A FRAMEWORK FOR THE DEVELOPMENT OF EXECUTABLE SYSTEMS ARCHITECTURE

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy at Institute of Business Administration, Karachi

by

Imran Khan

Master of Science, SZABIST, 2005
Master of Computer Science, University of Karachi, 2001
Post Graduate Diploma in Computer & Information Science, University of Karachi, 1999
Bachelor of Science, University of Karachi, 1996

Supervisor: Dr. Sajjad Haider

Department of Computer Science

Fall Semester 2014
Institute of Business Administration, Karachi, Pakistan
A FRAMEWORK FOR THE DEVELOPMENT OF EXECUTABLE SYSTEMS ARCHITECTURE

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy at Institute of Business Administration, Karachi

by

Imran Khan
Master of Science, SZABIST, 2005
Master of Computer Science, University of Karachi, 2001
Post Graduate Diploma in Computer & Information Science, University of Karachi, 1999
Bachelor of Science, University of Karachi, 1996

Supervisor: Dr. Sajjad Haider,
Department of Computer Science

Fall Semester 2014
Institute of Business Administration, Karachi, Pakistan
A FRAMEWORK FOR THE DEVELOPMENT OF EXECUTABLE SYSTEMS ARCHITECTURE

by

Imran Khan
A Dissertation Submitted to the Faculty of Computer Science in Partial Fulfillment of the Requirements for the Degree

of

Doctor of Philosophy
Computer Science

Committee:

Dr. Sajjad Haider Supervisor
Associate Professor
FCS, IBA, Karachi

Dr. Asim ur Rehman Khan Examiner
Professor
National University FAST, Karachi.

Dr. Sharifullah Khan Examiner
Associate Professor
SEECS, NUST, Islamabad

Dr. Sayeed Ghani
Associate Dean,
FCS, IBA, Karachi

Dr. Ishrat Husain,
Dean and Director,
IBA, Karachi.

Fall Semester 2014
Institute of Business Administration, Karachi, Pakistan
Dedication

To my parents and my entire family.
Acknowledgements

First of all, I thank Allah (subhana wa ta’al) for supporting me throughout this journey of doing research. Allah (swt) helped me when I was alone and stuck, with no one to refer to. Indubitably, the Greatest, He helped me get to the next level by helping me find the right kind of people, without whom this research would not have been possible. Dr. Sajjad, who got hold of my hand and guided me through my research, not only did he help me with the process of this work, but also provided encouragement and the advices which greatly improved my analysis and thinking. Research writing was another important area where Dr. Sajjad helped me a lot. His support throughout process is greatly appreciated. Dr. Shakeel Khoja and Dr. Zaheeruddin Asif patiently sat through my questions and provided excellent counseling, concerning the PhD process, in general, and my research, in specific. Their continuous moral support helped me pursue my journey. Dr. Asif, generously provided me with his time in order to discuss the research and provided excellent advices that guided the development of my methodology and ultimately its performance measurements. Dr. Khoja, provided counseling at every step of way along with remembering me in his prayers. I acknowledge Dr. Sayeed Ghani, Dr. Nasir Touheed and Dr. Shahid Qureshi for pushing me all the time to complete my tasks and helping me realize the importance of its success. Also, I acknowledge my colleagues, Dr. Quratul Ain, Ms. Saleha Raza and Ms. Asma Larik for
taking out time, to listen to my research presentation, and their valuable advices and Ms.
Rabail Qayum and her Writing Centre Team, of the Cowasjee Centre, for the number of
times they have sat with me helping me revise my work during dissertation writing sessions.
Finally, I would like to thank my entire family, friends and colleagues for their immense
support and their mere presence, if nothing else than their friendship, encouragement and
companionship. A very special thanks to my mother, it is all because of her prayers that I am
able to complete such an important phase of my life.
Table of Contents

List of Tables ................................................................................................................. xi
List of Figures .................................................................................................................... xii
Abstract .............................................................................................................................. xiv
Chapter 1 .......................................................................................................................... 1
Introduction ......................................................................................................................... 1
  1.2  MOTIVATION ............................................................................................................ 4
  1.3  OBJECTIVE ............................................................................................................. 7
  1.4  SUMMARY ................................................................................................................. 10
Chapter 2 .......................................................................................................................... 12
Technical Background ....................................................................................................... 12
  2.1  ONTOLOGY .............................................................................................................. 12
  2.1.1  Ontology Languages .......................................................................................... 14
  2.1.2  Uses of Ontology .............................................................................................. 16
  2.1.3  Ontology Editor .................................................................................................. 18
  2.2  SEMANTIC WEB RULE LANGUAGE (SWRL) ..................................................... 18
  2.3  UNIFIED MODELING LANGUAGE (UML) ............................................................. 20
  2.3.1  Class Diagram ................................................................................................... 23
  2.3.2  Activity Diagram .............................................................................................. 24
  2.4  PETRI NETS ............................................................................................................ 27
  2.5  COLOURED PETRI NETS (CPN) .......................................................................... 30
  2.5.1  Introduction ....................................................................................................... 30
  2.5.2  CPN Model ........................................................................................................ 31
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.5.3</td>
<td>CPN Tools</td>
<td>33</td>
</tr>
<tr>
<td>2.6</td>
<td>ILLUSTRATION OF CPN USING DINING PHILOSOPHER EXAMPLE</td>
<td>34</td>
</tr>
<tr>
<td>2.7</td>
<td>SUMMARY</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>Chapter 3</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>Literature Review</td>
<td>37</td>
</tr>
<tr>
<td>3.1</td>
<td>MODEL AND ITS PURPOSE</td>
<td>37</td>
</tr>
<tr>
<td>3.2</td>
<td>SYSTEMS AND SYSTEMS THINKING</td>
<td>39</td>
</tr>
<tr>
<td>3.3</td>
<td>ORIGIN OF SYSTEMS ENGINEERING</td>
<td>41</td>
</tr>
<tr>
<td>3.4</td>
<td>SYSTEMS ENGINEERING OR SYSTEMS ARCHITECTING</td>
<td>42</td>
</tr>
<tr>
<td>3.5</td>
<td>ARCHITECTURE FRAMEWORK</td>
<td>44</td>
</tr>
<tr>
<td>3.6</td>
<td>SUMMARY</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>Chapter 4</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>Consistent Systems Architecture Description and Behavior Framework (CSADBF)</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>SUMMARY</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td>Chapter 5</td>
<td>59</td>
</tr>
<tr>
<td></td>
<td>Ontology to UML mapping schemes</td>
<td>59</td>
</tr>
<tr>
<td>5.1</td>
<td>MAPPING BETWEEN ONTOLOGY AND UML</td>
<td>59</td>
</tr>
<tr>
<td>5.2</td>
<td>ILLUSTRATION OF ONTO-UML MAPPING SCHEME USING SAC-IBA</td>
<td>62</td>
</tr>
<tr>
<td>5.3</td>
<td>COMPARISON BETWEEN ONTO-UML ALGORITHM WITH THE PROTÉGÉ PLUG-IN (KNUBLAUCH'S APPROACH)</td>
<td>72</td>
</tr>
<tr>
<td>5.4</td>
<td>SUMMARY</td>
<td>77</td>
</tr>
<tr>
<td></td>
<td>Chapter 6</td>
<td>78</td>
</tr>
<tr>
<td></td>
<td>Onto-UML to CPN mapping schemes</td>
<td>78</td>
</tr>
<tr>
<td>6.1</td>
<td>MAPPING BETWEEN STATIC VIEW (ONTOLOGY, UML) AND DYNAMIC VIEW(CPN)</td>
<td>78</td>
</tr>
<tr>
<td>6.2</td>
<td>MAPPING SCHEME USING SAC-IBA</td>
<td>80</td>
</tr>
<tr>
<td>6.3</td>
<td>SUMMARY</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>Chapter 7</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td>SWRL rules to CPN Guard Conditions and Arc Inscription Mapping Scheme (OntoCPNRules)</td>
<td>86</td>
</tr>
</tbody>
</table>
List of Tables

Table 5.1 Ontology to UML mapping scheme (Onto-UML) .................................................. 61
Table 5.2 Components of SAC-IBA Ontology .................................................................. 73
Table 5.3 Ontology to UML mapping using Onto-UML ...................................................... 74
Table 5.4 Ontology to UML mapping using Protégé UML Plug-in (Knublauch’s methodology) ......................................................................................................................... 75
Table 5.5 Comparison (of Ontology to UML mapping) between Onto-UML & Protégé UML Plug-in ......................................................................................................................... 76
Table 5.6 Comparison between Onto-UML and Protégé Plug-in generated Class diagram .......................................................................................................................... 77
Table 6.1 Onto-UML to CPN mapping scheme .................................................................. 79
Table 7.1 Assumptions for OntoCPNRules .......................................................................... 89
Table 7.2 Ontology to CPN rules mapping Scheme (OntoCPNRules) ................................. 90
Table 7.3 Selected Case with example ................................................................................ 93
Table 7.4 Mapping Summary .............................................................................................. 98
Table 8.1 Create OP Ontology ............................................................................................ 104
Table 8.2 Generate OP Class Diagram from OP OWL Ontology ....................................... 104
Table 8.3 Create CPN model using Onto-UML to CPN mapping scheme ......................... 109
Table 8.4 Case 2 with example .......................................................................................... 113
Table 8.5 Eight cases with SWRL to guard condition mapping ......................................... 114
Table 8.6 Case 6 with example .......................................................................................... 116
Table 8.7 Case 8 with example .......................................................................................... 118
Table 9.1 The Operational Goals ....................................................................................... 124
Table 9.3 Case Studies’ Specifications ................................................................................ 143
Table 9.4 Summary of Results (SAC-IBA Case Study) ....................................................... 146
Table 9.5 Summary of Results (Order Processing Case Study) ........................................... 150
List of Figures

Figure 2.1 UML Diagrams.............................................................................................................. 21
Figure 2.2 Petri Nets (PN) .............................................................................................................. 29
Figure 2.3 CPN of Dining Philosopher.......................................................................................... 34
Figure 2.4 Dining philosopher after first transition ................................................................. 35
Figure 2.5: Dining Philosopher after 2nd transition ............................................................... 36
Figure 4.1. Consistent Systems Architecture Description and Behavior Framework (CSADBF) ................................................................................................................................. 56
Figure 5.1 SAC-IBA Ontology ...................................................................................................... 64
Figure 5.2 SAC-IBA class diagram ............................................................................................. 68
Figure 5.3 SAC-IBA Activity diagram ........................................................................................ 71
Figure 5.4 UML class diagram generated using Onto-UML tool............................................. 74
Figure 5.5 UML class diagram generated using Protégé UML Plug-in (Knublauch’s approach)............................................................................................................................................... 75
Figure 6.1 SAC-IBA Super page ................................................................................................. 83
Figure 6.2 SAC-IBA CPN model (without guard condition/Arc inscription)............................ 85
Figure 7.1 CPN model before and after execution of verifySACId transition ......................... 94
Figure 7.2 RACReader subpage with Arc Inscription ............................................................ 95
Figure 7.3 Graphical View of Overall mapping ........................................................................ 97
Figure 8.1 OP Ontology................................................................................................................ 102
Figure 8.2 OP class diagram ..................................................................................................... 103
Figure 8.3 OP Activity diagram ................................................................................................. 106
Figure 8.4 Graphical view of mapping of Ontology to UML mapping scheme.................... 107
Figure 8.5 Super page created from OP OWL ontology .......................................................... 110
Figure 8.6 OP Subpage created from OP OWL ontology (without guard conditions) ..... 111
Figure 8.7 CPN model before and after execution of Receive Equipment transition....... 115
Figure 8.8 CPN model before and after execution of Pick Staff transition ..................... 117
Figure 8.9 CPN model before and after execution of Pick Staff transition ...................... 119
Figure 8.10 Graphical view of mapping of OUCPN and OntoCPNRules ...................... 121
Abstract

The thesis presents a framework for executable systems architecture, termed as Consistent Systems Architecture Description and Behavior Framework (CSADBF). The framework shows how consistency can be maintained while modeling architectural description and behavior of a system. A combination of three established modeling techniques; ontology, UML (Unified Modeling Language), and Coloured Petri Nets (CPN), is used to develop this framework whereby each tool complements others in accomplishing the goal of consistency maintenance for the executable systems architecture. The framework suggests various mapping schemes that help in establishing strong concordance among different artifacts of these modeling techniques and maintaining consistency of overall system architecture. The first scheme maps OWL (Web Ontology Language) ontology to UML and is responsible for maintaining consistency of the architectural description. The second scheme maps combination of OWL ontology and UML, to CPN and is responsible for maintaining consistency between static and dynamic views. The third scheme ensures the behavioral consistency of the architecture by providing mapping between SWRL (Semantic Web Rule Language) and CPN guard conditions and arc inscription. Thus, the framework allows architects to model the systems architecture requirements in OWL ontology and UML and to analyze the behavior and performance of systems architecture in CPN. The thesis demonstrates the framework with the help of case studies and comparisons with other frameworks proposed in the literature. Evaluation of information capacity and correctness of
the mapping schemes is also demonstrated in the thesis using the same case studies. The empirical results proves that the transformation evolves in the proposed framework CSADBF is correct and lossless. The results also verifies preservation of information capacity which is evident from complete transformation of source model into the target model.
Chapter 1

Introduction

This dissertation presents a framework to model a system’s architecture. The presented framework helps in the maintenance of consistency across different views of the architecture and at the same time analyze their behavior and performance in an effective manner. This chapter presents the background of the thesis and discusses the motivation behind the work. It also outlines the main objectives as well as the scope of this research.

1.1 BACKGROUND

“System architecture can best be thought of as a representation of an existent (or to be created) system, and the process and discipline for effectively implementing the design(s) for such a system that defines the structure and/or behavior of a system”\(^1\). Modeling a system’s architecture (or Systems architecture) provides an abstract view of the system which helps an architect to manage complexities as well as enables him/her to visually analyze the proposed system. The primary purpose of architectural representations is to

---

\(^1\) http://en.wikipedia.org/wiki/System_architecture
ease communication among architecture’s stakeholders. It would be beneficial for the stakeholders to see the relevant components of a system in the form of a model. When designing an overall picture, architects usually use many diagrams. This results in a huge model, which may cause stakeholders to lose sight of the architecture (Firesmith, et al. 2008). Moreover, it is difficult to maintain consistency in the presence of many diagrams. The task of maintaining consistency becomes more challenging when it comes to executable architectures, where the behavioral performance of the architectural components can dynamically be visualized.

The consistency issue could be addressed by modeling a system’s architecture using a modeling language with strong semantics (formal semantics). The basic purpose of this language is to define architectural description. It should also provide support for visual representation of architectural component so that the behavior and performance of systems architecture can be analyzed visually. More importantly, it should restrict visual representation to only those objects which are defined in the architectural description.

There are varieties of executable formalism that support dynamic modeling with behavior analysis such as finite automata, event graphs, state machine, process algebra, process models, and Coloured Petri nets (CPN). Though all of these tools support development of dynamic models, a well-defined mathematical theory and graphical representation make CPN a better choice. Researchers have used CPN as a standalone tool (Kristensen, et al.
2004) to specify and simulate systems architecture, but due to its weak support for static
description of systems architecture, this method has limited application.

Lack of support for static description in CPN have encouraged researchers to explore other
language such as UML to complement CPN. In the last couple of decades, UML has
emerged as a primary tool for developing software/system architecture (Pohl, et al. 1999).
Although UML’s support for behavioral and structural modeling makes it well suited for
modeling complex relationships, it primarily captures static relationships. For example, the
behavior diagram can only be used to model the static view of system activities. It does not
take into account the dynamics of the system.

Few attempts have been made to make UML executable. Such enhancement includes
Executable UML (xUML), Executable and Translatable UML (xT UML) (Mellor and Balcer
2002), etc. These approaches are based on State Chart Variants and their goal is to generate
automatic code so this method also has a limited scope. Another variant of UML is SysML
(SysML 2003) which allows modeling a system from a Systems Engineering point of view.
Like UML, SysML also provides the static picture of the problem domain and does not
take into account the dynamic behavior.

Model Driven Architecture (MDA) (OMG 2001) is another approach for application design
and implementation which involves efficient use of system models in software
development process. The primary focus of MDA, however, is software architecture which makes it unsuitable for Systems architecture.

1.2 MOTIVATION

In lieu of the above discussion, one can claim that there is no existing modeling language that provides rigorous validation and verification of systems specification at the abstract level and at the same time supports dynamic analysis of its behavior. The discussion also highlights two dimensions of an executable architecture: Static and Dynamic. A Static model represents the architectural description of a problem domain and a static picture of its architectural behavior. A Dynamic model, on the other hand, shows how a system behaves before implementation. To model system architecture from both perspectives (Static and Dynamic), a strong concordance between them is required. The task of maintaining consistency becomes more complex when a system’s architecture has multiple views (architectural description) and (architectural behavior) and each view has multiple diagrams in it. The combination of these views is known as Executable System Architecture.

When dealing with an Executable System Architecture, semantics of the modeling language is a key factor that can help architects in restricting the introduction of inconsistencies in the model. Therefore, a requirement of a formal language has arisen; however, there is no existing modeling language with formal semantics that alone can provide support for executable architectural modeling. Creating a new language may solve
the inconsistency problem but it will take too much time to reach to the level of maturity and provide the kind of features that the existing languages are already providing in isolation. Thus, combining the strengths of existing languages is more efficient than creating a new one.

Most of the existing approaches (Wagenhals, et al. 2000) (Wagenhals, et al. 2003) (Wang and Dagli 2011) (Noguera, et al. 2009) have used a combination of UML/SysML and CPN as their primary languages for architectural description modeling (static view) and behavior modeling (dynamic view), respectively. However, there are certain issues with this combination such as CPN has strong semantics but at the same time it is weak at providing support for static description. Similarly, there are several limitations of UML as well. UML lacks formal semantics (Vanderperren and Dehaene 2005) (Bahill and Daniels 2003) (Evans 1998) (Whittle 2000) (Yi-zhi, Wang and Liu 2004) (OMG 2007), (Costal, Gómez and Guizzardi 2011) that limit verification of static view of the system it describes and that is why it is not fully sufficient for system architecture static modeling. It focuses on realization not conceptualization. It is also not machine readable which restricts querying the system architecture at a conceptual level. Lack of support for automated reasoning and inference in UML makes it weak at model consistency checking. Because of lack of concordance among the set of diagrams in UML, it does not restrict a user/system modeler from using one name throughout the modeling life cycle for classes and their instances, which reflects weak support for fully integrated data dictionary. Model designed in UML are not reusable/shared primarily because of its semi-formal nature. Furthermore, UML
attributes are not first class citizens, that is, attributes can neither be represented independent of the classes nor can they have inheritance-like classes. Moreover, UML provides weak support for modeling integrity constraints (or rules). It uses Object Constraint Language (OCL) to define integrity constraint that does not have formal semantics (Richters and Gogolla 2004) which makes it difficult to map the rules modeled in OCL to a formal language such as CPN. This is one of the reasons why the existing approaches do not provide any support for rules modeling in static view; instead the rules are directly modeled in dynamic view which causes inconsistency in a system’s architecture.

After analyzing the issues identified in system architectural modeling, it can be understood that the root cause lies in the static view, particularly in the selection of language for static view. Thus, there is a desperate need of a formal language that can replace UML completely or complement it with any of the exiting formal languages and limit the inconsistencies from entering the static view.

As discussed above, most of the existing approaches are unable to address three key issues: (1) seamless transformation between the different views of system architecture, (2) maintain consistency using an integrated data dictionary, and (3) verification of behavioral aspect. The first two issues are interrelated. Incomplete transformation between different views of a system’s architecture forces the architect to manually model the leftover architectural components that were not transformed. These manual activities without using
any data dictionary become the main source of introducing inconsistency in a system’s architecture. Besides these two issues, the existing approaches are also unable to verify the behavioral aspect which is due to insufficient support for modeling behavioral element, particularly operational rules at the abstract level within the problem domain’s architectural description. Even the definition of operational rules in static view alone cannot solve the problem of behavior verification unless these rules are verified through dynamic view using a suitable simulation tool. Therefore, the verification of operational rules through a simulation tool requires mapping of these rules from the static view to the dynamic view.

1.3 OBJECTIVE

This research aims at contributing to the theory of design and simulation of system architecture particularly executable system architecture by providing a novel framework. The framework will help the architects in creating a consistent view of both architectural description and behavior of a system. Moreover, it will assist the architects in conducting sensitivity analysis through logical and behavioral performance evaluation of a system. There are a number of objectives achieved while developing the presented framework that includes development of integrated data dictionary, maintenance of consistency between the multiple views, automatic transformation of business rules in dynamic view from the rules modeled in static view, and the overall semi-automatic transformation between static and dynamic views. In this way, the framework provides a complete information exchange mechanism that helps in the transformation within the various components of the architecture. The framework synergizes three well-founded existing modeling tools: OWL
ontology (in combination with SWRL), UML, and CPN. This research exhibits the limitation of these modeling tools to model an executable architecture and how their convergence can help in implementing the novel framework. The framework uses OWL ontology and UML for static view modeling and CPN for dynamic view modeling.

Even though UML is weak in formal semantics, the kind of richness that UML provides for modeling static view of a system’s behavior emphasizes that it is almost impractical to replace UML completely. As a result, complementing UML with a formal language is a better option. Essence of formalism in ontology (in particular Web Ontology Language (OWL) ontology) and its ability to model rules/constraints make it a suitable language for the complementing task. OWL is a formal language designed to represent rich and complex knowledge about things, groups of things, and relations between things. The architectural description can easily be modeled in OWL. To create a bridge between OWL and UML for this complementing task, there comes a need to provide a mapping scheme for automatically generating the UML class diagram from an OWL ontology so that the static view of behavior (UML activity) can be modeled using consistent UML class diagram. Couple of more schemes is required to complete the task. The first one is to create rules in OWL ontology using Semantic Web Rule Language (SWRL) and map these rules to guard condition and arc inscription in CPN. SWRL is a rule language that works in combination with OWL and supports rules modeling in OWL ontologies. The second mapping scheme is required to map the overall static view to dynamic view. These mapping schemes may
still not prevent the inconsistencies if they work in isolation. Hence, for maintaining the overall consistency, a framework is required that integrates the activity as a whole.

To start working with the presented framework, the system’s requirement specification in the form of a narrative story is needed, which is used to developed OWL ontology. These narrative stories belong to some problem domains and the domain information is only available to the domain experts. Therefore, these domains experts are needed for developing a new ontology as the presented framework does not provide any domain information. Narrative story should also contain the operational rules or business constraints. These rules will be modeled in SWRL embedded within OWL ontology. The existing ontology can also be taken if it is fulfilling requirement specification given in the narrative story, but these ontologies should also have business rules embedded in them.

