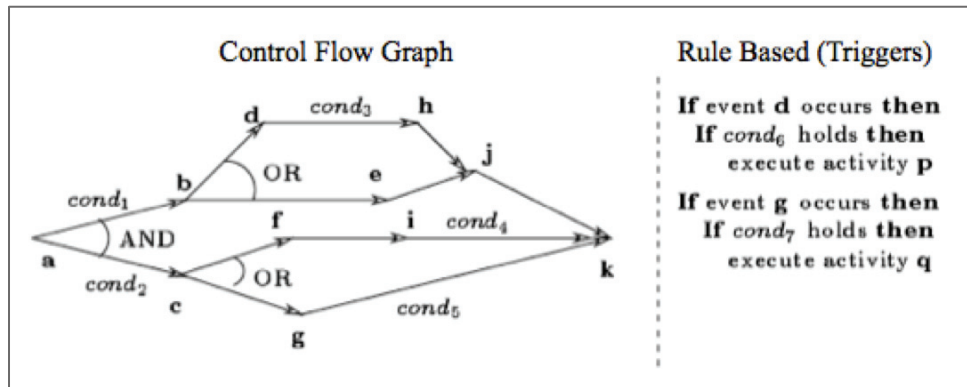


Figure 2.8 Different frameworks for modeling a workflow [113]



### 2.1.6 Workflow Formalisms

According to [114] a standard workflow model can be defined as a eight-tuple  $W (P, J_a, J_o, S_a, S_o, A, Trans, Name)$ , where:

- $P$  is a set of process elements that are divided into disjoint sets of AND-Joins ( $J_a$ ), OR-Joins ( $J_o$ ), AND-Splits ( $S_a$ ), XOR-Splits ( $S_o$ ), and activities ( $A$ ).
- $Trans \subseteq P \times P$  is a transition relation between the elements of a process.
- $Name \in N^A$  is a function to assign names to activities

There are several formalisms for describing workflows. We list some of these formalisms [115] here, however the details of these formalisms are out of the scope of this thesis:

**Petri-net:** A Petri-net [115] is a directed graph, and it consists of places (circles), transitions (bars) and arcs (edges from transitions to places or vice versa).

The activities in a workflow are represented as the transitions in a Petri-net, and the input of an activity is represented as an input place and the output of an activity is represented as an output place for the transition in a petri-net. Figure 2.9 illustrates the transformation of the UML Activity Diagrams constructs to Petri-nets [115].

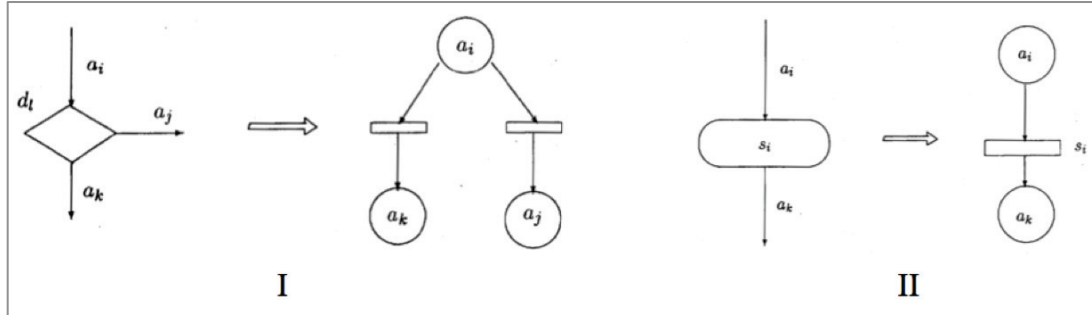


Figure 2.9 Transformation rule for a decision and activity state to Petri-net [115]

A PTN can be defined as a four-tuple  $C = (P, T, I, O)$  where [115]:

- $P$  is a set of places, which is finite:  $P = \{p_1, p_2, \dots, p_n\}$
- $T$  is a finite set of transitions:  $T = \{t_1, t_2, \dots, t_m\}$ . The set of  $P$  and  $T$  are disjoint,  $P \cap T = \emptyset$ .
- $I: T \rightarrow P^\infty$  is the input mapping function for from transitions to places.
- $O: T \rightarrow P^\infty$  is the output mapping function from transitions to places.

The dynamic behavior [115] of Petri-nets can be represented by assigning Tokens to the places of a Petri-net. A token in a place indicates the condition of that place. The position and number of tokens can change during the execution of a Petri-net.

A Petri-net can be executed by firing transitions, and it controls the quantity and spreading of tokens. A transition fires by eliminating tokens from the input places and assigning new tokens in the output places [115].

**Activity diagram:** An activity diagram is a state chart diagram of UML. All the states in the activity diagram are action states (activity node) and the transitions can be triggered by completion of the actions in the source states (the node that the edge leaves). An action state can model the execution of a procedure, it can be viewed as an atomic task, and it has an internal action and at least one outgoing transition [115].

For multiple transitions, a condition must be defined. An activity diagram can be defined as the tuple  $(S, S_0, A, C, D, In_s, Out_s, In_d, Out_d, In_c, Out_c)$  [115], where:

- $S = \{s_1, s_2, \dots, s_k\}$  is a set of activity states,  $S_0$  is an initial pseudo state.
- $A = \{a_1, a_2, \dots, a_k\}$  is a set of internal transitions.
- $C = \{c_1, c_2, \dots, c_n\}$  is a set of forks and joins.
- $D = \{d_1, d_2, \dots, d_p\}$  is a set of decisions.
- $In_s: S \cup \{s_0\} \cup Z \rightarrow A$  and  $Out_s: S \cup \{s_0\} \cup Z \rightarrow A$  are mappings, which define input and output transitions for states.
- $In_d: D \rightarrow A$  and  $Out_d: D \rightarrow A$  are mappings, which define input and output transitions for decisions.
- $In_c: C \rightarrow A$  and  $Out_c: C \rightarrow A$  are mappings, which define input and output transitions for forks and joins.

**EPC** [116]: It has three types of nodes, which are events (E), functions (F) and connectors (C). It can be formalized as a five-tuple  $(E, F, C, T, A)$  [116], where:

- E is a set of events. Events describe the situation before or after the execution of a function. Events represent pre or post condition of a function in a workflow.
- F is a set of functions. A function is related to an activity, such as a task in a workflow that has to be executed.

- $C$  is a set of logical connectors ( $\wedge, XOR, \vee$ ). The connectors connect activities and events to specify the flow of control. The " $\wedge$ " operator represents branching or synchronization in a workflow. The " $\{XOR, \vee\}$ " represent decision gateways in a workflow; based on the result of an event, one of the paths has to be followed.
- $T \in C \rightarrow \{\wedge, XOR, \vee\}$  is a function that specifies a connector type for a connector.
- $A \subseteq (E \times F) \cup (F \times E) \cup (E \times C) \cup (C \times E) \cup (F \times C) \cup (C \times F) \cup (C \times C)$  is a set of arcs. They connect functions, events and connectors. An arc acts as a sequence flow in a workflow.

## 2.2 BUSINESS PROCESS MODELING

According to Mending J. [19] business process modeling involves modeling of the business process in an organization. M. Weske [20] states that, a business process model includes a set of activity modeling and constraint execution.

According to [19], business process modeling has an important role in the lifecycle of a system development. It provides a process definition to model activities with the process modeling languages, and a workflow management system that monitors and controls the execution of the processes in a workflow.

Assaf A. [26] states that, "*Business Process Modeling Language (BPML) defines a formal model for expressing abstract and executable processes that address all aspects of enterprise business processes, including activities of varying complexity, transactions, data management, exception handling and operational semantics. BPML also provides a grammar in the form of an XML Schema for enabling the persistence and interchange of*

*definitions across heterogeneous systems and modeling tools.”*

According to [27] the modeling language consists of three elements:

- **Notation** specifies the visual elements, which can be used for the visualization of a model.
- **Syntax** defines a set of constructs with the rules, to describe how the constructs can be combined.
- **Semantics** provides meaning for the constructs defined in the syntax and it can be defined using ontologies.

In the next section, we review a number of modeling languages and notations for representing business processes and workflows, and then we gather state-of-the-art work on BPMN. We also provide a summary of workflow patterns analyses to explore the expressive power of BPMN and other modeling languages.

### 2.2.1 Event-Driver Process Chain (EPC)

Event-driven process chain (EPC) [19,31,32] is a semi-formal graphical modeling language for modeling business processes and workflows. Scheer W. developed EPC within the framework of ARIS [30] in the early 1990s. It was used in the ERP systems, which describe workflow (e.g. SAP R/3). EPC models processes as chains of events and triggers a function, which results in events again. The concept of EPC is very close to Petri nets [31].

A definition by Wang J. [31] states that, EPC represents events and functions in an ordered graph. It provides multiple connectors to execute multiple processes in parallel. It provided logical operators, such as OR, AND and XOR. EPC is very simple and easy to

understand technique to model business process.

EPC consists of the following elements [118]:

- *Events*: The passive elements, which describe the circumstances in which a function or process works. It is represented as hexagon.
- *Function*: The active elements that model the activities, and describe the transformations from the first state to the end state. Functions are represented as rounded rectangle.
- *Organization units*: It indicates the responsible person or unit for an assigned task.
- *Information or resource object*: It describes the objects in the real world and can be provided as the input data or output data for a function.
- *Process path*: It shows the sequence in EPC. They indicate the connection from or to other process.
- *Control flow*: Functions, process or logical connectors can be connected with other by control flow. It is represented as a dashed arrow.
- *Logical connector*: The logical connectors describe the relationships between the elements in a control flow. There are three different kinds of logical relationships in EPC: Branch/Merge, Fork/Join, OR.
- *Information flow*: The connection between input or output data and functions are represented by information flow.
- *Organization unit assignment*: It shows the connection between an organization unit and the assigned function.

A sample of EPC diagram is shown in Figure 2.10 [116].

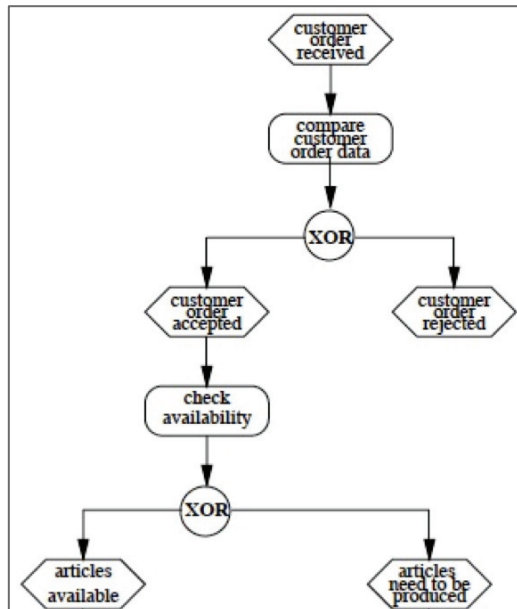


Figure 2.10 EPC diagram [116]

### 2.2.2 Petri-Nets

Petri net [29,89,90] is a formal, graphical and executable language to specify dynamic behavior. It was defined by C.A. Petri in the 1960s as a tool for modeling distributed systems. Petri nets have precise mathematical properties, which can be used workflow management [29].

Petri net [20] is a directed graph. It has two nodes, which are places (circles) and transitions (rectangles). There are directed arcs that connect the nodes. Places contain tokens, which represent the dynamic behavior (and being able to execute them).

According to Lohmann N. [90], a Petri net is a workflow net that has a source and a sink place, and there is an arc or a path from a source to a sink. A token in the source place represents a new event, and a token in the sink place represents a finished event.

Petri nets have a small number of modeling constructs; therefore they are limited to express the resources, structural, functional or operational perspectives [89].

The authors in [90] used the workflow patterns to analyze the expressive power of Petri nets for workflow and process modeling. We provide a summary of their results in Table 2.1 [90].

<b>Patterns that are easy to represent in Petri nets</b>
<i>Sequence</i> : An activity is started after the completion of another activity
<i>Parallel Split</i> : Activities can be executed simultaneously or in parallel
<i>Synchronization</i> : Waiting for all the incoming branches to be completed before going to the next step
<i>Exclusive Choice</i> : One of the branches has to be chosen
<i>Simple Merge</i> : Two or more branches are merged without any synchronization
<i>Deferred Choice</i> : One of the branches is chosen, but the decision is not based on data
<b>Patterns that are harder to represent in Petri nets</b>
<i>Multi-Choice</i> : A number of branches can be chosen
<i>Cancel Region</i> : A set of tasks that can not be executed
<b>Pattern that cannot be represented in Petri nets</b>
<i>General Synchronizing Merge</i> : Wait-and-see synchronizing construct

Table 2.1 Expressiveness of Workflow patterns in Petri nets [90]



### 2.2.3 Unified Modeling Language (UML)

According to [118] there are two types of diagrams for UML, which are Structural and Behavioral. Structural diagram is divided into three types of diagrams, which are [118]:

- *Class diagram*: It represents a set of classes, and the relationships between these classes. It shows the structure of the classes.
- *Component diagram*: it groups a set of objects into components, such as source codes or application documents.
- *Deployment diagram*: It shows the components that depend on the run time processes.

Behavioral Diagram is divided into four types of diagrams [118]:

- *Use case diagram*: It represents a set of classes, users (actors) and the relationships between these classes and users. It represents the functionality of the classes.
- *Interaction diagram*: It shows the communications between the objects.
- *State diagram*: It shows all the available states for objects, and how the state of an object can be changed. It includes two elements, which are state and transition.
- *Activity diagram*: It is the best way to model a workflow, among the listed diagrams. It includes a number of workflow constructs that can capture the workflow patterns.

UML Activity Diagram [25,33,34] is a semi-formal language from the Object Management Group (OMG). It is a case of UML state diagrams for modeling workflows. An UML activity diagram represents the step-by-step of a workflow, and the ordering

among the tasks, activities and states [25].

The basic elements of UML activity diagram are shown in Figure 2.11 [118].

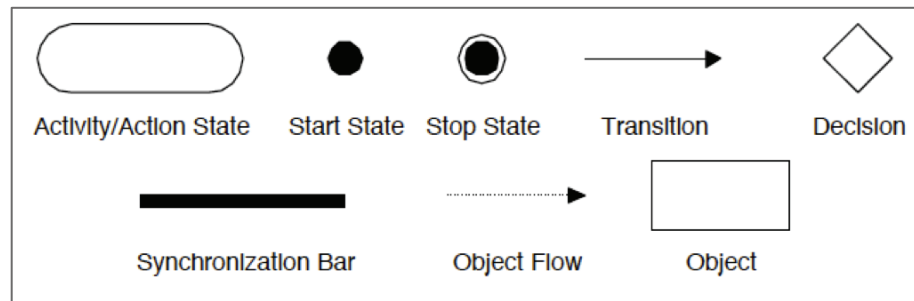


Figure 2.11 The basic elements of UML activity diagram [118]

#### 2.2.4 Business Process Execution Language (BPEL)

Business Process Execution Language (BPEL) [35,36,37,38] is a standard executable language, which specifies actions in the business processes by help of web services. BPEL uses Web Service Description Language [39] (WSDL), which is based on XML language, to describe the functionality of web services and how to access them.

BPEL, which is an orchestration language can specify an executable process to exchange messages with other systems, and an orchestration designer controls the messaging exchange [21].

BPEL process has the following concepts [36]:

- **Variables:** the data that are exchanged with web services can be stored in variables.
- **Handlers:** in the case of the occurrence of a fault, handlers can be used to handle the faults.
- **Basic and Structured Activities:** operations that have to be performed in a

process are specified by basic activities and structured activities are utilized for the definition of control flow.

- **PartnerLinkTypes:** the port types for a message exchange are defined with PartnerLinkTypes by indicating which partner acts according to which defined role in a partner link.

We also need to mention that some literature has proposed the use of BPMN to model a BPEL process, but this mapping can be very difficult since there are fundamental differences between these two languages [35,36]. It is difficult to create BPEL code from a BPMN diagram [36]. In addition there are a number of tools for partial mapping from BPMN to BPEL (e.g. BPMN2BPEL which is an open source tool), but [40] this is neither supported with semantics nor fully automated.

### **2.3 BUSINESS PROCESS MODELING NOTATION (BPMN)**

In this section, we gather state-of-the-art work on BPMN The Business Process Modeling Notation (BPMN) [27,34,35,41,42,43] is a semi formal modeling language, to model business process and web service processes. Business Process Management Initiative, which is now merged with the OMG organization, published the first version of BPMN in May 2004 [27]. The latest version of BPMN specification (BPMN v2.0), which has been recently released (March 2011), includes extensions to the notation and meta-model specification, however we used BPMN v1.1 for the purpose of this thesis.

According to [34], the main goal of BPMN is to provide a graphical notation that can be easily understood and use by all users, such as business analyst, technical people and business people [41].

The structural elements of BPMN facilitate the readability and provide three types of model [41]:

- **Private (internal) business processes:** It states business processes that are internal or private to an organization and are not accessible from outside. (Figure 2.12).
- **Abstract (public) processes:** It states the interactions between a private or, internal business process with another business process. (Figure 2.13).
- **Collaboration (global) processes:** It states the sequence of activities between two or more business entities. These activities represent the exchange of messages between those entities. (Figure 2.14).

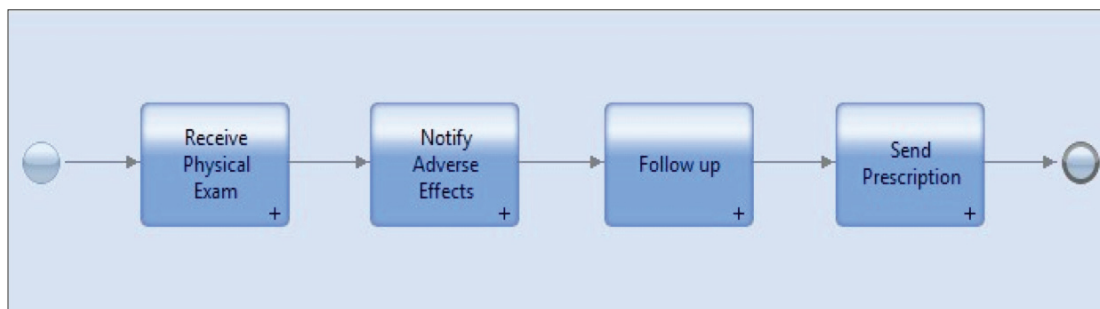


Figure 2.12 An example of a private business process

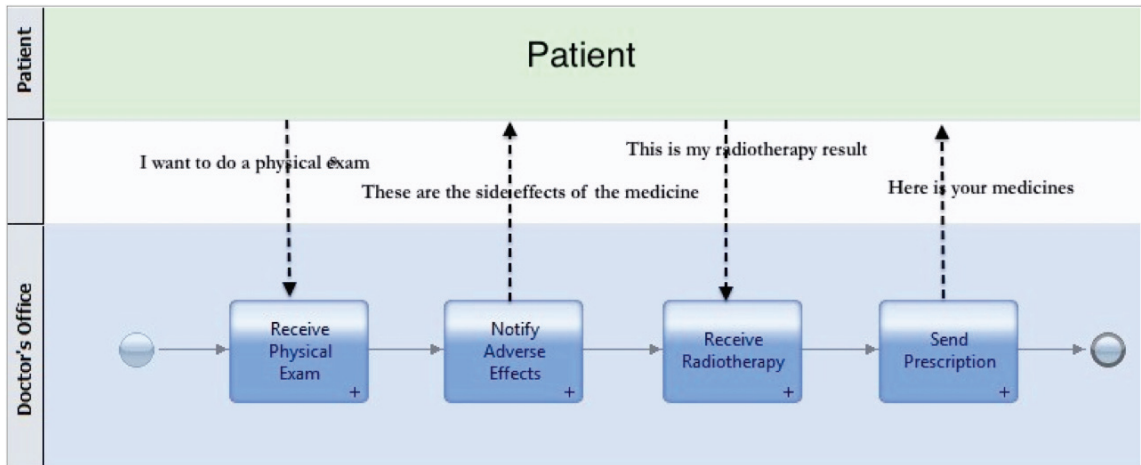


Figure 2.13 An example of an abstract business process

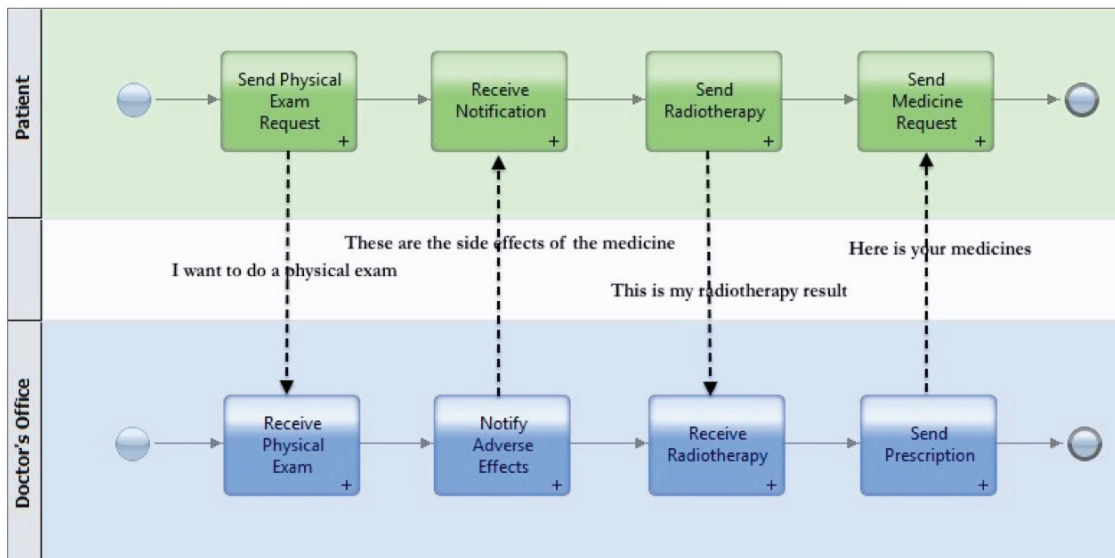


Figure 2.14 An example of a collaboration business process

BPMN provides a single diagram, called the *Business Process Diagram* (BPD). Business processes can be modeled and managed by this diagram. In addition it is easily understandable by all users [41].

BPMN elements can be categorized in four different classes [41]:

**The first group is called Flow Objects:** Flow objects are the primary graphical elements that can describe the behavior of a business process (Figure 2.15).

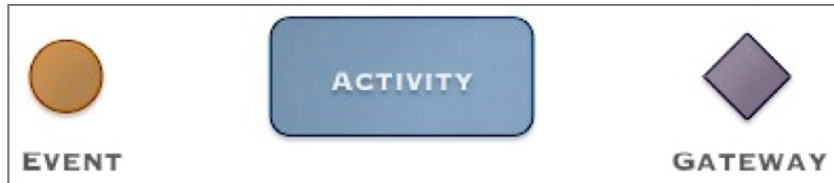


Figure 2.15 Flow objects [24]

Flow objects are defined in the following three groups:

- **Events** happen during the business process and are represented as circles. The complete list of BPMN events is shown in Figure 2.16. Events can have a trigger and results. There are three types of events that can affect the flow:
  - **Start Event:** starts a process flow.
  - **Intermediate Event:** happens during a process.
  - **End Event:** ends a process flow.

		Message	Timer	Exception	Cancel	Compensation	Rule	Link	Multiple	Terminate
Start										
Intermediate										
End										

Figure 2.16 The complete list of BPMN event types [24]

- **Activity** is the real work that the organization performs. An activity can be atomic or a compound activity that contains other activities. There are three types of activities, which are process, sub-process and task (Figure 2.17).

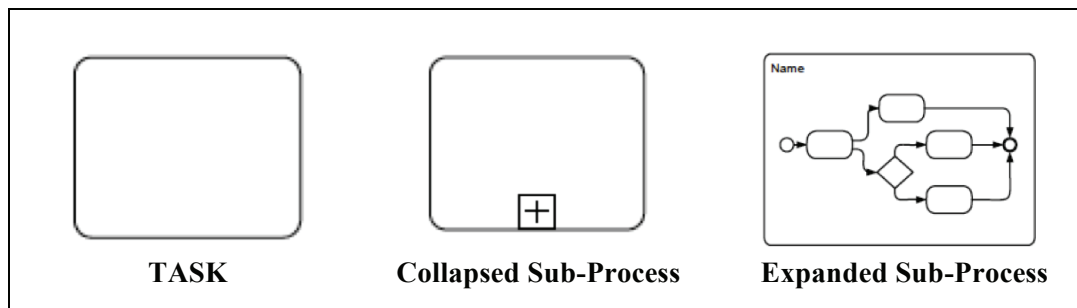


Figure 2.17 Activities [24]

- **Task** is an atomic activity and there are different types of tasks, such as “user task”, which is performed by a human, “send task”, which sends a message by executing the task and “service task” that provides a web service.
- **Sub-process** is a compound activity that contains another process. A sub-process can be expanded/collapsed, in order to show/hide the sub-process details.
- **Process** doesn't have a graphical representation. It is an activity that has to be performed within organizations.
- **Gateways** represent decisions in a process. They provide branching, forking, merging and joining of paths. The complete list of BPMN gateways is shown in Figure 2.18.

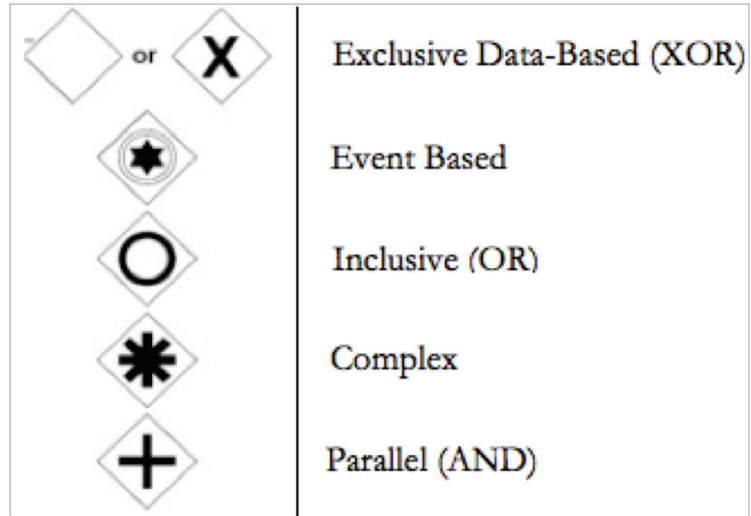


Figure 2.18 The complete list of BPMN gateway types [24]

The **second group** is called Connecting objects; they connect flow objects to each other in three ways (Figure 2.19):

- **Sequence Flow** represents the execution order of activities in a process and can be represented as arrows between flow objects.
- **Message Flow** shows the flow of message between different participants and are represented as dashed arrows.
- **Association** associates extra information to the flow objects, and are represented as dashed lines.