The OWL ontology developed in the first phase will be used to generate a UML class diagram automatically, which will be used to generate activity diagram semi-automatically. The combination of both will be used to generate the CPN model which would assist in analyzing the behavior and performance of the modeled system.

Hence, the overall research is defined in a problem statement as:
“Development of a framework for executable system architecture modeling that maintains consistency throughout the process.”

There are three sub problems of this research:

1. To enhance the consistency of static view by automatic creation of complete UML class diagram from OWL ontology and UML activity diagram from UML class diagram.

2. To create support for complex business rules modeling in static view and their behavior analysis in dynamic view by using the set of steps proposed to create business rules in SWRL (embedded within OWL) for their automatic transformation to both guard conditions and arc inscription in CPN.

3. To develop integrated data dictionary using the combination of OWL ontology, SWRL and UML to maintain consistency in the overall semi-automatic transformation process between static view and dynamic view.

1.4 SUMMARY

This chapter raised three key issues that affect systems engineering in today’s world. They are (1) seamless transformation between the different views of system architecture, (2) consistency maintenance using an integrated data dictionary, and (3) verification of behavioral aspect of systems architecture. The chapter began with the overall background of the research problem. It provided a detailed introduction, the motivation behind the research, the objectives that are going to be achieved from the research at present and the
scope of the overall research. The key contributions of the thesis were also outlined. In addition, the background information was also given to support the claim that the identified problem is indeed important and needs scholarly attention.
Chapter 2

Technical Background

This chapter discusses the technical background required to understand the presented framework. The framework consists of four techniques: OWL Ontology, SWRL, UML and Colored Petri nets (CPN). To understand and appreciate the framework, it is necessary to have some background knowledge of these four techniques.

2.1 ONTOLOGY

The term ‘ontology’ (or ontologia) was introduced for the first time in 1613 (Ingarden 1964), by two philosophers, Rudolf Göckel (Goclenius), in his Lexicon philosophicum and Jacob Lorhard (Lorhardus), in his Theatrum philosophicum. Basically it comes from the Greek ontos, for “being”, and logos, for “word”. Its first occurrence in English as recorded by the Oxford English Dictionary appears in Bailey’s dictionary of 1721, which defines ontology as ‘an Account of being in the Abstract’. In philosophy, it refers to the subject of existence, i.e., the study of being as such. More precisely, it is the study of the categories of things that exist or may exist in some domain (Sowa 2000).
An ontology is a formal theory within which not only definitions but also a supporting framework of axioms is included (perhaps the axioms themselves provide implicit definitions of the terms involved). The newly fashionable usage of ‘ontology’ as meaning just ‘conceptual model’ is by now firmly entrenched in many information systems circles. Gruber is to be given credit for having crystallized the new sense of the term by relating it to the technical definition of ‘conceptualization’ introduced by Genesereth and Nilsson (1987). In his study (Gruber 1993) Gruber defines the ontology as ‘the specification of a conceptualization’. An abstract, simplified view of the world is represented by Conceptualization. Every conceptualization based on concepts, objects and entities (Dragan, Djuri´c and Devedˇzi´ 2006) that exist in a domain of interest, and the relationships that exist among them. The other part of the above definition – specification – means a formal and declarative representation. In the data structure representing the ontology, the type of concepts used and the constraints on their use are stated declaratively, explicitly, and using a formal language. The formal representation implies that ontology should be machine-readable.

Kalfoglou (2001) highlights an important dimension of ontologies: “ontology is an explicit representation of a shared understanding of the important concepts in some domain of interest”. The word shared indicates that ontology captures some consensual knowledge (Dragan, Djuri´c and Devedˇzi´ 2006). It is not supposed to represent the subjective knowledge of some individual, but the knowledge accepted by a group or a community. Informally, the ontology of a certain domain is about its terminology (domain vocabulary),
all essential concepts in the domain, their classification, their taxonomy, their relations (including all important hierarchies and constraints), and domain axioms.

In the last couple of decades, ontology has become an active area of research and is having impact on many fields of computer and information science. Its importance has been recognized in research fields as diverse as Enterprise Modeling (Guizzardi and Wagner 2005), knowledge engineering (Gómez-Pérez 1997), knowledge representation (Guarino 1995), database design (Burg 1997), information modeling (Ashenhurst 1996), information integration (Wiederhold 1996), object-oriented analysis (Wand 1989), information retrieval and extraction (Reinoso-Ca

2.1.1 Ontology Languages

Ontology are usually developed using some language. Resource Description Framework (RDF) and Web Ontology Language (OWL) are the two widely used ontology representation languages around. RDF (Manola and Miller 2004) was developed by the W3C (World Wide Web Consortium) as a semantic network-based language to describe Web resources. OWL, or Web Ontology Language (Smith, Welty and McGuinness 2004) was also developed under the auspices of the W3C and evolved from DAML+OIL. OWL is currently the most popular ontology representation language.

Ontology representation languages, such as OWL and RDF, provide strong support for the semantics of systems architecture and representation of enterprise level conceptualization.
For enterprise conceptualization, RDF provides domain vocabulary definition, validation of ontologies, consistency checking, and increased expressivity in constraint representation. There are three sublanguages with different levels of complexity, which require increasing computation power but provide more expressive ways to pose restrictions.

- OWL Lite
- OWL DL
- OWL Full

The simpler is OWL Lite and the more complex and expressive OWL Full. In the middle there is OWL DL, which is based on Description Logics (Baader, et al. 2003) and provides a trade-off between complexity and expressiveness. OWL Lite is intended to support the building of simple classification hierarchies and simple constraints. To this end, the ability to specify constrains in OWL Lite is rather restricted; for example, the only cardinality values permitted in OWL Lite are 0 and 1. OWL DL reflects the description-logic foundation of its predecessor, DAML+OIL. OWL DL provides the maximum expressiveness, but also guarantees that all conclusions are computable and will finish in a finite time. OWL Full supports users who want maximum expressiveness and the syntactic freedom of RDF, but does not guarantee reasoning. OWL Full can be viewed as an extension of RDF, whereas OWL Lite and OWL DL can be viewed as extensions of a restricted view of RDF (Smith, Welty and McGuinness 2004).
2.1.2 Uses of Ontology

With respect to uses of ontology described in (Fishwick 2007): an ontology (Maedche and Staab 2001) (Fishwick and John A. 2004) (Lacy and William 2004) can span everything from a simple index or taxonomy to a semantic network that also defines a series of simple and complex logical constraints. The application of ontologies to modeling will gradually result in model taxonomies. Ontology provides standard terminology to the community and beyond, so that common understanding of concepts and relationships can be achieved, which, in turn, increases the potential for application interoperability and reuse of simulation artifacts.

A terminology defines a set of related terms, which may be classified to form taxonomy. When named relationships are added, it may be referred to as ontology. Specifically, ontology concerns the classification of concepts (or classes) as well as their subclasses, properties, and relationships to other concepts.

Let us now examine in greater detail, the problem of defining or describing a model in terms of (i) statics and (ii) dynamics. The statics of an entity define its type (types of properties) and immutable state. The statics can be described at a high level using, for example, a UML class diagram or OWL. The dynamics of an entity define its behavior. There are several ways to describe behavior in UML (e.g., sequence diagrams, collaboration diagrams, statechart diagrams, or activity diagrams). In addition, other formalisms such as process algebras, Petri nets, bond graphs, activity cycle diagrams, and
event graphs may be used. The current state of affairs is that there are several competing approaches and none are as successful as the approaches used for statics.

Ontology is ideal for describing things, so statics can be well handled. Dynamics or behavioral specifications are more challenging. For simple cases, preconditions and postconditions (or effects) may be expressed using an ontology language like OWL, while for more complex conditions a rule language like SWRL is more suitable. Pre- and postconditions (alternatively effects) may be modeled with a constraint language such as SWRL or UML’s object constraint language (OCL). SWRL rules are more than simple definitions of concepts, in that they reflect the relations between the concepts in addition to the definitions of the concepts themselves.

The construction of ontologies encourages the development of conceptually sound models, more effectively communicates these models, enhances interoperability between different models, and increases the reusability and sharing of model components (Reitsma and Jochen 2005).

The notion of combining ontologies with simulation has received much attention in the last decade (Lacy and William 2004) (Fishwick and John A. 2004) (Miller 2004) (Raubal and Werner 2004). If an ontology is also used to represent the internal structure of a model, then the model internals can be transformed into other dynamic modeling languages such as Coloured Petri nets etc. and it can also be compared in an automated fashion to determine which parts of the models are similar and which are different.
2.1.3 Ontology Editor

OntoStudio\(^2\) and Protégé\(^3\) are the two ontology editors used in this research. Protégé, an open source ontology development editor, currently the leading ontology development environment, was developed at Stanford University (protégé-owl 2004). Protégé has already been through a number of versions and modifications by research community.

2.2 SEMANTIC WEB RULE LANGUAGE (SWRL)

SWRL is an effort to combine ontologies (OWL-DL) and rules (RuleML) to capture operational or business rules (Ian Horrocks 2003). It has Horn-like rules structure that is combined with an OWL knowledge base. OWL-DL, by origin, is variable free whereas variables are used in RuleML. Thus, SWRL adds expressivity to OWL ontology and existing rule sets can be reused. SWRL extends the set of OWL axioms to include Horn-like rules. It thus enables Horn-like rules to be combined with an OWL knowledge base. All rules are expressed in terms of OWL concepts (classes, properties, individuals, literals…).

SWRL rule contains two parts:

1. an antecedent part, which is referred to as the body
2. a consequent part, which is referred to as the head.

\[ \text{Antecedent (or body)} \rightarrow \text{Consequent (or head)} \]

---

\(^2\) http://www.ontoprise.de/
\(^3\) http://protege.stanford.edu
The intended meaning can be read as: whenever the conditions specified in the antecedent satisfied, then the instructions specified in the consequent gets executed. Both the body and head consist of positive conjunctions of *atoms*:

\[ \text{atom} \land \text{atom} \land \text{atom} \ldots \rightarrow \text{atom} \]

Atoms in these rules can be of the form A(x), B(x,y), sameAs(x,y) or differentFrom(x,y), where A is an OWL description, B is an OWL property, and x,y are either variables, OWL individuals or OWL data values. SWRL provides seven types of atoms:

1. **Class Atoms** such as `Person(?x)` represents concept `Person` with instance ?x;
2. **Individual Property atoms** such as `hasBrother(?x, ?y)` represents a relationship `hasBrother` between ?x and ?y individual;
3. **Data Valued Property atoms** such as `hasAge(?x, ?age)` represents that ?x individual has a data value ?age of type `hasAge`;
4. **Different Individuals atoms** such as `differentFrom(?x, ?y)` represents that ?x and ?y are two different individual;
5. **Same Individual atoms** such as `sameAs(?x, ?z)` represents that ?x and ?z are same individuals;
6. **Built-in atoms** such as `swrlb:equals(?a, ?b)` represents that variable ?a and ?b have equal values,
7. **Data Range atoms** such as `[3, 4, 5](?x)` represent the ?x as an instance of the list that contains 3,4 and 5 as list elements.
The presented technique in this research supports all the seven types of atoms. There are a few limitations of SWRL. It does not allow disjunction and negation in the rules but there are no such consequences of this limitation in relation to the proposal except some extra effort is required to split the disjunction condition into separate rules. Moreover, explicit qualification over rules is also not supported. We have used SWRL built-ins for comparisons function and abox:setValue built-in function for rule modeling. SWRL built-ins for comparisons include the following functions: swrlb:equals, swrlb:greaterThanOrEqual, swrlb:lessThanOrEqual, swrlb:greaterThan, swrlb:lessThan, etc. The abox:setValue built-in function is used in the consequent part of every rule and is used to assign values to a particular variable used in the SWRL rule.

2.3 UNIFIED MODELING LANGUAGE (UML)

The Unified Modeling Language (UML) is a graphical language for modeling information systems. It was created by a consortium of 12 companies from various domains and was standardized in 1997 by the Object Management Group (OMG) (OMG, OMG Model-Driven Architecture 2001). The language is composed of different diagrams shown in Figure 2.1. UML is a general purpose modeling language initially intended for software intensive systems but it has a capability to communicate any form of information and should not be limited to software. UML is a productive language that can be used to lay out the system architecture of large scale systems.
There are 13 diagrams in the UML that may be grouped into two broad categories, each of which represents a particular aspect of the model. These two aspects of the model are the ‘structural’ and ‘behavioural’ aspects of the model. It is vital that both of these aspects exist for any system, otherwise the system is not fully defined.

The structural aspect of the model shows the ‘things’ or entities that a system comprised of and the relationships between them. It is crucial to remember that the structural aspect of the model shows ‘what’ the system looks like and ‘what’ it does, but not ‘how’. A structural aspect of the model may be thought of as a snapshot in time of any system. The structural aspect of the model may be realised by six UML diagrams shown in Figure 2.1: class diagrams, object diagrams, package diagrams, composite structure diagrams, deployment diagrams and component diagrams.

**Figure 2.1 UML Diagrams**
The behavioral aspect of the model shows the ‘how’ of the system. The behavioral aspect of the model demonstrates how a system behaves over time by showing the order in which things happen, the conditions under which they happen and the interactions between things.

The behavioral or dynamic aspect of the model may be realised using seven UML diagrams (Behavioral and Interaction Diagrams) shown in Figure 2.1: state machine diagrams, use cases, interaction diagrams (sequence, communication, interaction overview and timing diagrams) and activity diagrams.

We need to understand this important point that UML is not cure for all your problems. The main shortfall of the UML is seen when it comes to formal, mathematical specification. This criticism is that the UML is not a truly formal approach. Again, this is true, but the UML is intended to be a general-purpose modelling language with a scope that is as wide as possible. In addition, there is nothing to stop the UML from being used in conjunction with more formal techniques, such as Coloured Petri Nets (CPN) etc., in order to formally specify crucial parts of a system. In this way it provides a clear picture of the important problems at design time by preventing the designer from getting distracted by swarms of details that are better to suppress until later stages.

In the research at present we are using two UML diagrams, class diagram for structural modeling and activity diagram for behavioral modeling.
There are two basic elements that make up a class diagram, which are the ‘class’ and the ‘relationship’ and, at a very simple level, that is it! Clearly, there are many ways to expand on these basic elements, but providing that they are understood clearly and simply, the rest of the syntax follows on naturally and is very intuitive.

2.3.1 Class Diagram

A ‘class’ represents a type of ‘thing’ that exists in the real world and, hence, should have a very close connection to reality. Classes are almost always given names that are nouns, as nouns are ‘things’ and so are classes. This may seem a trivial point, but it can form a very powerful heuristic when assessing and analysing systems as it can be an indicator of whether something may appear as a class on a model.

The second element in a class diagram is a relationship that relates together one or more classes. Relationships should have names that form sentences when read together with their associated classes. Remember that the UML is a language and should thus be able to be ‘read’ as one would read any language. If a diagram is difficult to read, it is a fairly safe but that it is not a very clear diagram and should perhaps be ‘rewritten’ so that it can be read more clearly. Reading a good UML diagram should not involve effort or trying, in the same way that any sentence should not be difficult to read.
2.3.2 Activity Diagram

As described in (Grässle, Henriette and Philippe 2005), Activity diagrams, which are related to program flow plans, are used to illustrate activities. In the external view, we use activity diagrams for the description of those business processes that represent the functionality of the business system. It actually allows you to think functionally.

Because it is possible to explicitly describe parallel events, the activity diagram is well suited for the illustration of business processes, since business processes rarely occur in a linear manner and often exhibit parallelisms. Some of the components of activity diagram that we used in the research are explained below.

Activity diagrams are made up of three basic elements: one or more ‘Activity node’, one or more ‘Activity edge’ and one or more ‘Region’. There are three main types of ‘Activity node’ which are the ‘Activity invocation’ and the ‘Object’ and ‘Control node’. The ‘Activity invocation’ is where the main emphasis lies in these diagrams and it is through activity invocations that it is possible to establish traceability to the rest of the model via operations, activities and actions.

The ‘Activity edge’ element has two main types – ‘Control flow’ and ‘Object flow’ – both of which are the same as in UML 1.x and whose meaning is self-explanatory.

The other major element in an activity diagram is the ‘Region’ that has two main types: Interruptible activity region’ and ‘Activity partition’. An ‘Interruptible activity region’ allows a boundary to be put into a diagram that encloses any activity invocations that may
be interrupted. This is particularly powerful for software applications where it may be necessary to model different areas of the model that can be interrupted, for example, by a direct user interaction or some sort of emergency event. The ‘Activity partition’ is the mechanism that is used to visualize swim lanes that allow different activity invocations to be grouped together for some reason – in the case of swim lanes for responsibility allocation.

**Activity**: An activity diagram illustrates one individual activity. In our context, an activity represents a business process or set of business processes (Figure 3.16). Fundamental elements of the activity are actions and control elements (decision, division, merge, initiation, end, etc.): Elements are connected by “control flow” and “data flow” arrows. The execution of an activity can contain parallel flows. A border can surround the activity, meaning the entire activity diagram.

**Action**: An action is an individual step within an activity, for example, a calculation step that is not deconstructed any further. That does not necessarily mean that the action cannot be subdivided in the real world, but in this diagram will not be refined any further: The action can possess input and output information. The output of one action can be the input of a subsequent action within an activity. Specific actions are calling other actions, receiving an event, and sending signals.
**Data Store:** Data store acts as buffer in activity diagram. It carries data with it and helps to transfer between the action nodes and also between the swim lanes.

**Edge (Control Flow):** Edges, represented by arrows, connect the individual components of activity diagrams and illustrate the control flow of the activity: Within the control flow an incoming arrow starts a single step of an activity; after the step is completed the flow continues along the outgoing arrow. A name can be attached to an edge (close to the arrow).

**Decision Node:** The diamond below represents a conditional branch point or decision node. A decision node has one input and two or more outputs: Each output has a condition attached to it, which is written in brackets. If a condition is met, the flow proceeds along the appropriate output. An 'else' output can be defined along which the flow can proceed if no other condition is met.
**Initial Node:** The initial node is the starting point of an activity. An activity can have more than one initial node; in this case several flows start at the beginning of an activity. It is also possible that an activity has no initial node, but is initiated by an event (action: accepting an event).

**Activity Final Node:** The activity final node indicates that an activity is completed. An activity diagram can have more than one exit in the form of activity final nodes. If several parallel flows are present within an activity, all flows are stopped at the time the activity final node is reached.

### 2.4 PETRI NETS

A *Petri net* (PN) is a network of interconnected locations and activities, with rules that determine when an activity can occur, and specify how its occurrence changes the states of the locations associated with it (Christensen, Jørgensen and Kristensen 1997). Petri nets originated in the work of C. A. Petri in 1962, and have since been developed by many researchers in many countries (Petri 1962).
Petri nets can be used to model and simulate systems of any type. They are particularly useful in facilitating the design and analysis of complex distributed systems that handle discrete flows of objects and/or information.

Petri nets combine a well-defined mathematical theory with a graphical representation of the dynamic behavior of systems. There is an extensive mathematical formalism associated with Petri nets. This formalism completely defines what a Petri net is and how it behaves. The theoretical aspect of Petri nets allows precise modeling and analysis of system behavior, while the graphical representation of Petri nets enables visualization of the modeled system state changes. Although Petri nets are typically represented as graphs drawn on paper or on a computer screen, a Petri net is actually a mathematical object that exists independently of any physical representation.

There is no need to understand the mathematics of Petri nets in order to use them. Just as an engineer can use scientific theories to build useful devices without having to become a scientist, so a system designer can use Petri nets to build useful models without having to become a mathematician.

Petri nets have been developed over the years from a simple yet universally applicable paradigm to a more complex but far more convenient methodology, the *hierarchical colored Petri net*. Such nets are hierarchical in that they contain facilities for representing a model as a hierarchical structure, and are “colored” in that they allow data to have
different types and values (“colors”); earlier varieties of Petri nets allowed only boolean (“black and white”) data.

As shown in Figure 2.2, a PN model is graphically represented by a directed bipartite graph which is composed of circles, rectangle boxes and arcs.

![Figure 2.2 Petri Nets (PN)](image)

Circles (or ellipse) and rectangle boxes represent places and transitions, respectively. Places and transitions are called nodes. Together with the directed arc they constitute a net structure. An arc always connects a place with a transition or a transition with a place. It is illegal to have an arc connecting two nodes of same type. The arcs of the PN model are subdivided into two types: input arcs and output arcs. Input arcs are arrow-headed arcs from places to transitions and output arcs are arrow-headed arcs from transitions to places. PN models may have multiple arcs between places and transitions. Each arc is annotated with arc expression specifying how the state of the PN changes with the occurrence of transition. The arc expressions are written in the PN ML programming language and are built from
variables, constants, operators and functions. The places are used to represent the state of the modeled system. Each place can be marked with one or more tokens.

For brevity, hierarchical Colored Petri nets are usually called *Colored Petri nets (CPN)*. A CPN that models a system is called a *CPN model*.

## 2.5 COLOURED PETRI NETS (CPN)

### 2.5.1 Introduction

Introduced by Kurt Jensen in 1981 (K. Jensen 1981), Coloured Petri Nets (CPN) is a graphical oriented language to design, simulate and verification of systems (K. Jensen 1992) (K. Jensen 1994) (K. Jensen 1997) (Jensen and Kristensen 2009). It is composed of Petri nets (Petri 1962) (Reisig 1985) and a functional programming language CPN ML (Milner, et al. 1997). Petri nets are a powerful modeling formalism in computer science, system engineering, and many other disciplines. The CPN ML programming language which is based on the functional programming language Standard ML provides the primitives for the definition of data types, for describing data manipulation, and for creating compact and parameterisable models. CPN can be regarded as a general purpose modeling language which covers very broad class of systems characterized as concurrent systems. It is generally applicable to the situations where communication, synchronization and resource sharing is required. Typical application domains of CPN are communication protocols, data networks, distributed algorithms and embedded systems. CPN is, however, also applicable more generally for modeling systems where concurrency and
communication are key characteristics, examples of these are business processes and workflows, manufacturing systems, and agent systems.

The CPN language has few but very powerful primitives that make it possible to model systems and concepts at different levels of abstraction. CPN models are formal which means its modeling language has mathematical definition of syntax and semantics.

## 2.5.2 CPN Model

Every CPN model of a system describes states of the system and the events (transitions) that can cause the system to change states. In standard Petri Nets (PN) tokens are indistinguishable entities. The semantics of the model does not allow to follow the behavior of an individual token through the PN. CPN overcome this limitation by assigning each individual token a data value. This data value is called the *token colour*. It is the number of tokens and the token colours on the individual places that represent the state of the system. This is called a *marking* of the CPN model. The number of tokens present on specific places is constitutes the marking of those particular places. The initial CPN state is called the initial marking. Names of the places are written inside ellipse. Similar to Mnemonic names in traditional programming, these names have no formal meanings but they have huge practical importance of readability of CPN model.

The transitions represent the events that can take place in the modeled system. Similar to the places, the transition name is also written inside the box and it also does not have any formal meaning but they are important for the readability of the model. When a transition
occurs it removes tokens from its input places and it adds tokens to its output places. The movement of tokens in response to the firing of a transition is dictated by the inscriptions on the arcs that are incident on that transition. There are four inscriptions (all optional) that may be associated with a transition:

- Transition name inscription
- guard inscription
- Time inscription
- Code segment inscription

*Transition name inscription* has already been explained in the previous paragraph.

*Guard inscription* is a boolean expression. When a guard is present, it must evaluate to *true* for the bindings to be enabled, otherwise the binding is disabled and cannot occur. Before a guard has been added, the default text for the inscription is [].

*Time inscription* makes it possible to evaluate how efficiently a system performs its operations and it also makes it possible to model and validate real-time systems. Before a time inscription has been added, the default text for the inscription is @+.

*Code segment inscription* consists of a piece of sequential CPN ML code that is executed whenever the corresponding transition occurs in a simulation of the CPN model. Before a code segment has been added, the default text for the inscription is:

```plaintext
input ();
output ();
```
Time inscription and Code segment inscription are out of scope in this research. These inscriptions are used to define (operational) rules in CPN model. Arc inscription can also be used for the same purpose.

### 2.5.3 CPN Tools

CPN Tools\(^4\) is a tool for the editing, simulation, state space analysis and performance analysis of CPN models. It is originally developed by the CPN Group at Aarhus University between 2000 and 2010. From the autumn of 2010, CPN Tools is transferred to the AIS group, Eindhoven University of Technology, The Netherlands.

The graphical user interface (GUI) of CPN Tools has no conventional menu bars and pull-down menus, but is based on interaction techniques, such as *tool palettes* and *making menus*.

Sets of values or 'types' defined in CPN Tools are called Colour sets. Each place belongs to a Colour set. Tokens are values taken from the Colour set of the corresponding place. Next section is illustrating the CPN model using dining philosopher example.

\(^4\) CPN Tools can be downloaded from http://cpntools.org/
2.6 ILLUSTRATION OF CPN USING DINING PHILOSOPHER EXAMPLE

To understand how it works, consider Figure 2.3. It is a CPN model designed in CPN tools showing the executable model of dining philosopher. ‘Think’, ‘eat’ and ‘unused chopsticks’ in Circles/ovals are representing the places where as ‘take Chopsticks’ and ‘put down chopsticks’ in boxes are representing the transitions.

![Diagram of CPN model showing dining philosopher example](image)

**Figure 2.3** CPN of Dining Philosopher

Tokens are drawn as black dots within places representing the entities and objects that move around the places through transitions. In dining philosopher scenario, places such as ‘think’ and ‘unused chopsticks’ initially holds 5 token each representing 5 philosopher on dining table and 5 chopsticks on table, respectively. Figure 2.4 shows the first transition when one philosopher start eating after taking two unused

---

5 Dinning Philosopher CPN is an example CPN model provided as sample with CPN Tools by UoA.
chopsticks which reduces the token on ‘think’ and ‘unused chopsticks’ to 4 and 3, respectively whereas ‘eat’ has received its first token. If another philosopher start eating after taking two unused chopsticks then the token on ‘think’ and ‘unused chopsticks’ further reduced to 3 and 1, respectively whereas ‘eat’ has received its another token and now it holds two tokens as two philosophers are now eating. (See Figure 2.5)
Now no more philosopher can eat as there is only one chopstick is available. This deadlock situation is shown in Figure 2.5. In this example we can see the behaviour of dining philosopher scenario through resource utilization and deadlock detection.

2.7 SUMMARY

This chapter provided the overview of the technologies used in this research. The chapter began with the basic knowledge about ontology, SWRL, UML and CPN. All the relevant aspects of these technologies were covered in this chapter that needed to be known before going into the details of the presented framework.

The next chapter will provide details of the presented framework and explain how OWL ontology, SWRL, UML diagrams and CPN models complement each other in developing the presented framework.
Chapter 3

Literature Review

This chapter discusses the review of the relevant literature. In order to understand and appreciate the relevant work, it is necessary to have some background knowledge of what systems engineering and modeling is all about and why do we need it?

3.1 MODEL AND ITS PURPOSE

In the context of engineering, a model can be defined as:

“A model is defined as a simplification of reality that is created in order to better understand the system under development, as we cannot comprehend complex systems.” (Booch, Rumbaugh and Jacobson 1999)

From Wikipedia (Wikipedia 2014) we have the following definition of model in systems architecture:

“A system model is the conceptual model that describes and represents the structure, behavior, and more views of a system.”
The purpose of a model or modeling in general is even harder to capture. In an idealized sense, a model is the essence of science. Since the real world or real systems are so complex, models are constructed that can be manipulated logically or mathematically. Models help us dissect, understand, and make predictions about the real world. For science to be self-correcting, models (or hypothesis or theories) must be falsifiable. In other words, tests and experiments must be developed to show that the model has deficiencies that need to be corrected either by improving the model or discarding it. Like all other engineering disciplines, models are an essential means of communication in systems engineering as well.

In engineering, modeling is a useful starting point for describing and defining the system’s mission and operational environment, showing the interaction of a system with all external entities that may be relevant to its operation. It also provides a basis for formulating system operational scenarios that represent the different conditions under which it must be designed to operate. Four explicit ways in which we utilize a model are described in (Holt 2004):

1. To visualize a system. To quite literally get a mental picture of what the final system will look like from the user or operator’s point of view.
2. To specify a system. By specify, we mean to state specific needs or requirements concerning the system.
3. As a template for creation, for example, of a plan from which to build the final system.
4. In order to document decisions made throughout the project quite often, especially when things start to go wrong, people will look to lay blame or to try to establish why particular decisions were made at key points in the project. In effective modeling it is possible to capture thought processes throughout the life cycle of the project, which can be enormously powerful later on.

Conceptual integration is another reason to use model as described in (Brooks and Frederick 1995):

“Conceptual integrity is the most important consideration in system design. The architect should be responsible for the conceptual integrity of all aspects of the product perceivable by the user.”

Dickerson (Architecture and principles of systems engineering 2010) described that the key to conceptual integrity are modeling and modeling languages, architecture frameworks, and systems engineering standards.

3.2 SYSTEMS AND SYSTEMS THINKING

Systems are the inherent part of every aspect of our lives. It is defined in the INCOSE System engineering handbook (Haskins, et al. 2006):

“A system is a combination of interacting elements organised to achieve one or more stated purposes.”
Another definition of System (Hitchins 2003):

“A system is an open set of complementary, interacting parts with properties, and behaviors emerging both from the parts and from their interactions”

Hitchins emphasises the commonly accepted concepts of emergent behavior for systems. It is also clear from his definition that both the parts and the interactions between the parts cause emergent properties, capabilities, and behavior. Looking at any problem from systems perspective, it requires Systems thinking which is defined in (Wikipedia 2013) as:

“An approach to problem solving, by viewing "problems" as parts of an overall system, rather than reacting to specific part, outcomes or events and potentially contributing to further development of unintended consequences.”

Systems thinking has dramatically changed our perception to view the world around us and given us an understanding of how things, regarded as systems, influence one another within a whole. Ecosystem is a pertinent example of systems thinking in the world around us, in which various elements such as air, water, movement, plants, and animals work together to survive or perish. Other examples include various organizations working in this world, comprised of systems that consist of people, structures, and processes that work together to make an organization "healthy" or "unhealthy". Systems thinking enables humans to
understand the components of a system in the context of relationships with each other and with other systems, rather than in isolation. In systems science, it is argued that the only way to fully understand why a problem or element occurs or persists is to understand the parts in relation to the whole (Capra 1997).

3.3 ORIGIN OF SYSTEMS ENGINEERING

No particular date can be associated with the origin of systems engineering. Its principles have been practiced since the building of Pyramids and probably before, but it is recognized distinctly with the effects of World War II and specially in 1950s and 1960s when a number of books were published that identified Systems engineering as a distinct discipline (Kossiakoff and Sweet. 2010). Day by day increase in the development of the digital computer and the associated software technology driving it, are increasingly leading to the replacement of human control of systems by automation. Computer control is quantitatively increasing the complexity of systems and becoming the greater concern of the community. Systems engineering, as we know it today, has developed to meet all the challenges.

Subdividing the complex systems problems into individual building block may solve the problem, but it has a price to pay and that is integration. This means that each building block should fit with its neighbor as well as with its connecting external environment. This ‘fit’ must not only be physical but functional as well. Physical fit is accomplished by interfaces or multiple views or view point or static view, whereas functional relationships
are called interactions or behavioral view. Multiple views are part of everyday life, but not all of us recognise this perspective (Dickerson and Mavris 2010).

### 3.4 SYSTEMS ENGINEERING OR SYSTEMS ARCHITECTING

When most people hear the term “Systems Engineering”, they think about aircraft, motor vehicle, train, ship etc. but, in many cases, this is only the tip of the iceberg. The reason why people get confused is that the concept of systems is not clear. There is a range of systems that can be classified into different types. Holt (2004) listed the small set of types of systems, which can be taken to expand the context of systems in reader’s minds.

1. Large-scale, organizational-type systems, such as social and political systems
2. Living systems, such as the human body and the environment
3. Nontangible system, such as process models and software
4. Behavioral systems, such as martial arts, crossing the road, doing the shopping and going about everyday life
5. Physical systems, such as vehicles, buildings and home appliances

Any complete system that consists of a set of these systems or a sub set of these systems would be considered as “System of System”. Applying engineering to such systems is known as Systems engineering. ‘Applying engineering’ means problem solving with systems at any point in the life cycle.
Researchers (Rechtin 1992) (Kossiakoff and Sweet. 2010) have used the term “system architecting” virtually in the same context as “system engineering”. For this reason, both of these terms are used interchangeably in this thesis. Systems engineering as described in (Kossiakoff and William 2010):

“is an inherent part of project management—the part that is concerned with guiding the engineering effort itself—setting its objectives, guiding its execution, evaluating its results, and prescribing necessary corrective actions to keep it on course.”

To guide is defined as “to show the way or give direction”. This characterization emphasizes a process of deciding a path for others. This does not mean that systems engineers do not play a key role in system design. On the contrary, they are responsible for the overall activity. Important design decisions at every stage cannot be based entirely on quantitative knowledge, as they are for the traditional engineering disciplines, but rather must often rely on qualitative judgments balancing a variety of incommensurate quantities and utilizing experience in a variety of disciplines, especially when dealing with new technology. At this point it is important to know the major contributions of good systems engineering.

A dictionary definition of engineering described in (Kossiakoff and William 2010) is:
“The application of scientific principles to practical ends; as the design, construction and operation of efficient and economical structures, equipment and systems.”

According to this definition, efficiency and economy are the major contributions of good Systems engineering. To convert a new concept into an operational system typically requires many people with diverse skills, who devote years of effort to bring the system from concept to operational use. Systems engineers must guide and coordinate the design of each individual element as necessary to assure that the interactions and interfaces between system elements are compatible and mutually supporting. This will improve the efficiency of the system and make it economical as well.

Systems engineering is focused on the systems as a whole. It looks at the systems from the outside as well as from the inside. These include identification of customer requirements, the system operational environment, behavioral requirements, design validation etc. Systems engineers must guide and coordinate the design of each individual element as necessary to assure that the interactions and interfaces between system elements are compatible and mutually supporting. These interactions and interfaces are demonstrated using models and to formally perform this activity requires framework.

### 3.5 ARCHITECTURE FRAMEWORK

Historically, the term architecture is understood as images of fine building and drawing and plans of houses, but the same term is used by Systems engineers. It is basically:
“Formation or construction resulting from or as if from a conscious act resulting in a unifying or coherent form or structure” (Merriam-Webster 2014)

The ANSI/IEEE Standard 1471-2000 specification of architecture may be stated as:

“the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.”

The definition explicitly mentioned the word “principle” governing how the architecture is developed. Many other definitions exist, but they share several common themes: structure, utility, principle and beauty (Dickerson and Mavris 2010). This means that architecture is concerned not just with what is produced, but also how it is produced. A question may arise in the minds of the readers: is it really necessary to have architecture? The answer to the question, why an architecture must exist, is given in (Holt 2004). It helps:

- To aid understanding
- To highlight and manage complexity
- To communicate ideas and data

Similar to architecture, the word framework was not originally used in the context of systems engineering. The Oxford English Dictionary defines a framework as:
“A structure composed of parts framed together specially one designed for inclosing or supporting anything; a frame or skeleton.” (Oxford 2014)

An alternate definition

“A basic conceptional (the capacity, function, or process of forming or understanding ideas) structure (as of ideas).” (Merriam-Webster 2014)

Like architecture, many other definitions of framework exist, but they share the common theme of a basic underlying skeleton, or a structure. Pulling these definitions together, an architecture framework can then be described as:

“A basic underlying relational structure for forming or constructing an architecture.”

Several methodologies have been reported in the literature for developing software/systems architecture, however, the outcome of those methodologies is only static picture. The list includes the Zachman framework for enterprise architecture (Zachman 1987), The Open Group Architecture Framework (TOGAF) (TOGAF 2003), the Federal Enterprise Architecture (FEA) (CIO 1999), Gartner (formerly, the Meta Framework) (Bittler, Scott and Kreizman. October 21, 2005), IDEAS (International Defense Enterprise Architecture Specification for exchange) and Ontology-driven Open Architecture and Systems (OASIS)
The static forms of architecture are not easy to verify and validate, due to the fact that the collaboration and information flow among various components of architecture are represented in static way. Therefore, they are unable to support in evaluating the system’s performance, resource utilization, deadlock situations, etc. Overall, they provide insignificant information about how a system behaves in operational environment.

Exiting modeling languages do not provide both rigorous validation and verification of system specification at abstract level, and dynamic analysis of system behavior. In the last two decades UML emerged as a primary tool for developing software/system architecture (Pohl, et al. 1999). Its rich set of representations such as behavioral, structural and interaction diagrams makes it well suited to model complex relationships among entities comprising a large scale system. However, UML represents only static form of behavioral model and does not take into account the dynamics of the system. Some attempts in the past have been made with UML to make it executable such as Executable UML (xUML), Executable and Translatable UML (τUML) (Mellor and Balcer 2002), etc., but these approaches are based on State Chart Variants and their goal is restricted to generate automatic code. Another variant of UML is SysML (SysML 2003) which allows modeling a system from a Systems Engineering point of view. Like UML, SysML also provides the static picture of the problem domain and does not take into account the dynamic behavior. To overcome the issue of dynamic analysis of system behavior, CPN (Colored Petri Nets) is used as a supplement to UML diagrams to make architectural specification executable. Currently, much of the work in this field is concerned with the manual transformation
process with weak model consistency checking due to semi-formal semantics in UML (Hu and Shatz, 2004) (Pettit 2000). Petri Nets’ rule-based approach has also been used to assign formal execution semantics to UML notations, (Baresil 2001). CPN profile for UML is also found in some research (Hansen and Marius, 2001). CPN is also used as standalone tool to specify and simulate a system (Kristensen, et al. 2004), but because of its weak support for static description of system architecture, this method has a limited application.

There are also a few approaches on building CPN from Ontology. Lavee, et al. (2007) presents a methodology for mapping between video event ontology and CPN. They do not provide any automated process of transformation. In (Cranefield, Purvis and Hwang 2005) ontology is used with CPN for interaction protocols application. The MITRE Corporation developed executable architecture for the Department of Defense Architecture Framework (DODAF) that is Executable Architecture Methodology for Analysis (EAMA) (Tom and S. 2003). They use various communication network models, business process model and combat simulations.

DoDADF, a system engineering tool mainly used by the U.S engineering and acquisition communities (DoDADF 2010) provides an organized way to model system engineering information of any domain into a series of architecture products, and groups them into different “views” of the system. It provides a set of elements, rules, and relationships to ensure a common representation for understanding, comparing, and integrating different architectures. In 2009, DoDADF was revamped with a new data model, the DoDADF Meta Model (DM2) in its new release DoDADF 2.0. Introduction of DM2 shifts the focus of DoDADF from output products or documentation to the collection of the underlying data.
that represent the architecture. The DM2 replaces the Core Architecture Data Model (CADM) that was referenced in previous versions of DoDAF. The DM2 has three levels of abstraction: The Conceptual Data Model (CDM) describes the high level data constructs; the Logical Data model adds technical information and clarifies relationships; and the Physical Exchange Specifications (PES) specifies data types and implementation attributes (DoDAF 2010). DM2 introduces standard vocabulary in DoDAF that helps in the definition of the purpose, scope and information requirements of the architecture up-front and also provides a mechanism to store that information (Wennergren 2009). The products in previous versions of DoDAF are renamed as models; models can be documents, spreadsheets, or other graphical representations, and serve as templates for organizing and displaying data in a more understandable format. These models populated with architectural data become the architectural view and the collections of these views are referred to as viewpoints. Use of multiple views provides the depth for any system, but also makes the model a little complex, particularly for simple scenarios.

Only a few systems architecture modeling approaches (Wagenhals, et al. 2003) (Noguera, et al. 2009) (Wang and Dagli 2011) (Wagenhals, et al. 2000) have been reported in the literature that emphasize the interactive behavior among various components of system architecture that aim to establish concordance between static and dynamic views. Wagenhals, et al. have used dataflow diagram (Wagenhals, et al. 2000) and UML (Wagenhals, et al. 2003) to define the static view of a system’s architecture and then mapped the corresponding diagrams to Coloured Petri Nets (CPN) to build executable system architecture. They give preference to their object orientation approach over
structure analysis approach due to better expressive power of UML as well as familiarity of new engineers and architects with object orientation techniques. They have proposed a process for creating executable architectures by defining mapping among UML and CPN constructs. A subset of UML diagrams are used to represent the static view of a system’s architecture. One of the main challenges in their work is the maintenance of concordance among different UML diagrams. They have addressed it by developing an integrated data dictionary manually. The data dictionary contains definition and description of each and every element of all the diagrams used in the architecture. This manual maintenance of concordance, however, could also be regarded as a major limitation of their approach as it is prone to human error. The other major weakness in their approach is the manual addition of operational rules in the dynamic view of the architecture. These limitations restrict the seamless transformation between different views of system architecture. Weak semantics of UML diagrams also raises the issue of consistency (Vanderperren and Dehaene 2005) (Bahill and Daniels 2003) (Evans 1998) (Whittle 2000) (Yi-zhi, Wang and Liu 2004) (OMG 2007) (Costal, Gómez and Guizzardi 2011). Liles (2008) automated the manual transformation proposed by Wagenhals et al. Liles created a code in Rational System Developer tool that generated a file from UML that can be opened in CPN Tools. The transformation by Liles is limited to the mapping of an activity diagram to a CPN model.

Wang and Dagli (2011) have developed a framework that implements an executable system architecture by mapping SysML notations to CPN. For this purpose, they have introduced a new transformation scheme based on SysML Sequence diagram and have established concordance among Sequence diagrams, Activity diagrams and Block diagrams. They use
the principles of model driven architecture (MDA) in their framework. For architecture evaluation purposes, they have used BRITNeY Suite, a java application that runs on top of CPN that controls the simulation and generates graphical output such as message sequence charts (MSC) and state space graphs. They use these outputs to verify and validate the system architecture. Concordance between the diagrams in static view and the transformation between static and dynamic views is not completely automatic. In addition to this, the operational rules are manually added to CPN which leaves the consistency issues open.

Noguera et al. (2009) propose a UML-based framework in combination with OWL ontology and CPN. They use combination of UML activity diagram and OWL ontology for static view modeling and CPN for dynamic view modeling. They have developed ontology for UML activity diagram that helps to preserve the consistency of an Activity diagram. The framework also contains a mapping scheme that maps OWL ontology to CPN. All the constraints/rules are defined using decision boxes in the Activity diagram, which limit the expressiveness of the modeled operational rules since Activity diagrams have limited support for rules. In addition, the use of built-in CPN functions for rules mapping also restricts the rules mapping functionality. Moreover, the semi-automated process of transformation between OWL and CPN also raises the issue of consistency.

Grady (2009) used a combination of UML, SysML and four artifacts from traditional structured analysis and developed a Universal Architecture Description Framework. The framework covers the modeling of software and hardware specification under one umbrella. According to Grady, the proposed framework is efficient, but the framework can
further be improved by forming a stronger link between analytical and synthesis process from problem space to solution space that help to strengthen the verification process of requirement through design. In the proposed framework, Grady did not say much about model execution. Likewise, Guangsheng et al. (2006) suggest a manual approach for transforming OWL DL (Description Logic) and rules defined in Semantic Web Rule Language (SWRL) into Predicate Transition net (PrT-nets). The transformation allows rule inference using Petri nets. Zou, et al. (2010) propose accountability in business services through internet using combination of OWL and SWRL. They validate the OWL ontology in combination with SWRL rules through Pellet reasoner (Clark&Parsia 2006) and model action sequence of concepts in CPN. They, however, do not suggest any mapping between SWRL and CPN and rather use rules modeled in SWRL for reasoning.

3.6 SUMMARY

The chapter firstly explained the fundamental definitions such as Systems, Systems thinking, Systems engineering/Systems Architecture, Architecture Framework and then the purpose of the model was presented. It covered some of the close related approaches that were essential to be known before any new proposal. The key issues were described in the chapter that all the above mentioned approaches aim to solve (a) a seamless transformation between different views of system architecture and (b) maintenance of consistency. It was explained in the chapter that both issues were interrelated. It was also mentioned that incomplete transformation between different views of a system’s architecture forced the architect to manually model the leftover architectural components. These manual activities become the main source of introducing inconsistency in the system’s architecture. Besides
consistency maintenance, the chapter also discussed that the existing approaches were unable to verify the behavioral aspect in the static view, which was due to insufficient support for modeling behavioral element, particularly operational rules at the abstract level within the problem domain’s architectural description. It was mentioned that even the definition of operational rules in static view alone could not solve the problem of behavior verification, unless these rules were verified through dynamic view using a suitable simulation tool. It was emphasized in the chapter that the verification of operational rules through a simulation tool also required mapping of these rules from the static view to the dynamic view. To summarize, the chapter emphasized that the existing approaches have issues with consistency maintenance, data dictionary management, operational rules handling etc. and these issues are handled with presented framework.