Figure 2.19 Connecting objects [31]



**The third group** is called Swimlanes and they group the modeling elements in two ways (Figure 2.20):

- **Pools** represent involved participants and users in a process and also can be used to separate a set of activities in a pool from the activities in other pools.
- **Lanes** divide a pool into sub partitions and can be used to represent roles or departments. Lanes organize and categorize workflow elements.

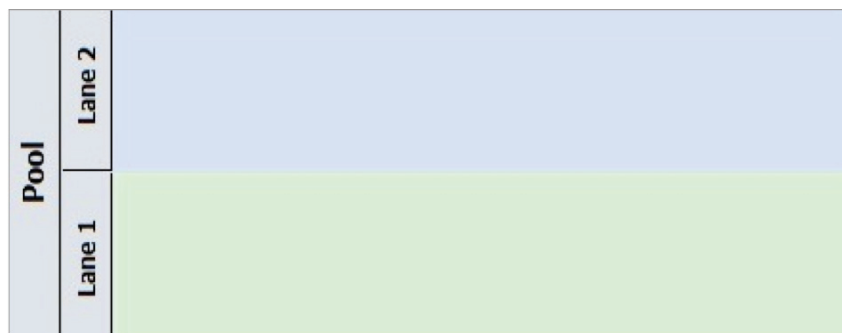


Figure 2.20 A pool with two lanes [31]

**The fourth group** is called Artifacts. Artifacts provide further information about a process. They don't affect the flow of interaction. There are three types of artifacts, which are provided in Figure 2.21.

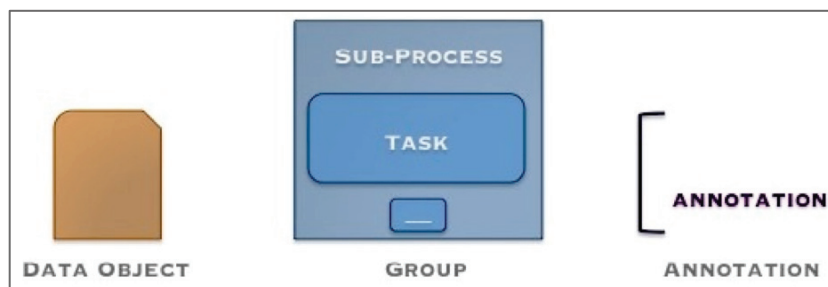


Figure 2.21 Artifacts [31]

- **Data Objects** provide information about activities or data exchanged between activities, and they don't have effect on the flow of process. Data objects are represented as a rectangle with a folded corner (Figure 2.22 and Figure 2.23).
- **Group** can be used for grouping flow objects with the same category. A group is represented as a dashed box that contains a group of flow objects.
- **Text Annotation** provides additional information about the model. It can be connected to an object by an association flow.

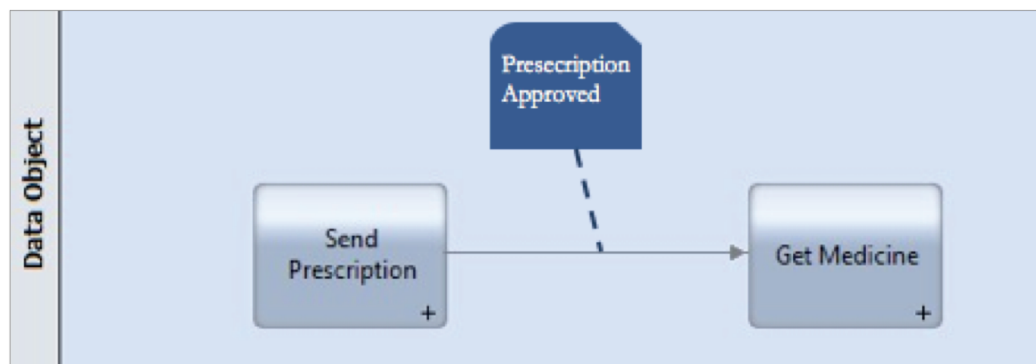


Figure 2.22 Attaching data object to the sequence flow: Prescription is being approved when it is sent from the Send Prescription task to the Get Medicine task

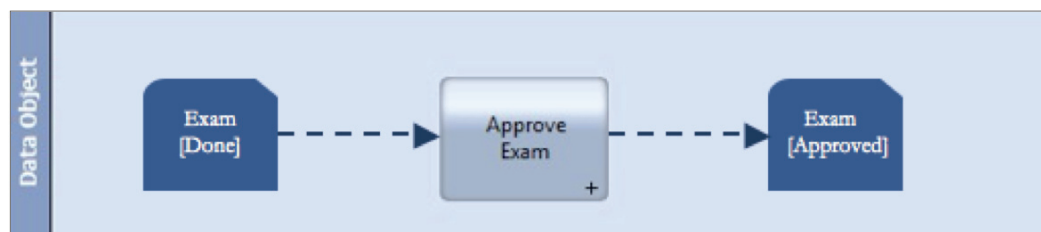


Figure 2.23 Association lines between data objects and a task: the state of Exam data object is changed from “Done” to “Approved” after the Approve Exam task

## 2.4 COMPARING BPMN WITH OTHER LANGUAGES

In this section, we provide a summary of workflow patterns analysis to analyze the expressive power of BPMN and other modeling languages. By providing this comparison, we conclude that BPMN is very easy to use and understand, since it is a graphical language, and it is more expressive than other languages concerning control flow structures. BPMN has more control flow elements than other languages; therefore it supports more workflow patterns [32,33,35].

The ARIS community has listed the comparison of EPC with BPMN below [32]:

- BPMN supports more workflow patterns than EPC, therefore it is more expressive than EPC. BPMN supports 24 of 43 patterns, while EPC supports 10.
- BPMN is more efficient than EPC. EPC requires events after OR and XOR gateways, while in BPMN the conditions are carried in the properties of sequence flows after the split gateway.
- Exceptions are well handled in BPMN, while EPC cannot handle exceptions that happen during the execution.
- BPMN is easier to read and understand.
- BPMN supports transactions and compensations patterns, while EPC has difficulty to support these patterns.

According to [33], UML is harder to understand than BPMN, since there are some model elements in BPMN that are not available in UML. In addition the main goal of BPMN is to be understandable by all users, which is not the same for UML.

The authors in [33] have conducted a workflow pattern framework analysis

between BPMN and UML. Their results showed that BPMN has a better representation power in control flow patterns and BPMN supports more data patterns than UML.

Another comparison of BPMN and UML in [34] indicates that:

- BPMN has a mathematical foundation that can be used for mapping to business process execution language (BPEL), whereas UML doesn't have a mathematical foundation and it does not define any execution meta-model.
- UML is a combination of diagrams that are not intended to communicate with each other; therefore UML can model part of applications with no details of implementation. However, BPMN defines a single diagram that provides multiple views for users.

The authors in [35] have done a comparison between BPMN and BPEL. Their result showed that BPEL has difficulty in modeling complex control flow patterns, such as the following patterns:

- Advanced branching and synchronization patterns are not well supported in BPEL, but BPMN supports these patterns by providing parallel gateways, which control the branching and merging flows.
- BPEL does not support the arbitrary cycle pattern.

L. Yun [89] has categorized the modeling constructs of the process modeling languages according to the six process perspectives. We provide the table of this categorization here:

	<b>Petri Net</b>	<b>EPC</b>	<b>UML</b>	<b>BPMN</b>
<b>Structural</b>	-	Process path	UML use case	Collapsed /Expanded sub-processes
<b>Operational /Functional</b>	-	Functions	Activity	Task, Process
<b>Control</b>	Transition node, Arc	Connector, Flow	Flow, Fork, Join, Decision, Merge	Sequence flow, Decision, Merge, Loop
<b>Resource</b>	-	Extension with information, Resource object	-	Data object
<b>Organizational</b>	-	Extension with role, Person	Partition, Swimlanes	Pool, Lane
<b>Data Transaction</b>	Token	Event	UML state diagram	Message flow with data object

Table 2.2 Modeling constructs of different business process modeling languages [89]

In this chapter, we provided an introduction to workflow, workflow management system, workflow modeling approaches and formalisms. We also reviewed a number of modeling languages and their limitations compared to BPMN. Because of those limitations we chose BPMN as our workflow modeling language, in addition based on [32,34,35,89,90] we highlight the main advantages of BPMN as follows:

- BPMN is more expressive than other modeling languages, since it supports most of the workflow patterns.
- BPMN is a graphical language, thus it is easy to understand and learn.
- Different user can have different views of the BPMN diagram.
- A set of attributes can be defined to provide a richer specification.
- The graphical elements can be extended for the domain purpose.

Based on these reviews, we chose BPMN as our modeling language to design standardized CP that can be executed through workflow execution engines. BPMN provide constructs to capture the complexity, and control-flow amongst multiple clinical tasks. It models the ordering among different tasks and activities. The use of BPMN formalisms to represent CP [1] clearly describes the operational aspects of clinical processes, such as (a) roles and responsibilities of care providers; (b) decision points and care options; (c) well-identified clinical/business rules; (d) operational constraints; (e) task scheduling; and (f) temporal constraints.

## **CHAPTER 3      COMPUTERIZING CP AND CPG**

The paper-based Clinical Practice Guidelines (CPG) and Clinical Pathways (CP) cannot be utilized at the point of health care, since it is difficult to integrate them in the active clinical practices. Paper based clinical pathways lack dynamicity, and are static. In addition, the maintenance of the healthcare business process lacks from continuous updating, since the medical guidelines change frequently, and there is no real time information for the clinical pathways [96].

CPG and CP can be computerized to reduce variations in quality of health care, to provide recommendations and to reduce costs [2]. Computerization simplifies decision support and execution; and multiple care processes can be executed simultaneously [1].

A number of approaches such as PROforma, Asbru, Gaston and SAGE have been proposed to computerize CPG and CP. Most of these formalisms represent medical knowledge as the “Task-Network Models” (TNM) in different approaches. TNM languages decompose recommendations into the network of tasks, and describe the relationships between these tasks. These computer formalisms of the clinical guidelines provide decision support at the point of care [10].

In this chapter we review some of these approaches, however the details of these approaches are out of the scope of this thesis. At the end of this chapter, we provide a comparison against the eight dimensions from Peleg [10] between some of these approaches and our CP ontology. By reviewing these approaches and providing a comparison from Peleg [10], we find out that these guideline models have some common constructs to represent guideline steps, and the combination of these constructs can

represent a workflow. However, they have a limited number of constructs and they cannot model workflow patterns properly. Therefore we use a standard workflow modeling language to model a CP.

### 3.1 PROFORMA

PROforma [8,9,10], a knowledge representation language, was implemented at the Advanced Computation Laboratory of Cancer Research, UK.

*PROforma* captures the knowledge and the structure of a guideline, which can be understood by a computer. PROforma is a combination of the two words *proxy* (“authorized to act for another”) and *formalize* (“give definite form to”) [8].

Guidelines are represented as a set of tasks. The tasks are modeled hierarchically into plans and are divided into four classes [9]:

- **Action** is a clinical activity or a task that needs to be executed (e.g. backup database).
- **Enquiry** is an action to request more information or data from the user (e.g. a nurse)
- **Decision** is a task, in which a decision has to be made, such as choice of diagnosis.
- **Plan** is a set of tasks that are combined together to accomplish a clinical objective. These tasks can be grouped, since they have common goals or they need to be executed at the same time during the execution.

There are two main implementations of the PROforma engine available [9], Arezzo, which is a commercial tool developed by *InferMed* Ltd. (London UK), and Tallis



implementation by Cancer Research UK.

Arezzo [9] includes a composer, which is a graphical authoring tool and a performer, which is an execution engine and application tester.

Tallis [9] consists of a composer to support the creation and modeling of guidelines, and a tester, which allows users to debug a guideline.

### **3.2 ASBRU**

Asbru, a skeletal plan-specification representation language, was developed at Ben Gurion University and the Vienna University of Technology. Asbru, which is a time oriented language represents clinical guidelines in XML [10].

The main features of Asbru are [18]:

- Temporal dimension of states and plans.
- Each plan can have its own intentions.
- Actions and states can be continuous.
- Plans can be executed in parallel, sequence or periodically.
- Verification and validation of the plan itself.
- Reuse of existing knowledge and acquired plans.

Asbru represents a protocol in a hierarchy of plans. Each plan has a name, a time label, which can be used to specify the duration of the plan's execution and the following main components [16]:

- **Preferences** are used to constrain the choice of a plan to accomplish a given goal or to describe the behavior of a plan.
- **Intentions** are the main goals at different stages of a plan, and are represented as

actions, or states that can be held during or after finishing a plan.

- **Conditions** are temporal patterns and they define the various phases for the execution of a plan. There are different conditions such as preconditions, activate, reactivate, complete and abort conditions.
- **Effects** describe the effects of a plan's execution on parameters.
- **Plan-Body** contains sub-plans and actions, which will be executed in a particular way (parallel, sequence or any order).

In the medical domain there is a temporal uncertainty for time aspects since we cannot always predict when something will happen or when it ends. Asbru includes time label that can be assigned to various components and the uncertainty can be represented in the starting time, ending time and duration [18].

AsbruView [10] is a visualization tool to model guidelines in the Asbru language. It can design the temporal views of plans that are written in Asbru.

However, S. Miksch [17] has listed the drawbacks of Asbru in her paper, which are:

- Acquisition of conditions (the temporal patterns) and time annotations are difficult, and temporal dimensions are often unknown.
- It is difficult to handle all the possible orders of the plan execution and the exceptions that might arise.

### **3.3 GASTON**

Gaston is developed in the Eindhoven University of Technology. The goal of this framework is to improve the use of the computerized guidelines and decision support systems [11].

The framework consists of [12]:

- A set of concepts, which are primitives, Problem Solving Method (PSM) and ontologies are used to represent guideline formalism.
- An authoring environment that allows authors to model guidelines.
- An execution environment to process and interpret guidelines by an execution engine.

Gaston architecture involves several steps [12]:

First a domain ontology must be defined. It contains a set of concepts for a specific domain and knowledge in terms of entities, properties and relations.

Then a method ontology must be defined, which models concepts as primitives and PSMs. Primitives describe a single guideline step and the internal structure of a PSM.

These ontologies can be defined in Protégé framework, which is a tool to develop knowledge-based systems [11].

Finally, a set of components that describe specifications for communication between the components of the execution time and other systems must be defined.

### **3.4 SAGE**

The SAGE (Standards-Based Active Guideline Environment) project [13] represents the integration of decision support systems for guidelines in the clinical information systems.

The SAGE project uses the standard terminologies and information models to encode the guideline content as the recommendation sets [13].

A combination of a clinical setting, the care provider and the relevant patient states, can define a context. A recommendation set relates decisions to actions in order to provide recommendations. Recommendation sets are modeled either as activity graphs or decision maps [13].

An activity graph represents guideline-directed processes. It can describe the relationships among different activities. Decision map represents recommendations by providing decisions at one point in time [13].

The SAGE project uses different levels of standard terminologies, which are required for encoding and executing guidelines. These standard terminologies use the vocabulary resources [13] of SNOMED CT, LOINC, and National Drug File-Reference Terminology (NDF-RT).

The SAGE project uses Protégé open-source knowledge based modeling environment to model guidelines, and GELLO a standardized language is used as the expression language of SAGE [13].

### **3.5 SEMANTIC-BASED CP WORKFLOW AND VARIANCE MANAGEMENT SYSTEM**

Semantic-based clinical pathway workflow and variance management system is a framework for modeling pathway from Y. Ye [14]. The proposed framework contains three components [14]:

- Clinical Pathway Ontology (CPO) and domain ontology
- Clinical pathway workflow management
- Variance Management

According to [14], the clinical pathway ontology and domain ontology can provide a semantic interoperability between the clinical pathway workflow and variance management. Clinical pathway workflow management executes and monitors the clinical pathways and variance management provides support for analyzing and handling variances during the reasoning.

Importing two existing ontologies, which are process ontology in OWL-S and time ontology develop a CPO. Process ontology defines terminologies to describe processes and their structures, and the time ontology provides temporal concepts and relations [14].

Semantic modeling in the framework is implemented by two modeling approaches [14]:

- A hierarchical modeling approach, which has two levels: the outcome flow level and intervention workflow level.
- A modeling approach, which is based on Semantic Web Rule Language (SWRL).

Applying a clinical pathway to the treatment of the individual patients may create some variances. In the framework, these variances are handled and analyzed by the

event-condition-action rules [14].

### **3.6 COMPARISON OF CPG FORMALISMS**

M. Peleg et al. [10] identified eight dimensions to compare six computer-interpretable Guideline Models (Asbru, EON, GLIF, GUIDE, PRODIGY, and PROforma). These eight dimensions are as follows [10]:

1. Organization of guideline components
2. The goals and intensions
3. Guideline actions modeling
4. Decisions
5. Expression languages for decision criteria
6. Interpretation of data
7. Medical concept model representation
8. Information model for patient

A number of tables in this study provide a comparison among these models based on each dimension. The details of these comparisons are out of the scope of this thesis. However, the study concluded that each of these models has strength in different dimensions and none of them performs well in all eight dimensions. A summarized comparison of these models by our CP ontology [92] is provided in Table 3.1 [10].

COMPARING COMPUTER-INTERPRETABLE GUIDELINE MODELS								
	D.1	D.2	D.3	D.4	D.5	D.6	D.7	D.8
<b>Asbru</b>	TNM	Expression	Medical actions	Switch constructs	XML	Temporal Abstraction	Variable name	Mapping guideline table
<b>EON</b>	TNM	Expression	Specialized medical actions	Switch constructs	RDF	Temporal Abstraction	Classification hierarchies	VMR
<b>GLIF</b>	TNM	Text String	Medical actions	Switch constructs	GELLO	General Abstraction	Classification hierarchies	HL7 RIM
<b>GUIDE</b>	TNM	Expression	Medical actions	Switch constructs	Formal Language	General Abstraction	Classification hierarchies	Mapping guideline table
<b>PRODIGY</b>	TNM	Text String	Specialized medical actions	Argumentation rules	Formal Language	General Abstraction	Classification hierarchies	VMR
<b>PROforma</b>	TNM	Expression	Medical actions	No commitment to a decision alternative	R2L	General Abstraction	Variable name	Mapping guideline table
<b>CP Ontology</b>	TNM	Free Text Expression	Specialized medical actions	Switch constructs	Logical Text	General Abstraction	Concept-URI	Concept-URI

Table 3.1 Comparing computer guideline models [10]

In this chapter we studied a number of approaches that have been proposed to computerize CPG and CP, and then we provided a comparison along the eight dimensions as mentioned above. By reviewing these approaches, we observed that most of these formalisms have some common constructs to represent guideline steps, such as actions, decisions and scheduling constraints. Table 3.2 [10] provides a summary of these constructs or modeling primitives.

	<b>MODELING PRIMITIVES</b>		
	<b>Branching/Scheduling</b>	<b>Action</b>	<b>Decision</b>
<b>Asbru</b>	Ordering, Completion or Continuation Condition	Plan	Precondition
<b>EON</b>	Branch Synchronization	Action	Decision
<b>GLIF</b>	Branch Synchronization	Action	Decision
<b>GUIDE</b>	Synchronization	Task	Deterministic Decision
<b>PRODIGY</b>	Branch	Action	Rules
<b>PROforma</b>	Branch Synchronization	Action	Decision

Table 3.2 Modeling constructs or primitives [10]

- Guideline actions represent clinical intervention, or actual tasks described by a clinical guideline.
- GLIF, EON and PROforma model decision as decision step (using switches), and Asbru does not use explicit construct to represent decisions, but it has exclusive pre-condition or argumentation rules.



- Scheduling constraints represent the temporal relationship between actions. All except Prodigy support cyclical and iterative graphs. Asbru uses ordering constraint to specify the order of sequence and completion/continuation condition.
- Parallel pathways can be modeled by providing a branch step and a synchronization step [10].

All of these formalisms contain a number of constructs to specify guideline steps such as actions, decisions and scheduling, and the combination of these constructs can represent a workflow. However, they have limited constructs and they cannot model workflow patterns. Therefore in order to design operationally and clinically pragmatic CP to ensure data interoperability, resource management and task prioritization, it is important to view CP as ‘specialized’ process workflows. However, the use of workflow modeling concepts in the design and optimization of CP is not yet well established, and as such there are no standard formalisms for the representation of CP in general, and CP as workflow models in particular.

There is a case for exploring the potential of business process modeling principles and workflow modeling formalisms—such as Business Process Modeling Notation (BPMN), Business Process Execution Language (BPEL), UML, etc.—to design standardized CP that can be executed through workflow execution engines.

Our main goal is to provide a CP design framework that uses a standard modeling notation to capture the control-flow amongst multiple clinical tasks with the workflow constructs that represent the workflow patterns [24].

## **CHAPTER 4      ONTOLOGIES IN USE: CP AND BPMN**

### **ONTOLOGIES**

The Semantic Web (SW) framework provides a representation formalism and semantically knowledge modeling in terms of ontologies, reasoning mechanisms and the reusability of knowledge models. Semantic web technologies offer OWL ontologies to both model and execute CP [1].

This chapter covers the research background in the realm of ontologies, specifically pertinent to our research. We present a description of the existing ontologies—(a) CP Ontology, and (b) BPMN ontology—that we have used in our research. In addition, we provide an introduction to ontology mapping with an overview of existing approaches and tools for ontology mapping.

#### **4.1    ONTOLOGY**

According to [44], a formal explicit specification of shared conceptualization defines an ontology. Conceptualization means that the represented knowledge is based on conceptualization and models a domain by providing its classes, concepts and the relations between those concepts. Concepts and relations in ontology must be understood according to their proposed conceptualization; therefore knowledge representation languages are used to provide a formal representation for the concepts and relations contained in an ontology. The explicit indicates that the knowledge, which is stated explicitly in the domain ontology is part of the machine process conceptualization, and

the concepts and the relationships exist between them in an ontology can be shared and reused between the people in the organizations and applications [27].

An ontology has the main following concepts [45]:

1. **Concepts** specify a set of entities within a specific domain. There are two types of concepts [45]:

- *Primitive* concepts have necessary conditions in order to be members of a class. For example, ‘Centrum’ is a multi-vitamin, that has vitamin ‘C’, but there could be other things that have vitamin ‘C’ and are not ‘Centrum’ multi-vitamin.
- *Defined* concepts have both necessary and sufficient description in order to be a member of the class. For example, Leukocytes are white cells that are produced from a multi potent cell known as a Hematopoietic. If an instance is a member of class **CELL** and it has at least one *is\_produced* relationship with a member of class **HEMATOPOIETIC**, then these conditions are sufficient to determine that that instance must be a member of **LEUKOCYTES**.

2. **Relations** describe the properties of the concepts and the interactions between them. There are two types of relations [45]:

- *Taxonomic relations* organize the concepts in a subclass and superclass structure. The most common form is known as the ‘is a kind of’ relationship. For instance, Chardonnay is a kind of white wine, which in turn is a kind of alcoholic drinks.
- *Associative relationships* relate concepts through the hierarchical tree

structures, such the nominative relationships that explain the names of concepts. For example, **PROTEIN** *hasProteinName* **PROTEINNAME**.

3. **Instances** are the ‘things’ (elements) represented by concepts. Instances describe the members of a class. For example, Zinc is an instance of the concept vitamin.
4. **Axioms** can be used to constrain values for classes or their instances. Such as a property axiom `<owl:ObjectProperty rdf:ID="hasProteinName">` specifies a property that the value is an instance of the **PROTEIN\_NAME** class.

Web Ontology Language (OWL) [53] defines instantiating and shares ontologies on the web. It is a semantic markup language that provides machine interpretable semantics, and a rich vocabulary [50]. There are many reasoners that support OWL, such as Pellet [54] and FaCT++ [55].

The OWL language describes relations between classes by providing a set of constructs (e.g. `unionOf`, `intersectionOf`, `disjointWith`, `equivalentClass`), defines cardinality for properties (e.g. `minCardinality=1`, `maxCardinality=3`) and can specifies the characteristics of a property (`equivalentProperty`, `FuncionalProperty`, `Transitive`) [53].

The OWL language provides three sub languages [53]:

- **OWL-Lite** supports classification hierarchy, and it is less complex than other sub languages. It supports simple constraints such as the cardinality constraints (OWL Lite supports only 0 and 1 cardinality values).
- **OWL-DL** is based on description logic, thus it provides more expressiveness than OWL-Lite and guarantees computational completeness. It makes sure that all computations are computed and completed in finite time. OWL-DL supports all OWL language constructs, however, it has some restriction rules such as a class

cannot also be an instance of another class.

Description Logics are knowledge representation formalisms and they are portions of the First Order Logic (FOL). They can be used to represent a formal and structured way of the knowledge of a domain [35].

- **OWL-Full** supports more expressiveness than the two other sub languages, but it has no computational guarantees. It doesn't guarantee that all the computation will be completed in finite time. OWL-Full allows a class to be treated as an instance as well. However, it is unlikely that a reasoner provides full support for the all the features of OWL-Full.

## **4.2 THE CP ONTOLOGY**

CPG are paper-based and they need to be computerized in order to be executed. As mentioned in chapter 3, CPG computerization demands the abstraction of medical and functional concepts from the paper-based CPG with respect to a CPG knowledge model. In our research, the CP ontology serves as the CPG knowledge model that comprises a semantic description of the high-level concepts, relations and constraints constituting a CPG. To computerize a CPG, a medical knowledge engineer will instantiate the CP ontology with domain and functional concepts from the CPG, and models the CPG's workflow in terms of procedural relations defined in the CP ontology. An instantiation of a paper-based CPG in terms of the CP ontology is regarded as the computerization of the CPG, such that it can now be executed through CPG execution engines [92].

The CP ontology from Shayegani S. [56,92] is used for the purpose of this thesis. It represents both the structures and the constructs of a CPG and the medical domain

knowledge within a CP or CPG [56]. The CP ontology includes a number of constructs for modeling workflow in CP, such as branching, synchronization, decision, flow of activities and etc.

The ontology represents the knowledge in CP by defining 50 classes, 161 properties and 589 instances. The Class names are denoted using **SMALL CAPS**, properties with *italics* and instances with underline.

A CP can be modeled as an instance of the **CLINIAL\_GUIDELINE** class in the CP ontology (Figure 4.1).

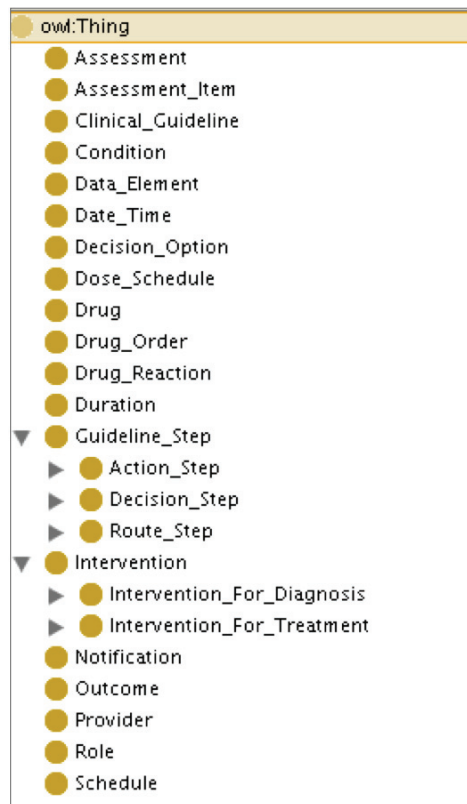


Figure 4.1 The CP ontology [56]

In the CP ontology, the sequences of activities are defined by two properties: (a) *first\_step*, a property of the **CLINICAL\_GUIDELINE** class and (b) *next\_step*, a property of

the **GUIDELINE\_STEP** class. The *first\_step* property is used to show the first step in a guideline and after the first step, the *next\_step* property indicates the sequence between two steps. We can move from one step to another step with the *next\_step* property. The range of the *next\_step* property can be either a **GUIDELINE\_STEP** or another **CLINICAL\_GUIDELINE**. These two properties represent the flow of a sequence in a workflow.

In the CP ontology, the **GUIDELINE\_STEP** represents the steps of a guideline and it has 3 main classes (Figure 4.2) [56,92]:

- **ACTION\_STEP** represents the clinical activities that are performed within a CPG's workflow and it has subclasses, such as **ASSESSMENT\_STEP** (to model a clinical assessment), **DIAGNOSTIC\_STEP** (diagnosis actions that are performed), **TREATMENT\_STEP** (the step that recommends a treatment in a guideline), **SCHEDULE\_STEP** (the step indicates that the activity needs to be scheduled to be performed later) and **NOTIFICATION\_STEP** (a step that indicates a notification for an activity needs to be sent to an external user).
- **DECISION\_STEP** represents a point where a decision has to be made for determining the next activities. The next step is based on the result of the decision. The next step is modeled by the *decision\_option* property, and each *decision\_option* may hold multiple instances of the **DECISION\_STEP** class. Each decision option indicates the next step that needs to be performed.

There are two subclasses of the **DECISION\_STEP** class:

- **PROVIDER\_DECISION\_STEP**: A healthcare provider should make a decision, and the next step is defined by his/her decision.

- **SYSTEM\_DECISION\_STEP**: The system makes a decision, when the decision logic is specified in the CPG. The decision is based on all the available information and data elements.
- **ROUTE\_STEP** represents the flow of activities in a CPG. It has 3 subclasses:
  - **BRANCH\_STEP**: It specifies the branching point in a CPG, where two or more steps need to be performed in parallel. The *branching\_step* property represents all the steps after a branching point, and it may hold multiple instances of the **GUIDELINE\_STEP** or the **CLINICAL\_GUIDELINE** class.
  - **LOOP\_STEP**: It specifies that one or more guideline steps needs to be repeated. It has four properties, which are *iteration* (to specify the number of times that a loop has to be repeated), *condition* (to specify when a loop should be terminated), *next\_step* (as long as a loop is not terminated, the *next\_step* property indicates the next step in a loop), *next\_step\_when\_loop\_ends* (it indicates the next step, when a loop is terminated).
  - **SYNCHRONIZATION\_STEP**: It synchronizes or merges the steps that are previously branched. It has a *preceding\_steps\_to\_be\_completed* property that specifies all the preceding steps need to be completed.



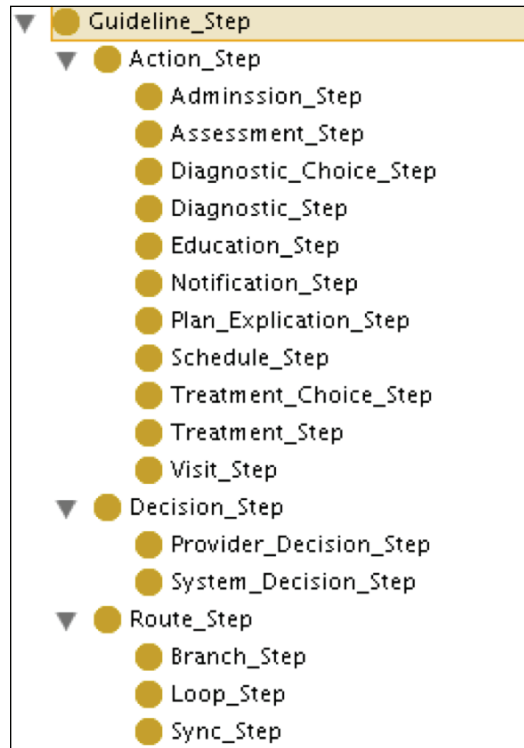


Figure 4.2 CP ontology - The subclasses of the **GUIDELINE\_STEP** class [56]

The provided CP ontology outlines the different clinical processes, their properties, constraints and relationships. An execution engine can execute an instantiation of CP from CP ontology with the patient data in order to provide recommendations [92]. However, the execution of a computerized CP is challenging, model specific, non-formal, non-standard (reusability, interoperability, analysis) and not connected to resources.

The CP ontology contains workflow, and it captures the workflow elements by providing different classes, such as **BRANCH\_STEP**, **LOOP\_STEP**, **SYNC\_STEP** and **ACTION\_STEP**.

Figure 4.3 [98] illustrates a clinical workflow, which is implemented by the standard workflow elements as we listed in Chapter 2. It has decision control construct to control the divergence and convergence of a flow, a process step to represent a general or

manual step in a process and a sequence flow construct to connect to steps. These elements can be captured by the constructs of our CPG ontology, as we listed before.

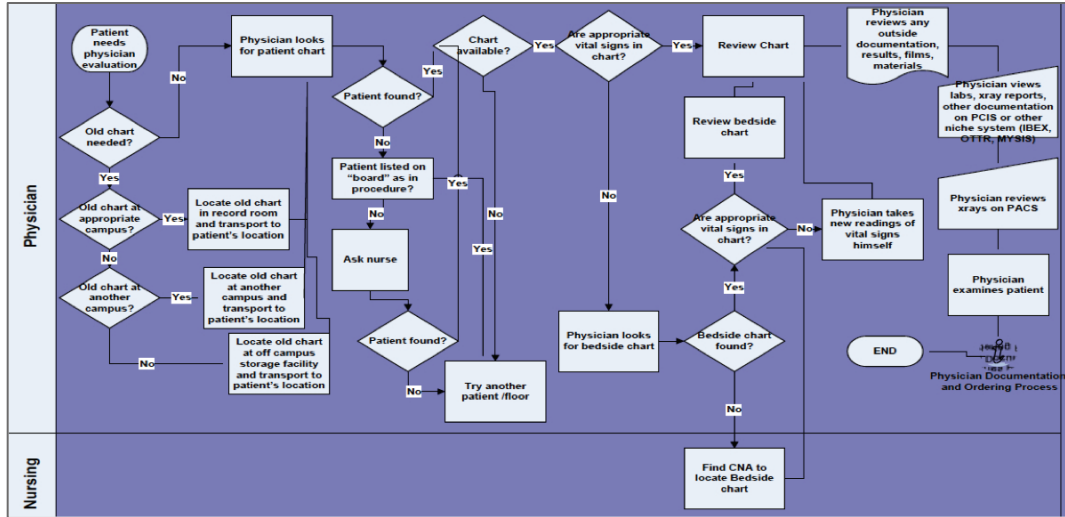


Figure 4.3 Physic patient evaluation workflow [98]

In order to design operationally and clinically pragmatic CP and to ensure data interoperability, resource management and task prioritization, we use BPMN, a business process modeling language to model CP as workflows. It allows us to design standardized CP, which clearly describes the operational aspects of clinical processes.

First we provide BPMN ontology that contains a semantic description of BPMN constructs, and then we establish a semantic interoperability (or ontology mapping) between the CP ontology and the BPMN ontology.

After explaining the BPMN ontology, we provide a table that lists the equivalence relationships between the constructs of the CP and BPMN ontologies. This table shows that the elements of the CP ontology exist in a workflow modeling language such as BPMN.

### 4.3 THE BPMN ONTOLOGY

The BPMN ontology is a formalization of the structural elements of the BPMN specification v1.1 in OWL-DL. It includes of a set of axioms for describing the BPMN elements and their combination for creating a Business Process Diagrams. For instance, it includes merging axiom to describe the correspondences between the BPMN ontology and a domain ontology. For example, a BPMN event can be used to describe the events of a domain ontology and not objects [57].

The structural assertion provides information about how to connect the graphical objects [57]:

- The **SEQUENCE\_FLOW** class has two properties (*source\_sequence\_ref*, *target\_sequence\_ref*), which states that two graphical elements are connected to it. For instance, the assertion *source\_sequence\_ref* (**sequence\_flow\_1**; **gateway\_5**) states that the *sequence\_flow\_1* originates from *gateway\_5*, and *target\_sequence\_ref* (**sequence\_flow\_1**; **activity\_2**) states that the target of the *sequence\_flow\_1* is *activity\_2*; both *gateway\_5* and *activity\_2* are graphical elements.

The process specific constraints [57] are expressions that state specific properties of a process. There are different types of process specific constraints [57]:

- *Containment constraints*: These constraints indicate that a BPD or some graphical elements contain other elements within them or not. For example the activity of patient admission is a sub process that contains an activity of registration:

$CPG:patient\_admission \subseteq BPMN:embedded\_sub\_process$

$CPG:patient\_admission \subseteq \exists BPMN:has\_embedded.$

$(BPMN:activity \sqcap CPG:registration)$

- *Enumeration constraints:* These constraints provide at least, at most and exactly enumerations to extend the containment constraints. For example, a gateway must have at least 2 outgoing gates:

$BPMN:Gateway \subseteq (\geq 2) BPMN:has\_gateway\_outgoing\_gate$

- *Precedence constraints:* These constraints state that some graphical elements should appear before others in a BPD. For example the activity of shipping is always preceded by an activity of payment:

$CPG:shipping \subseteq \forall BPMN:has\_sequence\_flow\_target\_ref \text{ } ^-$

$\forall BPMN:has\_sequence\_flow\_source\_ref.CPG:payment$

The BPMN ontology consists of 95 classes, 108 object properties (the relations between instances of two classes, for example *hasCondition* is an object property between two classes, **GATEWAY** and **CONDITION**) and 70 data properties (the relation between instances of classes and XML schema data types, for example *hasAge* (Integer data type), is a data property of the **AGE** class).

The BPMN elements are divided into two disjoint classes in the BPMN ontology [57]:

- **Graphical Elements** are the main elements to describe the business process, which we discussed in Chapter 2.
- **Supporting Elements** are used to specify the attributes of the graphical objects.

For example the supporting elements **INPUT\_SET** or **OUTPUT\_SET** are used to define attributes of the graphical object ‘**ACTIVITY**’, which describes the data requirements for input or output of the activity.

The constructs of BPMN ontology are the following [57]:

- The **BUSINESS\_PROCESS\_DIAGRAM** class collects a set of properties for a business process diagram (BDP), such as *id*, *name*, *version*, *author*, *creation\_date* and *pools*.
- Each BPD has one or more pools to represent all the participants in a process. Each **POOL** has a *process\_ref* and *has\_lanes* property. A pool has one or more lanes to organize activities within a pool.
- A **PROCESS** is the activity accomplished within a company. It includes a set of graphical elements to represent the activities. Each **PROCESS** class has *input\_set*, *output\_set* and *has\_graphical\_elements* properties. The *input\_set* or *output\_set* property defines the data requirement for input to a process or output from the process respectively. The *has\_graphical\_elements* property defines all the graphical objects that are included in a process (**ACTIVITIES**, **EVENTS**, **GATEWAYS** and **ARTIFACTS**).
- There are 3 types of events in the BPMN ontology, Start, Intermediate and End.
  - **START\_EVENT**: It specifies the start of a process. It has a *trigger* property to define the type of trigger for a start event. There are different triggers, such as **MESSAGE**, **TIMER** and **CONDITIONAL**. The **MESSAGE** trigger indicates that a process will start after receiving a message, the **TIMER** trigger means a process will start at a specific time/date and the

**CONDITIONAL** trigger means that a process will start if sets of conditions are evaluated to true.

- **INTERMEDIATE\_EVENT**: These events are between a start and end event, and they cannot start or terminate a process. However, they will affect the flow of a process. Each intermediate event has two properties; *has\_target* property indicates an intermediate event is attached to an activity and *has\_trigger* property defines the type of trigger for the event. There are different triggers for an intermediate event, such as: **CONDITIONAL**, **TIMER**, **CANCEL**, **ERROR**, **MESSAGE**, **COMPENSATION** and **SIGNAL**. Each of these events has different properties. The most important events in our study are **MESSAGE** event that has a *message\_ref* property, **TIMER** event that has two properties, *has\_timer\_cycle*, and *has\_timer\_date* and the **CONDITIONAL** event with a *condition\_ref* property.
- **END\_EVENT**: It specifies the end of a process.
- The **GATEWAY** class controls the divergence and convergence of a flow. Each gateway has a *gateway\_gate* property to indicate the number of gates (options) after a gateway. The range of the *gateway\_gate* property is the **GATE** class, which provides outgoing gates for gateways, for example a **GATEWAY** (Payment\_is\_Required) has two gates *yes*, *no*. There are three types of gateways:
  - **EXCLUSIVE\_GATEWAY**: It has two subclasses, which are **DATA\_BASED\_GATEWAY** that a decision is based on a set of data and **EVENT\_BASED\_GATEWAY** that a decision is based on an external event, such as receipt of a message.

- **PARALLEL\_GATEWAY**: It can be used for branching or merging, and has only one property, *gateway\_gate* property.
- The **SEQUENCE\_FLOW** class shows the flow of sequence in a BPD. It has three properties, *sour\_ref*, *target\_ref* and *condition\_expression*. We specify conditions for gateways in a sequence flow. The *condition\_expression* property has the range of an **EXPRESSION** class.
- The **EXPRESSION** class has an *expression\_body* property to provide the text of the expression.
- The **ACTIVITY** class represents the work that an organization performs. It has the following sub-classes: **MULTI\_INSTANCE\_LOOP**, **STANDARD\_LOOP\_ACTIVITY**, **SUB\_PROCESS**, **TASK**. There are three different sub processes: **EMBEDDED**, **REFERENCE** and **REUSABLE**. The **TASK** class can be a **SEND\_TASK**, **RECEIVE\_TASK**, **SCRIPT\_TASK**, **USER\_TASK**, **MANUAL\_TASK**, **ABSTRACT\_TASK**, **REFERENCE\_TASK** or **SERVICE\_TASK**. In this thesis, we used the **STANDARD\_LOOP\_ACTIVITY** to show a loop within a BPD and the **USER\_TASK** to represent the user activity. The **STANDARD\_LOOP\_ACTIVITY** has a *condition* property to specify a condition for a loop and a *counter* property to count the number of cycles.
- The **PROPERTY** class is used to provide the data elements. It has three properties, *name* (e.g. Age), *type* (Integer) and *value* (28).

The BPMN ontology used in this thesis is based on OWL-DL. However, in the literature review we found another BPMN ontology from Super project [60], which is in WSML-Flight (Web Service Management Layer-Flight) language [61].

As mentioned before, the CP ontology contains workflow, and it captures the workflow elements by providing different classes. We list the relationships between the workflow constructs in CP and BPMN ontologies in Table 4.1.

<b>WORKFLOW CONSTRUCTS</b>	
<b>CP Ontology</b>	<b>BPMN Ontology</b>
Clinical_Guideline	Business_Process_Diagram
Branch_Step, Syn_Step	Parallel_Gateway
Loop_Step	Standard_Loop_Activity
Decision_Step	Event_Based_Exclusive_Gateway, Data_Based_Exclusive_Gateway
Action_Step, Intervention_Step	User_Task
first_step, next_step, next_step_when_loop_ends, branching_steps	Sequence_Flow, sequence_flow_source_ref, sequence_flow_target_ref
inclusion_criteria, exclusion_criteria	Start_Event, Conditional_Event_Detail
decision_options, treatment_options	has_gateway_gate, has_sequence_flow_condition_expression
Condition	Condition, has_condition_expression
Data_Element	InputSet, OutputSet
Date_Time, Duration	Time_Date_Expression
Schedule	Timer_Intermediate_Event
Notification	Message_Intermediate_Event, Message

Table 4.1 The mapping between the workflow constructs in CP ontology and BPMN ontology

In the next section, we provide an introduction to ontology mapping, listing some of the matching techniques, and then we continue with an overview of some tools for ontology merging or mapping.



## 4.4 AN INTRODUCTION TO ONTOLOGY MAPPING

Ontology mapping [65,66,67] involves finding syntactic and semantic relationships between entities of different ontologies, and documenting semantic relations, mapping or correspondences using formal semantic mapping expressions (Figure 4.4 [67]).

There are different definitions for ontology mapping; in [65] ontology mapping is defined as finding correspondences that are similar in meaning but have different structures or name. Another definition in [66] states that ontology mapping tries to relate the elements of two given ontologies based on their structure and intended interpretations.

In our work, we consider the following definition for ontology mapping [67]:  
“Given two ontologies  $O_S$  and  $O_T$ , mapping from ontology  $O_S$  to another  $O_T$  means for each entity in ontology  $O_S$ , we try to find a corresponding entity, which has the same intended meaning in ontology  $O_T$ ”.

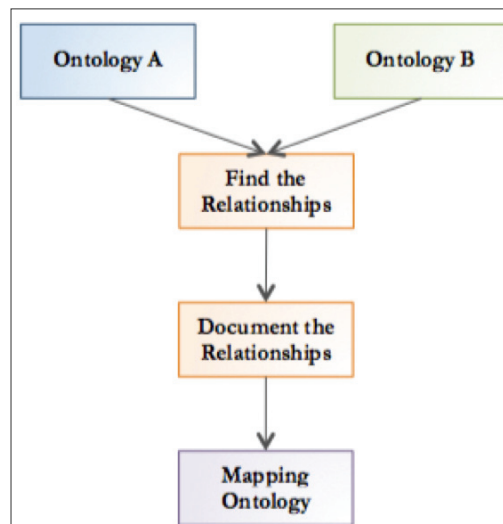


Figure 4.4 Ontology mapping steps [67]

The output of an ontology mapping exercise leads to the following [68]:

- The translation of the source ontology to the target ontology. The translation approaches specify ontologies in a standard form and translate them into a specific representation language [99]. An existing system such as Ontolingua [99] defines classes, properties, theories and functions. It translates definitions that are provided in a standard language into the forms that are required as the input for the other implemented representation system. In addition Dejing [101] uses first-order logic axioms to provide translation between ontologies; we provide an example of their work here: Ontology G1 has two properties *wife* and *married*, and G2 has the *partner* and *in\_marriage* properties. There is a first order logic axiom that describes the relationship between the domain and range (properties link instances from the domain to instances from the range [64]) of *has\_wife* and *has\_married* properties.

$$(\forall x, y) \text{has\_wife}(x, y) \rightarrow \text{has\_married}(x, y)$$

$x$  and  $y$  are variables that represent woman and man respectively. The facts that are expressed in G1 can be translated into G2 by replacing corresponding properties:

$$\text{has\_wife}(Angela, David) \rightarrow \text{has\_partner}(Angela, David)$$

$$\text{has\_married}(Angela, David) \rightarrow \text{in\_marriage}(Angela, David)$$

If we translate the axiom of G1 to G2, we have:

$$(\forall x, y) \text{has\_partner}(x, y) \rightarrow \text{in\_marriage}(x, y)$$

However, it's not always true, since a woman is a partner of a man doesn't mean that she must be in marriage with him.

- The merging of the two ontologies to create a new ontology; two given ontologies will be merged to a new third ontology. Prompt is a tool that supports the merging of two ontologies through ontology mapping [70]. We will provide an overview of Prompt later in this chapter.
- Ontology mapping can be represented by providing axioms to relate the elements of one ontology to the elements of another ontology [68], such as the MAFRA framework [79] that uses a semantic bridging ontology for encoding mapping, and an instance of this ontology includes semantic bridge instances to map an instance of the source entity to the instance of the target entity. For instance

```

<ConceptBridge rdf:ID="Individual-Man">
  <relatesSourceEntity rdf:resource="#user_task"/>
  <relatesTargetEntity rdf:resource="#diagnosis"/>
</ConceptBridge>

```

OWL itself provides tools to create axioms between entities. An example between two entities could be:

```

<rdf:RDF>
  <owl:ontology>
    <owl:imports rdf:resource="http://www.axiom.com/ont1"/>
    <owl:imports rdf:resource="http://www.axiom.com/ont2"/>
  </owl:Ontology>
  <owl:Class rdf:about="http://www.axiom.com/ont1#user_task">
    <owl:equivalentClass
      rdf:resource="http://www.axiom.com/ont2#admission"/>
  </owl:Class>
</rdf:RDF>

```

Shvaiko [100] provides a formalism to describe a mapping relationship. Their formalism is defined as a five-tuple  $(id, e, e', n, R)$ , where:

- $id$  is the unique identifier for a given mapping relation.

- $e, e'$  are the entities (classes or properties) in the source and target ontology respectively.
- $n$  is a mathematical confidence measure for the mapping relation between  $e, e'$ .
- $R$  is the relation between the entities  $e, e'$  (e.g. equivalence ( $=$ ), more general ( $\supseteq$ ), disjoint ( $\perp$ ), overlapping ( $\cap$ ))

The ontology mapping expresses the mapping relations by providing a representation language. Ontology representation languages provide more effective representation solutions for ontology mapping, since the ontology is expressive itself [69].

The mapping language specifies the actual mappings and the main goal of ontology mapping representation language [71] is to express a mapping relation. Therefore the expressivity of the mapping languages (the type of relations that can be expressed between the two ontology) is an important characteristic of these languages.

Two important tasks have to be accomplished in an ontology mapping process [68]: First we have to find the similarities and relationships between entities of two given ontologies and then we have to describe and represent the mappings relations between two ontologies in a standard mapping representation language.

## 4.5 MAPPING TECHNIQUES

There are several techniques to find similarities between entities of two ontologies, and to establish a semantic mapping between them [74,75,76]. We present some of these basic ontology-mapping techniques here.

### 4.5.1 Terminological Techniques

These techniques [74,75] calculate the similarity between text strings, which are a sequence of letters. There are two types of terminological techniques [74]:

**String-based** techniques compare the structure of text strings. A string is a sequence of letters, a set of words or a set of letters. These techniques don't consider the semantic of terms. An example of these techniques is the two terms *Book* and *Textbook* would have high degree of similarity, whereas *Book* and *Paper* have low degree of similarity.

These techniques are not strong enough and using these techniques in the mapping process alone is not enough. The following examples illustrate the weakness of these techniques.

The two terms *Person* and *Personality* have high degree of similarity since the text strings are quite similar, although they have different meanings. In another example the two terms *Drug* and *Medicine* are very distinct from each other, although the semantic concepts are generally the same.

**Language-based** techniques are based on Natural Language Processing (NLP) and are more complex than the previous technique. In these techniques strings are not treated as a sequence of characters, they are treated as a text. These techniques compare the

meaningful terms from the text and find the similarity between them.

Language-based techniques can be classified as intrinsic methods, which rely on algorithms only and extrinsic methods use external resources such as dictionaries to find the similarity between the terms [75].

Intrinsic methods reduce each term to a standardized form that can be easily understood; they find similarities between terms that have syntactical variations. Extrinsic methods use external resources such as dictionaries to find the similarity between lexical variations for a term, for example the two words *Physician* and *Doctor* have the same meaning [75].

#### 4.5.2 Structural Techniques

The structural techniques will compare the structure of elements (e.g. classes) in ontologies. They can either compare the internal structure of elements, such as properties or cardinality, which is called internal structure or they can compare the relationship of each element with other elements, which is called external structure [74].

**Internal Structure** [74] techniques compare the properties, attributes, relations and cardinality of elements. For instance if one ontology *O1* has an element *Text* with two attributes (*Name.String*, *DateOfPublish.Date*) and ontology *O2* has an element *Manual* with two attributes (*ManualName.String*, *DateOfRelease.Date*), these techniques will give a very high similarity for these two elements, since the data types of these two attributes in the two elements are the same.

These techniques are easy to implement but they are not correct all the time. Two elements with different concepts may have properties that have the same data types, or properties of two elements may have different data types. For instance if one ontology *O1*

has an element *Person* with two attributes (*Name.String*, *DateOfBirth.Date*) and ontology *O2* has an element *Car* with two attributes (*Name.String*, *ModelDate.Date*), the internal structure techniques will suggest that these two elements are similar. However, they have different semantics.

**External Structure** [74] techniques are based on the relationship of an element with other elements. The external structure method suggests that if two entities are similar, then there could be some similarity with their adjacent entities. Ontology is considered as a graph, in which each node is an element and edges specify the relations between nodes and are labeled by a name.

#### 4.5.3 Extensional Techniques

These techniques [74,75] compare the instances of two elements (class). These techniques match two elements when they have the same set of instances. These techniques are useful when there is not enough information about the concept, but the concepts have some instances. For instance if one ontology *O1* has an element *Car* with two instances (*Audi* and *BMW*) and ontology *O2* has an element *Vehicle* with the same instances (*Audi* and *BMW*), then these techniques will suggest these two entities are similar by comparing all the instances.

#### 4.5.4 Semantic Techniques

These techniques [74,75,76] match elements in the ontologies based on their semantic interpretation. These methods justify their results based on the theoretical models [75]. An example of these techniques is based on Description Logics.

**Semantic techniques based on Description Logic (DL):** According to [74] in these techniques, provide the necessary expressivity to matches concepts in a semantic manner. The relation can be described with respect to the subsumption test, which establishes the relations between the concepts in a semantic manner. For instance if ontology *O1* has three entities (*University*, *College*, and *Department*) and *University* is a *College* with more than 10 *Departments* and ontology *O2* has three entities (*Academy*, *Institution*, *Building*) and *Academy* is *Institution* with more than 6 *Buildings*. It is also declared that *College* is equivalent to *Institution* and all *Buildings* are *Departments*, then these techniques will suggest that *University* is equivalent to *Academy*.

In this section, we listed the basic techniques for finding similarities between entities based on terminological, structural, extensional and semantic methods. However, not all of these techniques are equally applicable to any domain, the best technique is to find the appropriate combination of these techniques for a selected domain.

## **4.6 TOOLS AND FRAMEWORKS**

In this section, we provide an overview of some tools for ontology mapping that creates mapping relations between two given ontologies, and ontology merging that merges two given ontologies into one target ontology. The details of these tools are out of the scope of this study.