The next chapter will cover the technical background required to understand the presented framework.
Chapter 4

Consistent Systems Architecture Description and Behavior Framework (CSADBF)

This chapter provides the details of the proposed framework: Consistent Systems Architecture Description and Behavior Framework (CSADBF). As described earlier, the existing systems architecture modeling approaches have issues with maintaining consistency among multiple views of systems architecture. The common reasons for this inconsistency are:

a. the absence of an integrated data dictionary for systems architecture modeling
b. manual definition of operational rules in the dynamic view
c. the use of language with weak semantics for static view modeling

The CSADBF introduces OWL ontology as the starting point for modeling the static view and uses it as an integrated data dictionary. The framework helps to maintain the
consistency of system architecture by generating the dynamic view from the static view in an automated manner. For this purpose the framework proposes three mapping schemes:

1. Ontology to UML mapping scheme (Onto-UML)
2. Onto-UML to CPN mapping scheme (OUCPN)
3. SWRL rules to CPN guard conditions and arc inscription (OntoCPNRules)

The static view contains the structural part of the architecture. The combination of OWL ontology, UML class diagram and UML activity diagram form the overall static view. The dynamic view contains the functional part of the architecture which shows the interactive behavior of system entities and it comprises of CPN. The framework enables the architect to automatically create this dynamic view from the structural and behavioral description modeled in the static view. The rules mapping scheme helps an architect to create operational rules in SWRL embedded in the OWL ontology and then mapping these rules to CPN guard conditions and arc inscription. Mapping to guard condition and arc inscription increases the consistency of a system’s architecture and enables the architect to verify the behavior of these rules. Figure 4.1 shows the abstract view of the framework.

The figure contains three types of boxes: single line box, double line box and the broken line box. These boxes represent different stages of the framework. The single line box represents the stage where an architect’s involvement is required. The broken line box suggests that no architect is required whereas the double line box represents the final stage of CSADBF with partial involvement of the architect. The light gray portion of Figure 4.1 represents mapping between OWL ontology and UML. The dark gray portion represents mapping of OWL ontology, SWRL rules and UML activity diagram to CPN. These
mapping schemes are described in detail in the next chapter with illustration. Swift Attendance Card for IBA University (SAC-IBA) case study is chosen for the illustration purpose and it goes as follows:

![Diagram of Consistent Systems Architecture Description and Behavior Framework (CSADBF)](image)

**Figure 4.1.** Consistent Systems Architecture Description and Behavior Framework (CSADBF)

Before the start of each semester, students register in courses being offered. Each course has one or more sections and the students are enrolled in a section. Each student is required to get registered in sections of relevant courses. Without registration, a student cannot attend a class. Attendance in a class is marked only for students who have registered in the course during the registration period. It is worth nothing that all the operational activities at IBA are automated except attendance. Students can avail four absences in a course during a semester. A student is marked absent if he is not present at the beginning of a lecture. After the last lecture, the instructor hands over the attendance register to the
concerned program office that enters the attendance records in the database. It is quite obvious that this manual attendance process consumes around 5-10 minutes in a class of around 50 students. When aggregated over 28 lectures during a semester, this time loss would add up to 140+ minutes. A smarter attendance system, thus, would save significant amount of time which can be utilized for teaching.

In this case study, we assume that IBA has provided RFID tagged Swift Attendance Card (SAC) to every student admitted to IBA. SAC allows information to be encoded and retrieved by a radio signal. SAC would contain a unique code that IBA will match against the student’s unique code stored in the central database. Central database contains information such as student enrolled, course allocation to teachers, course offered in a semester, rooms allocation to course, overall time table, attendance of students, etc. We also assume that in every class room of IBA, RFID Access Control Readers (RACR) are installed and connected to the central database through wired connection. For simplicity, we use one room with RACR installation.

The SAC based attendance process can be described as follows: A student enters into one of the rooms equipped with RACR. If student has a SAC, he waves the card in front of the RACR. The sensor built in to the RACR reads the card and if the student is registered in a course that is scheduled in that particular class room in that time slot of the day then the sensor light gets green with a beep and then turns red. Otherwise it remains red with no beep. The information read by the sensor is sent to the central database over LAN. Student authorization is done by comparing the student code read from the tag with the student
code stored in the database and then by comparing the course, scheduled in that classroom during the current time slot, with the course in which the student is enrolled. If all conditions are satisfied then the student is marked present. Students who do not show up for the class are marked absent. In case a student comes late to the class; she is also marked absent but late arrival is stored in the database as the explanation for her absence. The instructor may mark such students present if she agrees with the explanation provided by students for coming late to the class.

SUMMARY

This chapter discussed the overall picture of the presented framework with its different components. The chapter began with the description about the reason for inconsistencies in systems architecture. Then it explained how these issues could be solved by using presented framework, CSADBF. The framework comprised of three mapping schemes: Onto-UML, OUCPN and OntoCPNRules. These mapping schemes were explained in detail in the next chapters individually.
Chapter 5

Ontology to UML mapping schemes

The previous chapter described the overview of CSADBF. This and the following chapters will be discussing the mapping schemes in detail by illustrating the steps with the help of case study. SAC-IBA case study described in the previous chapter is used for illustration purpose.

5.1 MAPPING BETWEEN ONTOLOGY AND UML

The ontology to UML mapping transforms an OWL ontology to a UML Class Diagram. In addition, it also generates a skeleton of the UML activity diagram. Overall, the mapping consists of three stages. In the first stage, the OWL ontology is manually created from the architectural description given in the narrative story of the problem domain. Strong semantics in ontology restricts inconsistencies and enables the architect to validate the architectural description at the abstract level. Once the ontology is validated, it is then used as data dictionary throughout the framework.

In the second stage, a UML class diagram is generated from the OWL ontology automatically which in turn is used in generating the skeleton of a UML Activity Diagram.
automatically. The skeleton includes swim lanes, corresponding objects/classes and the activities under each lane. Connections between different activities in the Activity diagram are created manually by the architect. It must be mentioned that if the architect wants to define a new class or activity then the architect must add the required description in the ontology first. The revised ontology will then be used to generate an updated class diagram and activity diagram. The process ensures consistency across all diagrams in the static view.

A prototype of this mapping scheme has been implemented in Java language; the tool is named as Onto-UML. Its comparison with exiting UML back-end plug-in in Protégé is given in Section 5.3. The tool accepts an OWL ontology (developed in Protégé) as input and automatically generates UML class diagram as output. For the development of this tool, two APIs are used: JENA API\(^6\) and UML2 API\(^7\). JENA API is used to extract the components of OWL ontology and UML2 API is used for the generation of UML Class diagram. Further details are given in Appendix I. The mapping scheme is presented in Table 5.1.

In CSADBF, the *concepts* are logically divided into two categories:

a. active concepts

b. non-active concepts

---

\(^6\) leading Java based open source API for OWL and RDF http://jena.sourceforge.net/

\(^7\) UML2 plug-ins for Eclipse contains UML2 API http://www.eclipse.org/modeling/mdt/?project=uml2
Active concepts are the ones which have object properties while the non-active concepts do not have object properties. These active and non-active concepts are mapped to concrete and association classes, respectively, in the UML class diagram and are also used as swimlanes and datastore node, respectively, in the UML activity diagram.

Table 5.1 Ontology to UML mapping scheme (Onto-UML)

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Define the scope and purpose of the system architecture of the problem domain.</td>
</tr>
<tr>
<td>2.</td>
<td>Find existing ontology if any ontology then move to step 5 otherwise move to step 3.</td>
</tr>
<tr>
<td>3.</td>
<td>Create Ontology of narrative story.</td>
</tr>
<tr>
<td>3.1</td>
<td>Assemble the domain vocabulary.</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Pick all common nouns and proper nouns. Represent them using concepts and concepts hierarchy in generalized to specialized or specialized to generalized order or both.</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Pick all characteristics of the nouns from narrative use cases and represent them as data type properties of the concepts.</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Pick all the verbs that are associated with nouns then represent them in ontology using object property.</td>
</tr>
<tr>
<td>3.2</td>
<td>Classify the concepts as active and non-active concepts. (Active concepts are those concepts which have object properties and non-active concepts are those which do not have object properties.)</td>
</tr>
<tr>
<td>3.3</td>
<td>Associate data type properties to concepts and define the aspects of the properties such as property value type, domain and range.</td>
</tr>
<tr>
<td>3.4</td>
<td>Define the characteristics of object properties such as domain/range concepts and cardinality.</td>
</tr>
<tr>
<td>4.</td>
<td>Generate UML class diagram from the Ontology created in Step 3 or from an existing ontology found in Step 2.</td>
</tr>
<tr>
<td>4.1</td>
<td>Map all concepts in ontology to classes in UML. All active concepts are mapped to concrete class where as non-active concept are mapped to association class. Map all concept hierarchies to class hierarchies using subclasses (UML generalization/specialization).</td>
</tr>
<tr>
<td>4.2</td>
<td>Map all properties/slots (data type properties) associated with concepts in ontology to class attributes. If a data type property is associated with more than one domain then it will map to attributes of more than one class.</td>
</tr>
<tr>
<td>4.3</td>
<td>Map all object type properties in ontology to binary UML association in the class diagram. Map domain concept of an object property to navigable end and range concept to non-navigable end in a particular UML binary association. Create corresponding operations/functions and then associate them with navigable end of a class of that particular association.</td>
</tr>
<tr>
<td>4.4</td>
<td>Map all cardinalities in Ontology to cardinalities in Class diagram.</td>
</tr>
<tr>
<td>5.</td>
<td>Create activity diagram using the narrative story of the problem and the ontology developed.</td>
</tr>
<tr>
<td>5.1</td>
<td>Draw activity symbol to enclose all the actions, control flows and other elements that make up the activity.</td>
</tr>
<tr>
<td>5.2</td>
<td>Draw swim lanes (activity partition) for every concrete class.</td>
</tr>
<tr>
<td>5.3</td>
<td>Add activity nodes (operation actions) in swim lanes using the operations from concrete classes.</td>
</tr>
<tr>
<td>5.4</td>
<td>Add datastore nodes for every association class.</td>
</tr>
<tr>
<td>5.5</td>
<td>Create control flow arcs between the activity nodes (operation actions); and object flow arcs between activity nodes (operation actions) and datastore (using narrative story).</td>
</tr>
<tr>
<td>5.6</td>
<td>Add decision nodes (with help of domain expert) and connect them with activity nodes (operation actions) using control flow arcs.</td>
</tr>
</tbody>
</table>
5.2 ILLUSTRATION OF ONTO-UML MAPPING SCHEME USING SAC-IBA

1. Define the scope and purpose of the System Architecture.

The purpose of current case study is to model the behavior of RFID tagged Swift Attendance Card (SAC) for marking attendance. The scope of the model is to show the conceptualization of IBA University attendance through SAC device.

2. Find existing ontology if finds any ontology then move to step 5 otherwise move to step 3.

There is a large collection of ontologies electronically available on the web. For example, Ontolingua ontology library (http://www.ksl.stanford.edu/software/ontolingua/) or the DAML ontology library (http://www.daml.org/ontologies/) or some of the commercial ontologies (e.g., UNSPSC (www.unspsc.org), RosettaNet (www.rosettanet.org), DMOZ (www.dmoz.org)). If there exist ontology similar to what any one going to build then reuse that ontology and enhanced further as per the requirement. For SAC-IBA, however, there does not exist any relevant ontology online, therefore, a new ontology needs to be developed from scratch.

3. Create an Ontology of narrative story.

Based on the scope of the problem domain and the desired results, only relevant nouns and verbs are then extracted and mapped to concepts, properties/slots and relations.
3.1. Assemble the domain vocabulary.

3.1.1. Pick all common nouns and proper nouns. Represent them as concepts and concepts hierarchy in generalized to specialized or specialized to generalized order or both.

Concepts: SAC, Student, RACReader, ApplicationSoftware, AttendanceDB, TimeTable, DisplayMsg, Invalid SACMsg, PresentMsg, etc.

3.1.2. Pick all characteristics associated with a noun from narrative story and represent them as datatype properties of the concept.

Datatype Properties: SACId, StudentId, CourseCode, CurrentTime, Msg, AttendanceDate, Validate, AbsentCount, RoomNo, TimeSlot, etc.

3.1.3. Pick all verbs that are associated with nouns and represent them in ontology using object properties.

Object Properties: showSAC, sendsSAC, sendStudentInfo, receiveSACInfor, verifySACId, checkSchedule, markedAttendance, etc.

3.2. Classify the concepts as active and non-active concepts.

Active Concepts: Student, RACReader, ApplicationSoftware

Non-active Concepts: SAC, AttendanceDB, TimeTable, DisplayMsg etc.

Figure 5.1 shows the complete view of SAC-IBA ontology
3.3. Associate *datatype properties* to concepts and define the aspects of the property such as property *value type, domain and range*.

A value type describes the data type of a property associated with each concept.

**Concept:** SAC

**Property:** SACId

**Value type:** Number

**Concept:** Student

**Property:** StudentId
3.4 Define the characteristics of object property such as domain/range and cardinality.

Relation: showSAC
Concepts: Student, SAC
Domain: Student
Range: SAC
Cardinality: 1..1

Relation: senseSAC
Concepts: RACReader, SAC
Domain: RACReader
Range: SAC
Cardinality: 1..1

This step is repeated for every object property.

4. Generate UML Class diagram from the Ontology created in Step 3 or from the existing ontology found in Step 2.

4.1 Map all concepts in ontology to classes in UML. All active concept are mapped to concrete class where as non-active concept are mapped to association class.
Classes: SAC, Student, RACReader, ApplicationSoftware, AttendanceDB, TimeTable, DisplayMsg, Invalid SACMsg, PresentMsg, etc.

Concrete Classes: Student, RACReader, ApplicationSoftware

Association Classes: SAC, AttendanceDB, TimeTable, DisplayMsg etc.

Map all properties (data type properties) associated with concepts in ontology to class attributes.

Attribute: SACId, StudentId, CourseCode, CurrentTime, PresentCount, Msg, AttendanceDate, Validate, AbsentCount, RoomNo, TimeSlot, etc.

4.2. Map all object type properties in ontology to binary UML association in the class diagram. Map domain concept of relation to navigable end and range concept to non-navigable end in a particular UML binary association. Create corresponding operations/functions and then associate them with navigable end of a class of that particular association.

Associations:

showSAC, sensSAC, sendStudentInfo, recieveSACInfo, verifySACId, checkSchedule, seeNotEnrolledMsg, seeInvalidSACMsg etc.

Class (Operation/function):

Student (showSAC, seePresentMsg, seeNotEnrolledMsg, seeInvalidSACMsg)

RACReader (senseSAC, verifySACId, verifySACInfo, etc.)
4.3. Map all cardinalities in ontology to cardinalities in a class diagram.

Association: showSAC
Navigable end: Student
Non Navigable end: SAC
Cardinality: 1..1

Association: recievePresentMsg
Navigable end: RACReader
Non Navigable end: PresentMsg
Cardinality: 1..*

Figure 5.2 shows the complete view of SAC-IBA class diagram generated from SAC-IBA ontology (Figure 5.1).
Figure 5.2 SAC-IBA class diagram

The figure shows that there are thirteen classes created from thirteen concepts. For every active concept a concrete class is created. Student, RACReader and Application Software are the three concrete classes. All the concrete classes have methods which are created
against the object properties of active concepts. This highlights the strong consistency between the two diagrams.

Step 5 creates the UML activity diagram from Class diagram that provides support for modeling ordering and sequence of business processes. Activity nodes, Data nodes and Swim lanes are automatically created from the UML class diagram, while Decision box, Data flow arrows and Control flow arrows are created manually with the help of a domain expert.

5  Create activity diagram using the narrative story of the problem and generated UML class diagram. (Figure 5.2)

5.1 Draw activity symbol to enclose all the actions, control flows and other elements that make up the activity.

5.2 Draw swim lanes (activity partitions) for every concrete class.

Swim lanes: Student, RACReader, ApplicationProgram

5.3 Add activity nodes (operation action) in swim lanes using the operations for concrete classes.

Activity node:

Student swim lane:

showSAC, sendsSAC, sendStudentInfo, recieveSACInfo, etc.

RACReader swim lane:

senseSAC, verifySACId, verifySACInfo, etc.

5.4 Add datastore nodes for every association classes.

Datastore node: SAC, AttendanceDB, TimeTable, DisplayMsg etc.
Create control flows arcs between the activity nodes (operation actions); and object flow arcs between activity nodes (operation actions) and datastore (using narrative story).

Manual activity with help of domain expert.

5.5 Add decision nodes (with help of domain expert) and connect them with activity nodes (operation actions) using control flow arcs.

Manual activity with help of domain expert.

Figure 5.3 shows the complete view of activity diagram of SAC-IBA case study.
Figure 5.3 SAC-IBA Activity diagram
The next section describes the comparison between the Onto-UML and existing Knublauch’s approach (Knublauch 2003)

5.3 COMPARISON BETWEEN ONTO-UML

ALGORITHM WITH THE PROTÉGÉ PLUG-IN

(KNUBLAUCH'S APPROACH)

Like Onto-UML, there exist some approaches (Vasilecas and Trinkunas 2006) (Knublauch 2003) that transform OWL ontology to other modeling languages. Vasilecas, et al. (2006)’s approach provides support for transformation of Enterprise ontology to Conceptual model in the form of Entity Relationship (ER) model and the purpose of transformation is information system development. Whereas, Knublauch (2003) provides export feature (as Protégé UML back-end plug-in) for transforming OWL ontology to UML class diagram. The transformation of Knublauch (2003) is comparatively similar to Onto-UML except the class diagram generated from protégé plug-in does not contain any function or method, and association classes which should be created from the object properties and non-active concepts, respectively. Therefore, Knublauch’s (2003) transformation doesn’t consider to be complete. The detail comparison between Onto-UML and Knublauch’s (2003) approaches is given below using SAC-IBA case study.

Protégé UML back-end plug-in provides export feature for ontologies that are developed in protégé, to UML. The supported format for ontology is Protégé knowledge model
Figure 5.1 is showing the ontology of SAC-IBA case study developed using Onto-UML algorithm.

For simplicity, only the communication between Student and RACReader is taken for comparison. Details about the concepts, data type properties and object type properties involved in this communication are listed in the table below.

**Table 5.2 Components of SAC-IBA Ontology**

<table>
<thead>
<tr>
<th>SAC-IBA Ontology</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Concepts</strong></td>
</tr>
<tr>
<td>Student, SAC, InvalidSACMsg, DisplayMsg, InvalidSACMsg, PresentMsg, NotEnrolledMsg, RACReader</td>
</tr>
<tr>
<td><strong>Data type properties/attribute</strong></td>
</tr>
<tr>
<td>SACId, StudentId, CourseCode, Msg, Validate, RoomNo</td>
</tr>
<tr>
<td><strong>Object type properties/relations</strong></td>
</tr>
<tr>
<td>showSAC, seeNotEnrolledMsg, seeInvalidSACMsg, seePresentMsg, displayInvalidSACMsg, displayNotEnrolledMsg, displayPresentMsg, receiveNotEnrolledMsg, receivePresentMsg, sendStudentInfo, sensSAC, verifySACId</td>
</tr>
</tbody>
</table>

According to Onto-UML algorithm, Student and RACReader are the *Active Concepts* whereas SAC, InvalidSACMsg, DisplayMsg, InvalidSACMsg, PresentMsg, NotEnrolledMsg are *Non-Active Concepts*. Active concepts are those concepts which have object type property whereas Non-active concepts are those concepts which do not have object type property. Non-active concepts become the part of object properties of active concepts. Table 5.3 is showing the mapping of Ontology to UML.
Table 5.3 Ontology to UML mapping using Onto-UML

<table>
<thead>
<tr>
<th>Active Concepts</th>
<th>Concrete Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-active Concepts</td>
<td>Association Classes</td>
</tr>
<tr>
<td>Data type properties/attribute</td>
<td>Attributes</td>
</tr>
<tr>
<td>Object type properties/relations</td>
<td>Methods</td>
</tr>
</tbody>
</table>

Figure 5.4 is showing UML class diagram, generated from SAC-IBA Ontology (Figure 5.1) using Onto-UML. Figure 5.5 is showing the UML class diagram generated using Protégé UML plug-in. It can be clearly seen that class functions and association classes present in Figure 5.4 are missing in Figure 5.5. Protégé UML plug-in transformed the non-active concepts to concrete classes instead of association classes.

Figure 5.4 UML class diagram generated using Onto-UML tool
Figure 5.5 UML class diagram generated using Protégé UML Plug-in (Knublauch’s approach)

Table 5.4 is showing the mapping of Ontology to UML using Protégé UML Plug-in.

**Table 5.4 Ontology to UML mapping using Protégé UML Plug-in (Knublauch’s methodology)**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Active Concepts</td>
<td>Concrete Classes</td>
</tr>
<tr>
<td>Non-active Concepts</td>
<td>Concrete Classes</td>
</tr>
<tr>
<td>Data type properties/attribute</td>
<td>Attributes</td>
</tr>
<tr>
<td>Object type properties/relations</td>
<td>Associations</td>
</tr>
</tbody>
</table>

The main difference between Protégé UML plug-in (Knublauch’s methodology) and Onto-UML, is its scope. Onto-UML algorithm is developed for Systems Architecture modeling (particularly executable System Architecture) whereas Protégé UML plug-in was developed for import/export of ontologies in multiple file formats. Table 5.5 is showing
the overall comparison of Ontology to UML transformation, between Onto-UML and Protégé Plug-in.

**Table 5.5** Comparison (of Ontology to UML mapping) between Onto-UML & Protégé UML Plug-in

<table>
<thead>
<tr>
<th>Ontology Components</th>
<th>UML (Using Onto-UML)</th>
<th>UML (Using Protégé Plug-in)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Active Concepts</strong></td>
<td>Concrete Classes</td>
<td>Concrete Classes</td>
</tr>
<tr>
<td><strong>Non-active Concepts</strong></td>
<td>Association Classes</td>
<td>Concrete Classes</td>
</tr>
<tr>
<td><strong>Data type properties/attribute</strong></td>
<td>Attributes</td>
<td>Attributes</td>
</tr>
<tr>
<td><strong>Object type properties/relations</strong></td>
<td>Methods</td>
<td>Associations</td>
</tr>
</tbody>
</table>

It can be seen in the Table 5.6 that the non-active concepts are transformed into association class using Onto-UML and concrete class using Protégé plug-in, whereas, the object properties are transformed into methods using Onto-UML and associations using Protégé plug-in.
Table 5.6. Comparison between Onto-UML and Protégé Plug-in generated Class diagram

<table>
<thead>
<tr>
<th>Comparison of UML class diagram generated using Onto-UML and Protégé Plug-in</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Concrete Class</td>
</tr>
<tr>
<td>Association Class</td>
</tr>
<tr>
<td>Class Attribute</td>
</tr>
<tr>
<td>Association</td>
</tr>
<tr>
<td>Methods</td>
</tr>
<tr>
<td>Generalization</td>
</tr>
</tbody>
</table>

Table 5.6 is showing the comparison between the components of UML class diagram generated using Onto-UML and Protégé Plug-in. It shows that the class diagram generated through Protégé UML plug-in does contain neither association classes nor class methods. Since, Association classes and methods are the essential UML components required for the model transformation to CPN executable model. Therefore, Protégé UML plug-in is not fully suitable for the presented framework.