### **4.6.1 PROMPT**

The PROMPT suite [77] includes a set of tools to merge ontologies, align ontologies and versioning of ontologies. It has an interactive process to merge ontologies,



and a user makes many decisions. PROMPT either performs additional actions based on the users choices or provides a new set of suggestions. It can identify inconsistencies and conflicts between ontologies after performing updates.

The components of PROMPT suit are developed as plug-ins of Protégé [64]. These components are [77]:

- **PROMPT** or **iPrompt** is a merging tool, which provides suggestions for merging the elements. It gets two ontologies  $O_1$  and  $O_2$  as input, and creates a new merged ontology  $O_m$ . The merging process is based on the similarity of class names. Figure 4.5 [77] shows a screenshot of PROMPT. The main window (**A**) in the background shows a list of suggestions at the left side and the explanation for the selected suggestion at the bottom. The right side of the window (**B**) shows the merged ontology. The front screen (**C**) shows the two source ontologies side by side.

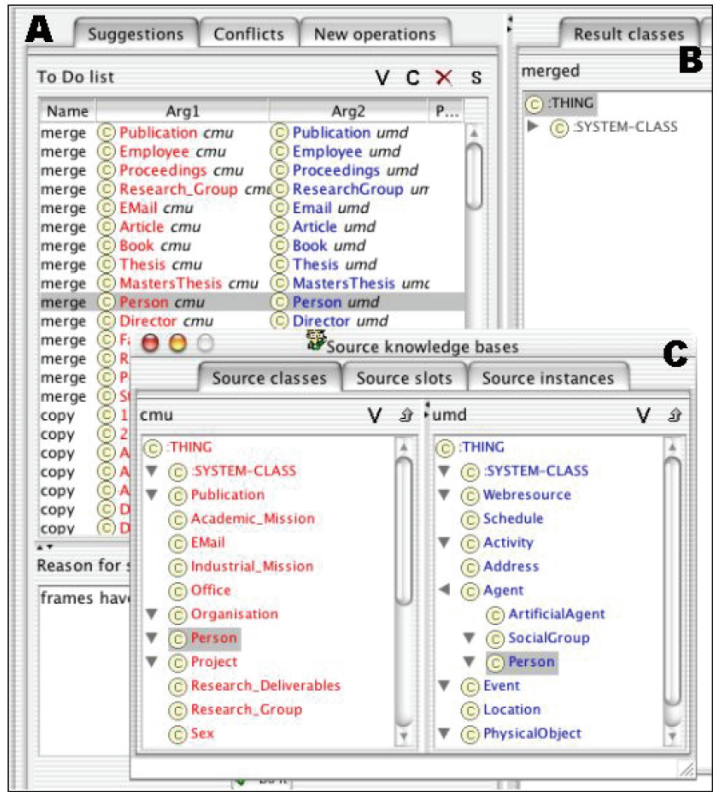


Figure 4.5 The snapshot of the ontology merging process in Prompt [77]

The PROMPT algorithm defines a set of steps for merging two ontologies, which are shown in Figure 4.6.

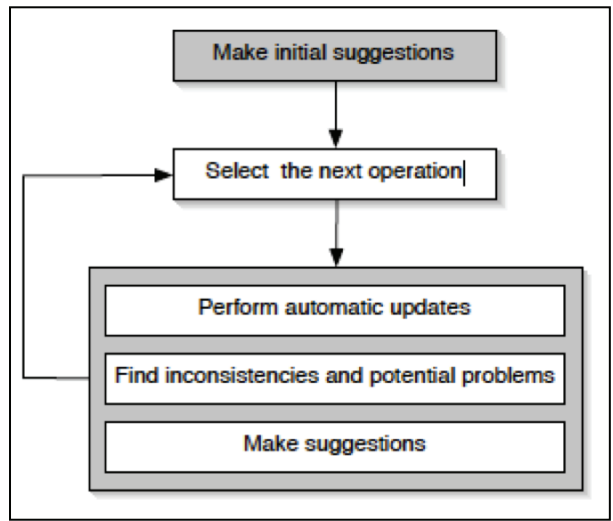


Figure 4.6 The flow of Prompt algorithm [77]

In the first step, the system makes the initial suggestions based on the lexical similarities, and then the list of suggested merging is shown to the user. In the second step, the user chooses one of the suggested merging from the list and then the system performs the merging operation and the additional changes based on the type of the operation. The system uses the confirmed correspondences from the user and performs structural analysis based on the structure of ontologies, and then creates a new list of suggestions. In addition, it states the conflicts from the performed operations and the possible solutions. An example of these conflicts could be name conflicts, when there is more than one frame (class, slot, instance) in the merged ontology, for instance we copy the class **WAGE** to the merged ontology and then we copy the slot *wage* that may exist in the source ontology. There will be a name conflict in the merged ontology. In the last step, the user responds to the suggestions and after that the next merging suggestion can be selected from the list [77].

- **PROMPTDiff** compares the structure of two versions of an ontology to identify whether there are changes in the frames, in the properties only or frames that have changed their names and also other parts of their definitions.
- **AnchorPROMPT** is an alignment tool. It finds relationship between concepts and provides additional information. It extends PROMPT by finding more similarities between ontologies, which are not identified by PROMPT. It takes two pairs of terms as the input in the source ontologies and creates new pairs of matching terms. The results can be used in PROMPT and provide new suggestions to the user.

- **PROMPTFactor** enables users to extract a new ontology from an existing ontology. The terms of the resulting ontology are well defined, and the algorithm copies all the terms that are required in order to maintain the semantics of the descriptions.

The PROMPT is developed as plug-in of the Protégé [64] environment. Protégé ontology environment has an Open Knowledge Base Connectivity (OKBC) [64] knowledge model. OKBC is frame based and it has three different types, which are classes (a set of entities), slots (relations between classes) and instances.

#### 4.6.2 GLUE

GLUE [78] is a system that uses a machine-learning technique to create mappings between two ontologies. For each concept in the source ontology, it finds the related concept in the target ontology. GLUE finds one to one mappings between concepts of ontologies, which can be seen as taxonomies.

GLUE uses two taxonomies [78], in which concepts are referred as nodes, and edges refers to is-a relationships in the taxonomies. The result is a set of similarity measures. It identifies the concepts of a given taxonomy that are similar to the concepts of another taxonomy. It has three modules [78]:

- **Distribution Estimator:** It takes two taxonomies O1 and O2 and then a machine-learning technique is applied to calculate the joint probability distributions for every pair of concepts. It computes four probabilities namely,  $P(A, B)$ ,  $P(\bar{A}, B)$ ,  $P(A, \bar{B})$ ,  $P(\bar{A}, \bar{B})$ . For example  $P(A, B)$ , is the probability that an instance belongs to both **A** and **B**, or  $P(A, \bar{B})$ , is the probability that an

instance belongs to **A**, but not to **B**.

- **Similarity Estimator:** A similarity measure such as *Jaccard coefficient* is applied to the results of the previous probabilities. The similarity can be computed by  $(P(A \cap B)/P(A \cup B))$ . The output will be a similarity matrix for the concepts of two given taxonomies.
- **Relaxation Labeler:** Searching for the similarity measures that satisfy the domain constraints in a similarity matrix. A set of similarity measures is the output of GLUE.

#### 4.6.3 MAFRA

The Mapping FRAMework (MAFRA) [79] is a mapping representation approach. It has an ontology called the Semantic Bridge Ontology (SBO). An instantiation of this ontology provides an ontology-mapping document. It provides mapping relations between concepts and attributes. It can provide conditional mappings as well [80]. The SBO includes the following concepts [79]:

- The **SEMANTIC BRIDGE** class states the relations of the source entities to target entities, based on their types and cardinality. Concepts and properties of the source ontology map into concepts and properties of the target ontology. Each bridge has a transformation service that determines the required procedure for doing this mapping transformation, and in addition the required information that the user has to provide to the execution engine.
- The class **SERVICE** provides the resources that are responsible to describe the transformations. These resources may describe the characteristics of services, such as name and location for the execution engine.

- The class **RULE** states the constraints and relevant information for a transformation.
- The class **TRANSFORMATION**, which is an obligatory class, specifies the procedure of the transformation for each semantic bridge, and it uses the *inService* relation for linking the procedure to the execution engine.
- The class **CONDITION** represents the required conditions that should be evaluated to true, in order to execute a semantic bridge.
- The composition modeling primitive belongs to the **SEMANTIC\_BRIDGE** class by the *hasBridge* relation. It allows a semantic bridge to combine various different bridges, and then to call and process bridge by bridge during the execution of transformations.
- The alternative modeling primitive is supported by the **SEMANTICBRIDEALT** class. It groups multiple mutual exclusive semantic bridges.
- The Lift & Normalization module translates the ontologies into RDF(S) in the mapping process, and it is a required module for MAFRA. The Lift & Normalization module defines a uniform representation that normalizes the ontologies for the mapping process.

The MAFRA maps classes and attributes by providing *ConceptBridge* and *AttributeBridge* (Figure 4.7 [79]).

However, MAFRA does not support mappings between properties and instances [81], unlike our mapping representation language that provide mapping between properties and instances.

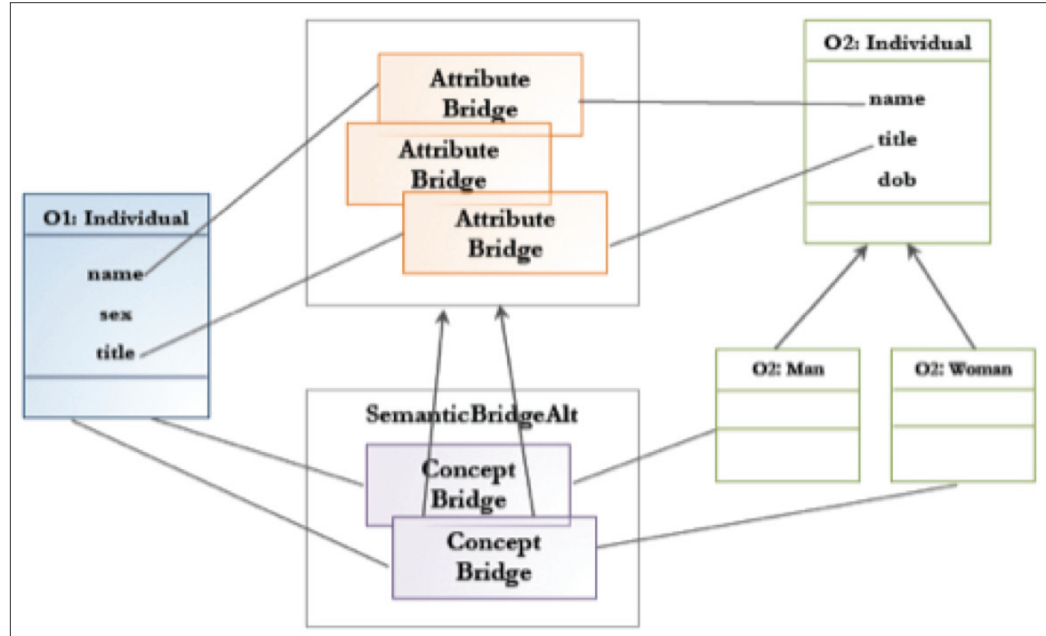


Figure 4.7 Concept and Attribute Bridge in MAFRA [79]

In this section, we have provided an overview of two tools for ontology mapping. These tools and other existing tools provide different algorithm to (semi) automatize the mapping process, they may provide a graphical user interface that allow user to relate the corresponding entities. However, they may only generate few mappings and each tool may use different algorithms to generate mapping ontology, and the results are based on different formats [82]. In the next chapter, we propose a common ontology mapping representation language that allows the alignment of semantic relations between two ontologies. We formalize the semantic correspondences between each type of entity (classes, properties and instances) of our ontologies. We explain our mapping process that documents the semantic relations between the entities of our two ontologies, and generates mapping tables. We define different types of constructs to capture the relations between the entities of our two ontologies.

## CHAPTER 5 ONTOLOGY MAPPING

In this chapter we present our work on ontology mapping to establish semantic interoperability between two ontologies—i.e. the CP ontology and the BPMN ontology. Clinical Pathways (CP) model the sequence of tasks, constraints, decision points and actor roles, to perform a specific clinical procedure, based on the operational policies of the institution [1, 2]. From a business process re-engineering perspective, a CP encapsulates the workflow about how to conduct a specific healthcare procedure for a specific disease/outcome in a specific healthcare setting. The intent of our ontology mapping exercise is to establish an interoperability framework that enables the translation of clinical workflows to a standard process workflow formalism—semantic interoperability, therefore, involves the mapping of the clinical concepts to the workflow concepts such that a clinical workflow can be represented as a process workflow and executed by a workflow execution engine. We represent clinical workflows using a CP ontology that outlines the different clinical processes, their properties, constraints and relationships, and process workflows using a BPMN ontology that contains a semantic description of BPMN constructs. Ontology mapping, therefore, is the alignment of semantic relations between the clinical and workflow ontologies such that a clinical process defined in the CP ontology is mapped to a standard BPMN workflow element in the BPMN ontology.

To further specialize the BPMN ontology towards clinical workflows, we extended our BPMN ontology to provide more salient mapping expressions, such that the extended BPMN ontology is very close to our CP ontology.



In the next chapter, we execute our BPMN-based CP in the Lombardi workflow engine (developed by IBM), whereby users can view the execution of the CP and make necessary adjustments to optimize the CP.

Lombardi does not provide the same level of workflow expressiveness and abstraction as the BPMN specification. BPMN is a much richer workflow representation formalism. We establish a semantic interoperability between the BPMN and Lombardi ontologies, to provide a richer specification for the Lombardi constructs. At the end, we model a number of existing CP using our framework and we will present our results. The overall ontology-mapping framework is shown in Figure 5.1.

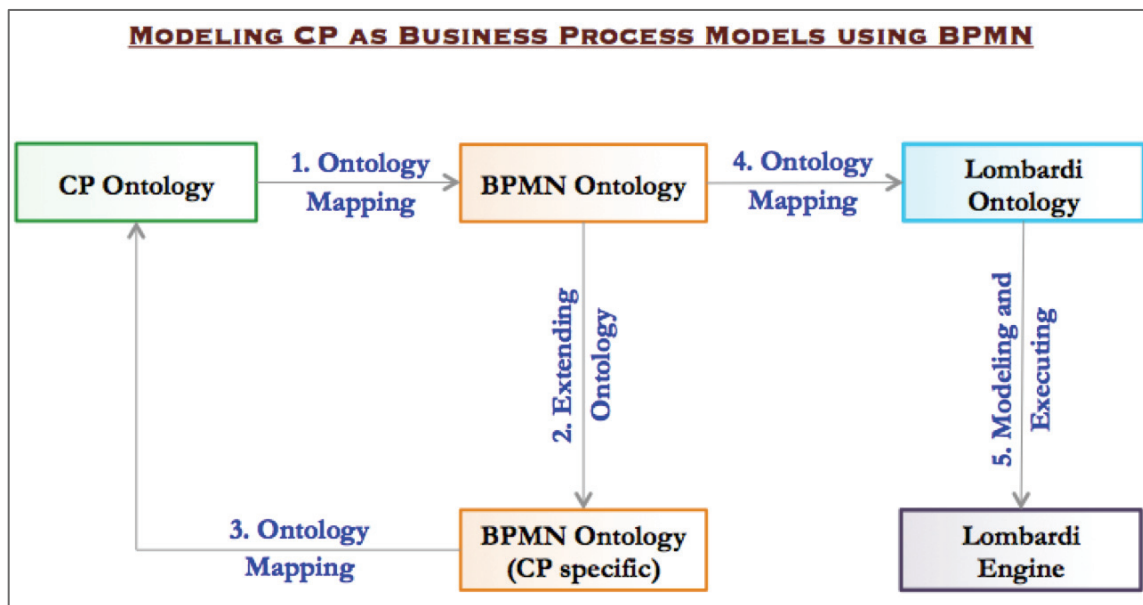


Figure 5.1 The overall ontology-mapping framework

## 5.1 CP-BPMN ONTOLOGY MAPPING

To achieve ontology mapping, we specify the correspondences between classes, properties and instances between the candidate ontologies. These correspondences are based on the terminological technique [74,75] (entity names, labels), for instance the *author* property of the CP ontology will be mapped to the *has\_business\_process\_diagram\_author* property of the BPMN ontology. In addition to the terminological technique, we specify the correspondences based on the interpretation of entities [74,75,76] when the labels are not the same. These interpretations are based on the semantic description of the entities. For instance the range of the *first\_step* property of the CP ontology will be mapped to the *sequence\_flow\_target\_ref* property of the **SEQUENCE\_FLOW** class in the BPMN ontology, and to show that this is a first step in a workflow, we write a constraint that that the source element of the sequence flow is the **START\_EVENT**.

We define a mapping expression language that allows the alignment of relations between two ontologies—the relations are represented in terms of *mapping expressions* (discussed below). The mapping expressions, therefore allow the mapping of a CP to a business workflow represented using the BPMN modeling language. The mapping expressions are written in OWL language and exported to the Terse RDF Triple Language (Turtle) [83] syntax to make them more readable. The mapping expressions are represented in a *mapping ontology*—the mapping ontology basically establishes semantic mappings between the CP and BPMN ontologies, such that the concepts in the mapping expressions will have their domain and ranges defined as concepts in the CP and BPMN ontologies.

Our ontology *mapping process* consists of four steps:

1. *Extracting and Analyzing Concepts*: First, we extract all the classes, properties and constraints of both our CP and BPMN ontologies. We list all the classes and their related properties and constraints to discover the semantic relations. The domain and range of each property is captured as well. There are existing constraints, such as timing constraints, scheduling constraints and resource constraints that are represented in the BPMN ontology and need to be analyzed, since these constraints can help us later for mapping and encoding purposes. For instance there is a constraint in the BPMN ontology that after the **EVENT\_BASED\_GATEWAY** (decision by the user), a timer or signal intermediate event should be appeared, but any object can be connected to a **DATA\_BASED\_GATEWAY** (decision by the system).
2. *Mapping Discovery*: We discover the mapping relations between the entities (classes, properties) of our CP and BPMN ontologies. These mappings are based on the terminological technique [74,75] (entity names, labels), and the interpretation of entities. These interpretations are based on the semantic description of the entities [74,75,76,79,80,100]. For instance there is a *next\_step* property in the CP ontology for going from one step to the next step, and in the BPMN ontology there is a **SEQUENCE\_FLOW** class that has two properties, which are the *sequence\_flow\_source\_ref* and *sequence\_flow\_target\_ref* properties. These two properties of the **SEQUENCE\_FLOW** class are used to represent the flow in a workflow. The domain of the *next\_step* property should be mapped to the *sequence\_flow\_source\_ref* property of the **SEQUENCE\_FLOW** class, and the range

of the *next\_step* property should be mapped to the *sequence\_flow\_target\_ref* property of the **SEQUENCE\_FLOW** class in the BPMN ontology. We provide this mapping later.

We create instances and constraints in our mapping expressions to map a class or property to another class or property, when the direct mapping is not possible. We demonstrate this mapping discovery in our mapping tables later.

3. *Documenting*: After discovering the mapping relations between our CP and BPMN ontologies, we need to document these relations. We represent our mapping expressions as a five-tuple  $M(id, T, e, e', R)$  [100], where:

- $id$  is the unique identifier for the given mapping relation.
- $T$  is the type of mapping relation. We define four types of constructs in our mapping ontology (Class, Property, Class-Property, Property-Instance). We explain these constructs later.
- $e, e'$  are the entities (class, property or instance) in the source and target ontology respectively.
- $R$  is the relation between the entities  $e, e'$ . (e.g. `equivalentClass`, `equivalentProperty`, `equivalentInstance`).

4. *Consistency Checking*: After documenting our mapping expressions, we check the consistency of our mapping ontology. We export all of our mapping expressions to an OWL ontology that we call *mapping ontology*. The CP and BPMN ontologies will be imported to this mapping ontology as well. The mapping expressions in this ontology act as the bridges between these two ontologies.

We check the consistency of this mapping ontology, first by using the Pellet Reasoner in Protégé, and then we model a number of existing CP to a BPMN based workflow. We make sure that the existing CP can be encoded to the BPMN ontology, and the control flow patterns and conditions are captured based on our mapping expressions. After encoding the existing CP in our BPMN ontology, we check the consistency in Protégé again to make sure that there is no inconsistency because of the existing constraints in the BPMN ontology. In the case of inconsistency, we correct the mapping expression and then we repeat this process again. The output is a consistent mapping ontology.

## 5.2 CP-BPMN MAPPING EXPRESSIONS

The mapping expressions contains constructs [82] to express relations between the different entities of the two ontologies:

- **Class-to-Class mapping (37 CCMappings):** Mapping a class to another class (or instance of the class).
- **Property-to-Property mapping (48 PPMappings):** Mapping a property (either object or data property) to another property (or instance of the property).
- **Class-to-Property mapping (6 CPMappings):** Mapping between a property and a class (or instance of the class).
- **Property-to-Instance mapping (79 PVMappings):** Mapping between a property and an instance. Unlike the MAFRA framework that does not support mapping between properties and instances, in our mapping expression language an instance

may be mapped as a value of a target property, or a source property/class may be mapped to an instance of the target entity.

We provide examples for each of these constructs later. In our mapping expressions we used a set of OWL properties such as cardinality, union, intersection, and equivalent (*owl:cardinality*, *owl:UnionOf*, *owl:IntersectionOf*, *owl:equivalent*, *owl:oneOf*). For instance, we indicate the relation between a class in the CP ontology, which is the same as another class in the BPMN ontology by the *owl:equivalent* property (*a owl:equivalent b*).

The mapping expressions together with these OWL properties enable the translation of clinical workflows to the BPMN workflow language. Therefore a defined clinical process in the CP ontology can be mapped to a standard BPMN workflow element in the BPMN ontology based on these mapping expressions.



### 5.2.1 Step 2: Mapping Discovery

We discover the mapping relations between entities (classes, properties) of our CP and BPMN ontologies. We discover the relationships between entities based on the terminological technique, and the interpretation of entities when the labels are not the same [74,75,76,79,80,100], as we have provided examples in the previous section. We create instances and constraints in our mapping expressions to map a class or property to another class or property, when the direct mapping is not possible.

For instance as is listed in Table 5.1, to map the **NOTIFICATION** class of the CP ontology to the **MESSAGE** class of the BPMN ontology, we have to write a constraint that a message intermediate event has a message event detail, and the message event detail

has a message reference property. After writing this constraint, we map the **NOTIFICATION** class to the range of the *message\_reference* property, which is the **MESSAGE** class.

The mappings between the classes of the CP and BPMN ontologies are listed in Table 5.1. It can be noted that whilst the CP ontology provides a fine-grained classification of the **ACTION\_STEPS** (such as **ADMISSION\_STEP**, **DIAGNOSTIC\_STEP**, etc.), the same is not the case in the BPMN ontology where there is a single high-level concept **USER\_TASK**.

THE MAPPING BETWEEN THE CLASSES OF CP AND BPMN ONTOLOGIES	
CP	BPMN
Action_Step	User_Task
Admission_Step Assessment_Step Diagnostic_Choice_Step Diagnostic_Step Education_Step Notification_Step Plan_Explication_Step Schedule_Step Treatment_Choice_Step Treatment_Step Visit_Step	(User_Task) and (BPMN_E. Cat = "Admission_Step") (User_Task) and (BPMN_E. Cat = "Assessment_Step") (User_Task) and (BPMN_E. Cat = "DiagnosticChoice_Step") (User_Task) and (BPMN_E. Cat = "Diagnostic_Step") (User_Task) and (BPMN_E. Cat = "Education_Step") (User_Task) and (BPMN_E. Cat = "Notification_Step") (User_Task) and (BPMN_E. Cat = "Plan_Explication_Step") (User_Task) and (BPMN_E. Cat = "Schedule_Step") (User_Task) and (BPMN_E. Cat = "Treatment_Choice_Step") (User_Task) and (BPMN_E. Cat = "Treatment_Step") (User_Task) and (BPMN_E. Cat = "Visit_Step") <ul style="list-style-type: none"> <li>• "Categories" can be for the user-defined semantics.</li> <li>• Instead of "Categories", we can use "Documentation" as well.</li> <li>• BPMN_Element <i>EquivalentTo</i> Graphical_Element                Graphical_Element <i>EquivalentTo</i> Flow_Object                Flow_Object <i>EquivalentTo</i> Activity</li> </ul>
Intervention_For_Diagnosis Diagnostic_Imaging Group_of_Diagnostic_Process	(User_Task) and (BPMN_E. Cat = "Intervention_Diagnosis") (User_Task) and (BPMN_E. Cat = "Diagnostic_Imaging") (User_Task) and (BPMN_E. Cat = "GroupDiagnosticProcess")
Laboratory_Exam Physical_Exam Procedure_To_Diagnosis Intervention_For_Treatment Procedure_For_Treatment Prescription Radiotherapy	(User_Task) and (BPMN_E. Cat = "Laboratory_Exam") (User_Task) and (BPMN_E. Cat = "Physical_Exam") (User_Task) and (BPMN_E. Cat = "ProcedureToDiagnosis") (User_Task) and (BPMN_E. Cat = "InterventionForTreatment") (User_Task) and (BPMN_E. Cat = "ProcedureForTreatment") (User_Task) and (BPMN_E. Cat = "Prescription") (User_Task) and (BPMN_E. Cat = "Radiotherapy")
Provider_Decision_Step	Event_Based_Exclusive_Gateway 
System_Decision_Step	Data_Based_Exclusive_Gateway 




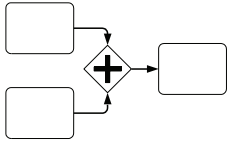

THE MAPPING BETWEEN THE CLASSES OF CP AND BPMN ONTOLOGIES	
CP	BPMN
Branch_Step	Parallel_Gateway 
Sync_Step	Parallel_Gateway 
Loop_Step	(StandardLoop_Activity) and (BPMN_E. Cat = "Loop_Step")
Data_Element	Property
Provider	Member
Role	Role
Duration	Time_Date_Expression (Timer_Event_Detail <i>has_timer_event_time_date</i> Time_Date_Expression)
Decision_Option	Gateway <i>has_out_going_gate</i> Gate (Gate <i>has_gate_outgoing_sequence_flow_ref</i> Sequence_Flow) 
Date_Time	Time_Date_Expression
Condition	Expression
Notification	Message Message_Intermediate_Event <i>has_message_event_detail</i> Message_Event_Detail Message_Event_Detail <i>has_message_event_message_ref</i> Message
Clinical_Guideline	Business_Process_Diagram

Table 5.1 The mapping between the classes of the CP and BPMN ontologies

In Table 5.2, we list some of the relationships between the object properties of the CP ontology and the BPMN ontology.


THE MAPPING BETWEEN THE PROPERTIES OF CP AND BPMN ONTOLOGIES	
OBJECT PROPERTY	
DOMAIN	RANGE
<b>expected_duration</b>	
(Action_Step or Provider_Decision_Step)	Duration
owl:oneOf (User_Task or event_based_gateway)	Time_Date_Expression
<p>These two properties should be mapped to the Timer_Intermediate_Event class, if they affect the task and workflow, and are not only an expression.</p> <p style="text-align: center;">Timer_Intermediate_Event <i>has_intermediate_event_target</i> owl:oneOf (User_Task or event_based_gateway)</p> <p style="text-align: center;">Timer_Event_Detail <i>has_timer_event_time_cycle</i> Time_Date_Expression</p>	
<b>schedule</b>	
(Schedule_Step or Procedure_For_Treatment)	Schedule
owl:oneOf ((User_Task and BPMN_Element.Category="Schedule_Step") or (User_Task and BPMN_Element.Category="Procedure_For_Treatment"))	Timer_Intermediate_Event
	
<b>condition_to_go_forward</b>	
Sync_Step	Condition
Parallel_Gateway	Expression <i>sequence_flow_condition_expression</i> Condition <i>Condition has_expression Expression</i>
DATA PROPERTY	
CPG	BPMN
<b>Loop_Step</b>	<b>Standard_Loop_Activity</b>
iterations	has_standard_loop_counter
<b>Clinical_Guideline</b>	<b>Business_Process_Diagram</b>
author	has_business_process_diagram_author

Table 5.2 The mapping between the properties of the CP and BPMN ontologies. The blue font indicates that the class or the property belongs to the BPMN ontology

As can be noted from Table 5.2, we have provided some mappings for our object and data properties. We map the domain and range of each object/data property of the CP ontology to the domain and range of another object/data property of the BPMN ontology. In addition, we map the domain and range of a property to a class of the BPMN ontology. For instance we map the domain of the *expected\_duration* object property, which can be either the **ACTION\_STEP** class or the **PROVIDER\_DECISION\_STEP** class to the **USER\_TASK** and **EVENT\_BASED\_GATEWAY** classes of the BPMN ontology. We use *OWL:oneOf (User\_Task, Event\_Based\_Gateway)* property to indicate that the domain can be mapped to one of these classes. The range of the *expected\_duration* property, which is the **DURATION** class, is mapped to the **TIME\_DATE\_EXPRESSION** class of the BPMN ontology.

We cannot always map the domain and range of each property directly to another class in BPMN; sometimes we have to create instances and provide some constraints in our mapping expressions. For instance as is listed in Table 5.2, the range of the *condition\_to\_go\_forward* object property, which is the **CONDITION** class is mapped to the **EXPRESSION** class of the BPMN ontology, but first we have to create an instance for the **SEQUENCE\_FLOW** class and the **CONDITION** class, and then we write two constraints, which are the instance of the sequence flow has a condition (instance) and the condition has an expression. These two constraints enable us to map the **CONDITION** class to the **EXPRESSION** class, and to provide the semantic description that this mapped expression belongs to the condition of a sequence flow.

These mapping expressions develop a high-level semantic mapping between the CP ontology and the BPMN ontology. It allows the alignment of semantic relations between

two ontologies and thus ensures that a clinical process defined in the CP ontology is mapped to a standard BPMN workflow element. In the next section, we provide some examples from our mapping expressions that we have documented.

### 5.2.2 Step 3: Documenting the Mappings

After discovering the mapping relations between our CP and BPMN ontologies, we need to document these mappings. As mentioned before, we write these mappings in a text file and then we export this file to an ontology, which we call it mapping ontology. To document our mappings, first of all we have defined four types of constructs, which we have explained before (ClassMapping, PropertyMapping, ClassPropertyMapping, PropertyInstanceMapping), and our mapping expression can be formalized as a five-tuple  $M(id, T, e, e', R)$  [100], where  $id$  is the unique identifier for the given mapping relation,  $T$  is the type of a mapping relation (or the type of construct),  $e, e'$  are the entities (class, property or instance) in the source and target ontology respectively and  $R$  is the relation between the entities  $e, e'$ . (e.g. equivalentClass, equivalentProperty, equivalentInstance).

#### 5.2.2.1 Class-Class Mapping

In the Class-Class mapping, we map a class directly to another class or an instance of a class. For instance, in our mapping ontology, the **ACTION\_STEPS** (the second row of Table 5.1) or the **INTERVENTION\_STEPS** (the third row of Table 5.1) will be mapped to the **USER\_TASK** class of the BPMN ontology.

The mapping is done in the following way:

- The **USER\_TASK** is a **BPMN\_ELEMENT**, so it inherits all of its properties.
- Each **BPMN\_ELEMENT** has a *has\_category* object property with the range of the **CATEGORY** class, and the **CATEGORY** has a *name* data property with the string data type.
- Each **ACTION\_STEPS** or **INTERVENTION\_STEPS** is a **USER\_TASK** class and can be mapped to it. However, in order to differentiate the action steps (e.g. admission, diagnosis, assessment etc.) from each other, we create an instance for the **USER\_TASK** class and then we map the action/intervention steps to these instances (e.g. **ADMISSION\_STEP** to the Adm\_Instance of the **USER\_TASK**).
- Each instance of the **USER\_TASK** class has a category property and its value is the name of the **ACTION/INTERVENTION\_STEP**. Figure 5.2 shows the actual mapping for the **ADMISSION\_STEP**.

```

# Admission_Step

:CC01 a :CCmapping ;
    rdfs:label "(CC01) Admission_Step - Adm_Instance-Category.Name=Admission_Step" ;
    rdfs:comment "
The Admission_Step class in CPG will be equivalent to the instance of the User_Task class in
BPMN and has a Category property with the value of Admission_Step."

:Adm_Instance          a :CreateInstance .
:bpmnElement_AdmStep  a :CreateInstance .
:category_AdsStep     a :CreateInstance .

:Adm_Instance          :hasClass    bpmn:user_task .
:bpmnElement_AdmStep  :hasClass    bpmn:BPMN_element .
:category_AdsStep     :hasClass    bpmn:category .

:Adm_Instance    owl:EquivalentTo    :bpmnElement_AdmStep .

:bpmnElement_AdmStep    bpmn:has_BPMN_element_category    :category_AdsStep .
:category_AdsStep       bpmn:has_category_name              "Admission_Step" .

:CC01 :hasSource :CC01S .
:CC01 :hasTarget :CC01T .

:CC01S :hasClass cpg:Admission_Step .
:CC01T :hasClass :Adm_Instance .

:CC01S owl:equivalentClass :CC01T .

```

Figure 5.2 The actual class-class mapping for the **ADMISSION\_STEP** to the instance of the **USER\_TASK** class

In another example for the class-class mapping, we map the **PROVIDER\_DECISION\_STEP** class of the CP ontology to the **EVENT\_BASED\_GATEWAY** class of the BPMN ontology and the **SYSTEM\_DECISION\_STEP** class to the **DATA\_BASED\_GATEWAY** class. The mapping is a direct class-to-class mapping, and there is no need to create any instances or constraints. We map the **PROVIDER\_DECISION\_STEP** class of the CP ontology to the **EVENT\_BASED\_GATEWAY** class of the BPMN ontology, since the value of the condition property for the

**EVENT\_BASED\_GATEWAY** by default is ‘none’, and it means that there is no condition and the provider has to make a decision, and we map the **SYSTEM\_DECISION\_STEP** class of the CP ontology to the **DATA\_BASED\_GATEWAY** class of the BPMN ontology, since there is a constraint in the BPMN ontology such that we have to specify a condition for the **DATA\_BASED\_GATEWAY**. The actual mapping is shown in Figure 5.3.

```

                                # Provider_Decision_Step

:CC02 a :CCmapping ;
      rdfs:label "(CC02) Provider_Decision_Step - event_based_exclusive_gateway" ;
      rdfs:comment "The Provider_Decision_Step class in CPG will be equivalent to the
event_based_exclusive_gateway class in BPMN.".

:CC02 :hasSource :CC02S .
:CC02 :hasTarget :CC02T .

:CC02S :hasClass   cpg:Provider_Decision_Step .
:CC02T :hasClass   bpmn:event_based_exclusive_gateway.

:CC02S owl:equivalentClass :CC02T .

                                # System_Decision_Step

:CC03 a :CCmapping ;
      rdfs:label "(CC03) System_Decision_Step - data_based_exclusive_gateway" ;
      rdfs:comment "The System_Decision_Step class in CPG will be equivalent to the
data_based_exclusive_gateway class in BPMN.".

:CC03 :hasSource :CC03S .
:CC03 :hasTarget :CC03T .

:CC03S :hasClass   cpg:System_Decision_Step .
:CC03T :hasClass   bpmn:data_based_exclusive_gateway.

:CC03S owl:equivalentClass :CC03T .

```

Figure 5.3 The actual class-class mapping for the provider and system decision steps to the event and data based exclusive gateways

The differences between the **DATA\_BASED\_GATEWAY** and **EVENT\_BASED\_GATEWAY** are:

- The decision for the **DATA\_BASED\_GATEWAY** is made by the system according to a set of data. For the **EVENT\_BASED\_GATEWAY**, the provider makes the decision and there is no data.
- For the **EVENT\_BASED\_GATEWAY**, the condition of the sequence flow is set to 'none', but for the **DATA\_BASED\_GATEWAY** we have to specify a condition.
- The alternatives of the **EVENT\_BASED\_GATEWAY** are based on an event that occurs at the point in the process. There is a construct in the BPMN ontology that after an **EVENT\_BASED\_GATEWAY**, the next object should be either a timer or signal intermediate event, but any object can be connected to a **DATA\_BASED\_GATEWAY**. In our study for modeling a CP to a BPMN based workflow, this construct is demonstrated as the system waits for a certain amount of time to receive a response from the provider. The response can be a yes or no message that determines which path should be taken (Figure 5.4).

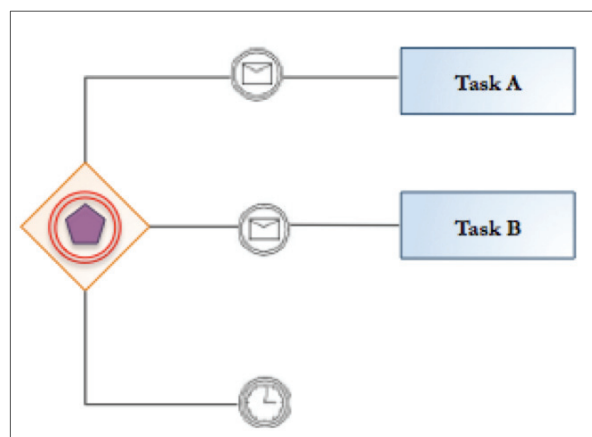


Figure 5.4 The **EVENT\_BASED\_GATEWAY** construct



### 5.2.2.2 Property-Property Mapping

The previous mappings (Figure 5.2, Figure 5.3) were the class-class mappings, now we provide examples for the property-property mapping.

We map the data properties directly to each other, the domain and range of the data properties are mapped before, in the class-to-class mappings; we only specify to what class the data property belongs. The mappings for data properties are the simplest type of mapping, and they are different from the object property or class mappings.

We only map a data property to another data property. However, in class-class mapping, we may have to create instances or constructs before mapping a class to another class, as the provided examples in the previous section.

The mappings for the object properties are different from the mapping of the data properties. We always cannot map an object property directly to another object property, since each object property in the CP ontology has a domain and range, which are different from the domain and range of the object property in the BPMN ontology. Therefore we have to map the domain and range of each object property in the CP ontology to the domain and range of an object property in the BPMN ontology. We provide an example later.

Figure 5.5 illustrates a data property mapping. It provides the actual mapping of the *author* property of the **CLINICAL\_GUIDELINE** class of the CP ontology, to the *has\_business\_process\_diagram\_author* property of the **BUSINESS\_PROCESS\_DIAGRAM** class of the BPMN ontology. We map data properties directly to each other. In this example the *author* property can be mapped directly to the

*has\_business\_diagram\_author*, which is a data property of the **BUSINESS\_PROCESS\_DIAGRAM** class of the BPMN ontology.

```
# Author Property

:DPP01 a :PPmapping ;
    rdfs:label "(DPP01) author - has_business_process_diagram_author" ;
    rdfs:comment "The author property of the clinical_guideline class will be equivalent
to the has_business_process_diagram_author property of the business_process_diagram
class.".

:DPP01 :hasSource :DPP01S .
:DPP01 :hasTarget :DPP01T .

:DPP01S :hasClass          cpg:Clinical_Guideline .
:DPP01S :hasThingsToMap    cpg:author .

:DPP01T :hasClass          bpmn:business_process_diagram .
:DPP01T :mappedToProperty  bpmn:has_business_process_diagram_author .
```

Figure 5.5 The actual property-property mapping for the *author* property to the *has\_business\_process\_diagram\_author* property

In another example for the property-property mapping, we map the *date\_time\_format* and *date\_time\_value*, data properties of the **DATE\_TIME** class of the CP ontology to the data properties of the **TIMES\_DATE\_EXPRESSION** class, which are *has\_expression\_expression\_language* and *has\_expression\_expression\_body*. The actual mappings are shown in Figure 5.6.

```

### Date_Time - Time_Date_Expression (Data Properties)

# datetime_format

:DPP02 a :PPmapping ;
    rdfs:label "(DPP02) datetime_format-has_expression_expression_language" ;
    rdfs:comment "The datetime_format property of Date_Time in CPG will be
equivalent to time_date_expression class.has_expression_expression_language property in
BPMN.".

:DPP02 :hasSource :DPP02S .
:DPP02 :hasTarget :DPP02T .

:DPP02S :hasClass          cpg:Date_Time .
:DPP02S :hasThingsToMap   cpg:datetime_format .

:DPP02T :hasClass          bpmn:time_date_expression .
:DPP02T :mappedToProperty bpmn:has_expression_expression_language .

#####          datetime_value

:DPP03 a :PPmapping ;
    rdfs:label "(DPP03) datetime_value-has_expression_expression_body" ;
    rdfs:comment "The datetime_value property of Date_Time in CPG will be
equivalent to time_date_expression class.has_expression_expression_body property in
BPMN.".

:DPP03 :hasSource :DPP03S .
:DPP03 :hasTarget :DPP03T .

:DPP03S :hasClass          cpg:Date_Time .
:DPP03S :hasThingsToMap   cpg:datetime_value .

:DPP03T :hasClass          bpmn:time_date_expression .
:DPP03T :mappedToProperty bpmn:has_expression_expression_body .

```

Figure 5.6 The actual property-property mapping for the properties of the **DATE\_TIME** class of the CP ontology

The next mapping (Figure 5.7) will map the domain and range of the *condition\_to\_go\_forward* object property to the related classes in the BPMN ontology.

The mapping is done in the following way:

- The domain of the *condition\_to\_go\_forward* object property is the **SYNC\_STEP** class and the range is the **CONDITION** class. We have mapped the **SYNC\_STEP** and the **CONDITION** class to the **PARALLEL\_GATEWAY** and **EXPRESSION** class before in our class-class mapping. However, this is an object property mapping and we want to specify how this object property can be mapped to the BPMN ontology.
- In order to do this mapping, we need to create instances and constructs in our mapping expression. We create instances for the **PARALLEL\_GATEWAY** (pg\_syms), **GATE** (pg\_gate), **SEQUENCE\_FLOW** (pg\_sf) and **EXPRESSION** (pg\_ex) classes of the BPMN ontology. We also have to write the following constructs as well:

```
:pg_syms bpmn:has_gateway_gate (:pg_gate [owl:cardinality "synGateCounter"])  
    :pg_gate bpmn:has_gate_outgoing_sequence_flow_ref :pg_sf .  
:pg_sf bpmn:has_sequence_flow_condition_expression :pg_ex .
```

We specify that a parallel gateway has a number of gates by using the *owl:cardinality* property, and each gate has an outgoing sequence flow ref and each sequence flow has a condition expression.

- We map the domain of the *condition\_to\_go\_forward* property to the instance of **PARALLEL\_GATEWAY** class (pg\_syms) and the range to the instance of the **EXPRESSION** class (pg\_ex). The provided constructs provide the semantic descriptions that this expression belongs to an outgoing sequence flow from a gate of a parallel gateway.

```

# condition_to_go_forward

:PP01 a :PPmapping ;
      rdfs:label "(PP01) condition_to_go_forward - parallel_Gateway (domain)" ;
      rdfs:comment "The domain of condition_to_go_forward property in CPG will be
mapped to instance the parallel_gateway in BPMN."

pg_syns a :CreateInstance .
:pg_gate a :CreateInstance .
:pg_sf a :CreateInstance .
:pg_ex a :CreateInstance .

:pg_syns :hasClass bpmn:parallel_gateway .
:pg_gate :hasClass bpmn:gate .
:pg_sf :hasClass bpmn:sequence_flow .
:pg_ex :hasClass bpmn:expression .

:pg_syns bpmn:has_gateway_gate (:pg_gate [owl:cardinality "synGateCounter"]).
:pg_gate bpmn:has_gate_outgoing_sequence_flow_ref :pg_sf .
:pg_sf bpmn:has_sequence_flow_condition_expression :pg_ex .

# Domain

:PP01 :hasSource :PP01S .
:PP01 :hasTarget :PP01T .

:PP01S :hasProperty cpg:condition_to_go_forward .
:PP01S :hasThingsToMap "domain" .

:PP01T :hasClass bpmn:parallel_gateway .
:PP01T :mappedToInstance :pg_syns .

#Range

:PP02 a :PP02mapping ;
      rdfs:label "(PP02) condition_to_go_forward - condition (range)" ;
      rdfs:comment "The range of condition_to_go_forward property in CPG will be
mapped to instance values for the expression class in BPMN."

:PP02 :hasSource :PP02S .
:PP02 :hasTarget :PP02T .

:PP02S :hasProperty cpg:condition_to_go_forward .
:PP02S :hasThingsToMap "range" .

:PP02T :hasProperty bpmn:has_sequence_flow_condition_expression .
:PP02T :mappedToInstance :pg_ex .

```

Figure 5.7 The actual property-property mapping for the domain and range of the *condition\_to\_go\_forward*

### 5.2.2.3 Class-Property Mapping

The next mapping (Figure 5.8) will map the *next\_step* object property of the CP ontology to the **SEQUENCE\_FLOW** class of the BPMN ontology. This is a class-property mapping, since the *next\_step* is an object property in the CP ontology and the **SEQUENCE\_FLOW** is a class in the BPMN ontology. The mapping is done in the following way:

- There is a **SEQUENCE\_FLOW** class in BPMN ontology. Each **SEQUENCE\_FLOW** class has two object properties, which are *has\_sequence\_flow\_source\_ref* and *has\_sequence\_flow\_target\_ref*.
- The range of the *has\_sequence\_flow\_source\_ref* is an object, which is the source of the flow and the range of the *has\_sequence\_flow\_target\_ref* is an object, which is the target of the flow.
- We map the domain of the *next\_step* property to the *has\_sequence\_flow\_source\_ref* property of the **SEQUENCE\_FLOW** class and the range of the *next\_step* property to the *has\_sequence\_flow\_target\_ref* property of the **SEQUENCE\_FLOW** class (Figure 5.9).

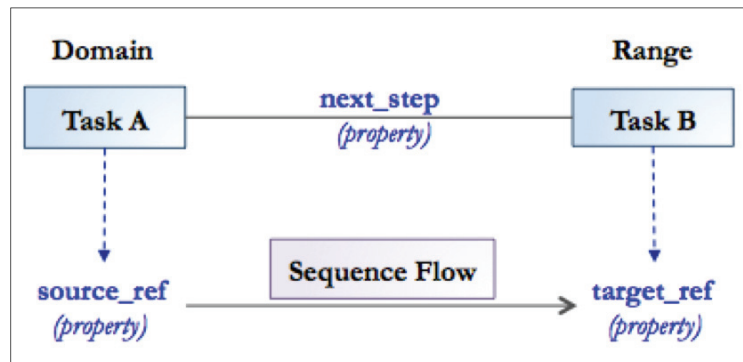


Figure 5.8 Mapping the *next\_step* property to the **SEQUENCE\_FLOW** class

```

# next_step Property

#Domain

:CP01 a :CPmapping ;
      rdfs:label "(CP01) next_step - Sequence_flow (domain)";
      rdfs:comment "The domain of the next_step property in CPG will be mapped to the
has_sequence_flow_source_ref property of the Sequence_flow class in BPMN."

:CP01 :hasSource :CP1S .
:CP01 :hasTarget :CP1T .

:CP1S :hasProperty      cpg:next_step .
:CP1S :hasThingsToMap  "domain" .

:CP1T :hasClass         bpmn:sequence_flow .
:CP1T :mappedToProperty bpmn:has_sequence_flow_source_ref .

#Range

:CP02 a :CPmapping ;
      rdfs:label "(CP02) next_step - Sequence_flow (range)";
      rdfs:comment "The range of the next_step property in CPG will be mapped to the
has_sequence_flow_target_ref property of the Sequence_flow class in BPMN."

:CP02 :hasSource :CP2S .
:CP02 :hasTarget :CP2T .

:CP2S :hasProperty      cpg:next_step .
:CP2S :hasThingsToMap  "range" .

:CP2T :hasClass         bpmn:sequence_flow .
:CP2T :mappedToProperty bpmn:has_sequence_flow_target_ref .

```

Figure 5.9 The actual property-class mapping for the *next\_step* property to the **SEQUENCE\_FLOW** class

#### 5.2.2.4 Property-Instance Mapping

An example of property-instance mapping (Figure 5.10), maps the *branching\_steps* property to the instance of the **PARALLEL\_GATEWAY** class. The mapping is done in the following way:

- The domain of the *branching\_steps* property is the **BRANCHING\_STEP** class and its range is the next activity.
- The **Parallel\_Gateway** class has a property, which is *has\_gateway\_gate\_property* and its range is the **GATE** class. The **GATE** class indicates the options after a gateway, for example a Yes gate and No gate. We indicate the number of gates by using the *owl:cardinality* property. The value of this property is a variable (e.g. `branchCounter`) that should be calculated during the execution.
- We create an instance for the **PARALLEL\_GATEWAY (PG)**, **Gate (GA)** and **SEQUENCE\_FLOW (SF)** classes, and then we write these constructs in the mapping file:

```
PG has_gateway_gate (GA [owl:cardinality "branchCounter"])
```

```
GA has_gate_outgoing_sequence_flow_ref SF
```

Each parallel gateway has a number of gates (or options) and each gate has an outgoing sequence flow reference. This **SEQUENCE\_FLOW** class has a *sequence\_flow\_source\_ref* property, with the range of the **GATE** class, and a *sequence\_flow\_target\_ref* property, with the range of the next activity.

- We map the domain of the *branching\_steps* property to **PG** (the instance of parallel gateway). The range of *branching\_steps*, which is the next activity, will



be mapped to the *has\_sequence\_flow\_target\_ref* property of the SF (instance of the **SEQUENCE\_FLOW** class).

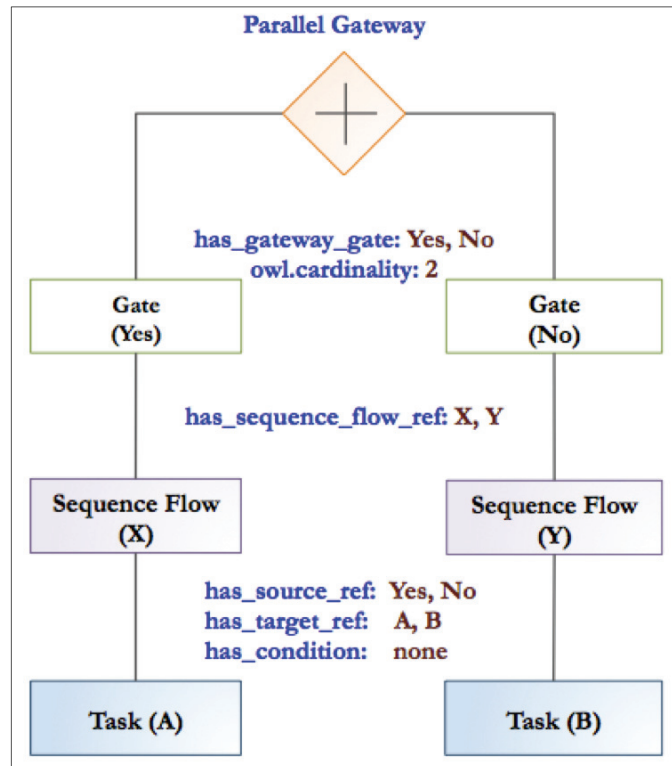


Figure 5.10 Mapping the *branching\_steps* property to the **PARALLEL\_GATEWAY** class

As is shown in the Figure 5.9, the **SEQUENCE\_FLOW** has a *condition* property, which by default is 'none' for the parallel gateway. In the case of data- or event-based gateway, we can set a condition in each sequence flow by using the *has\_condition* property, with the range of the **EXPRESSION** class. The actual mapping is shown in Figure 5.11.

```

                                #Branching_Step

:parallelGT_BS a :createInstance .
:GT_BS        a :CreateInstance .
:SF_BS        a :CreateInstance .
:parallelGT_BS :hasClass    bpmn:parallel_gateway .
:GT_BS        :hasClass    bpmn:gate .
:SF_BS        :hasClass    bpmn:sequence_flow .

: parallelGT_BS bpmn:has_gateway_gate    (:GT_BS [owl:cardinality "branchCounter"]) .
:GT_BS        bpmn:has_gate_outgoing_sequence_flow_ref    :SF_BS .

                                #Domain

:PV01 a :PVmapping ;
      rdfs:label "(PV01) branching_step-parallel_gateway (domain)" ;
      rdfs:comment "The domain of branching_step property in CPG will be mapped to the
instance of the parallel_gateway in BPMN. Number of the branches will be calculated." .

:PV01 :hasSource    :PV1S .
:PV01 :hasTarget    :PV1T .

:PV1S :hasProperty    cpg:branching_steps .
:PV1S :hasThingsToMap "domain" .

:PV1T :hasClass    bpmn:parallel_gateway .
:PV1T :mappedToInstance    :parallelGT_BS .

                                #Range

:PV02 a :PVmapping ;
      rdfs:label "(PV02) branching_step-parallel_gateway (range)" ;
      rdfs:comment "The range of the branching_step property in CPG will be mapped to
the sequence_flow_target_ref property of the SF_BS instance (sequence_flow)." .

:PV02 :hasSource    :PV02S .
:PV02 :hasTarget    :PV02T .

:PV02S :hasProperty    cpg:branching_steps .
:PV02S :hasThingsToMap "range" .

:PV02T :hasInstance    : SF_BS .
:PV02T :mappedToProperty    bpmn:has_sequence_flow_target_ref .