5.4 SUMMARY

This chapter discussed the first mapping schemes (Ontology to UML) of the presented framework, CSADBF. The mapping scheme was explained step by step using SAC-IBA case study. The chapter also presented the detail comparison between the class diagram generated from Protégé Plug-in and Onto-UML (the tool developed using the first mapping scheme of CSADBF). Each diagram was discussed in details and missing artifact in Protégé Plug-in generated class diagram was identified.
Chapter 6

Onto-UML to CPN mapping schemes

Onto-UML to CPN mapping is the second mapping scheme of CSADBF, which generates CPN constructs automatically from the architectural description defined in the ontology and the static behavior modeled in the UML activity diagram. This chapter will be discussing the mapping schemes in detail by illustrating the mapping steps with the help of SAC-IBA case study.

6.1 MAPPING BETWEEN STATIC VIEW (ONTOLOGY, UML) AND DYNAMIC VIEW(CPN)

The dynamic view comprises of Coloured Petri nets (CPN) and it is generated automatically by following the steps given in Onto-UML to CPN mapping scheme presented in Table 6.1. Every element of the CPN model has a direct connection with the elements of OWL ontology and UML activity diagram. For example, places and transitions on the super page are created from the non-active concepts and active concepts, respectively whereas they (intermediate places & transitions) are created using control flow/data flow arrows (in activity diagram) and action nodes (in activity diagram)/active concepts (in OWL ontology), respectively on sub pages. The sub pages are created against
the transitions present on suppr page. The *global declarations* are created using the *data type* and *object type properties*, respectively. The *input and output arrows* in CPN are created from UML activity diagram. It must be mentioned that few intermediate places need to be defined manually for smooth process flow. The left over part of mapping SWRL *rules* on arc inscriptions or *guard conditions* is covered in the next chapter. These rules are applied based on the object property present in SWRL rules.

**Table 6.1** Onto-UML to CPN mapping scheme

<table>
<thead>
<tr>
<th>Step 1. Map all datatype properties in OWL ontology to standard declarations in ML (CPN)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Map datatype property names to color set and their range (datatypes) to respective datatypes in ML.</td>
</tr>
<tr>
<td>2. Read domain(concepts) of datatype properties, if a concept name appears as domain with more than</td>
</tr>
<tr>
<td>One datatype properties then create a product color set of those datatype properties.</td>
</tr>
<tr>
<td>3. Create variables for both types of color set created in Step 1.1 and Step 1.2</td>
</tr>
<tr>
<td>Step 2. Create super page.</td>
</tr>
<tr>
<td>1. Map all active concepts to substitution transitions.</td>
</tr>
<tr>
<td>2. Map all non-active concepts to places in CPN and assign respective color set from step 1.2</td>
</tr>
<tr>
<td>3. Create a place as starting point (that holds some tokens to start the simulation) against the class of activity partition</td>
</tr>
<tr>
<td>(swim lane) that contain initial activity node.</td>
</tr>
<tr>
<td>4. Create arcs on superpage using activity diagram. Map object flow arrows coming out of data store node to the arcs</td>
</tr>
<tr>
<td>between places and transitions and object flow arrows coming into the data store node to the arcs between</td>
</tr>
<tr>
<td>transitions and places.</td>
</tr>
<tr>
<td>5. Assign variables (created in step 1.3) to arc expressions.</td>
</tr>
<tr>
<td>Step 3. Create subpage for every substitution Transitions (created in step 2.1).</td>
</tr>
<tr>
<td>1. Delete the transition that is copied from the super page and leave the input and output ports intact</td>
</tr>
<tr>
<td>2. Create transitions on the subpage against the object properties of active concepts. The newly created transitions</td>
</tr>
<tr>
<td>correspond to the action node in the activity diagram.</td>
</tr>
<tr>
<td>3. Create intermediate places between the transitions using data that flows through control flow/data flow arrows in</td>
</tr>
<tr>
<td>the activity diagram.</td>
</tr>
</tbody>
</table>
| 4. Input and output arcs on subpages are created from control flow/data flow arrows in the activity diagram.
6.2 MAPPING SCHEME USING SAC-IBA

Step 1. Map all *datatype properties* to *standard declarations* in ML(CPN)

*Concrete Classes:* Student, RACReader, ApplicationSoftware

*Association Classes:* SAC, AttendanceDB, TimeTable, DisplayMsg etc.

*Attribute:* SACId, StudentId, CourseCode, CurrentTime, PresentCount, Msg,

AttendanceDate, Validate, AbsentCount, RoomNo, TimeSlot, etc.

*CPN Declaration*

*Standard Declarations*

```
colset SACId = int;
colset StudentId = int;
colset RoomNo = int;
colset CurrentTime = date;
colset Validation = bool;
colset CourseCode = string;
colset AttendanceStatus = string;
colset TimeSlot = int;
colset DayTitle = string;
colset Message = string;
colset SACDB = list SACId;
colset SAC = product SACId*StudentId*RoomNo;
colset tdxRoom = product StudentId*RoomNo;
```
colset CourseSchedule = product CourseCode*DayTitle*RoomNo*TimeSlot;

colset StudentEnroll = product StudentId*CourseCode;

colset StEnList = list StudentEnroll;

var sacdb: SACDB;
var sc: SAC;
var scid: SACId;
var validate: Validation;
var chkschlst: CourseScheduleList;
var crcd: CourseCode;
var stenrlst: StEnList;
var stcrtmatdy: StxCrxTmxAttxDy;
var mrkatt: Attendance;

Step 2. Create super page.

Step 2.1. Map all active concepts to substitution transitions.

Substitution Transition: Student, RACReader, Application Software
Step 2.2. Map all non-active concepts to places in CPN and assign respective color set from step 1.

Places: SAC, AttendanceDB, TimeTable, DisplayMsg etc.

Step 2.3. Create a place as starting point that holds some tokens to start the simulation.

Starting point place: Student

Step 2.4. Create arcs on superpage using activity diagram. Map object flow arrows coming out of data store node to the arcs between places and transitions and object flow arrows coming into the data store node to the arcs between transitions and places.

Input arcs: Student to SAC, RACReader to PresentMsg, RACReader to NotEnrolledMsg, etc.

Output arcs: SAC to RACReader, PresentMsg to Student, NotEnrolledMsg to Student etc.

Step 2.5. Assign variables (created in step 1) to arc expressions.

Arcs expressions: sacdb:SACDB, sc: SAC, scid: SACId, validate: Validation, chkschlst:CourseScheduleList, crcd:CourseCode, stenrlst:StEnList etc.

Figure 6.1 shows the super page created using the mapping scheme provided in Table 6.1. Ellipse representing the Places (for concrete classes) and Rectangle box representing the
Transition (for association classes). Input and Output Arcs are linking the Places with Transitions.

Figure 6.1 SAC-IBA Super page

Step 3. Create subpage for every substitution transition (created in step 2.1).

Substitution Transition: Student, RACReader, Application Program

Subpage: StudentSubPage, RACReaderSubPage, ApplicationProgramSubPage

Step 3.1. Delete the transition that is copied from the super page and leave the input and output port intact.

Step 3.2 Create transitions on subpage against the object properties of the active concepts. The newly created transitions corresponds to the action node in the activity diagram.

Transitions on Student subpage: showSAC, seePresentMsg, seeNotEnrolledMsg, seeInvalidSACMsg

Transition on RACReader subpage: senseSAC, verifySACId, verifySACInfo, sendStudentInfo, displayInvalidSACMsg,
displayPresentMsg, displayNotEnrolledMsg,
receivePresentMsg, receiveNotEnrolledMsg

Transition on Application subpage: checkSchedule, pickTimeoftheday,
receiveSACInfo, markedAttendance,
verifyStudentEnrollement, sendPresentMsg,
sendNotEnrolledMsg

Step 3.3. Create intermediate places between the transitions using data that flows through control flow/data flow arrows in the activity diagram.

Intermediate places on RACReader subpage: SACId, StudentInfo,
InvSACMsg, PresMsg,
EnrollMsg

Intermediate places on Application subpage: Time&Day, StRm,
CourseCode,
StudentAttendance,
EnrollmentMsg,
AttendanceMsg

Step 3.4. Input and output arc on subpages are created from control flow/data flow arrows in the activity diagram.

CPN Tools\(^8\) is used to draw the executable models of SAC-IBA case study. Figure 6.2 shows the CPN model generated from SAC-IBA Ontology and activity diagram without

\(^8\)CPN Tools can be downloaded from http://cpntools.org/
guard condition and if … then … else condition on Arc Inscription. Eight transitions
(Figure 6.2) are created against 8 object properties (Figure 5.1).

**Figure 6.2** SAC-IBA CPN model (without *guard condition/Arc inscription*)

### 6.3 SUMMARY

This chapter introduced the CSADBF’s second mapping scheme, Onto-UML to CPN. In
continuation with the previous chapter, the mapping scheme was illustrated step by step
using SAC-IBA case study and the output of the scheme, CPN model without *guard
condition* and *arc inscription*, was shown in Figure 6.2. Rule mapping on CPN subpage is
demonstrated in the next chapter.
Chapter 7

SWRL rules to CPN Guard Conditions and Arc Inscription Mapping Scheme (OntoCPNRules)

The previous two chapters showed how consistency is maintained across static and dynamic views with the help of the prescribed mapping schemes. To verify the behavior shown in the simulation of CPN model, however, requires definition of operational rules in static view and its mapping to dynamic view. This chapter describes the steps of third and the final mapping scheme of the presented framework that transforms operational rules in SWRL to guard conditions and arc inscription in CPN. The scheme is named as Ontology to CPN rules mapping (OntoCPNRules). The mapping scheme is also illustrated (step by step) in the chapter using SAC-IBA case study.

7.1 OntoCPNRules MAPPING SCHEME

The mapping scheme works under the assumptions given in Table 7.1. It is assumed that every SWRL must contain one individual property with at least one comparison function. Actually it is mainly for the purpose of maintaining
consistency between models. The mapping of SWRL rule to guard condition depends upon the presence of individual property atom in any rule. This individual property atom contains the object property and the active concept with which the object property is associated. The same active concept and object property is used to create the subpage in CPN and the transitions on that subpage, respectively. Since every rule is mapped to the guard condition of the corresponding transitions so the presence of exactly one individual property ensures that the rule is mapped to the guard condition of exactly one transition which is created against the same object property that is used in the rule.

None of the rules can have empty head or body even though it is allowed in SWRL. Mapping to “if..then..else” in guard conditions is based on the presence of individual property in the rules. Either a particular individual property is present in one rule or many, it is mapped on to only one “if..then..else” statement. Guards only allow comparison between single token values. It also has limited support for complex scenarios such as handling of list data types and recursive functions. List data types in combination with recursive functions enable the comparison of a token value with multiple token values of the same data type. The routing of tokens of different color sets on multiple arcs coming out of a transition is also not possible with Guard conditions as the output of if-expressions always have the same type. One solution to this problem is to use code segment instead of Guard conditions and use recursive functions that return a tuple. The other solution is to use Arc inscription and to call recursive functions from the respective arcs. Unlike code segment, Arc inscription does not restrict the number of input and output parameters as
well as the parameters do not have to be tuple only. To summarize the whole discussion, guard conditions only allow comparisons between values of single tokens. There are certain scenarios when a single token value is required to compare against multiple token values. This is only achievable using recursive functions called from Arc inscriptions. Table 7.2 shows the steps of mapping the constructs of SWRL to CPN guard conditions and arc inscription. Arc inscription mapping also helps in selecting subsequent transition based on the output of the recursive function.

The generalized syntax of SWRL rule for the Ontology to CPN rules mapping scheme is defined as:

(For Guard condition)

Individual Property atom ∧ Built-ins for comparisons atom ∧ atom ∧ atom ……→
abox:setValue built-in atom

(For Arc Inscription)

Individual Property atom ∧ Built-ins for comparisons atom ∧ atom ∧ atom ……→
abox:setValue built-in atom ∧ Individual Property
Table 7.1 Assumptions for OntoCPNRules

1. Every rule must contain exactly one individual property atom in the antecedent part because every rule will be mapped to a Guard condition of only one transition that is created against the individual property (or object property).
2. In addition to antecedent part, the rule may also contain one individual property atom in the consequent part and in that case the rule will be mapped to an arc inscription.
3. For the arc inscription there must be minimum two SWRL rules that have same individual property atoms in the antecedent part and different individual property atom in the consequent part.
4. Rules having individual property in both antecedent and consequent part cannot have same individual property atom in both parts.
5. Every rule must contain at least one SWRL built-ins for comparison.
6. A rule can use exactly one `abox:setValue` built-in function in the consequent section of the rule.
7. The head (consequent) or body (antecedent) with no atoms is allowable in the SWRL Member Submission but for the OntoCPN rules head (consequent) or body (antecedent) cannot have no-atom state.
8. No built-in function other than the SWRL built-ins is allowed for comparisons in the antecedent.
9. Disjunction (a logical OR operation) is not directly supported by SWRL but can be implemented by splitting the disjunction into separate rules.
10. Conjunction (a logical AND operation) can only be written within one rule.
11. Reading of SWRL rules in particular order is not supported.
12. Conjunctions of disjunction are not allowed but disjunctions of conjunction rules are allowed.
13. Rules with nested if statements are not supported.
14. SWRL does not provide any support to write rules for “else” part of logic. This condition would therefore be manually inserted in CPN Guard conditions.
In the generalized syntax, an individual property atom consists of an OWL object property and two arguments representing OWL individuals. For example, in the predicate hasBrother(?x, ?y), hasBrother is an OWL object property whereas ?x and ?y are the OWL individuals. Built-ins for comparison atoms are one of the set of built-ins for SWRL. There can be more than one built-in for comparison functions in one SWRL rule which becomes part of rest of the atoms. An operation rule may comprise of one SWRL rule or a set of SWRL rules. The sets are formed on the basis of common Individual Property (object property) present in more than one rule. For Arc Inscription mapping, the mapping of
SWRL rules to “if … then … else” is based on the presence of *individual property* in the *consequent part* in addition to the *individual property* present in the *antecedent part*. The steps given in Table 7.1 are explained in the next section with the help of multiple cases.

### 7.2 ILLUSTRATION OF ONTO-UML to CPN MAPPING SCHEME USING SAC-IBA

One transition *verifySACId* from Figure 6.2 is selected in the sequel to demonstrate rule mapping mechanism of OntoCPNRules. All mappings from SWRL rules to Arc Inscription are based on a particular *Individual Property atoms* present in the antecedent and the consequent part of the rules. Even if a particular *Individual Property* is present in multiple rules, it is mapped to only one *if statement*. The distinguishing factor, between the two *if statements* of Arc Inscriptions, is the presence of same individual property atom in the antecedent part and a different individual property atom in the consequent part. In case of multiple SWRL rules, the corresponding if statement on Arc inscription may contain multiple *if (conditions)* within one *if..then..else* statement or it may have multiple *else if (conditions)*. *Comparison functions* are also mapped on equivalent comparison functions in CPN.

The general criterion for all such cases is given below:
Number of SWRL rule = 1 | n where n >1

Number of Comparison Function = 1 | 2 | n where n >1

Number of common Comparison Function between rules = 1

Consequent in every rule = same | different | mix (same, different)

If any Individual Property that appears once in a rule that contain exactly one built-in for comparison function then the rule will be mapped on to one if statement with one condition.

If any Individual Property that appears in multiple SWRL rules with exactly two built-in for comparison function where one built-in for comparison function with its compared values remain same in all of those rules and the consequent also remain same then the rules will be mapped to one if statement with multiple conditions using andalso and orelse. If any Individual Property that appears once in a rule that contain multiple built-in for comparison functions then the rule will be mapped on to one if statement with multiple conditions for every built-in for comparison. A detailed explanation of the above mentioned criterion is provided in Chapter 8 with the help of eight cases. Out of those eight cases, we have picked one case and will use it to demonstrate the SWRL to Arc Inscription mapping.

Table 7.3 shows the selected case along with an example. Two or more SWRL rules (with different object property in the consequent of each rule) with exactly one comparison functions are mapped to multiple if statements with one condition. The built-in function given in the consequent part of every SWRL rule is mapped to if ... condition ... then
statement. Each *if* statement embedded in the recursive function is called from a separate 
Arc Inscription coming out of the transition which is created against the object property 
present in the consequent part of the SWRL rule.

**Table 7.3 Selected Case with example**

<table>
<thead>
<tr>
<th>SWRL</th>
<th>Arc Inscription</th>
</tr>
</thead>
<tbody>
<tr>
<td>atom ^ atom ^ ... ^ individual</td>
<td>if (condition)</td>
</tr>
<tr>
<td>property atom ^ Built-ins for</td>
<td>then</td>
</tr>
<tr>
<td>comparisons atom ^ abox:setValue</td>
<td>&lt;statement&gt;</td>
</tr>
<tr>
<td>built-in atom ^ individual property atom</td>
<td>else</td>
</tr>
<tr>
<td>atom ^ atom ^ ... ^ individual</td>
<td>&lt;statement&gt;</td>
</tr>
<tr>
<td>property atom ^ Built-ins for</td>
<td>if (condition)</td>
</tr>
<tr>
<td>comparisons atom ^ abox:setValue</td>
<td>then</td>
</tr>
<tr>
<td>built-in atom ^ individual property atom</td>
<td>&lt;statement&gt;</td>
</tr>
<tr>
<td>atom</td>
<td>else</td>
</tr>
<tr>
<td></td>
<td>&lt;statement&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;statement&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;statement&gt;</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Example: Verify SACId on the basis of following criteria: if the SACId sensed by RACReader is same as the SACId present in the DE then studentInformation (student id and room information) otherwise return "invalid SAC msg".

```prolog
SWACId(Student, ?s) ^ SACId(hasSACDS, ?db) ^ Validate(hasSACDS, ?r) ^ RoomNo(SACInfo, ?no) ^ verifySACId(?r, ?db) ^ sample(?s, ?db) ^ abox:setValue(?, "true") ^ sendStudentInfo(?, SACInfo_1) ^ SACInfo_1 = ?no + ?s
```

```prolog
if #1(sc) = sacdb
then #1(#2(sc), #3(sc))
else empty
```

(Recursive function with embedded if statement)

```prolog
fun vsacStm(sacdb, (scid, st, rm)) = if sacdb=[]
then empty
else if (scid=List.hd sacdb)
then #1(sc, rm)
else vsacStm(List.tl sacdb, (scid, st, rm));
```

(Function calling from the Arc Inscription)

```prolog
vsacStm(sacdb, sc)
```

```prolog
SWACId(Student, ?s) ^ SACId(hasSACDS, ?db) ^ Validate(hasSACDS, ?r) ^ Msg(InvalidSACMsg, ?msg) ^ verifySACId(?r, ?db) ^ differentFrom(?s, ?db) ^ abox:setValue(?, "false") ^ displayInvalidSACMsg(?r, ?msg)
```

```prolog
if #1(sc) = sacdb
then #1("Invalid SAC")
else empty
```

(Recursive function with embedded if statement)

```prolog
fun vsacEMsg(sacdb, (scid, st, rm)) = if sacdb=[]
then #1("invalid sac"
else if (scid=List.hd sacdb)
then empty
else vsacEMsg(List.tl sacdb, (scid, st, rm));
```

(Function calling from the Arc Inscription)

```prolog
vsacEMsg(sacdb, sc)
```
Figure 7.1 shows the pictorial representation of the case with recursive function calling from arc inscriptions. The figure shows the state of CPN model before and after the execution of the “verifySACId” transition. There are six places in the model: SACId, StudentInfo, hasSACDB, SACInfo, InvSACMsg and SACInvalidMsg. SACId is carrying 4 tokens that consist of SACId. StudentInfo is also carrying 4 tokens that contain RoomNo and StudentId. hasSACDB is carrying 4 tokens that consist of SACId. It can be seen in the executed model (After) that 2 tokens are verified as valid and 2 tokens are carrying “Invalid SAC” message.

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Before Model" /></td>
<td><img src="image2.png" alt="After Model" /></td>
</tr>
</tbody>
</table>

**Figure 7.1** CPN model before and after execution of verifySACId transition

Figure 7.2 shows *RACReader subpage* with *recursive function* calling from the *arc inscriptions*. The figure also shows the body of the *recursive functions* within *global declaration*. The generated *if statements* is embedded within the *recursive function*. It must be mentioned that the *recursive function* has been added manually. However, the manual
activity has been performed without affecting the overall consistency. For instance, no new color set or if condition is being introduced in the model.

Figure 7.2 RACReader subpage with Arc Inscription

Figure 7.3 shows the complete picture of the SAC-IBA case study with overall mapping between different views. For simplicity, only one subpage (RACReader) is selected for mapping. It can be seen that every component of the dynamic view (CPN) is created from the information provided in the static view (OWL ontology and activity diagram). The figure also shows that the behaviour defined in the SWRL rules have been completely mapped to CPN. The mapping, thus, proves that the transformation is seamless and consistency is maintained throughout the architecture. The OWL ontology with embedded
SWRL rules plays the role of a single repository for capturing the requirements of a problem domain and restricts inconsistency in the static view of a system’s architecture.
Figure 7.3 Graphical View of Overall mapping
The next section will describe the step by step mapping of Onto-UML to CPN mapping using SAC-IBA case study.
7.3 SUMMARY

This chapter introduced the CSADBF’s third and the final mapping scheme, OntoCPNRules. The chapter was mainly focused on the demonstration of rules mapping between SWRL rules and arch inscription. In continuation with the last couple of chapters, OntoCPNRules was illustrated step by step using SAC-IBA case study and completed the seamless transformation, one of the key features of CSADBF. In order to demonstrate the rules mapping mechanism of OntoCPNRules, the chapter used verifySACld transition from the CPN subpage (Figure 6.2). The general criterion for different cases was also described. The chapter concluded with mapping summary and the overall graphical view of mapping of CSADBF.
Chapter 8

Illustration of CSADBF using Order Processing Case Study

This chapter demonstrates the proposed methodology using another case study. The illustrative example is based on an order processing supply chain system for a factory. The description of the system is not accurate but it has been created for the purpose of explanation. A fictitious garment factory, Ideal Garment Factory (IGF) is taken as a model factory for the illustration of requirements and operational concepts.