```

Figure 5.11 The actual property-instance mapping for the **BRANCHING\_STEP** to the **PARALLEL\_GATEWAY** class

In the last example we provide another property-instance mapping for the domain and range of the *acceptable\_duration\_of\_results* property of the CP ontology to the BPMN ontology. The mapping is done in the following way:

- The domain of the *acceptable\_duration\_of\_results* is the **ACTION\_STEP** class. As mentioned earlier the **ACTION\_STEP** class can be mapped to the **USER\_TASK** class of the BPMN ontology.
- The range of the *acceptable\_duration\_of\_results* is the **DURATION** class of the CP ontology. The **DURATION** class can be mapped to the **TIME\_DATE\_EXPRESSION** class of the BPMN ontology.
- In BPMN, a **TIMER\_INTERMEDIATE\_EVENT** can be attached to the **USER\_TASK** class with the *has\_intermediate\_event\_target* object property. The range of this property is the **USER\_TASK** class.
- Each **TIMER\_INTERMEDIATE\_EVENT** has a *trigger* property, which may define the details of this event, such as time and date.
- We create an instance for the **TIMER\_INTERMEDIATE\_EVENT** (TIE), **TIMER\_EVENT\_DETAIL** (TED), **TIME\_DATE\_EXPRESSION** (TDE) and the **USER\_TASK** (UT) class. These instances will enable us to write the following expressions in our mapping ontology.

```

:TIE    bpmn:has_intermediate_event_trigger    :TED .
:TIE    bpmn:has_intermediate_event_target    :UT .
:TED    bpmn:has_timer_event_time_date        :TDE .

```

- After writing these expressions, we map the domain (**ACTION\_STEP**) of the *acceptable\_duration\_of\_results* property to the UT instance, and the range (**DURATION**) to the TDE instance. The actual mapping is shown in Figure 5.12.

```

                                # acceptable_duration_of_results

:TIE    a :CreateInstance .
:TED    a :CreateInstance .
:TDE    a :CreateInstance .
:UT     a :CreateInstance .
:TIE    :hasClass bpmn:timer_intermediate_event .
:TED    :hasClass bpmn:timer_event_detail .
:TDE    :hasClass bpmn:time_date_expression .
:UT     :hasClass bpmn:user_task .
:TIE    bpmn:has_intermediate_event_target    :UT .
:TIE    bpmn:has_intermediate_event_trigger   :TED .
:TED    bpmn:has_timer_event_time_date       :TDE .

                                #Domain

:PV02 a :PVmapping ;
      rdfs:label "(PV02) acceptable_duration_result - Timer_intermediate_event target" ;
      rdfs:comment "The domain of acceptable_duration_result property in CPG will be
mapped to instance values for the range of has_event_target in BPMN." .

:PV02 :hasSource :PV02S .
:PV02 :hasTarget :PV02T .

:PV02S :hasProperty      cpg:acceptable_duation_of_results_if_available .
:PV02S :hasThingsToMap   "domain" .

:PV02T :hasProperty      bpmn:has_intermediate_event_target .
:PV02T :mappedToInstance :UT .

                                #Range

:PV03 a :PVmapping ;
      rdfs:label "(PV03) acceptable_duration - Time_date_expression" ;
      rdfs:comment "The range of accepted_duration property in CPG will be mapped to
instance values for the range of time_date_expression class in BPMN." .

:PV03 :hasSource :PV03S .
:PV03 :hasTarget :PV03T .

:PV03S :hasProperty      cpg:acceptable_duation_of_results_if_available .
:PV03S :hasThingsToMap   "range" .

:PV03T :hasClass         bpmn:time_date_expression .
:PV03T :mappedToInstance :TDE .

```

Figure 5.12 The actual property-instance mapping for the *acceptable\_duration\_of\_results* to the instances of the **USER\_TASK** and the **TIME\_DATE\_EXPRESSION** class of the BPMN ontology

In this section we described the ontology mapping process to establish a semantic interoperability between the CP and BPMN ontologies. We demonstrated through examples how the mapping ontology was achieved through a range of the mapping expressions developed by us. This ontology mapping ensures that a clinical process defined in the CP ontology is mapped to a standard BPMN workflow element. We would like to point out that after establishing the mapping, we then encoded six different CP in the BPMN ontologies by using our mapping expressions (discussed in detail later in Chapter 6).

### 5.2.3 Step 4: Consistency Checking and the Output

After documenting our mapping expressions, we check the consistency of our mapping ontology. We export all of our mapping expressions to an OWL ontology that we call mapping ontology. This mapping ontology includes the CP and BPMN ontologies as well. Our mapping expressions act as the bridges between these two ontologies and link the classes and properties of our mapping expressions to the classes and properties of the CP and BPMN ontologies.

The consistency of this mapping ontology, is checked by using the Pellet Reasoner in Protégé, and by modeling a number of existing CP to a BPMN based workflow to make sure that the existing CP can be modeled to a BPMN based workflow based on our mapping expressions. We make sure that the existing CP can be encoded to the BPMN ontology, and the control flow patterns and conditions are captured based on our mapping expressions.

We considered six already encoded CP in the CP ontology. Each CP is modeled as an instance of the **CLINICAL\_GUIDELINE** class in the ontology and it has a goal. To relate

this instance to our BPMN ontology, first we created an instance of the **BUSINESS\_PROCESS\_DIAGRAM** class in the BPMN ontology, and then an instance of the **CLINICALPATHWAY** class and the *has\_business\_process\_diagram* property. Each clinical pathway has a business process diagram. We provide the details of this encoding later in Chapter 7.

After encoding existing CP in our BPMN ontology, we check the consistency in Protégé again to make sure that there is no inconsistency. In the case of an inconsistency, we correct the mapping and then we repeat this process again.

After checking the consistency of our mapping expressions, we have a mapping ontology. This mapping ontology is consistent, and it contains a set of expressions that allow the alignment of semantic relations between two ontologies and thus ensures that a clinical process defined in the CP ontology is mapped to a standard BPMN workflow element.

### **5.3 EXTENDED BPMN ONTOLOGY**

The CP ontology provides a fine-grained classification of the **ACTION\_STEPS** (such as admission step, diagnostic step, etc), the same is not the case in the BPMN ontology where there is a single high-level concept **USER\_TASK**. The specialized **ACTION\_STEPS** described in the CP ontology are therefore mapped as instances of the concept **USER\_TASK**.

To differentiate the instances of the **USER\_TASK** class from each other, we use the **BPMN\_ELEMENT** class and its *category* data property of the BPMN ontology. We write a constraint that an instance of the **USER\_TASK** is equivalent to an instance of the

**BPMN\_ELEMENT**, and the instance of the **BPMN\_ELEMENT** has a *category* data property (string data type), and the range of this property (*category*) is the name of the action step (such as admission step, diagnostic step, etc.). However, it is not the best way, since we have a lot of instances in our mapping expressions that are not that useful, the only purpose they have is to differentiate the created instances from each other.

We extended our BPMN ontology to provide more significant mapping expressions, such that the extended BPMN ontology is very close to our CP ontology.

We copied all the **ACTION/INTERVENTION\_STEPS** with all of their properties to our BPMN ontology as the subclasses of the **USER\_TASK** class (Figure 5.13). We now map each step directly to the corresponding step and there is no need to create the instances anymore. The actual mapping is shown in the Figure 5.14.

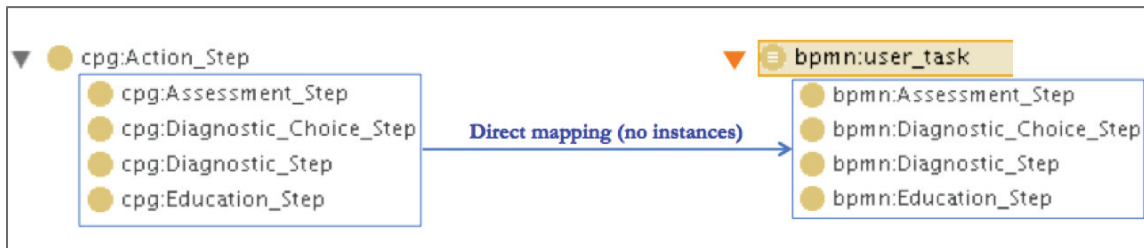


Figure 5.13 Copying the subclasses of the **ACTION\_STEPS** to the subclasses of the **USER\_TASK** class (only some of the subclasses are shown here)

```

# Admission_Step

:CC01 a :CCmapping ;
    rdfs:label "(CC01) Admission_Step – Admission_Step" ;
    rdfs:comment "
The Admission_Step class in CPG will be equivalent to the Admission_Step, subclasses of
the User_Task."

:CC01 :hasSource :CC01S .
:CC01 :hasTarget :CC01T .

:CC01S :hasClass cpg:Admission_Step .
:CC01T :hasClass bpmn:Admission_Step .

:CC01S owl:equivalentClass :CC01T .

```

Figure 5.14 The actual class-class mapping for the subclasses of the **ACTION\_STEPS** to the subclasses of the **USER\_TASK** class

In this chapter, we presented our work on ontology mapping to establish semantic interoperability between two ontologies (the CP ontology and the BPMN ontology).

The intent of our ontology mapping exercise was to establish an interoperability framework that enables the translation of clinical workflows to a standard process workflow formalism—semantic interoperability, therefore, involves the mapping of clinical workflows concepts to workflow concepts such that a clinical workflow can be represented as a process workflow.

To achieve ontology mapping, we specified the correspondences between classes, properties and instances between the candidate ontologies. We defined four steps in our ontology mapping process, which were extracting and analyzing concepts, mapping discovery, documenting and consistency checking. By processing these four steps, we



achieved a mapping ontology that includes a set of mappings between the concepts of our CP and BPMN ontologies.

In the next chapter, we execute our BPMN-based CP in the Lombardi workflow engine (developed by IBM), whereby users can view the execution of the CP and make necessary adjustments to optimize the CP.

## **CHAPTER 6      MODELING AND EXECUTION OF CP IN LOMBARDI**

In this chapter we explain our work pertaining to the modeling and execution of CP that are modeled and executed in Lombardi, a modeling tool from IBM. It is important to note that Lombardi is a tool and not a workflow language, and it is not based on the BPMN specification. It only provides a few constructs for modeling and executing business process. BPMN is a rich workflow representation formalism.

In order to represent our BPMN-based CP in terms of Lombardi constructs, we need to define a mapping ontology that establishes semantic mappings between the elements of Lombardi and BPMN. This mapping ontology represents the Lombardi constructs in terms of our BPMN ontology, and it will provide a richer specification for the Lombardi constructs.

This mapping ontology enables us to model our BPMN-based CP model by the Lombardi constructs. After modeling our BPMN-based CP model in Lombardi, we will execute our model and it will result to the CP execution.

To provide this mapping ontology, first we need to develop a Lombardi ontology to systemically describe the Lombardi constructs, and then we establish its semantic interoperability with the BPMN ontology—note that both ontologies represent workflows but offer different levels of workflow abstractions in terms of workflow components and constraints.

## 6.1 INTRODUCTION TO LOMBARDI

IBM WebSphere Lombardi v.7.1 [7,84] is one of the first BPM tools for modeling BPMN. It provides an environment to improve business process applications.

Lombardi provides design, simulation, rules definition, process execution and monitoring functions. The architecture of Lombardi is shown in Figure 6.1 [7]. The details of these components are out of the scope of this thesis. In this thesis we only use the Authoring Environment component to design and execute CP.

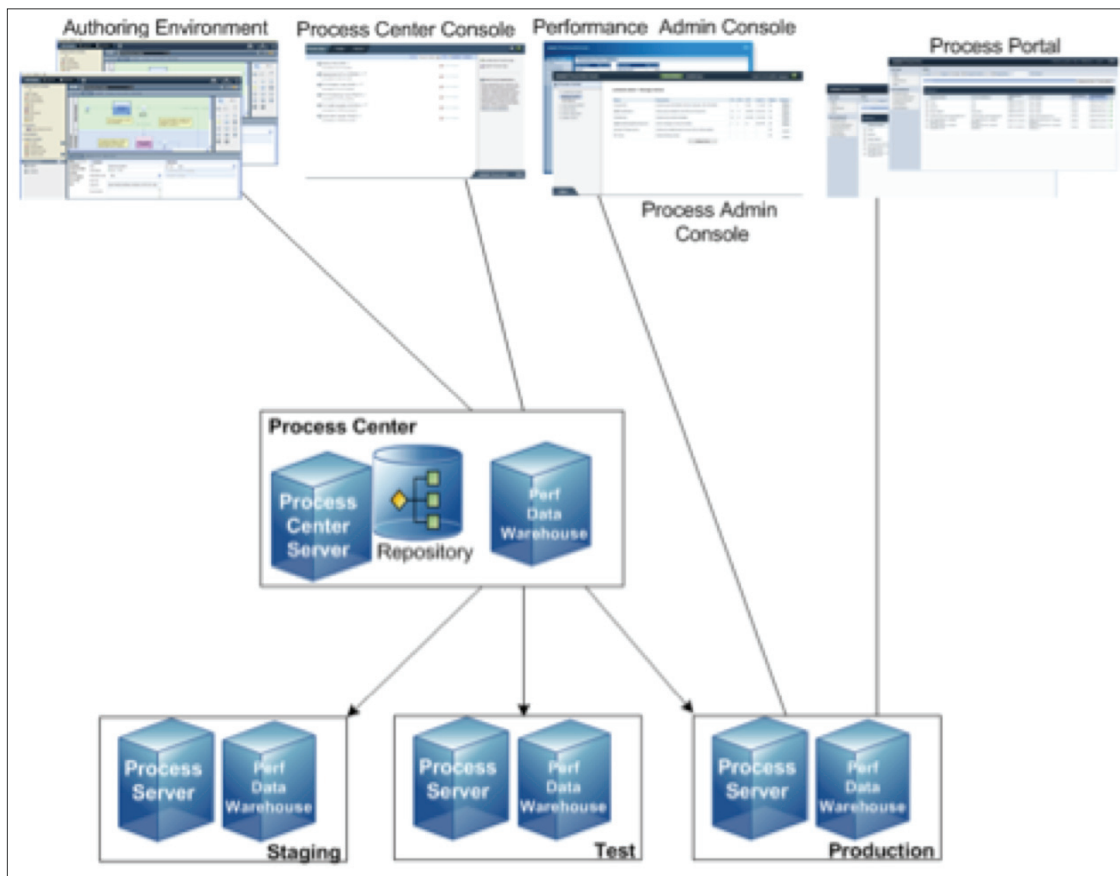


Figure 6.1 The architecture of IBM WebSphere Lombardi [7]

According to [7], the main components of Lombardi are:

**Lombardi Authoring Environment:** In the authoring environment, users can create process models.

**The Process Center:** It includes Process Center Server and Performance Data Warehouse. Users can run their process applications and store performance data for testing.

**Process Center Console:** The applications that are ready for running and testing, can be installed by the administrator in the process center console.

**Process Portal:** End users perform the assigned tasks.

**Performance Data Warehouse:** The process center server will pass the tracked data at regular intervals to the performance data warehouse. These data can be used create for the reporting purposes.

We used the Authoring Environment to design CP. The constructs that are available in this environment are shown in Figure 6.2 [7]. The BPMN ontology provides a much richer workflow representation formalism than Lombardi.



Figure 6.2 The constructs of Lombardi [7]

## 6.2 LOMBARDI ONTOLOGY

We create an ontology to semantically describe the elements of Lombardi so that they are interoperable with other workflow ontologies. We model a number of existing CP to a Lombardi based workflow by providing this ontology. We use some visualization plug-ins in Protégé such as OntoGraph [85] or Jambalaya [86] to visualize this modeling, and to make it easier to follow the steps as the Lombardi environment. We provide an example of these encodings in Chapter 7.

The Lombardi ontology is in OWL language, and it has 56 classes, 40 object properties and 25 data properties. The classes and some of the subclasses of the Lombardi ontology are shown in Figure 6.3.

To create an ontology for Lombardi, first we classified all the constructs of Lombardi. We categorized these constructs by their intended purposes and labels. We use the same structure as our BPMN ontology.

For instance, Lombardi has different events such as intermediate tracking event, start message event, end event, etc. We created an **EVENT** class in our Lombardi ontology, and as in the BPMN ontology we created 3 types of events as the subclasses of the **EVENT** class, which are **START**, **INTERMEDIATE** and **END**. We included all of the events of Lombardi as the subclasses of these classes based on their intended purposes and labels. For instance the **END\_EVENT** and the **END\_EXCEPTION\_EVENT** are under the **END\_EVENT** class. We created the **EVENT\_TYPE** class to specify the type of the event by the *has\_event\_type* property.

We captured and categorized all of the properties that are available for these elements in Lombardi, and then created object and data properties for these elements. For instance, the **MESSAGE\_INTERMEDIATE\_EVENT** has the following properties: *has\_message\_ref* (to specify the body of a message), *has\_message\_condition* (we specify the condition for the event with this property, and when it is evaluated to true the event will be activated) and *has\_intermediate\_event\_target* (to which activity it belongs).

The **TIMER\_EVENT** class is an event that can be used for scheduling or delaying an activity. It has the following properties: *has\_custom\_date* (with the range of the **EXPRESSION** class to specify the expression for the event) and *has\_intermediate\_event\_target* (to which activity it belongs). We used the same procedure for the gateways, activities, etc.

Lombardi has five different gateways, which are *conditional\_join\_or* (merge two or more paths based on a condition), *conditional\_split\_or* (split two paths based on a condition), *decision\_gateway\_xor* (choose one of the several paths based on a condition), *simple\_join\_and* (merge all paths for synchronization) and *simple\_split\_and* (branching). We included all of these gateways under the **GATEWAY** class. We also created the **GATEWAY\_TYPE** class to specify the type of the gateway by the *has\_gateway\_type* property. We created object properties for these gateways based on the existing properties in Lombardi, such as *has\_gateway\_input\_set* (to specify the input data requirement for a gateway to make a decision), *has\_gateway\_output\_set* (the data set, which will produced or passed to the next element), *has\_gateway\_lane* (to specify in which lane, the gateway is located) and *has\_out\_sequence\_flow\_ref* (the outgoing sequence flow from the gateway, and as in the BPMN ontology it has three properties, which are *sequence\_source\_ref*, *sequence\_target\_ref* and *sequence\_flow\_condition*).

There are different types of activities in Lombardi. We included all of these activities under the **ACTIVITY** class. However, based on their labels, we created subclasses for the **ACTIVITY** class. For instance, **HUMAN\_SERVICE**, **RULE\_SERVICE** and **WEB\_SERVICE** are all under the **LOMBARDI\_SERVICE** class. We explain these services later.



Figure 6.3 Lombardi ontology



### **6.3 BPMN-LOMBARDI ONTOLOGY MAPPING**

Lombardi does not provide the same level of workflow expressiveness and abstraction as the BPMN specification. BPMN is a much richer workflow representation formalism. We established a semantic interoperability between the BPMN and Lombardi ontologies.

The ontology mapping specifies the correspondences between classes, properties and instances between the candidate ontologies.

The procedure for creating the mapping ontology is similar to the ontology mapping presented in Chapter 5. Ontology mapping is represented through four different OWL constructs that document the mapping expressions. The constructs are: 46 Class-to-Class mapping expressions, 57 Property-to-Property (object and data properties) mapping expressions, 6 Class-to-Property mapping expressions and 2 Property-to-Instance mapping expressions. The relationships between the classes of the BPMN and Lombardi ontologies are listed in Table 7.1.

THE MAPPING BETWEEN THE CLASSES OF BPMN AND LOMBARDI ONTOLOGIES	
BPMN	Lombardi
User Task	Activity Lombardi Service Human Service
Service Task	Activity Lombardi Service Web Service
Script Task	Activity Java Script
Sub Process	Activity Nested Process
Standard Loop Activity	Activity Simple Loop
Multi Instance Loop Activity	Activity Multi Instance Loop
Exclusive Gateway	Decision Gateway (XOR) *
Inclusive Gateway	Conditional Split (OR) * Conditional Join (OR)
Parallel Gateway	Simple Split (AND) Simple Join (AND)
Start Event	Start Event
Start Event <i>has_start_event_trigger</i> Message Event Detail	Start Message Event
Message Intermediate Event	Intermediate Message Event
Error Intermediate Event	Intermediate Exception Event
Timer Intermediate Event	Timer Event
Cancel Intermediate Event	Terminate Event
End Event	End Event

THE MAPPING BETWEEN THE CLASSES OF BPMN AND LOMBARDI ONTOLOGIES	
BPMN	Lombardi
End Event <i>has_end_event_trigger</i> Error Event Detail	End Exception Event
Business Process Diagram	Business Process Definition
Assignment	Assignment
Condition	Expression
Expression	Expression
Time Date Expression	Expression
Graphical Elements	Graphical Elements
Lane	Lane
Message	Message
Annotation	Notes
Participant	Participant
Pool	Pool
Process	Process
Property	Property
Role	Role
Sequence Flow	Sequence Flow
Input Set <i>has_input_set_property_input</i> Property	Input Set
Output Set <i>has_output_set_property_output</i> Property	Output Set

Table 6.1 The mappings between the classes of BPMN and Lombardi ontologies

In the Lombardi ontology, we use Lanes to differentiate the subclasses of the **USER\_TASK** class (**ADMISSION\_STEP**, **ASSESSMENT\_STEP**, etc.) from each other.

Lanes represent departments within an organization. Lanes separate the events and activities of each lane (e.g. department, section, step) with the other lanes during the process execution. In addition a person or a human resource can be assigned to a lane to hold or to be responsible for all the activities within the lane during the execution [7].

The mapping between the subclasses of the **USER\_TASK** class and the **HUMAN\_SERVICE** class is done in the following way:

- Each subclass of the **USER\_TASK** class can be mapped to the **HUMAN\_SERVICE** class of the Lombardi ontology.
- To differentiate these subclasses from each other, we create an instance for the **HUMAN\_SERVICE** and **LANE** classes.
- Each instance of the **HUMAN\_SERVICE** class has an *activity\_lane* property, and its range is the instance of the **LANE** class.
- Each instance of the **LANE** class has a *lane\_name* data property, and its value is the name of the subclass of the **USER\_TASK** class. By these expressions, we indicate that each step has a lane and the name of the lane is the same as the name of the step (Figure 6.4). The actual mapping is shown in Figure 6.5.

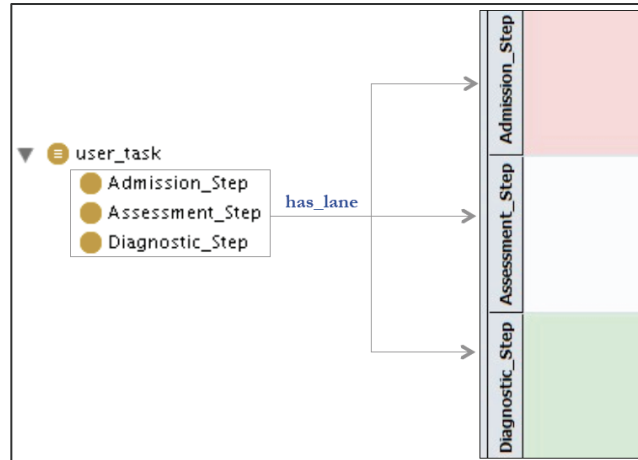


Figure 6.4 Creating separate lanes for each of the subclasses of the **USER\_TASK** class

```

#Admission_Step

:CC01a :PVmapping ;
      rdfs:label "(CC01) admission_step-human_service" .
      rdfs:comment "
The Admission_Step class in BPMN will be equivalent to the instance of the
Human_Service class with an admission lane property."

:HS_Adm    a :CreateInstance .
:AdmLane   a :CreateInstance .

:HS_Adm    :hasClass    lombardi:human_service .
:AdmLane   :hasClass    lombardi:Lane .

:HS_Adm    lombardi:has_activity_lane    :AdmLane .
:AdmLane   lombardi:has_lane_name    "Admission_Step" .

:CC01      :hasSource    :CC1S .
:CC01      :hasTarget    :CC1T .

:CC1S      :hasClass    bpmn:Admission_Step .
:CC1T      :hasClass    :HS_Adm .

:CC1S      owl:equivalentClass    :CC1T .

```

Figure 6.5 The actual Class-Class mapping for the **ADMISSION\_STEP** class to the instance of the **HUMAN\_SERVICE** class

## 6.4 MODELING CP IN LOMBARDI

We modeled 3 different CP in Lombardi. After modeling each clinical pathway, we can execute the model by going through from the start event to the end event. The diagram for the PMRT CP is shown in Figure 6.6 [56], and Figure 6.7 illustrates its modeling in the Lombardi modeling environment.

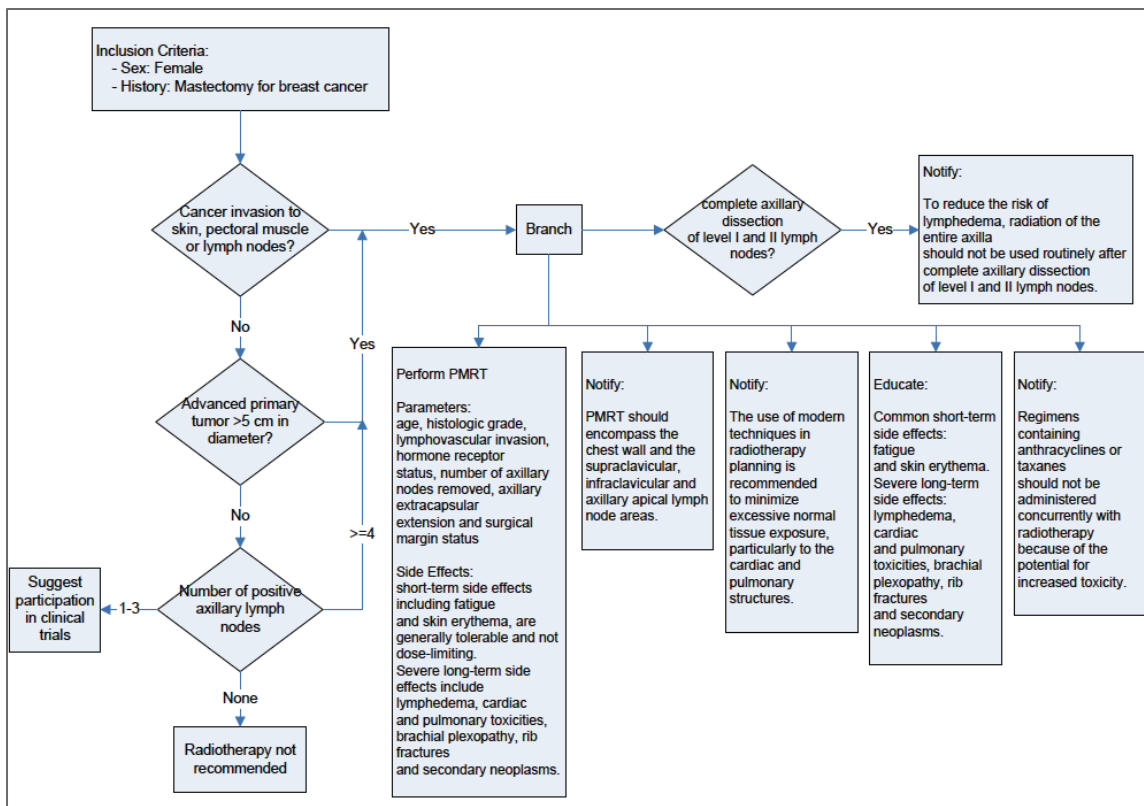


Figure 6.6 PMRT CP [56]

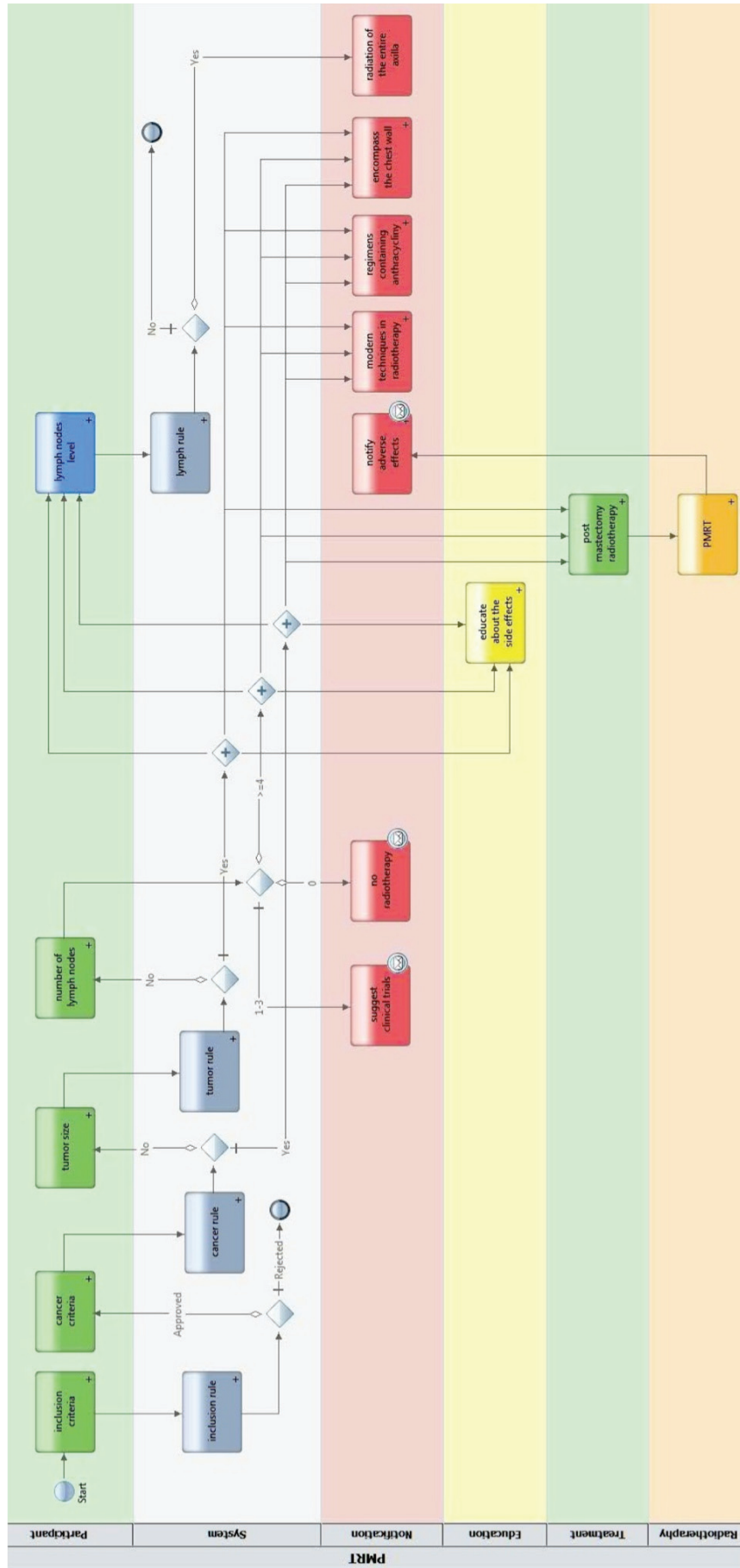


Figure 6.7 PMRT CP in Lombardi

### **Lombardi constructs:**

- Lanes [7] represent departments within an organization. They separate the events and activities of each lane (e.g. department, section, step) with the other lanes during the process execution. In addition a person or a human resource can be assigned to a lane to be responsible for all the activities within the lane during the execution. Lombardi by default has a system lane. The system lane contains all the activities that have to be executed by the Lombardi Engine. It will execute all the created services (such as the rule service or web service) during the execution. In addition to the system lane, we also added the Participant lane. We include all the human services (to create interactive services) in this lane. The human services include coaches (user interface) to interact with the user. For instance, the **INCLUSION\_CRITERIA** human service in the Participant lane has an interface that allows the participant (e.g. physician) to specify the inclusion or exclusion criteria of a guideline.
- Lombardi has three types of variables [7]:
  - Private: These variables are local variables and can be only used within the current process.
  - Input: These variables can be passed into the current process. For instance, we pass these variables to another activity or a gateway.
  - Output: these variables can be passed out from the current process to a parent process.



- Human service [7] is used when you want to create an interaction service. It contains a coach component that creates an interface. It provides buttons, forms, fields, etc.
- Rule service [7] is used when you want to specify a condition for a specific process. A rule service performs the JavaScript expression based on the input data, and it passes output variables to the next task.
- Lombardi has five different gateways [7] namely Simple Split (the process follows all available paths), Simple Join (merge several paths into a single path after the runtime execution of each individual path), Conditional Split (one or more path can be followed based on conditions that you specify), Conditional Join (merge several paths into a single path based on a condition) and Decision Gateway (only one of the several paths can be followed, depending on a condition).

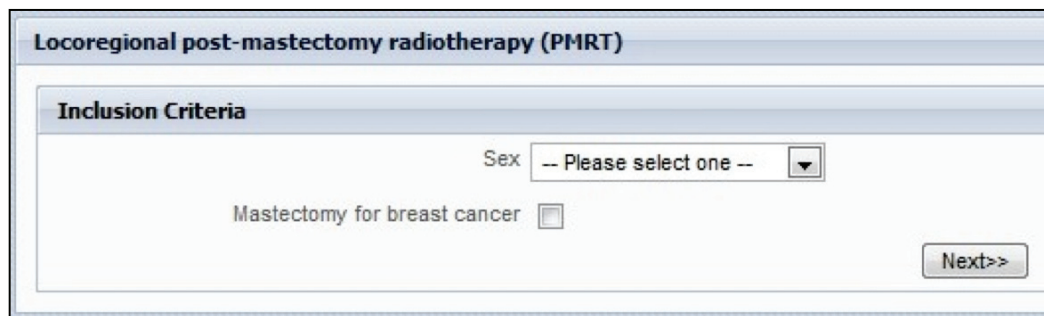
In each decision/conditional gateway [7] you can specify a condition to determine, which path has to be followed. The gateway gets an input variable from the previous task and based on the value of the variable makes a decision. In the case of simple split (parallel gateway) there is no condition and all paths are enabled. They can be executed in parallel or sequentially.

- A message intermediate event can be attached to an activity in order to send a message to an external participant. The settings for sending a message have to be defined. In addition to the message intermediate event, a timer intermediate event can be attached to an activity to schedule an activity.

## Modeling in Lombardi:

In Lombardi, in addition to the system and provider lanes, we create lanes to differentiate each Action/Intervention\_Step from each other. Each Step has a different lane; for instance, in Figure 6.7 the pink lane represents the Notification\_Step and the yellow lane represents the Education\_Step.

The first activity after the start event in the participant lane (the first lane) is the Inclusion/Exclusion criteria task. It has a graphical interface that enables the user to specify the symptoms of a patient (Figure 6.8). After submitting the information, it passes the data variables to the next task, which is a task with a rule service.



The screenshot shows a web-based form titled "Locoregional post-mastectomy radiotherapy (PMRT)". Below the title is a sub-section titled "Inclusion Criteria". Inside this section, there is a label "Sex" followed by a dropdown menu containing the text "-- Please select one --". Below that is a label "Mastectomy for breast cancer" followed by an unchecked checkbox. In the bottom right corner of the form, there is a button labeled "Next>>".

Figure 6.8 Inclusion Criteria interface, designed by a coach component

The rule service (Figure 6.9) performs our specified JavaScript expression based on the input data from the provider. The result is a Boolean variable that will be passed to a gateway to check whether the pathway should be continued or not.

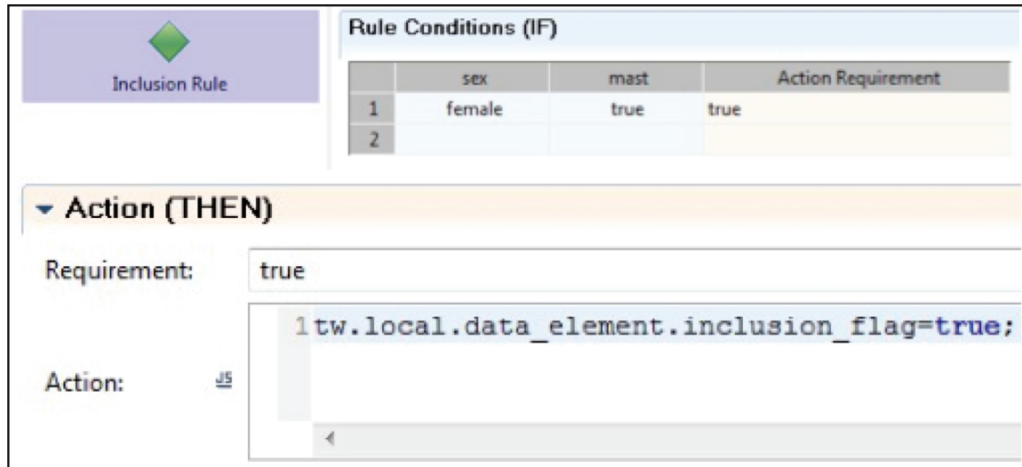


Figure 6.9 Rule service component

We use gateways to model the decision options. For instance, the number of lymph nodes in the PMRT CP is modeled by a gateway. The gateway gets an input variable from the previous task, which indicates the number of lymph nodes. The gateway determines which path has to be followed based on the value of the input variable and our defined decision options in the gateway [Figure 6.10].



Figure 6.10 Decision options in a gateway

We modeled scheduling and messaging by attaching the timer intermediate event and message intermediate event to an activity respectively. We may specify the settings for sending a message, or in the case of the timer intermediate event we can delay an activity before performing the next activity (Figure 6.11).

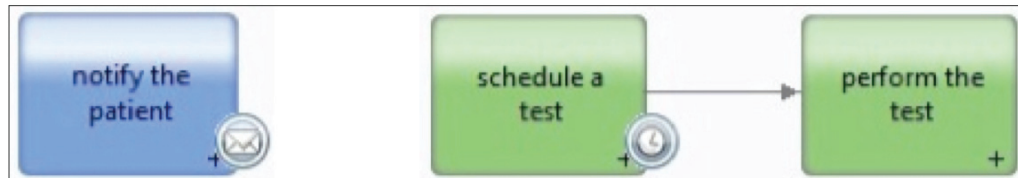


Figure 6.11 Attaching a message or timer intermediate event to an activity

We execute our pathway after finishing the modeling. It goes step by step and executes each task. The provider has an ability to monitor the running processes and tasks. The provider can terminate a process at any time.

In this chapter, we have explained how to model and execute CP in Lombardi, a modeling tool from IBM. BPMN and Lombardi offer different levels of workflow abstractions in terms of workflow components and constraints. Lombardi does not provide the same level of workflow expressiveness and abstraction as the BPMN specification. BPMN is a much richer workflow representation formalism. In order to represent CP in Lombardi, first we developed a Lombardi ontology to formalize the structure of the Lombardi constructs and then we established its semantic interoperability with the BPMN ontology. We provided mapping expressions between the BPMN and Lombardi ontologies to express the relations between the Lombardi constructs and BPMN.

## CHAPTER 7 EVALUATION

To evaluate our CPG-BPMN and BPMN-Lombardi mapping expressions, we encoded six CP in our ontologies. These CP are already encoded in the CP ontology from Shapoor [56]. We provided an overview of the CP ontology in Chapter 4. Each CP is modeled as an instance of the **CLINICAL\_GUIDELINE** class in the ontology and it has a goal.

We model a number of existing CP to a BPMN and Lombardi based workflows. We make sure that the existing CP can be modeled to a BPMN and Lombardi based workflows based on our mapping expressions. It can be based on the ability to capture the control flow patterns and conditions.

After encoding the CP to our BPMN and Lombardi ontologies, first we checked the consistency by the Pellet Reasoner to make sure that there is no inconsistency in the classes and the domain and range of our properties, and then the resulting encoding of these CP to our ontologies are verified by their clinical pathway diagrams and the existing descriptions in their guidelines.

In addition, we modeled these CP in the Lombardi environment, which allows us to compare step by step these encodings with the steps of our model in Lombardi. We use some visualization plug-ins in Protégé such as OntoGraph [85] to visualize this modeling, and to make it easier to follow the steps of the Lombardi environment.

## 7.1 THE ENCODED CP IN THE CP ONTOLOGY

We considered six already encoded CP in the CP ontology [87]:

- Diagnosis and treatment of Acute Otitis Media (AOM), aiming to increase the accuracy of the diagnosis of AOM, and optimizing management of AOM.
- Locoregional Post Mastectomy Radiotherapy (PMRT), aiming to improve locoregional control, which increases disease-free survival and overall survival.
- Treatment of Cataract in Adults (CAT), aiming to resolve cataract disease.
- Protocol for Macroscopic and Microscopic Urinalysis (UA) aiming to avoid unnecessary testing in routine cases.
- Dysphagia Care in MND (DCM)
- Treatment of Gallstones in Adults (GALLA)

Each of these CP was encoded separately as an instantiation of the CP ontology (Figure 7.1) and with the variety of complexity, GALLA being the simplest and PMRT being the most complex one. The complexity measure is based on the existing control patterns in the CP. The number of decisions, loops, scheduling and notification events, demonstrate the complexity measure.

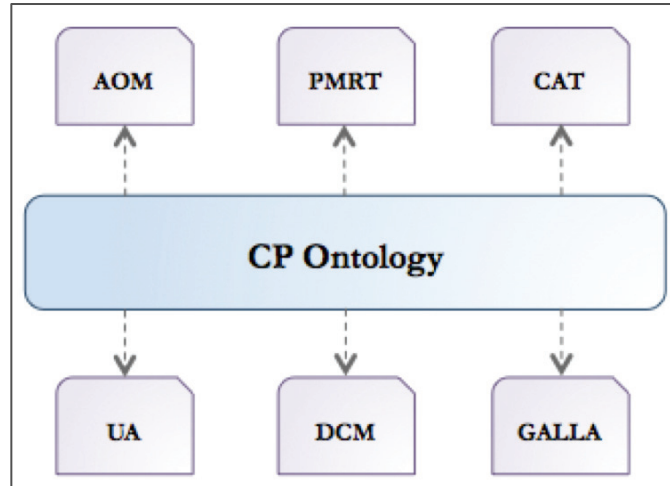


Figure 7.1 The instantiations of the CP ontology

## 7.2 ENCODING CP IN BPMN AND LOMBARDI ONTOLOGIES

We encoded the mentioned CP in the BPMN ontologies. CPG is modeled as an instance of **CLINICAL\_GUIDELINE** in the CP ontology. To relate this instance to our BPMN ontology, first we created an instance of the **BUSINESS\_PROCESS\_DIAGRAM** class in the BPMN ontology, and then an instance of the **CLINICALPATHWAY** class and the *has\_business\_process\_diagram* property.

Each **BUSINESS\_PROCESS\_DIAGRAM** has a **POOL**, and each **POOL** has a **PROCESS**. Each **PROCESS** has a number of **GRAPHICAL\_ELEMENTS**, which include the **START\_EVENT** and the **END\_EVENT**. The process starts from the **START\_EVENT**, with a *trigger* property to indicate the inclusion/exclusion criteria and a *connecting\_object* property, with the range of the **SEQUENCE\_FLOW** class (Figure 7.2).

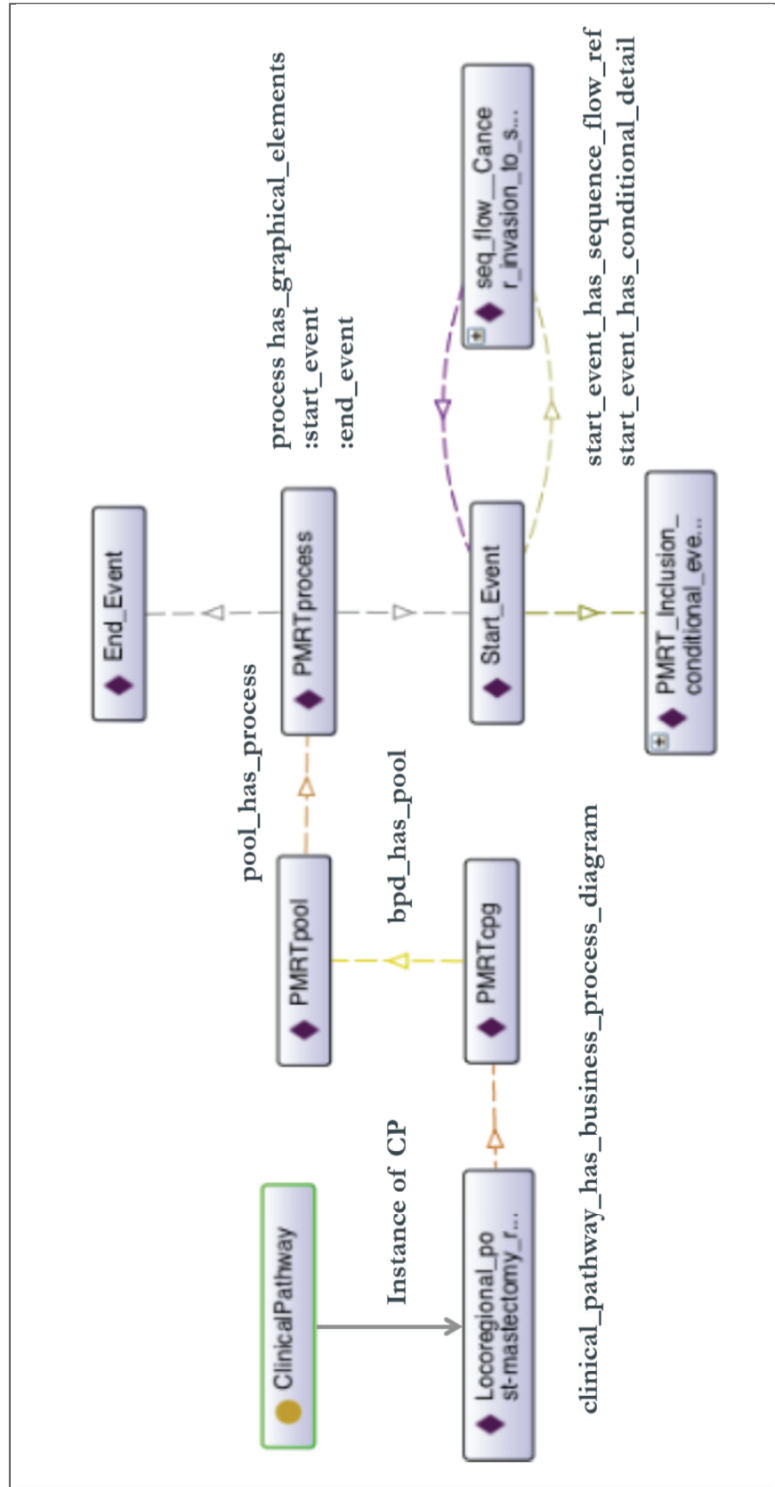


Figure 7.2 The initial part of PMRT encoding in the BPMN ontology



We encoded the instantiations of the CP ontology in our Lombardi ontology, which does not provide the same level of workflow expressiveness and abstraction as the BPMN specification.

As in the BPMN encoding, we created an instance of the **CLINICAL\_PATHWAY** class and the *has\_business\_process\_diagram* property. A main difference is that in the Lombardi ontology, lanes are used to differentiate the sub-classes of the **USER\_TASK** class. Each activity has an *activity\_lane* property that determines to which lane the activity belongs.

In addition in Lombardi there is no *trigger* property, therefore we cannot model the inclusion/exclusion criteria in the **START\_EVENT**. In order to include these conditions, the first task after the **START\_EVENT** is a task (**RULE\_SERVICE**) that includes inclusion/exclusion criteria and then goes to the next step. The initial part of AOM encoding in the Lombardi ontology is shown in Figure 7.3.

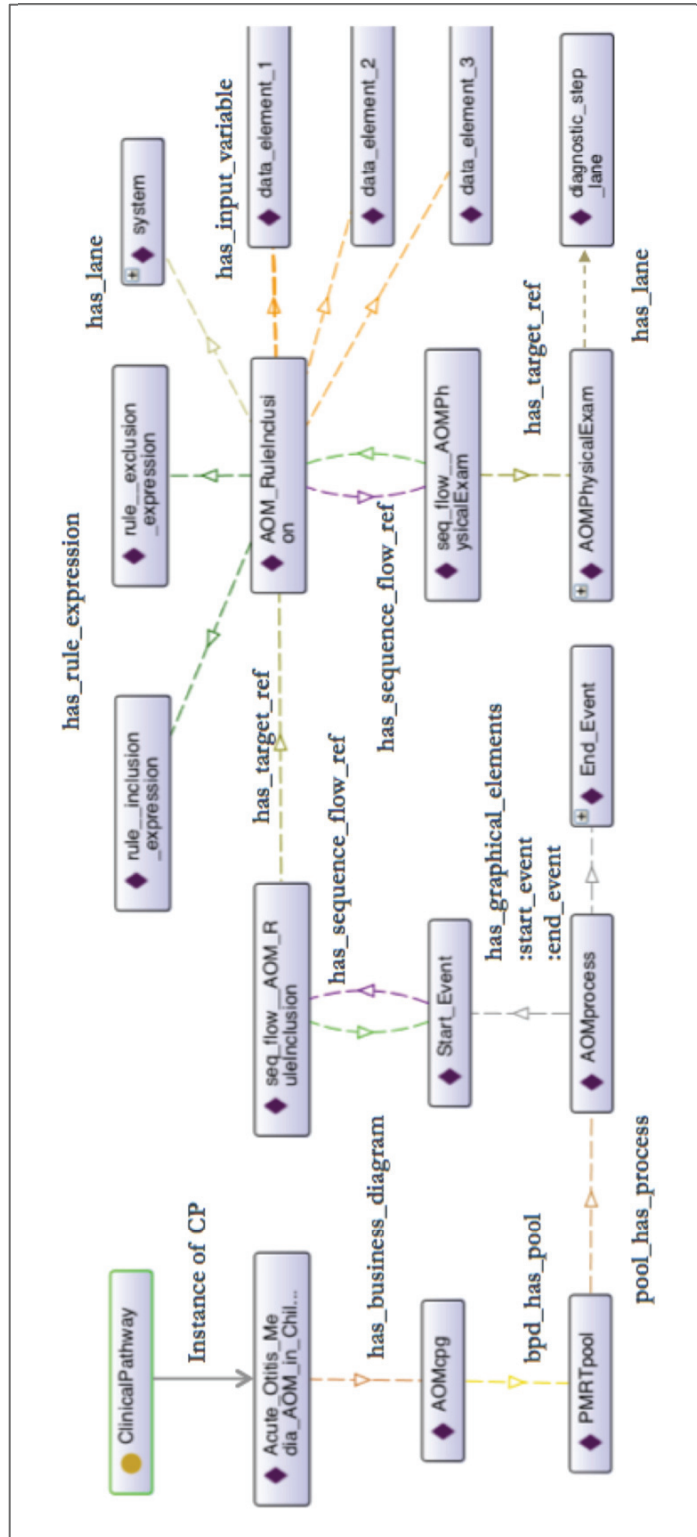


Figure 7.3 The initial part of AOM encoding in the Lombardi ontology

In this chapter, we evaluated our CP-BPMN and BPMN-Lombardi mapping expressions by encoding six CP in our BPMN and Lombardi ontologies. We verified that the existing CP can be modeled to a BPMN and Lombardi based workflows based on our mapping expressions by capturing the control flow patterns and conditions. We verified the resulting encoding of these CP to our ontologies by their clinical pathway diagrams and the existing descriptions in their guidelines.

After encoding CP in our ontologies, we used the OntoGraph visualization plug-in in Protégé to visualize our encoding results as well. It gives us the BPMN/Lombardi-based CP model in a visual format. It provides users a better understanding of CP, which are represented as the BPMN language and Lombardi constructs. Our objective is to compare our BPMN/Lombardi-based CP model, step by step with the CP diagrams to make sure that: (a) we have represented all of the existing steps in a CP; (b) we have captured the flow of control and decision steps in a CP. Our evaluation is based on the completeness of this objective.

## CHAPTER 8 CONCLUSION

We have developed a semantic interoperability framework whereby clinical processes/pathways can be conveniently mapped to business process notations thus enabling CP to be executed and simulated for adjusting various cost functions.

Our semantic interoperability framework allows healthcare professionals to model a CP using modeling constructs that they are familiar with, and then we transform their CP model to a business process model.

The use of ontologies, at both representation and mapping levels, allows for the semantic description of concepts and their relations, with provisions for semantic classification of healthcare concepts to ensure the right level of conceptual granularity in the representation scheme.

We executed our BPMN-based CP in the Lombardi workflow engine, whereby users can view the execution of the CP and make necessary adjustments to optimize the CP. However, Lombardi does not provide the same level of workflow expressiveness and abstraction as the BPMN specification. BPMN is a much richer workflow representation formalism. Therefore, we established a semantic interoperability between the BPMN and Lombardi ontologies. The mapping ontology between the BPMN and Lombardi ontologies provides a richer specification for the Lombardi constructs.

To evaluate our semantic interoperability framework, we modeled a number of existing CP to a BPMN based workflow—the CP are rendered in a visual format and can be interactively executed to study and optimize the CP. The semantic description of the CP tasks ensures that the transformation of a CP to a BPMN workflow maintains the

clinical pragmatics of the CP and that it can be actively connected with health data from HIS. The graphical notation of the CP enables rapid user feedback and adjustments to optimize performance metrics. The interactive execution of designed CP allows determining process bottlenecks, costs, resource requirements and decision options.