We start with a description of how the factory works. Order Processor receives product orders of varying sizes (Small, Big, Very Big) and quantities and picks staff (Novice, Expert, Super Expert) to process orders. Currently, orders are assigned to staff members irrespective of their skill levels. As a result, staff members are not properly utilized. To improve the overall supply chain of IGF’s order processing systems, the system needs to work under the following operational rules. Small size order are assigned to Novice staff where as Big and very Big orders are given to Expert and Super Expert staff, respectively.

Two types of Equipment tools (Fast and Slow) are also given to staff based on the following criteria. Fast tools are given when the size of the order is Big or Very Big with quantity greater then 5, otherwise Slow tool is given. Based on the quantity of the order, the order processor gives the following discounts:
– 20% discount for quantity >=10
– 15% discount for quantity >=8
– 10% discount for quantity >=5
– 5% discount for quantity >=3
– Otherwise zero discount

Once the product is ready, the order processor sends the product for shipment.

8.1 ILLUSTRATION OF STATIC VIEW OF CSADBF (ONTOLOGY TO UML MAPPING SCHEME)

8.1.1 Phase I: Ontology Development

To model the desired behavior, this step extracts relevant nouns and verbs from the narration and mapped them to concepts and properties/slots. Table 8.1 shows the extracted components from the narrative story using the ontology creation scheme described in Table 5.1 (Step 3).

Figure 8.1 shows the graphical view of the OP ontology. The next phase will use this ontology as input and automatically generates UML class diagram which will be further used for behavior modeling in UML activity diagram.

8.1.2 Phase II: UML Class Diagram Creations

In Phase II of Onto-UML, OP class diagram is automatically created from OP OWL ontology as explained in Table 5.1 (Step 4). Table 8.2 shows elements of the generated UML class diagram.
It can be seen in Table 8.2 that all the active concepts such as: Person, OrderProcessor are transformed to concrete classes and non-active concepts such as Product, Equipment_Pool, Staff_Pool, Discount are transformed to association classes.

**Figure 8.1 OP Ontology**

Associations between classes are created from object properties in the ontology. Object properties are also mapped on to operations/functions of classes. Figure 8.2 shows the UML class diagram generated via Onto-UML tool. The details about Onto-UML tool are described in Appendix I.
Figure 8.2 OP class diagram
### Table 8.1 Create OP Ontology

<table>
<thead>
<tr>
<th>Step 1</th>
<th>Step 1.1</th>
<th>Step 1.2</th>
<th>Step 1.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concepts:</td>
<td>Person, Order, Product, OrderProcessor, Equipment_Pool, Staff_Pool, Discount</td>
<td>Datatype Properties/slots:</td>
<td>Object Property:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OrderQuantity, OrderSize, StaffType, EquipmentType, DiscountPercent</td>
<td>givesOrder, receivesOrder, picksStaff, picksEquipment, preparesOrder, releaseEquipment, releaseStaff, givesDiscount, releaseProduct, receivesOrder</td>
</tr>
<tr>
<td>Active Concept:</td>
<td>Person, OrderProcessor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-active Concept:</td>
<td>Order, Product, Equipment_Pool, Staff_Pool, Discount</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**This step is repeated for all the concepts and their respective attributes in the OP Ontology.**

**This step is repeated for all the properties in the OP Ontology.**

### Table 8.2 Generate OP Class Diagram from OP OWL Ontology

<table>
<thead>
<tr>
<th>Step 2</th>
<th>Step 3</th>
<th>Step 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes: Person, Order, Product, OrderProcessor, Equipment_Pool, Staff_Pool, Discount</td>
<td>Associations:</td>
<td>Class (Operation/function):</td>
</tr>
<tr>
<td>Concrete Classes: Person, OrderProcessor</td>
<td>receiveOrder, picksStaff, picksEquipment, preparesOrder, releaseEquipment, releaseStaff, givesDiscount, releaseProduct</td>
<td>Person (givesOrder, receivesProduct)</td>
</tr>
<tr>
<td>Association Classes: Product, Equipment_Pool, Staff_Pool, Discount</td>
<td></td>
<td>Order Processor(receiveOrder, picksStaff, picksEquipment, preparesOrder, releasesEquipment, releasesStaff, givesDiscount, releasesProduct)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Operation: givesOrder</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NAVIGABLE end: Person</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Non-Navigable end: Order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Operation: receivesOrder</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NAVIGABLE end: OrderProcessor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Non-Navigable end: Order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This step is repeated for all the object properties in the OP Ontology.</td>
</tr>
</tbody>
</table>

**This step is repeated for all the Relations in the OP Ontology.**

**This step is repeated for all the cardinalities in the OP Ontology.**
8.1.3 Phase III: UML Activity Diagram Creation

As per the mapping defined in Table 5.1, *swim lanes* are created against *concrete classes*, *activity nodes* within *swim lanes* are created against every operation of *concrete classes* and finally the *data store nodes* are created against *association classes*. The graphical view of OP activity diagram is shown in Figure 8.3. It shows two *swim lanes* which are created against the *concrete classes*: *Person* and *OrderProcessor*. In each *swim lane* there are *activity nodes* which are connected with other *activity nodes* through *control flow arcs* whereas *object flow arcs* connect *activity node* with *datastore nodes*. As mentioned earlier, *decision nodes* are created with the help of a domain expert.
**Figure 8.3** OP Activity diagram

Figure 8.4 shows the complete mapping of ontology to UML. The mapping from ontology to UML reflects strong concordance because each and every artifact created in the UML class diagram and activity diagram is created from the structural components defined in the ontology which results in maintenance of consistency in the architectural description.
Figure 8.4 Graphical view of mapping of Ontology to UML mapping scheme
8.2 ILLUSTRATION OF DYNAMIC VIEW OF CSADBF (ONTO-UML TO CPN MAPPING SCHEME)

In addition to maintaining consistency in the static view, the CSADBF framework also supports consistency maintenance in the dynamic view by automatic generation of CPN models. For the OP example, OWL ontology and activity diagram are used to generate a CPN. Table 8.3 is showing the elements of CPN created using the steps described in Table 6.1.
### Table 8.3 Create CPN model using Onto-UML to CPN mapping scheme

<table>
<thead>
<tr>
<th>Step 1.1 (colset and variable Declaration)</th>
<th>Step 2 (Superpage)</th>
<th>Step 2.5</th>
<th>Step 3 (Subpage)</th>
<th>Step 3.1</th>
<th>Step 3.2</th>
<th>Step 3.3</th>
<th>Step 3.4</th>
<th>Step 3.5</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>colset:</strong> OrderQuanti		y, OrderSize, StaffType, EquipmentType, DiscountPercent</td>
<td><strong>variables:</strong> ordprd:OrderProduct, PersonOrder, ProductWithDiscount, ProductShipped, OrderProductStaff, OrdPrdStfEquip</td>
<td><strong>Places:</strong> Person, Product, EquipmentPool, StaffPool, Discount</td>
<td><strong>Input Arcs:</strong></td>
<td><strong>Person Subpage</strong></td>
<td><strong>Transitions:</strong></td>
<td><strong>Input Arcs:</strong></td>
<td><strong>Output Arcs:</strong></td>
<td>Manual addition of Intermediate places</td>
</tr>
<tr>
<td>product colset: OrderProduct, PersonOrder, ProductWithDiscount, ProductShipped, OrderProductStaff, OrdPrdStfEquip</td>
<td><strong>Starting Place:</strong> Person</td>
<td><strong>Input Port:</strong> Person</td>
<td><strong>Transition:</strong> givesOrder</td>
<td><strong>Output Port:</strong> Order</td>
<td><strong>Transition:</strong> ProductReceived</td>
<td><strong>This step is repeated for all Object Property of Particular active concepts.</strong></td>
<td><strong>This step is repeated for all Object Property of Particular active concepts.</strong></td>
<td><strong>This step is repeated for all Object Property of Particular active concepts.</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Transition:</strong> OrderProcessor</td>
<td><strong>Output Port:</strong> Product</td>
<td><strong>This step is repeated for all the Subpages (active concept in ontology) on Superpage.</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The *global declarations* in CPN is created from *datatype properties* in OWL ontology while the *active concepts* in OWL ontology are transformed into *substitution transition* and *non-active concepts* into *places*. The creation of *input* and *output arcs* on the *super page* is based on the mapping of *dataset node* from the activity diagram.

![Super page created from OP OWL ontology](image)

**Figure 8.5** Super page created from OP OWL ontology

Figure 8.5 shows *super page* of the CPN model created in CPN tools. In addition to super page, the mapping scheme also creates *subpages* for every *substitution transition*. Figure 8.6 shows *subpage* for *OrderProcessor substitution transition*. 
So far, all the transitions are without guard conditions. The next subsection describes how guard conditions are created from the operational rules defined in the static view using OntoCPNRules methodology.

8.3 ILLUSTRATION OF OPERATIONAL RULES MAPPING

To illustrate Ontology to CPN rules mapping scheme, we use OWL ontology and global declaration of CPN created in the previous steps. The OWL ontology constructs are used in SWRL rules whereas CPN global declarations are used in CPN guard conditions.
The eight transitions shown in Figure 8.6 are created against the eight object properties defined in OWL ontology (Figure 8.2). Three transitions from the figure, Receive order, Receive Equipment and Give Discount, are selected in the sequel to demonstrate rule mapping by OntoCPNRules using eight different cases. Each case illustrates a scenario with the help of the OP ontology. All mappings from SWRL rules to guard conditions are based on the presence of individual property atom. Whether a particular individual property is present in a single rule or multiple rules, it is mapped to only one if statement. In case of multiple SWRL rules, the corresponding guard conditions may contain multiple if (conditions) within one if..then..else statement or it may have multiple else if (conditions). Comparison functions are also mapped to equivalent comparison functions in CPN. The general criterion for all the cases is given below:

Number of SWRL rule = 1 | n where n >1

Number of Comparison Function = 1 | 2 | n where n >1

Number of common Comparison Function between rules = 1

Consequent in every rule = same | different | mix (same, different)

Table 8.5 shows the eight cases of mapping SWRL rules to CPN guard conditions. To explain the overall functionality, three cases: Case 2, 6 and 8 have been picked and are explained in detail below.
Case 2:

Table 8.4 shows Case 2 which has a single SWRL rule with multiple comparison functions. The resulting if statement obtained after mapping contains multiple conditions and andalso logical operator.

Figure 8.7 demonstrates Case 2 using the Receive Equipment transition. There are three places in the model: Order Staff, Equipment Pool and Order Staff with Equipment. Order Staff is carrying 4 tokens that carry product size and product quantity details.

<table>
<thead>
<tr>
<th>SWRL</th>
<th>Guard Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>atom ∧ atom ∧ ... ∧ individual property</td>
<td>If (condition) andalso (condition)</td>
</tr>
<tr>
<td>atom ∧ built-ins for comparisons atom</td>
<td>andalso (condition)...</td>
</tr>
<tr>
<td>∧ built-ins for comparisons atom</td>
<td>then</td>
</tr>
<tr>
<td>Built-ins for comparisons atom</td>
<td>&lt;statement&gt;</td>
</tr>
<tr>
<td>......abox:setValue built-in atom</td>
<td>else</td>
</tr>
<tr>
<td>If (condition) andalso (condition) andalso (condition)...</td>
<td>&lt;statement&gt;</td>
</tr>
<tr>
<td>then</td>
<td></td>
</tr>
<tr>
<td>&lt;statement&gt;</td>
<td></td>
</tr>
<tr>
<td>else</td>
<td></td>
</tr>
<tr>
<td>&lt;statement&gt;</td>
<td></td>
</tr>
<tr>
<td>Example: Pick Fast Equipment when the product size is Big and product quantity is greater than equal to 5.</td>
<td>[if #1(ordprd)=Big andalso #2(ordprd) &gt;=5 then equipment=Fast else ...]</td>
</tr>
</tbody>
</table>

Equipment Pool is carrying 2 tokens containing two types of equipment: Fast and Slow. The after execution state of the CPN model shows that all 4 tokens are given “Slow” equipment because none of the token is satisfying the guard condition which is “Big” size order with quantity greater than 5.
### Table 8.5 Eight cases with SWRL to guard condition mapping

<table>
<thead>
<tr>
<th>Case #</th>
<th>SWRL</th>
<th>Guard Condition</th>
</tr>
</thead>
</table>
| Case #1 | Number of SWRL rule = 1  
Number of comparison Function = 1 | number of if.. condition = 1 |
| Case #2 | Number of SWRL rule = 1  
Number of Comparison Function = n | number of conditional statement within if.. condition = n  
logical operator = andalso |
| Case #3 | Number of SWRL rule = n  
Number of Comparison Function in every rule = 1  
Consequent in every rule = same | number of conditional statement within if.. condition = n  
logical operator = oralso  
else if condition = no |
| Case #4 | Number of SWRL rule = n  
Number of Comparison Function in some of the rules = n  
Number of Comparison Function in rest of the rules = 1 | number of conditional statement within if.. condition = n  
else if condition = no  
logical operator = andalso  
logical operator = oralso |
| Case #5 | Number of SWRL rule = n  
Number of Comparison Function in every rule = 2  
Consequent in every rule = same | number of conditional statement within if.. condition = n  
else if condition = no  
logical operator = andalso  
logical operator = oralso |
| Case #6 | Number of SWRL rule = n  
Number of Comparison Function in every rule = 1  
Consequent in every rule = different | number of conditional statement within if.. condition = n  
else if condition = yes |
| Case #7 | Number of SWRL rule = n  
Consequent in every rule = different | number of conditional statement within if.. condition = n  
else if condition = yes |
| Case #8 | Number of SWRL rule = n  
Consequent in some rule = same  
Number of Comparison Function in some rules = n  
Number of Comparison Function in other rules = 1 | number of conditional statement within if.. condition = n  
else if condition = yes  
logical operator = andalso  
logical operator = oralso |
Figure 8.7 CPN model before and after execution of Receive Equipment transition

Case 6:

Table 8.6 shows Case 6 which models a situation where we have multiple SWRL rules containing single comparison function and every rule is having different consequent. The if statement is generated with else if conditions which is due to the presence of different consequent in every rule. Every comparison function is mapped between if and else if conditions with single conditional statement. No if condition or else if condition contain more than one conditions.
Table 8.6 Case 6 with example

<table>
<thead>
<tr>
<th>SWRL</th>
<th>Guard Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>• atom ∧ atom ∧ ... ∧ atom ∧ atom ∧ atom ∧ atom ∧ atom ∧ atom</td>
<td>If (condition) then &lt;statement&gt; else if (condition) then &lt;statement&gt; else if (condition) then &lt;statement&gt; else &lt;statement&gt;</td>
</tr>
<tr>
<td>• atom ∧ atom ∧ ... ∧ atom ∧ atom ∧ atom ∧ atom ∧ atom ∧ atom</td>
<td>• atom ∧ atom ∧ ... ∧ atom ∧ atom ∧ atom ∧ atom ∧ atom ∧ atom</td>
</tr>
<tr>
<td>• atom ∧ atom ∧ ... ∧ atom ∧ atom ∧ atom ∧ atom ∧ atom ∧ atom</td>
<td>• atom ∧ atom ∧ ... ∧ atom ∧ atom ∧ atom ∧ atom ∧ atom ∧ atom</td>
</tr>
</tbody>
</table>

Example: Pick the staff to process orders on the basis of following criteria: Novice staff when the order size is Small, Expert staff when the order size is Big and Super Expert staff when the order size is Very Big.

- OrderSize(Order, ?o) ApickStaff(?op, ?s) Aswrlb:equal(?o, "Big")
  abox:setValue(?s, "Expert")
- OrderSize(Order, ?o) ApickStaff(?op, ?s) Aswrlb:equal(?o, "Small")
  abox:setValue(?s, "Novice")
- OrderSize(Order, ?o) ApickStaff(?op, ?s) Aswrlb:equal(?o, "VeryBig")
  abox:setValue(?s, "SuperExpert")

Figure 8.7 demonstrates Case 6 with the Pick Staff transition. The after execution state of the CPN model shows that all 4 tokens satisfy the guard condition. 1 token is given Expert Staff because it qualify the guard condition of Big size order, 1 token is given Super Expert Staff because of Very Big size order whereas the remaining 2 tokens are given Novice Staff because of Small size order.
Case 8:

Table 8.7 shows Case 8. The scenario gets very complex as we allow *mix (same or different) types of Consequents* in multiple rules with *single comparison function* as well as *multiple comparison functions*. Case 8 comprises of multiple sets of rules with common individual property. Each set is differentiated on the basis of different *Consequent*. On the whole Case 8 follows a super set of mappings that consist of mappings defined in Case 2, 3, 4, 5 and 6. Different *Consequent* in every set of rules results in the mapping of rules to separate *else if conditions* (as in Case 6). If a subset of rules has both *multiple comparison functions* and *single comparison function* then mapping defined in Case 4 or 5 is used. If a subset has only a single comparison function then it follows the mapping of Case 3 otherwise in case of set of rules containing *multiple comparison functions*, Case 2 is followed.
### Table 8.7 Case 8 with example

<table>
<thead>
<tr>
<th>SWRL</th>
<th>Guard Condition</th>
</tr>
</thead>
</table>
| **Rules with multiple comparison function with different consequents:** | If ((condition) andalso (condition)) orelse (condition)...
| • atom ∧ atom ∧ ... ∧ atom ∧ built-ins for comparisons atom ∧ built-ins for comparisons atom...
| → abox:setValue built-in atom                                        | then<br>_else if (condition) orelse (condition) then<br>_else if (condition) orelse (condition) then<br>_else if (condition) then<br>_else ... |
| • atom ∧ atom ∧ ... ∧ atom ∧ built-ins for comparisons atom ∧ built-ins for comparisons atom...
| → abox:setValue built-in atom                                        | else<br>_else if (condition) then<br>_else if (condition) then<br>_else if (condition) then<br>_else ... |

**Example:** Give discount on orders on the basis of the following criteria: 20% discount to Big size orders with quantity 5 or more, 20% discount to any order with quantity greater than 8, 15% discount to Small size order with quantity 5 or more, 10% discount to Big size orders with quantity 3 or more, 5% discount to any order with quantity 3 or more.

- OrderSize(Order, ?o) ∧ OrderQuantity(Order, ?oq) ∧ giveDiscount(?op, ?d) ∧ swrlb:equal(?o, "Big") ∧ swrlb:greaterThanOrEqual(?oq, "5") → abox:setValue(?d, "20")
- OrderQuantity(Order, ?oq) ∧ giveDiscount(?op, ?d) ∧ swrlb:greaterThanOrEqual(?oq, "8") → abox:setValue(?d, "20")
- OrderSize(Order, ?o) ∧ OrderQuantity(Order, ?oq) ∧ giveDiscount(?op, ?d) ∧ swrlb:equal(?o, "Small") ∧ swrlb:greaterThanOrEqual(?oq, "5") → abox:setValue(?d, "15")
- OrderSize(Order, ?o) ∧ OrderQuantity(Order, ?oq) ∧ giveDiscount(?op, ?d) ∧ swrlb:equal(?o, "Big") ∧ swrlb:greaterThanOrEqual(?oq, "3") → abox:setValue(?d, "10")
- OrderQuantity(Order, ?oq) ∧ giveDiscount(?op, ?d) ∧ swrlb:greaterThanOrEqual(?oq, "3") → abox:setValue(?d, "5")

[if (#1(ordprd)=Big andalso #2(ordprd)>=5) orelse #2(ordprd)>=8 then ds=20
else if #1(ordprd) = Small andalso #2(ordprd)>=5 then ds=15
else if #1(ordprd)=Big andalso #2(ordprd)>=3 then ds=10
else if #2(ordprd)>=3 then ds= 5
else ...]
Figure 8.8 illustrates Case 8 with the help of the Give Discount transition. The after execution state of the CPN model shows that 2 tokens are given 20 percent discount which qualifies the criteria of order quantity greater than 8. One token of Big size order with order quantity greater than 3 got 10 percent discount and one token that qualify the criterion of order quantity greater than 3 received 5 percent discount.

![Diagram of CPN model before and after execution of Pick Staff transition](image)

Figure 8.9 CPN model before and after execution of Pick Staff transition

Figure 8.9 shows a complete view of the mapping schemes proposed in this dissertation. For simplicity, only one CPN subpage is shown in the mapping. It can be seen that almost every component of the CPN subpage is created with the help of mappings described in the previous sections. The figure shows that transitions, places and guard conditions are created from OWL ontology and SWRL rules whereas the arcs are created using UML activity diagram. The guard conditions (highlighted with oval callout box) are created against the three sets of SWRL rules (Case 6, Case 2 and Case 8). The object properties in SWRL (highlighted with ellipse) are responsible for the mapping of these rules to relevant
transition in CPN. This clearly shows that consistency is maintained across each view of the architecture.
Figure 8.10 Graphical view of mapping of OUCPN and OntoCPNRules
8.4 SUMMARY

This chapter illustrated the presented framework CSADBF using Order Processing case study and explained that CSADBF supports two types of consistency: specification consistency and functional consistency. The specification consistency suggested that the architectural description remained consistent across all diagrams: OWL ontology, UML class diagram, UML activity diagram and CPN. The functional consistency, on the other hand, dealt with consistency of behavior modeled in both static and dynamic views, that was, the CPN simulation should be in accordance with the operational rules defined in SWRL rule and the activities modeled in UML activity diagram. The illustration clearly showed that the framework was expressive and extensible and could be applied to any scenario with similar level of complexity demonstrated in our work. The chapter had demonstrated how both levels of consistency are achieved by the proposed CSADBF framework.

The overall purpose of this chapter was to show that CSADBF is a generalized framework and other purpose was to know that it was almost impossible to generate a complete CPN model from OWL ontology with embedded SWRL without any manual interference but at the same time it was possible to maintain the consistency of the artifacts used and the rules modeled. This ensured structural and behavioral consistency for a system’s architect.
Chapter 9

CSADBF Transformation Process and Its Evaluation

The previous chapter demonstrates CSADBF using another case study. In this chapter correctness of CSADBF is evaluated. In transformation systems, correctness is based on information capacity. The three mapping schemes that CSADBF comprised, involves transformation between the different views of systems architecture. Therefore, correctness is also important in the case of CSADBF. A correct and lossless transformation ensures that all the components in a source model are represented in the target model. The evaluation criteria for the lossless and correct transformation are defined:

a. Preservation of information capacity:

This criterion finds out whether the source model (i.e., OWL ontology) is completely transformed into the target model (i.e., CPN).
b. Correct identification and transformation of components:

The correctness of the transformation is evaluated by this criterion. It is significant to find the quality of the generated CPN model.

9.1 OPERATIONAL GOALS

Every transformation system has different goals and operational requirements which should be identified explicitly. These goals should be analyzed carefully to check the correctness of the system and to prevent from occasional errors and misconceptions. Therefore, a collection of operational goals are identified for developing correctness criteria. Table 9.1 shows the main operational goals that are defined for measuring the quality and correctness of the generated CPN using CSADBF.

Table 9.1 The Operational Goals

<table>
<thead>
<tr>
<th>Goal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1: Building standard ontology by using default heuristic rules.</td>
<td>Some default rules are used for building Ontology (i.e., an OWL concept for every Noun (subject &amp; object), OWL data type properties for characteristics attributes of nouns and OWL object properties for verbs (actions)).</td>
</tr>
<tr>
<td>G2: Preserving requirement definition (narrative story) of the problem domain using a formal modeling language.</td>
<td>Ensuring the preservation of requirement definition in the form of integrated data dictionary using a formal language (OWL ontology).</td>
</tr>
<tr>
<td>G3: Preserving consistency of static view (automatically generating UML class diagram &amp; UML activity diagram from Owl ontology).</td>
<td>Onto-UML mapping scheme is prescribed to preserve the static view consistency. Lossless mapping is done between OWL ontology and UML class diagram; and between UML class diagram and activity diagram.</td>
</tr>
<tr>
<td>G4: Handling of active concept, non-active concept and object property properly during the transformation</td>
<td>The steps given in Onto-UML mapping scheme help to distinguish between active and non-active concepts and transform them into concrete and</td>
</tr>
</tbody>
</table>
G5: Preservation of consistency while taking assistance from domain expert for modeling ordering and sequencing of actions in activity diagram.

G6: Preserving consistency between static view and dynamic view (without rules modeling)

G7: Ensuring behavioral consistency between static and dynamic view.

G8: Proper handling of complex operational rules in both static view and dynamic view.

<table>
<thead>
<tr>
<th>Goals</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G5</td>
<td>Preservation of consistency while taking assistance from domain expert for modeling ordering and sequencing of actions in activity diagram. The domain expert is only allowed to create arrows between the existing nodes.</td>
</tr>
<tr>
<td>G6</td>
<td>Automatic creation of CPN model from the combination of OWL ontology and UML activity diagram using Onto-UML to CPN mapping scheme.</td>
</tr>
<tr>
<td>G7</td>
<td>Ensuring proper definition of operational rules in OWL ontology (static view) using SWRL and its transformation in the form of CPN guard conditions and arc inscriptions (dynamic view). Simulations feature of CPN can be used to verify the behavioral consistency between the views.</td>
</tr>
<tr>
<td>G8</td>
<td>Use of SWRL enables the handling of complex rules in static view. Guards in CPN only allow comparison between single token values. They also have limited support for complex scenarios such as handling of list data types and recursive functions. List data types in combination with recursive functions enable the comparison of a token value with multiple token values of the same data type.</td>
</tr>
</tbody>
</table>

G1 to G5 goals are achieved using Onto-UML mapping scheme. G6 is achieved through Onto-UML to CPN mapping scheme. G7 and G8 goal are achieved using OntoCPNRule mapping scheme.

In the next sections, the correctness and accuracy of CSADBF has been proved through (i) mathematical proof and (ii) experimental results.
9.2 MATHEMATICAL PROOF

On the basis of information capacity, the correctness of the proposed framework is evaluated. The mappings which preserve information capacity should completely transform the source model into the target model. This target model should be reversible so that it can be transformed back to the original source model (Miller, Yannis and Ramakrishnan 1993).

The following definitions have been taken from (Miller, Yannis and Ramakrishnan 1993):

**Definition:** Let A and B be sets. A mapping (binary relation) \( f: A \rightarrow B \) is **functional** if for any \( a \in A \) there exists at most one \( b \in B \) such that \( f(a) = b \); **injective**, if its inverse is functional; **total**, if it is defined on every element of \( A \); and **surjective** (onto) if its inverse is total. A functional, injective, total, surjective mapping is a **bijection**.

**Definition 2:** An information capacity preserving mapping between the instances of two schemas \( S_1 \) and \( S_2 \) is a total, injective function \( f:I(S_1) \rightarrow I(S_2) \).

**Definition 3:** \( S_2 \) dominates \( S_1 \) through \( f \), iff \( f:I(S_1) \rightarrow I(S_2) \) is an information capacity preserving mapping, such that \( S_1 \preceq S_2 \).

In CSADBF, OWL ontology is a source model and CPN is the target model. UML class diagram and UML activity diagram are the intermediate models. In the context of different transformation and integration tasks, (Miller, Yannis and Ramakrishnan 1993) identified a
set of operational goals. According to these goals, the operational goal of the proposed framework is:

*The target model (i.e., CPN) can be used to trace back to the source model (i.e., OWL ontology).* To achieve this goal, a total function $f: I(S_1) \rightarrow I(S_2)$ is needed as defined above, but it is also needed that $f$ must not lose any information. An instance of $S_1$ should uniquely determine an instance of $S_2$, i.e., $f$ should also be injective so that its inverse $f^{-1}$ is well-defined. Therefore, $f$ must be an information capacity preserving mapping for achieving this goal, therefore, $S_1 \preceq S_2$.

### 9.3 LEMMAS

Four lemmas are provided to show that the transformation is information capacity preserving and is correct by adopting the evaluation technique of (Miller, Yannis and Ramakrishnan 1993). Before defining the lemmas, some notations and concepts are defined which are used in evaluation.

- Let $T$ be a transformation, $S_1$ be a source schema (i.e., OWL ontology) and $S_2$ be a target schema (i.e., CPN). For transforming OWL ontology into CPN, some intermediate models (UML class diagram and UML activity diagram) $M_{CL}$ and $M_{AD}$ have been created so $T$ is divided into three sub transformations, $T_1$, $T_2$ and $T_3$. $T_1$ is in charge of creating intermediate models $M_{CL}$ from $S_1$ and $M_{AD}$ from $M_{CL}$. $T_2$ is responsible for transforming
the combination of $S_1$ and $M_{AD}$ into $S_2$. $T_3$ is responsible for transforming the SWL rules in $S_1$ into the guard conditions in $S_2$.

- Let $AC, NC, OP, DP$ and $SR$ be the sets which represent active concepts, non-active concepts, object property, datatype property and SWRL rule of source ontology $SO$, respectively. Let $DP_i$ be a set of datatype properties owned by either $AC_j$ and $NC_k$ or both, and $OP_i$ be a set of object properties only owned by $AC_j$.

- $DP = \{DP_i : DP_i$ is datatype property\}$
- $OP = \{OP_i : OP_i$ is object property\}$
- $SR = \{SR_i : SR_i$ is SWRL rule\}$
- $AC_i = \{DP_k, OP_j : DP_k$ is datatype property owned by $AC_i, OP_j$ is object property owned by $AC_i\}$
- $NC_i = \{DP_j : DP_j$ is datatype property owned by $NC_i\}$
- $AC = \{AC_i : AC_i$ is active concept\}$
- $NC = \{NC_i : NC_i$ is active concept\}$

9.3.1 Lemma 1

Let $C(SO)$ denote the set of all concepts of some OWL ontology $SO$ and $C_i \in C(SO)$.

The categories of the concepts are identified by the classification imposed on $C(SO)$ by $T_1$. These categories are correct iff

$$\forall C_i \in (AC \lor NC)$$
Proof: $T_1$ divides the $C(SO)$, such that each concept $C_i$ falls exactly into its one category.

One property, $hasObjectproperty$ has been defined for explaining the conditions.

Following cases are used to identify correct category of $C_i$:

Case 1: $C_i \in AC \iff \neg(C_i \cap \forall hasObjectproperty. \bot)$

Description: $C_i$ is an $AC$ iff $C_i$ has object properties.

Case 2: $C_i \in NC \iff (C_i \cap \forall hasObjectproperty. \bot)$

Description: $C_i$ is an $NC$ iff $C_i$ does not have object properties.

9.3.2 Lemma 2

Classes $CL$ of intermediate model (class diagram) $M_{CL}$ are created by $T_1$. $T_1$ is correct and information capacity preserving transformation iff

- a. concrete class $CL_{AC}$ of $M_{CL}$ are created for $\forall C_i \in AC$, such that $CL_{AC} \subseteq CL$
- b. association class of $CL_{NC}$ of $M_{CL}$ are created for $\forall C_i \in NC$, such that $CL_{NC} \subseteq CL$

Therefore, $CL_{AC} \sqcup CL_{NC} = CL$

Proof: This part of the transformation is total because

- a. $CL_{AC_i}$ is created as a concrete class of intermediate model (class diagram) for each concept $C_i$ which comes under the category of active concept. $CL_{AC_i}$ is a concrete
class because it contains methods that come from object properties $OP$ of active concept. Therefore, $CL_{AC} \subseteq CL, OP_i \subseteq OP, OP \in AC, OP \neq \emptyset$

b. $CL_{NC_i}$ is created as an association class of intermediate model (class diagram) for each concept $C_i$ which comes under the category of non-active concept. $CL_{NC_i}$ is an association class because it does not contain methods which is because of absence of object properties. Therefore, $CL_{NC} \subseteq CL, OP_i \subseteq OP, OP \notin NC, OP = \emptyset$.

The classes of intermediate model (class diagram) are created for each OWL ontology concept; The example is shown in Figure 5.2. Overall, there is thirteen concepts in SAC-IBA case study. Three concrete classes are created for three active concepts and ten association classes are created for ten non-active concepts.

The transformation is also injective since its inverse maps

- all concrete classes $CL_{AC}$ that have methods back to original active concept of $OWL$ ontology.
- all association classes $CL_{NC}$ that do not have any methods back to original non-active concept of $OWL$ ontology.

Therefore, this part of transformation is proved to be total and injective.
9.3.3 Lemma 3

Let $P(C_i)$ denote the set properties of concept $C_i$. $P(C_i)$ is further divided into two subsets $DP(C_i)$ and $OP(C_i)$ which represent the set of datatype properties and object properties, respectively, such that $DP(C_i) \subseteq P(C_i)$ and $OP(C_i) \subseteq P(C_i)$. Here $DP_i \in DP(C_i)$ and $OP_i \in OP(C_i)$. $T_1$ is a correct transformation iff

a. class member variables $CL(MV)$ are created $\forall DP_i \in C_i$ and added to their respective classes $CL_j$.

b. class member function $CL(MF)$ are created $\forall OP_i \in C_i$ and added to their respective classes $CL_j$.

c. association between classes $CL(AS)$ are created $\forall OP_i \in C_i$ and added in between concrete class $CL_{AC}$ and association class $CL_{NC}$.

**Proof:** This part of the transformation is total because

a. $CL(MV_i)$ is created as a class member variable of class $CL$ of intermediate model (class diagram) for each property $DP_i$ which is a datatype property of concept $C_i$.

b. $CL(MF_i)$ is created as a class member function of class $CL$ of intermediate model (class diagram) for each property $OP_i$ which is an object property of concept $C_i$.

d. $CL(AS_i)$ is created as an association between concrete class $CL_{AC}$ and association class $CL_{NC}$ of intermediate model (class diagram) for each property $OP_i$ which is an object property of concept $C_i$. 
The transformation is also injective since its inverse maps all class member variables \( CL(MV) \) and class member function \( CL(MF) \) back to their original \( S_1 \) properties because all properties of OWL ontology are maintained by the transformation. Therefore, it is concluded that this part of the transformation is both \textbf{total} and \textbf{injective}, thus, it is information capacity preserving transformation.

9.3.4 Lemma 4

Swim lanes \( SL \) and Data store node \( DN \) of intermediate model (activity diagram) \( M_{AD} \) are created by \( T_1 \). \( T_1 \) is correct and information capacity preserving transformation \textit{iff}

\begin{enumerate}[a.]
  \item swim lanes \( SL \) of \( M_{AD} \) are created for \( \forall CL_i \in CL_{AC} \)
  \item data store node \( DN \) of \( M_{AD} \) are created for \( \forall CL_i \in CL_{NC} \)
\end{enumerate}

\textbf{Proof:} This part of the transformation is total because

\begin{enumerate}[c.]
  \item \( SL \) is created as a swim lane of intermediate model (activity diagram) for each class \( CL_i \) which comes under the category of concrete class.
  \item \( DN \) is created as a data store node of intermediate model (activity diagram) for each class \( CL_i \) which comes under the category of association class.
\end{enumerate}

The swim lanes of intermediate model (activity diagram) are created for concrete classes of intermediate model (class diagram) which are created for each OWL ontology concept;
The example is shown in Figure 5.3. Overall, there are thirteen concepts in SAC-IBA case study. Three swim lanes are created for three concrete classes which are created for three active concepts and ten data store nodes (with two duplicate data store nodes) are created for ten association classes which are created for ten non-active concepts (as shown in Figure 5.3).

The transformation is also injective since its inverse maps

- all swim lanes SL back to original concrete classes of class diagram.
- all data store nodes DN back to original association classes of class diagram.

Therefore, this part of the transformation is proved to be total and injective.

### 9.3.5 Lemma 5

Let $CM(CL_i)$ denote the class members of class $CL_i$. $CM(CL_i)$ is further divided into two subsets $MV(CL_i)$ and $MF(CL_i)$, which represent the set of member variable and member function, respectively, such that $MV(CL_i) \subseteq CM(CL_i)$ and $MF(CL_i) \subseteq CM(CL_i)$. Here, $MV_i \in MV(CL_i)$ and $MF_i \in MF(CL_i)$. $T_1$ is a correct transformation iff

a. activity diagram arrow inscription $AD(AI)$ are created $\forall MV_i$

b. activity diagram action node $AD(AN)$ are created $\forall MF_i$ and added to their respective swim lanes $SL_i$. 

133
**Proof:** This part of the transformation is total because

c. \( AD(AI_i) \) is created as an activity arrow inscription of arrow \( A_j \) of intermediate model (activity diagram) for each class member \( CL(MV_i) \) which is a class member variable of class \( CL_i \).

d. \( AD(AN_i) \) is created as a activity action node of arc \( A_j \) of intermediate model (activity diagram) for each class member \( CL(MF_i) \) which is a class member function of class \( CL_i \).

The transformation is also injective since its inverse maps all activity arrow inscription \( AD(AI) \) and activity action node \( AD(AN) \) back to their original \( S_1 \) class members because all class members of UML class diagram are maintained by the transformation. Therefore, it is concluded that this part of transformation is both **total** and **injective**. Thus, it is information capacity preserving transformation.

**9.3.6 Lemma 6**

Given the source schema OWL ontology \( S_1 \) and the intermediate models (UML class diagram and UML activity diagram) \( M_{CL} \) and \( M_{AD} \), the transformation \( T_2 \) is an information capacity preserving transformation iff

a. colorset \( CS \) are created \( \forall DP_i \) and added to the global declaration of CPN model

b. substitution transitions \( ST \) are created \( \forall AC_i \) and added to the CPN super page.
c. places $P$ are created $\forall NC_i$ and added to the CPN super page and assigned a respective colorset.

d. arcs $AR(SP)$ are created $\forall DA_i$ and added to the CPN super page between places and transition $s$, and transitions and places.

e. transitions $TR$ are created $\forall OP_i$ and added to the respective CPN subpage.

f. arcs $AR(SB)$ are created $\forall A_i$ and added to the respective CPN subpage between places and transitions, and transitions and places.

**Proof:** The transformation is total because

a. $CS_i$ is created as a colorset of global declaration in the target schema CPN model $S_2$ for each property $DP_i$ which is a datatype property of concept $C_i$. Each $CS_i$ is assigned to respective places.

b. $ST_i$ is created as a substitution transition on super page in the target schema CPN model $S_2$ for each concept $AC_i$ which is an active concept of concept $C_i$.

c. $P_i$ is created as a place on super page in the target schema CPN model $S_2$ for each concept $NC_i$ which is a non-active concept of concept $C_i$.

d. $AR(SP)_i$ is created as an arc on the super page in the target schema CPN model $S_2$ for each arrow $DA_i$ which is data flow arrow of arrow $A_i$. Arcs are created between each place $P_i$ and substitution transition $ST_i$ and vice versa.

e. $TR_i$ is created as a transition on subpage in the target schema CPN model $S_2$ for each property $OP_i$ which is an object property of concept $C_i$. 

135
f. \( AR(SB) \) is created as an arc on the subpage in the target schema CPN model \( S_2 \) for each arrow \( CA_i \) which is control flow arrow of arrow \( A_i \) and each arrow \( DA_i \), which is data flow arrow of arrow \( A_i \). Arcs are created between each place \( P_i \) and transition \( TR_i \) and vice versa.

The transformation is also injective since its inverse maps CPN places and transitions, and its colorset back to the component of source models because information about OWL concepts, properties and activity diagram arrows are properly maintained by the transformation. Therefore, it is proved that the transformation is both total and injective and preserves the information capacity.

9.3.7 Lemma 7

Given SWRL rules with the source schema OWL ontology \( S_1 \), the transformation \( T_3 \) is an information capacity preserving transformation iff

a. guard conditions \( GC \) are created \( \forall SR_i \), when object property \( OP \) is present only in the antecedent part of SWRL rule

b. arc inscriptions \( AI \) are created \( \forall SR_i \), when object property \( OP \) is present in both the antecedent and consequent part of SWRL rule
Proof: The transformation is total because

c. \( G_C_i \) is created as a guard condition in the target schema CPN model \( S_2 \) for each SWRL rule \( SR_i \) or set of \( SR_i \). \( GC \) is attached to the transition \( TR \) present on the subpage of CPN model. The creation of \( GC \) and its attachment to the transition \( TR \) depends upon the presence of object property \( OP_i \) in each \( SR_i \). If there are multiple rules \( SR_i \) having same \( OP_i \) then the set of multiple rules will be mapped onto one \( GC \). Presence of \( OP_i \), also helps to associate the \( GC \) to respective transition \( TR \). Creation of \( GC \) in \( S_2 \) follows the mapping given in Table 8.5.

d. \( AI_i \) is created as rules mapping on an arc inscription in the target schema CPN model \( S_2 \) for each SWRL rule \( SR_i \) or set of \( SR_i \). \( AI \) is attached to the arcs \( AR(SB) \) coming out of transitions present on the subpage of CPN model. The creation of \( AI \) and its attachment to arc \( AR(SB) \) depends upon the presence of object property \( OP_i \) in the antecedent and consequent part of each \( SR_i \).

The transformation is also injective since its inverse maps Guard conditions and arc inscriptions, back to the SWRL rules because information about SWRL antecedent and consequent are properly maintained by the transformation. Therefore, it is proved that the transformation is both total and injective and preserves the information capacity.
9.3.8 Conclusion of Lemma 1-7

Therefore, these six lemmas proved that transformation $T$ of OWL ontology $S_1$ into CPN model $S_2$ is correct and information capacity preserving transformation. Three sub transformations of $T$: $T_1$, $T_2$ and $T_3$ involves in the creation of target model $S_2$. The categories of concepts $C(SO)$ are identified such that $\forall C_i \in (AC \lor NC)$. A class diagram of classes $CL$ is created for these categories in the first part of the transformation $T_1$. $CL$ consists of set of nodes:

$$N^{CL} = \{N^{CLAC}, N^{CLNC}\}$$

This defines:

- a set $N^{CLAC}$ concrete classes – nodes which have both member variable and member function.
- a set $N^{CLNC}$ association classes – nodes which have only member variable

According to OWL ontology these nodes are used to represent following concepts:

- $N^{CL}$ represent $[AC \in C \land NC \in C \land C \neq \phi]$
- $N^{CLAC}$ represent $[AC \in C \land C \neq \phi \land OP \neq \phi]$
- $N^{CLNC}$ represent $[NC \in C \land C \neq \phi \land OP = \phi]$

Besides these nodes the class diagrams have some other components as well:

- $N^{MV}$ represent $\sum_{i=1}^{\text{C}} [DP \in C \land C \neq \phi]$

138
• $N^{MF}$ represent $\sum_{i=1}^{AC}[OP \in AC \land AC \neq \phi]$

• $N^{CL(AS)}$ represent $\sum_{i=1}^{AC}[OP \in AC \land AC \neq \phi]$

The first output of transformation $T_1$ is intermediate model (UML class diagram). $T_1$ contains a set of classes $CL$ (with member variables and with/without member functions), member variables $MV$, member function $MF$ and associations between classes $CL(AS)$, here:

- total number of $N^{CLAC} = \text{total number of concrete classes}$
- total number of $N^{CLNC} = \text{total number of association classes}$
- total number of $N^{MV} = \text{total number of member variables}$
- total number of $N^{MF} = \text{total number of member functions}$
- total number of $N^{CL(AS)} = \text{total number of associations between classes}$

In SAC-IBA case studies, there are three active concepts, ten non-active concepts, fifteen datatype properties and nineteen object properties. Therefore, $CL$ contains thirteen nodes $N^{CL}$ corresponding to the thirteen concepts which is subdivided into three nodes $N^{CLAC}$ corresponding to active concepts, ten $N^{CLNC}$ corresponding to non-active concepts so three concrete classes and ten association classes are created. Fifteen member variables $MV$ corresponding to datatype properties; nineteen member function $MF$ and nineteen associations between classes $CL(AS)$, corresponding to object properties, are also created. The class diagram is shown in Figure 5.2.
The second output of transformation $T_1$ is another intermediate model (UML activity diagram). In the second part of $T_1$ UML activity diagram $AD$ is created from UML class digram $CL$. $AD$ consists of set of components. $T_1$ contains a set of swim lanes $SL$, data store nodes $DN$, action nodes $AD(AN)$:

$$N^{AD} = \{N^{SL}, N^{DN}, N^{AD(AN)}\}$$

Here:

- total number of $N^{AD} = \text{total number of activity diagram elements}$
- total number of $N^{SL} = \text{total number of swim lanes}$
- total number of $N^{DN} = \text{total number of data store nodes}$
- total number of $N^{AD(AN)} = \text{total number of action nodes}$

According to OWL ontology these components are used to represent following concepts:

- $N^{AD}$ represent $[CL_{AC} \in CL \land CL_{NC} \in CL_{AC} \land MF(CL) \in CL_{AC} \land CL \neq \phi]$
- $N^{SL}$ represent $[CL_{AC} \in CL \land CL \neq \phi \land MF(CL) \neq \phi]$
- $N^{DN}$ represent $[CL_{NC} \in CL \land CL \neq \phi \land MF(CL) = \phi]$
- $N^{AD(AN)}$ represent $\sum_{i=1}^{AC}[MF(CL) \in CL_{AC} \land MF(CL) \neq \phi]$

Following the creation of class diagram, activity diagram $AD$ contains three nodes $N^{SL}$ corresponding to concrete classes, ten $N^{DN}$ corresponding to association classes so three swim lanes and ten data store nodes are created. Fifteen activity diagram arc inscription
AD(AI) corresponding to datatype properties and nineteen action node \( N^{AD(AN)} \) corresponding to member functions are also created. The activity diagram is shown in Figure 5.3.