Our contributions to this study are:

- We developed a semantic interoperability (mapping ontology) framework between the CP ontology and the BPMN ontology. In our framework, we defined a mapping expression language that allows the alignment of relations between two ontologies—the relations are represented in terms of *mapping expressions*. The mapping expressions are represented in a *mapping ontology*—the mapping ontology establishes semantic mappings between the CP and BPMN ontologies, and ensures that a clinical process defined in the CP ontology is mapped to a standard BPMN workflow element.
- We extended our BPMN ontology to provide more clinically salient mapping expressions, such that the extended BPMN ontology is very close to our CP ontology.
- We encoded six clinical pathways in the BPMN ontology to evaluate the mapping expressions of our mapping ontology.
- We executed our BPMN-based CP in the Lombardi workflow engine, whereby users can view the execution of the CP and make necessary adjustments to optimize the CP.
- Since Lombardi provides fewer constructs than BPMN ontology, first we developed an ontology for Lombardi to formalize the structure of the Lombardi

constructs, and then we established a mapping ontology between the BPMN and Lombardi ontologies in order to provide a richer specification of concepts for the Lombardi constructs.

## 8.1 ENHANCING THE CP ONTOLOGY

We propose some extensions to enhance our CP ontology in order to capture the more complex workflow structure of the clinical pathways. We propose these extensions by studying the constructs of the BPMN ontology. The following constructs are shown in Figure 8.1 as well [24]:

**MULTI\_INSTANCE\_LOOP:** In addition to the **SIMPLE\_LOOP** class we can add the **MULTI\_INSTANCE\_LOOP**. The instances of this loop are performed in parallel or sequentially. It has a *numeric\_expression* property that determines the number of times that the activity has to be repeated, and it is evaluated only once before starting the activity.

**GROUP:** We may group a set of the guideline steps, since they share the same category.

**INPUT\_SET AND OUTPUT\_SET:** We may define a set of data requirements (variables) for the input or output of a task.

**REFERENCE, SERVICE, SEND and RECEIVE tasks:** We can reference another task that has already been defined. A **REFERENCE** task shares the same behavior of another task and it shares all the attributes of that task. An action step of the guideline can be a **SERVICE** task that provides a service (e.g. Web service). A simple task could be also a **SEND** or a **RECEIVE** task. The **SEND** task sends a message to an external user and then it

is completed, and the **RECEIVE** task waits for receiving a message from an external participant and then it is completed.

**PRE\_CONDITION and POST\_CONDITION:** Tasks may have pre-conditions or post-conditions that should be satisfied before executing a task or proceeding to the next step.

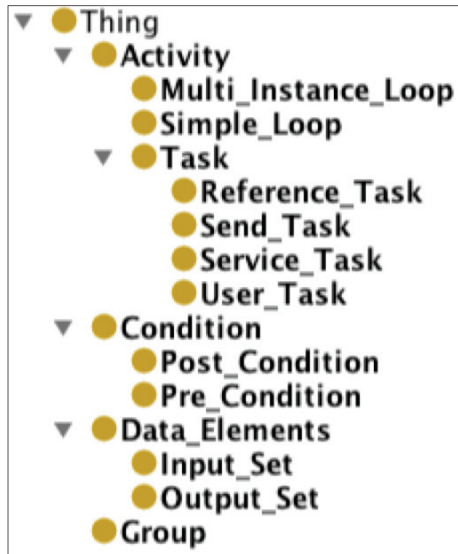


Figure 8.1 Class hierarchy of the new constructs for the domain ontology

## 8.2 LIMITATIONS AND FUTURE WORK

Apart from the above-mentioned constructs that can be implemented in more detail, the other major limitations of this study, which can be our future work, are as follows:

- We may develop an execution engine for the BPMN ontology, based on our mapping ontology.
- In the BPMN ontology, there is no construct to define the outcome of a clinical guideline, or what happens next in the case of achieving or not achieving the outcome.

- We may provide additional constructs to capture complex temporal aspects of tasks. The current BPMN ontology only provides a single time data expression class for a task, and it doesn't provide any construct for modeling temporal aspects, such as start date/time, duration and end date/time.
- Finally, we may add constructs or provide another ontology to capture information about a situation, which involves the real time processing of information from an evolving situation in order to understand what is happening, and to provide a high level reasoning support [91].



## BIBLIOGRAPHY

- [1] Daniyal, A., Sibte, S. & Abidi, R. Semantic Web-based modeling of Clinical Pathways using the UML Activity Diagrams and OWL-S. *Work* 88-99 (2010).
- [2] De Bleser, L. et al. Defining pathways. *Journal of Nursing Management* 14, 553-563 (2006).
- [3] Leong, T.Y., Kaiser, K. & Miksch, S. Free and open source enabling technologies for patient-centric, guideline-based clinical decision support: a survey. *Yearbook of medical informatics* 46, 74-86 (2007).
- [4] Cocos C., MacCaulla, W., Kramer B. and Latzel M., An Ontology-Based Approach to Decision Support for Healthcare Workflows. In proceedings of 2nd Workshop of Ontologies in Biomedicine and Life Sciences, IMISE-REPORT Nr. 2/2010 (September 2010), pp. 13-17.
- [5] Aalst, W.M.P.V.D., Barros, A.P., Ter Hofstede, A.H.M. & Kiepuszewski, B. Advanced Workflow Patterns. *Cooperative Information Systems* 1901, 18–29 (2000).
- [6] Cabral, L., Norton, B. & Domingue, J. The business process modelling ontology. Management Workshop: Semantic Business Process Management (SBPM 2009) at European Semantic Web Conference - ESWC 2009, Greece, ACM International Conference Proceeding Series.
- [7] WebSphere Lombardi Edition 7.2.0 Authoring Environment User Guide, IBM, 2010. Available: <http://www-01.ibm.com/software/integration/lombardi-edition/library/documentation>
- [8] Fox, J., Johns, N. and Rahmzadeh, A. Disseminating medical knowledge: the PROforma approach. *Artificial Intelligence in Medicine* 14, 157-181 (1998).
- [9] Sutton, D.R. & Fox, J. The syntax and semantics of the PROforma guideline modeling language. *Journal of the American Medical Informatics Association* 10, 433-443 (2003).
- [10] Peleg, M. et al. Comparing Computer-interpretable Guideline Models: A Case-study Approach. *Journal of the American Medical Informatics Association* 10, 52-68 (2003).
- [11] De Clercq, P.A., Blom, J.A., Hasman, A. & Korsten, H.H. GASTON: an architecture for the acquisition and execution of clinical guideline-application tasks. *Medical informatics and the Internet in medicine* 25, 247-263 (2007).

- [12] De Clercq, P.A., Hasman, A., Blom, J.A. & Korsten, H.H. Design and implementation of a framework to support the development of clinical guidelines. *International Journal of Medical Informatics* 64, 285-318 (2001).
- [13] Tu, S.W. et al. The SAGE Guideline Model: Achievements and Overview. *Journal of the American Medical Informatics Association* 14, 589-598 (2007).
- [14] Yan, Y.m Zhibin, J., Dong. Y., Gang, D., A semantics-based clinical pathway workflow and variance management framework. *Service Operations and Logistics, and Informatics, 2008. IEEE/SOLI 2008. IEEE International Conference on*, vol.1, no., pp.758-763, 12-15 Oct. 2008
- [15] ASBRU. Internet: [http://www.openclinical.org/gmm\\_asbru.html](http://www.openclinical.org/gmm_asbru.html) [June, 1, 2011].
- [16] The Asbru Language. Internet: [http://www.asgaard.tuwien.ac.at/plan\\_representation/asbru\\_doc.html](http://www.asgaard.tuwien.ac.at/plan_representation/asbru_doc.html) [June, 1, 2011].
- [17] Miksch, S., Hammermuller, K., Asbru, a Plan-Representing Language Modelling Time-oriented, Skeletal Plans in Sport. Available: [http://www.ifs.tuwien.ac.at/~silvia/pub/publications/mik\\_css99.pdf](http://www.ifs.tuwien.ac.at/~silvia/pub/publications/mik_css99.pdf).
- [18] Bosse, T., An Interpreter for Clinical Guidelines in Asbru. Master thesis, Vrije Universiteit Amsterdam, 2001.
- [19] Mendling, J. Detection and Prediction of Errors in EPC Business Process Models. *Information Systems Journal* 1-491 (2007).
- [20] Weske, M. *Business Process Management: Concepts, Languages, Architectures*. Process Management 54, 398 (Springer-Verlag New York Inc: 2007).
- [21] Business Process with BPEL4WS: Understanding BPEL4WS. Internet: <http://www.ibm.com/developerworks/library/ws-bpelcoll> [June, 3, 2011].
- [22] OWL Web Ontology Language. Internet: <http://www.w3.org/TR/owl-guide> [June, 5, 2011].
- [23] Event-driven process chain. Internet: [http://en.wikipedia.org/wiki/Event-driven\\_process\\_chain](http://en.wikipedia.org/wiki/Event-driven_process_chain) [June, 5, 2011].
- [24] Omg, T., Final, O.M.G., Specification, A., Recommendation, T.F.T.F. & Catalog, O.M.G.S. *Business Process Modeling Notation Specification*. Management 308 (2006). Available: [http://www.omg.org/bpmn/Documents/OMG\\_Final\\_Adopted\\_BPMN\\_1-0\\_Spec\\_06-02-01.pdf](http://www.omg.org/bpmn/Documents/OMG_Final_Adopted_BPMN_1-0_Spec_06-02-01.pdf).

- [25] Dumas, M. & Ter Hofstede, A.H.M. UML Activity Diagrams as a Workflow Specification Language. Intl Conf Unified Modeling Language UML 49, A11 (2001).
- [26] Arkin, A. & Pryor, M., Business Process Modeling Language. Intellectual Property 43, 1-67 (2002).
- [27] Markovic, I., Semantic Process Modeling. Germany, 2009.
- [28] Vanderaalst, W. & Ter Hofstede, A.H.M. YAWL: yet another workflow language. Information Systems Journal 30, 245-275 (2005).
- [29] Petri Nets. Internet: <http://www.informatik.uni-hamburg.de/TGI/PetriNets>. [June, 8, 2011].
- [30] Architecture of Integrated Information Systems. Internet: [http://en.wikipedia.org/wiki/Architecture\\_of\\_Integrated\\_Information\\_Systems](http://en.wikipedia.org/wiki/Architecture_of_Integrated_Information_Systems) [June, 8, 2011].
- [31] Tsai, A., Wang J., Tepfenhart, W., Rosea, D., EPC Workflow Model to WIFA Model Conversion, Systems, Man and Cybernetics, 2006. SMC '06. IEEE International Conference on , vol.4, no., pp.2758-2763, 8-11 Oct. 2006.
- [32] BPMN vs. EPC. Internet: <http://www.ariscommunity.com/users/ivo/2011-04-11-bpmn-vs-epc-revisited-part-1> [June, 8, 2011].
- [33] Ad, U.M.L. A Notation Evaluation of BPMN and UML Activity Diagrams. Information Systems Journal (2006).at [http://www.soberit.hut.fi/T-86/T-86.5161/2006/UML\\_PBMN\\_Final\\_Presentation.pdf](http://www.soberit.hut.fi/T-86/T-86.5161/2006/UML_PBMN_Final_Presentation.pdf).
- [34] Owen, M. & Raj, J. BPMN and Business Process Management. Business (2003).
- [35] Martin, V. & Dustdar, S. A View Based Analysis of Workflow Modeling Languages. 14th Euromicro International Conference on Parallel Distributed and NetworkBased Processing PDP06 276-290 (2006).
- [36] Recker, J. & Mendling, J. On the Translation between BPMN and BPEL: Conceptual Mismatch between Process Modeling Languages. CAiSE 2006 Workshop Proceedings Eleventh International Workshop on Exploring Modeling Methods in Systems Analysis and Design EMMSAD 2006 521-532 (2006).
- [37] Business Process Execution Language. Internet: [http://en.wikipedia.org/wiki/Business\\_Process\\_Execution\\_Language](http://en.wikipedia.org/wiki/Business_Process_Execution_Language) [June, 10, 2011].

- [38] Which is Simpler: BPMN or BPEL. Internet: <http://www.activevos.com/blog/bpel/bpmn-or-bpel-which-is-simpler/2009/11/19> [June, 10, 2011].
- [39] Web Service Description Language. Internet: <http://www.w3.org/TR/wsdl> [June, 10, 2011].
- [40] Abramowicz, W., Filipowska, A., Kaczmarek, M. & Kaczmarek, T. Semantically enhanced Business Process Modelling Notation. Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management SBPM 2007 251, 88-91 (2007).
- [41] Number, O.M.G.D. Business Process Model and Notation ( BPMN ). Business 538 (2011).at <http://www.omg.org/spec/BPMN/2.0/PDF>.
- [42] Seitz, C., Patterns for Semantic Business Process Modeling. Augsburg, 2008.
- [43] BPMN Overview. Internet: <http://www.bpmnforum.com> [June, 20, 2011].
- [44] Gruber, T.R. A Translation Approach to Portable Ontology Specifications by A Translation Approach to Portable Ontology Specifications. Knowledge Creation Diffusion Utilization 5, 199-220 (1993).
- [45] Stevens, R., Goble, C.A. & Bechhofer, S. Ontology-based knowledge representation for bioinformatics. Briefings in Bioinformatics 1, 398-414 (2000).
- [46] W3C. Internet: <http://www.w3.org> [June, 20, 2011].
- [47] Heflin, J. & Hendler, J. A Portrait of the Semantic Web in Action. IEEE Intelligent Systems 16, 54-59 (2001).
- [48] XML. Internet: <http://www.w3.org/TR/2000/REC-xml-20001006> [June, 20, 2011].
- [49] HTML. Internet: <http://en.wikipedia.org/wiki/HTML> [June, 20, 2011].
- [50] Gasevic, D., Djuric, D. & Devedzic, V. Model Driven Architecture and Ontology Development. Educational Technology 315 (Springer-Verlag: 2006).
- [51] Su, X. & Gulla, J.A. Semantic Enrichment for Ontology Mapping. Natural Language Processing and Information Systems 217-228 (2004).
- [52] RDF. Internet: <http://www.w3.org/TR/rdf-schema> [June, 20, 2011].
- [53] OWL Guide. Internet: <http://www.w3.org/TR/2004/REC-owl-guide-20040210> [June, 20, 2011].

- [54] Pellet. Internet: <http://clarkparsia.com/pellet> [June, 20, 2011].
- [55] FaCT++. Internet: <http://owl.man.ac.uk/factplusplus> [June, 20, 2011].
- [56] Shayegani, S., Towards Computerization And Merging Of Clinical Practice Guidelines, Halifax (2007).
- [57] Di Francescomarino, C., Ghidini, C., Rospocher, M., Serafini, L. & Tonella, P. Semantically-aided business process modeling. The Semantic WebISWC 2009 5823, 114-129 (2009).
- [58] Di Francescomarino, C., Ghidini, C., Rospocher, M., Serafini, L. & Tonella, P. Reasoning on semantically annotated processes. ServiceOriented 5346, 132-146 (2008).
- [59] Koschmider, A. & Oberweis, A. Ontology Based Business Process Description. Order A Journal On The Theory Of Ordered Sets And Its Applications 5, 321-333 (2005).
- [60] Super Project. Internet: <http://www.ip-super.org> [July, 10, 2011].
- [61] WML Flight. Internet: <http://www.w3.org/Submission/WSML/#wsml-flight> [July, 10, 2011].
- [62] IRIS. Internet: <http://www.iris-reasoner.org> [July, 10, 2011].
- [63] Bruijn, J.D., Lausen, H., Polleres, A. & Fensel, D. The web service modeling language wsml: An overview. Lecture Notes in Computer Science 4011, 590 (2006).
- [64] Horridge, M., Knublauch, H., Rector, A., Stevens, R. & Wroe, C. A Practical Guide To Building OWL Ontologies using the protege-owl plugin. The University Of Manchester 27, (2004).
- [65] Dou, D., McDermott, D. & Qi, P. Ontology Translation on the Semantic Web. *Science* 2, 35-57 (2004).
- [66] Calvanese, D., Giacomo, G.D. & Lenzerini, M. Ontology of integration and integration of ontologies. Description Logics 49, 10-19 (2001).
- [67] Tang, J. Multiple strategies detection in ontology mapping. *www* 5, 1040-1041 (2005).
- [68] Marques, D., A Survey of Recent Research in Ontology Mapping. School of Interactive Arts & Technology, (2005).

- [69] Yang, K., Steele, R., A Framework for Ontology Mapping for the Semantic Web, NSW Australia (2007)
- [70] PROMPT. Internet: <http://protege.stanford.edu/plugins/prompt/prompt.html> [July, 16, 2011].
- [71] Thomas, H., Sullivan, D.O. & Brennan, R. Ontology Mapping Representations: a Pragmatic Evaluation. Management 228-232 (2009).
- [72] Noy, N.F. Semantic integration: a survey of ontology-based approaches. ACM Sigmod Record 33, 65-70 (2004).
- [73] Madhavan, J., Bernstein, P.A., Domingos, P. & Halevy, A.Y. Representing and Reasoning about Mappings between Domain Models. Artificial Intelligence 80-86 (2002).
- [74] Euzenat, J. & Shvaiko, P. Ontology matching. booksgooglecom 333 Springer, (2007).
- [75] Romero, M., Naya, J., Loureiro, J., Ezquerra, N., Ontology Alignment Techniques. IGI Global, (2008).
- [76] Euzenat, J. & Valtchev, P. Similarity-based ontology alignment in OWL-Lite. Processing 333-337 (2004).
- [77] Noy, N.F. & Musen, M.A. The PROMPT suite: interactive tools for ontology merging and mapping. International Journal of Human-Computer Studies 59, 983-1024 (2003).
- [78] Doan, A., Madhavan, J., Domingos, P. & Halevy, A. Ontology matching: A machine learning approach. Science 1-20 (2004)
- [79] Maedche, A., Motik, B., Silva, N. & Volz, R. MAFRA-a MApping FRAmework for distributed ontologies. Lecture Notes in Computer Science 235–250 (2002).
- [80] Euzenat, J. & Shvaiko, P. Frameworks and formats: representing alignments. Ontology Matrching 219-244 (2007).
- [81] Lei, Y. An instance mapping ontology for the semantic web. Proceedings of the 3rd international conference on Knowledge capture KCAP 05 67 (2005).
- [82] Scharffe, F. & De Bruijn, J. A Language to Specify Mappings Between Ontologies. Proc of the Internet Based Systems IEEE Conference SITIS05 260-264 (2005).
- [83] Terse RDF Triple Language. Internet: <http://www.w3.org/TR/turtle> [July, 20, 2011].

- [84] Modeling your business processes with IBM WebSphere Lombardi. Internet: [http://www.ibm.com/developerworks/websphere/library/techarticles/1101\\_wang/1101\\_wang.html?ca=drs-](http://www.ibm.com/developerworks/websphere/library/techarticles/1101_wang/1101_wang.html?ca=drs-) [Aug, 6, 2011].
- [85] OntoGraph. Internet: <http://protegewiki.stanford.edu/wiki/OntoGraf> [Aug, 6, 2011].
- [86] Jambalaya. Internet: <http://protegewiki.stanford.edu/wiki/Jambalaya> [Aug, 6, 2011].
- [87] BCG Guidelines. Internet: [www.bcguidelines.ca](http://www.bcguidelines.ca) [Aug, 6, 2011].
- [88] Macroscopic and Microscopic Urinalysis Guideline. Internet: [http://www.bcguidelines.ca/guideline\\_urinalysis.html](http://www.bcguidelines.ca/guideline_urinalysis.html) [Aug, 6, 2011].
- [89] Lin, Y. & Interoperability, S. Semantic Annotation for Process Models. (2008).
- [90] Lohmann, N., Verbeek, E. & Dijkman, R. Petri Net Transformations for Business Processes – A Survey. *Transactions on Petri Nets and Other Models of Concurrency II* 5460, 46-63 (2009).
- [91] Matheus, C.J., Kokar, M.M. & Baclawski, K. A core ontology for situation awareness. *Proceedings of the Sixth International Conference on Information Fusion 1*, 545-552 (2003).
- [92] Sibte, S., Abidi, R. and Shayegani, S., Modeling the Form and Function of Clinical Practice Guidelines: An Ontological Model to Computerize Clinical Practice Guidelines. *Knowledge Management for Health Care Procedures* 81-91 (2009).
- [93] Essex, D. The many layers of workflow automation. *Healthcare informatics the business magazine for information and communication systems* 17, 121-122, 124-130 (2000).
- [94] V. Kabilan, T. A. Halpin, K. Siau, J. Krogstie, Eds. *Proceedings of the 10th International Workshop on Exploring Modeling Methods in Systems Analysis and Design EMMSAD 05 Caise* , 557-568 (2005).
- [95] Lee, J., Kim, J., Cho, I. & Kim, Y. Integration of workflow and rule engines for clinical decision support services. *Studies In Health Technology And Informatics* 160, 811-815 (2010).
- [96] Alexandrou, D., Xenikoudakis, F. & Mentzas, G. SEMPAT: Adapting Clinical Pathways by Utilizing Semantic Technologies. 2009 13th Panhellenic Conference on Informatics 125-130 (2009).



- [97] Du, G. & Jiang, Z., The integrated modeling and framework of clinical pathway adaptive workflow management system based on Extended Workflow-nets (EWF-nets). 2008 IEEE International Conference on Service Operations and Logistics and Informatics 914-919 (2008).
- [98] Morgenstern D., Essentials of Clinical Workflow Analysis. Internet: [http://www.masstech.org/ehealth/CPOE University/WFACSC.pdf](http://www.masstech.org/ehealth/CPOE%20University/WFACSC.pdf) [Dec, 20, 2011].
- [99] Gruber, T.R., 1993. A Translation Approach to Portable Ontology Specification. *Knowledge Acquisition*, 5(2)(2), p.199-220. Technical Report. Available at: <http://tomgruber.org/writing/ontolingua-kaj-1993.htm> [Dec, 23, 2011].
- [100] Shvaiko, P., and Euzenat, J., A Survey of Schema based Matching Approaches. Technical Report DIT-04-087, Informatica e Telecomunicazioni, University of Trento, 2004.
- [101] Dou, D. & McDermott, D. Deriving axioms across ontologies. *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems AAMAS 06* 952 (2006).
- [102] Bergmann S., Design and Implementation of a Workflow Engine. Thesis chapter, Rheinische Friedrich-Wilhelms-Universität, 2009.
- [103] R.E., Eshuis, R. & Wieringa, R. A Formal Semantics for UML Activity Diagrams - - Formalising Workflow Models. *Science* 02, 1-44 (2001).
- [104] WfMC Workflow Management Coalition Terminology & Glossary. *Management* 39, 1-65 (1999).
- [105] Workflow Implementation Guideline V.7.1, IBM. Available at: [http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/topic/com.ibm.mam.doc.7.1/pdf/mam71\\_workflow\\_imp\\_guide.pdf](http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/topic/com.ibm.mam.doc.7.1/pdf/mam71_workflow_imp_guide.pdf). [Jan, 16, 2012].
- [106] Morgenstern D., Essentials of Clinical Workflow Analysis. Massachusetts Technology, New Healthcare Institute, CPOE University. Available at: [http://www.masstech.org/ehealth/CPOE University/WFACSC .pdf](http://www.masstech.org/ehealth/CPOE%20University/WFACSC.pdf) [Jan, 16, 2012].
- [107] Georgakopoulos, D., Hornick, M. & Sheth, A. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases* 3, 119-153 (1995).
- [108] Stoilov T., Stoilov K., Lyutov N., Workflow Technology as a Tool for Automation of Business Systems. International Scientific Conference, Bulgarian Academy of Science, Varna Free University, 668-674 (2008).



- [109] Ouvry A.S., Workflow analysis and modeling in medical IT projects. *MEDICAMNUDI* 46, 47-54 (2002).
- [110] Hollingsworth, D. Workflow Management Coalition The Workflow Reference Model. *Management* 1-55 (1995).
- [111] W. van der Aalst and K. van Hee, Workflow Management: Models, Methods, and Systems (Cooperative Information Systems). The MIT Press, Mar. 2004.
- [112] Borgida, A. Description Logics for Data Bases. *Order A Journal On The Theory Of Ordered Sets And Its Applications* 472-494 (2003).
- [113] Davulcu, H., Kifer, M., Ramakrishnan, C.R. & Ramakrishnan, I.V. Logic based modeling and analysis of workflows. *Proceedings of the seventeenth ACM SIGACTSIGMODSIGART symposium on Principles of database systems PODS 98* 25-33 (1998).
- [114] Kiepuszewski, B., Ter Hofstede, A.H.M. & Van Der Aalst, W.M.P. Fundamentals of control flow in workflows. *Acta Informatica* 39, 143-209 (2003).
- [115] Tricković, I. FORMALIZING ACTIVITY DIAGRAM OF UML BY PETRI NETS. *emisamsorg* 30, 161-171 (2000).
- [116] Van Der Aalst, W. Formalization and verification of event-driven process chains. *Information and Software Technology* 41, 639-650 (1999).
- [117] Lu, R. & Sadiq, S. A Survey of Comparative Business Process Modeling Approaches. *Business Information Systems* 4439, 82-94 (2007).
- [118] Ferdian A Comparison of Event-driven Process Chains and UML Activity Diagram for Denoting Business Processes. *Harburg Technische Universitat Hamburg* Master, 1-42 (2001).
- [119] Yang, P., Yang, Z. & Lu, S. Formal Modeling and Analysis of Scientific Workflows Using Hierarchical State Machines. *Third IEEE International Conference on eScience and Grid Computing eScience 2007* 619-626 (2007).
- [120] Yildiz, U., Guabtni, A. & Ngu, A.H.H. Towards scientific workflow patterns. *Proceedings of the 4th Workshop on Workflows in Support of LargeScale Science* 1-10 (2009).
- [121] Ceusters, W. & Smith, B. Semantic Interoperability in Healthcare State of the Art in the US A position paper with background materials prepared for the project. *Health San Francisco* 1-33 (2010).

- [122] Oemig, F. & Blobel, B. Semantic interoperability between health communication standards through formal ontologies. *Studies In Health Technology And Informatics* 150, 200-204 (2009).
- [123] Orgun, B., Dras, M., Nayak, A. & James, G. Approaches for semantic interoperability between domain ontologies. *Expert Systems* 25, 179-196 (2006).
- [124] Ducrou, A.J. University of Wollongong Thesis Collection Complete interoperability in healthcare: technical, semantic and process interoperability through ontology mapping and distributed enterprise integration techniques. *Techniques* (2009).