In contrast to \( T_1 \), transformation \( T_2 \) uses two models: sources schema \( S_1 \) (i.e., OWL ontology) and intermediate model (i.e., UML activity diagram), as an input. The output of transformation \( T_2 \) is \( S_2 \) a target schema (i.e., CPN model) contains set of places \( P \), substitution transitions \( ST \), arcs on super page \( AR(SP) \), transitions on subpage \( TR \), arcs on subpage \( AR(SB) \) and colorset \( CS \), here:

- total number of \( N^P \) = total number of places
- total number of \( N^{ST} \) = total number of substitution transitions
- total number of \( N^{TR} \) = total number of transitions on subpages
- total number of \( N^{AR(SP)} \) = total number of arcs on super page
- total number of \( N^{AR(SB)} \) = total number of input/output arcs on subpage
- total number of \( N^{CS} \) = total number of colorset

According to OWL ontology and UML activity diagram, these components are used to represent following concepts:

- \( N^{ST} \) represent \([AC \in C \land NC \in C \land C \neq \phi]\)
- \( N^P \) represent \([AC \in C \land C \neq \phi \land OP \neq \phi]\)
• $N^{TR}$ represent $\sum_{i=1}^{AC}[OP \in AC \land AC \neq \phi]$

• $N^{CS}$ represent $\sum_{i=1}^{C}[DP \in C \land C \neq \phi]$

• $N^{AR(SP)}$ represent $[DA \in A \land A \neq \phi]$

• $N^{AR(SB)}$ represent $[CA \in A \land A \neq \phi]$

The target schema (i.e., CPN model) $S_2$ contains three nodes $N^{ST}$ corresponding to active concepts, ten nodes $N^P$ corresponding to non-active concepts and nineteen arcs on super page $N^{AR(SP)}$ corresponding to object properties. Three $N^{ST}$ becomes the reason for the creation of three subpages which contain nineteen $N^{TR}$ corresponding to object properties. The subpages also contain thirty eight input/output arcs $N^{AR(SB)}$ corresponding to control flow/dataflow arrows on UML activity diagram. $S_2$ also contains fifteen colorset corresponding to the datatype property in OWL ontology.

The final output of transformation $T$ is $T_3$ may contains guard condition $GC$ or filter condition on arc inscription $AI$ or both, here;

• total number of $N^{GC}$ = total number of guard conditions

• total number of $N^{AI}$ = total number of filter condition on arc inscription

The target schema (i.e., CPN model) $S_2$ of SAC-IBA case study contains two $N^{AI}$ corresponding to SWRL rules. Therefore, overall these seven lemmas proved that
transformation $T$ of OWL ontology $S_1$ into CPN model $S_2$ is correct and information capacity preserving transformation.

### 9.4 EXPERIMENTAL RESULTS

For the evaluation of CSADBF framework, two case studies are used. The specifications of these case studies have been shown in Table 9.2.

**Table 9.2 Case Studies’ Specifications**

<table>
<thead>
<tr>
<th>Case Studies</th>
<th>Concepts</th>
<th>Properties</th>
<th>SWRL Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Active</td>
<td>Non-active</td>
<td>Data type</td>
</tr>
<tr>
<td>SAC-IBA Case Study</td>
<td>3</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>Order Processing Case Study</td>
<td>2</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

### 9.4.1 SAC-IBA Case Study

SAC-IBA case study contain three active concepts, ten non-active concepts, fifteen data type properties, nineteen object properties and two SWRL rules. The transformations or results of SAC-IBA case study are validated by comparing the generated CPN with (i) the intermediate models (UML activity diagram), and (ii) OWL ontology model and its components. CPN Super page (transition, places), Subpage (transition, places), guard
conditions, arc inscriptions and global declarations (colorset, variables etc.) are compared with (i) the intermediate swim lanes (concrete classes), data stores node (association classes), and (ii) OWL ontology concepts (i.e., active concepts, non-active concepts), data type properties, object properties and SWRL rules respectively. Taking the example of SAC-IBA case study to prove that transformation is complete and correct according to input models, following formulas are used which are extracted from the mathematical proof provided in above section:

**UML Class Diagram Creation:**

Total number $N^{CL} = [AC \in C \land NC \in C \land C \neq \phi] = 13$

- total number of $N^{CLAC} = [AC \in C \land C \neq \phi \land OP \neq \phi] = 3$
- total number of $N^{CLNC} = [NC \in C \land C \neq \phi \land OP = \phi] = 10$
- total number of $N^{MV} = \sum_{i=1}^{C}[DP \in C \land DP \neq \phi]$
  here $C = 13$, therefore,
  $$= [(2+3+4+3+2+3+6+7+1+1+1+1)] = 34$$
- total number of $N^{MF} = \sum_{i=1}^{AC}[OP \in AC \land OP \neq \phi]$
  here $AC = 3$, therefore,
  $$= [(4+7+8)] = 19$$
- total number of $N^{CL(AS)} = \sum_{i=1}^{AC}[OP \in AC \land OP \neq \phi]$
  here $AC = 3$, therefore,
  $$= [(4+7+8)] = 19$$
**UML Activity Diagram Creation:**

total number of $N^{SL} = [AC \in C \land C \neq \phi \land OP \neq \phi] = 3$

total number of $N^{DN} = [NC \in C \land C \neq \phi \land OP = \phi] = 12$ (10 unique data nodes)

total number of $N^{AD(AN)} = \sum_{i=1}^{AC}[MF(CL) \in CL_{AC} \land MF(CL) \neq \phi]$

here $AC = 3$, therefore,

$$= [(4+7+8)] = 19$$

With the help of domain expert 24 data flow arrows $DA$ and 21 control flow arrows $CA$ has been drawn. So, overall in the three swim lanes there are 5, 19 and 21 arrows.

**CPN Model Creation:**

total number of $N^{ST} = [AC \in C \land C \neq \phi \land OP \neq \phi] = 3$

total number of $N^{P} = [NC \in C \land C \neq \phi] = 10$

total number of $N^{TR} = \sum_{i=1}^{AC}[OP \in AC \land AC \neq \phi]$

here $AC = 3$, therefore,

$$= [(4+7+8)] = 19$$

total number of $N^{AR(Sp)} = [DA \in A \land A \neq \phi] = 24$

total number of $N^{AR(SB)} = \sum_{i=1}^{SL}[CA \in A \land DA \in A \land A \neq \phi]$

here $SL = 3$, therefore,

$$= [(5+19+21)] = 45$$

total number of $N^{CS} = \sum_{i=1}^{C}[DP \in C \land C \neq \phi] = 15$
Table 9.3 Summary of Results (SAC-IBA Case Study)

<table>
<thead>
<tr>
<th>concepts, Properties &amp; Rules</th>
<th>OWL ontology (Source Model)</th>
<th>UML Class diagram (Intermediate Model)</th>
<th>UML Activity diagram (Second Source Model)</th>
<th>CPN Model (Target Model)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active concepts</td>
<td>Classes, Member Variables, Member Functions &amp; Associations</td>
<td>Swim lanes, Data store nodes, Action nodes, &amp; Arrows</td>
<td>Substitution Transitions, Places, colorsets, Transitions, input/output arcs</td>
<td></td>
</tr>
<tr>
<td>Non-active concepts</td>
<td>Association Classes</td>
<td>Data store nodes</td>
<td>Places (on super pages), Places (input nodes, output nodes, i/o nodes on subpages)</td>
<td></td>
</tr>
<tr>
<td>Data type properties</td>
<td>Member Variables</td>
<td>Arc inscriptions</td>
<td>Colorsets</td>
<td></td>
</tr>
<tr>
<td>Object properties</td>
<td>Member Functions</td>
<td>Action nodes (in swim lanes)</td>
<td>Transitions (on subpages)</td>
<td></td>
</tr>
<tr>
<td>Object Properties</td>
<td>Associations</td>
<td>Data flow arrows (in swim lanes)</td>
<td>Input/Output Arrows (Super page)</td>
<td></td>
</tr>
<tr>
<td>SWRL rules</td>
<td></td>
<td>Control Flow/Data flow Arrows</td>
<td>Input/Output Arrows (Subpage)</td>
<td></td>
</tr>
</tbody>
</table>

Table 9.3 shows the summary of the output results coming out of above mathematical calculations. The calculation has proved that transformation generated from three mapping
schemes of CSADBF is complete and correct according to input models. three substitution transitions and ten places on super page are created in the target model (CPN) corresponding to three active concepts and ten non-active concepts from the source model (OWL ontology). Similarly, nineteen transitions (7+4+8) and ten places (input nodes, output nodes, i/o nodes) on three subpages are created corresponding to nineteen object properties (7+4+8) and ten non-active concepts. Twenty four Input/output arcs on super page are created for twenty four Data flow arrows (in the swim lanes) of activity diagram. Forty five Input/output arcs on three subpages (5+19+21) are created for forty five control flow arrows in three swim lanes (5+19+21) of activity diagram. Fifteen colorset for fifteen datatype properties and two filter conditions on two arc inscriptions corresponds to the two SWRL rules. Hence, it is proved that the transformation performed using CSADBF is complete.

9.4.2 Order Product Case Study

Order Product case study contain two active concepts, five non-active concepts, five data type properties, ten object properties and three sets (1+3+5) of SWRL rules. The transformations or results of Order Product case study are also validated by comparing the generated CPN with (i) the intermediate models (UML activity diagram), and (ii) OWL ontology model and its components. CPN Super page (transition, places), Subpage (transition, places), guard conditions, arc inscriptions and global declarations (colorset, variables etc.) are compared with (i) the intermediate swim lanes (concrete classes), data stores node (association classes), and (ii) OWL ontology concepts (i.e., active concepts,
non-active concepts), data type properties, object properties and SWRL rules respectively.

Taking the example of Order Product case study to prove that transformation is complete and correct according to input models, following formulas are used which are extracted from the mathematical proof provided in above Section 9.3:

**UML Class Diagram Creation:**

Total number \( N^{CL} = [AC \in C \land NC \in C \land C \neq \phi] = 7 \)

total number of \( N^{CLAC} = [AC \in C \land C \neq \phi \land OP \neq \phi] = 2 \)

total number of \( N^{CLNC} = [NC \in C \land C \neq \phi \land OP = \phi] = 5 \)

total number of \( N^{MV} = \sum_{i=1}^{C} (DP \in C) \land DP \neq \phi \)

here \( C = 7 \), therefore,

\[ = [(2+2+2+1+1+1+1)] = 10 \]

total number of \( N^{MF} = \sum_{i=1}^{AC} (OP \in AC \land OP \neq \phi) \)

here \( AC = 2 \), therefore,

\[ = [(2+8)] = 10 \]

total number of \( N^{CL(AS)} = \sum_{i=1}^{AC} (OP \in AC \land OP \neq \phi) \)

here \( AC = 2 \), therefore,

\[ = [(2+8)] = 10 \]
UML Activity Diagram Creation:

- total number of $N^{SL} = [AC \in C \land C \neq \phi \land OP \neq \phi] = 2$
- total number of $N^{DN} = [NC \in C \land C \neq \phi \land OP = \phi] = 5$
- total number of $N^{AD(AN)} = \sum_{i=1}^{AC} [MF(CL) \in CL_{AC} \land MF(CL) \neq \phi]$

  here $AC = 2$, therefore,

  $$= [(2+8)] = 10$$

With the help of domain expert 10 data flow arrows $DA$ and 15 control flow arrows $CA$ has been drawn. So, overall in the two swim lanes there are 3 and 22 arrows.

CPN Model Creation:

- total number of $N^{ST} = [AC \in C \land C \neq \phi \land OP \neq \phi] = 2$
- total number of $N^{P} = [NC \in C \land C \neq \phi] = 5$
- total number of $N^{TR} = \sum_{i=1}^{AC} [OP \in AC \land AC \neq \phi]$

  here $AC = 2$, therefore,

  $$= [(2+8)] = 10$$

- total number of $N^{AR(SP)} = [DA \in A \land A \neq \phi] = 10$
- total number of $N^{AR(SB)} = \sum_{i=1}^{SL} [CA \in A \land DA \in A \land A \neq \phi]$

  here $SL = 2$, therefore,

  $$= [(3+22)] = 25$$

- total number of $N^{CS} = \sum_{i=1}^{C} [DP \in C \land C \neq \phi] = 5$
Table 9.4 Summary of Results (Order Processing Case Study)

<table>
<thead>
<tr>
<th>OWL ontology</th>
<th>UML Class diagram</th>
<th>UML Activity diagram</th>
<th>CPN Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Source Model)</td>
<td>(Intermediate Model)</td>
<td>(Second Source Model)</td>
<td>(Target Model)</td>
</tr>
<tr>
<td>Concepts, Properties &amp; Rules</td>
<td>Classes, Member Variables, Member Functions &amp; Associations</td>
<td>Swim lanes, Data store nodes, Action nodes, &amp; Arrows</td>
<td>Substitution Transitions, Places, colorsets, Transitions, input/output arcs</td>
</tr>
<tr>
<td>Active concepts</td>
<td>Concrete Classes</td>
<td>2</td>
<td>Swim lanes</td>
</tr>
<tr>
<td>Non-active concepts</td>
<td>Association Classes</td>
<td>5</td>
<td>Data store nodes</td>
</tr>
<tr>
<td>Data type properties</td>
<td>Member Variables</td>
<td>5</td>
<td>Arc inscriptions</td>
</tr>
<tr>
<td>Object properties</td>
<td>Member Functions</td>
<td>(2+8) = 10</td>
<td>Action nodes (in swim lanes)</td>
</tr>
<tr>
<td>Object properties</td>
<td>Associations</td>
<td>10</td>
<td>Data flow arrows (in swim lanes)</td>
</tr>
<tr>
<td>SWRL rules</td>
<td>Control Flow/Data flow Arrows</td>
<td>(3+22) = 25</td>
<td></td>
</tr>
</tbody>
</table>

Table 9.4 shows the summary of the output results coming out of above mathematical calculations. The calculation has proved that transformation generated from three mapping schemes of CSADBF is complete and correct according to input models. Two substitution transitions and five places on super page are created in the target model (CPN) corresponding to two active concepts and five non-active concepts from the source model (OWL ontology). Similarly, ten transitions (2+8) and five places (input nodes, output nodes, i/o nodes) on two subpages are created corresponding to ten object properties (2+8).
and five non-active concepts. Ten Input/output arcs on super page are created for ten Data flow arrows (in the swim lanes) of activity diagram. Twenty five Input/output arcs on two subpages (3+22) are created for twenty five control flow arrows in two swim lanes (3+22) of activity diagram. Five colorset for five datatype properties and three guard conditions on three transitions corresponds to the three sets of SWRL rules. Hence, it is proved that the transformation performed using CSADBF is complete.

9.5 SUMMARY

This chapter illustrated the correctness of the presented framework CSADBF using two examples: SAC-IBA case study and Order Processing case study. The transformation results of all the mapping schemes of CSADBF are mathematically proved. It is evident from the experimental results and mathematical proves that the transformation evolves in the proposed framework CSADBF is correct and lossless which verifies that all the components in a source model are represented in the target model.
Chapter 10

Comparative Analysis

As mentioned in the Introduction section, several architecture frameworks have been developed in the past but only a handful of them emphasize on the interactive behavior modeling. The list includes the framework presented by Wagenhals et al. (2003), Liles (2008), Noguera et al. (2009) and Wang et al. (2011). Liles’ framework is an automated version of Wagenhals et al.’s framework so we treat them together. The comparison made below is on the basis of automation and expressive power of each framework. The automation feature provides the following benefits: (a) reduction in human errors (b) change management and (c) behavioral concordance. A rather detailed description of these concepts is provided below as it helps in understanding the scope of the comparison.

- Automation refers to automated creation of various views of an architecture with no or little human intervention.

- One of the major advantages of automation is the reduction in human introduced errors. It is worth mentioning that this criterion only covers the errors made in the design phase; implementation errors are not considered in this analysis.

- Change management is linked with information availability or reusability.
- Behavioral concordance tells us how much the behavior shown in the dynamic view is aligned with the behavior modeled in the static view.

- Expressive power tells us about the number of diagrams that are used to model a system’s architecture. A higher number of diagrams help in better communication among various stakeholders.

When it comes to automation, it is safe to say that none of the existing approaches provides complete transformation between the static and dynamic views. All of them require some manual effort while transforming static description to behavioral one. The issue is important because it is the utmost desire of an architect to use a framework that guards the architect from introducing errors. Wagenhals et al. approach does not fully restrict the architect from injecting error. The errors can be injected in static as well as dynamic views. The manual maintenance of concordance among various diagrams is most prone to error injection. The automation of Wagenhals et al.’s framework by Liles somewhat solves this issue but the effort is only limited to activity diagram transformation to Colored Petri Nets. Wang et al.’s approach is also prone to errors in a similar way. Noguera et al. approach is less prone to errors as compared to Wagenhals’ and Wang’s approach. Noguera et al.’s framework provides partial reduction in human error. The ontology of activity diagram in their approach restricts the human error. However, manual addition of guard conditions in CPN is a potential source of error injection. In CSADBF, on the contrary, the human error is reduced to its greatest extent which is due to almost non-existent human intervention while making multiple transformations between the static and dynamic views.
Change management or reusability is another important criterion for comparison. Modeling a domain through a large number of diagrams with manual maintenance of concordance among them make Wagenhals et al. and Wang et al. approaches less effective. It is a common practice that changes (clarity) in the requirements happen throughout the requirement analysis phase. If an approach requires manual transformation among various diagrams after each round of changes then the approach cannot claim to poses a good change management capability. When we compare the existing approaches in terms of reusability, we find that Wagenhals et al., Liles and Wang et al.’s approaches do not support reusability or shared conceptualization whereas Noguera et al. and CSADBF does it. The use of ontology in the later approaches also enables shared conceptualization. The CSADBF is even better than Noguera et al.’s approach as it also automatically maps operation rules in a static view to the dynamic view.

It is worth mentioning that except CSADBF, operational rules are directly modeled in the dynamic view by almost all the other proposed approaches without giving much detail in the static view. Wagenhals et al., Wang et al. and Noguera et al. use decision boxes in the activity diagram for modeling operational rules. These decision boxes provide very limited support for modeling operational rules and thus restrict an architect from modeling complex rules also in the static view. CSADBF, on the other hand, models operational rules using SWRL and provides a mapping scheme for transforming these rules to *guard conditions* in CPN.
In terms of expressive power, Wagenhals et al.’s and Wang et al.’s frameworks have relatively more expressive power than CSADBF and Nogeura et al. and Liles frameworks. Use of large number of diagrams in Wagenhals et al. and Wang et al. approaches leverage the architect to model a system’s architecture from multiple dimensions. Improvements in the expressive power also improve the communication between the architect and the stakeholders. Unlike Wagenhals et al., Wang et al. uses SysML which is a subset of UML 2 specially designed for System architecture modeling.

Based on the above comparison, it can be claimed that none of the proposed framework is completely suitable for all types of scenario. If we summarize our findings, Wagenhals et al. and Wang et al. frameworks are more or less similar in their capabilities except that Wang et al. use SySML instead of UML. Liles has an advantage over Wang et al. because of automatic transformation but the advantage comes at the cost of less expressiveness. Noguera et al. handles consistency in a better way but is less expressive. All the approaches lack proper handling of operational rules and do not provide any transformation that maps these rules from the static view to the dynamic view. The proposed CSADBF framework addresses this issue and provides an automated transformation that maps operational rules defined in SWRL to CPN guard conditions and arc inscription. A bunch of other automated transformations also improve the overall consistency of the framework and reduces the possibility of injecting human errors in the architecture as well as provide better support for change management. The use of ontology also helps in maintaining an integrated dictionary which also eliminates redundancy in the architecture.
Chapter 11

Conclusion and Future Research Directions

The framework presented in this dissertation is an effort to solve inconsistency problem in executable systems architecture. To overcome the challenges of linking together the interfaces of a system’s architecture, a set of automated mappings are provided in the framework. These mappings serve as glue that binds various components and interfaces of the system’s architecture together. In addition, the automated mappings reduce the chances of human errors. The framework supports automatic generation of UML Class diagram and constructs of UML Activity diagram from OWL ontology. The framework also eliminates the redundancy by using ontology as data dictionary. A mapping is also provided that transforms OWL ontology and UML Activity diagram to CPN. The framework also allows architects to model complex rules using SWRL and then map them to *guard conditions* and *arc inscription* in CPN. As a result of these automated mappings, the CSADBF framework not only makes the architecture consistent across different views but it also makes the architect’s job of explaining the working of the proposed system to various stakeholders easier. Provision of list data types, recursive functions and function calling
from Arc Inscription strengthen the overall framework and make it more scalable. Experimental results with mathematical proves demonstrate that transformation involves in the framework is correct and the system preserves information capacity.

There are few areas where there is room for improvements. To get the desired architecture, currently no single tool is available that allows modeling of all four techniques, namely, OWL, UML, SWRL and CPN, in an integrated development environment. Further efforts would be directed to implement such a tool in the future. Secondly, the present OntoCPNRules is workable under the assumptions provided in Table 7.1. Same rules can be mapped to code segment inscription with some minor changes in the mapping technique to allow more flexibility in the CPN model. Similarly, OntoCPNRules can be further extended to model the sequence of operational rules which defines before and after sequence execution of a business process. Furthermore, the CPN generated from CSADBF can be further analyzed via state space analysis. Finally, the time inscription in CPN can also be used in the future research to do temporal logic based analysis.
Appendix I
Appendix I

Onto-UML Tool

Prerequisites

To start using Onto-UML (OntoUML.jar), you must have following software on your machines.

1) Java Runtime Environment 1.6 or later

Download web link: http://java.sun.com/javase/downloads/index.jsp

2) Eclipse 3.4 (Eclipse Modeling Tools (includes Incubating components) )

Download web link:
Eclipse Modeling Tools (includes Incubating components)


Following are the steps to generate UML Class diagram from Ontology using Onto-UML tool.

Step 1:

Check java version on your machine. It should be 1.6 or later.
Step 2:

Copy OntoUML.jar to any location on your hard disk and execute using following command.

`C:\java -jar OntoUML.jar newfolder`

“newfolder” is the name of folder in which it will save .uml file.

System will ask you for .owl file.
Select any .owl file and press enter.

System has created FPModel.uml file in newfoder.
Step 3:

Open Eclipse and create new project.
Right click on the project and import FPModel.uml file (previously created using Onto-UML tool).
Step 4:

Right click on FPModel.uml and click on initialize class diagram and then press next and finish.

Rearrange the class diagram.
REFERENCES


