Validation of UML Models for Interactive Systems with CPN and SPIN

Parne Balu Laxman

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Odisha, India

Validation of UML Models for Interactive Systems with CPN and SPIN

Thesis submitted in partial fulfillment of the requirements for the degree of

Master of Technology

in

Computer Science and Engineering

(Specialization: Software Engineering)

by

Parne Balu Laxman

(Roll No.: 211CS3070)

under the supervision of

PROF. S. K. Rath

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela, Odisha, 769 008, India

June 2013

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Odisha, India.

Certificate

This is to certify that the work in the thesis entitled ***Validation of UML models for Interactive Systems with CPN and SPIN*** by ***Parne Balu Laxman*** is a record of an original research work carried out by him under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of Master of Technology with the specialization of Software Engineering in the department of Computer Science and Engineering, National Institute of Technology Rourkela. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Place: NIT Rourkela
Date: June 3, 2013

Prof. S. K. Rath
Professor, CSE Department
NIT Rourkela, Odisha

Acknowledgment

I am grateful to numerous local and global peers who have contributed towards shaping this thesis. At the outset, I would like to express my sincere thanks to Prof. S. K. Rath for his advice during my thesis work. As my supervisor, he has constantly encouraged me to remain focused on achieving my goal. His observations and comments helped me to establish the overall direction of the research and to move forward with investigation in depth. He has helped me greatly and been a source of knowledge.

I extend my thanks to our HOD, Prof. A. K. Turuk for his valuable advices and encouragement.

I am really thankful to my all friends. My sincere thanks to everyone who has provided me with kind words, a welcome ear, new ideas, useful criticism, or their invaluable time, I am truly indebted.

I must acknowledge the academic resources that I have got from NIT Rourkela. I would like to thank administrative and technical staff members of the Department who have been kind enough to advise and help in their respective roles.

Last, but not the least, I would like to dedicate this thesis to my family, for their love, patience, and understanding.

Parne Balu Laxman
Roll: 211CS3070

Abstract

Unified Modelling Language (UML) is considered to be the standard language for object-oriented modeling and analysis. However, UML cannot be used for automatic analysis and simulation. The system model developed on the basis of UML tool is not executable in nature. So, behavior of the model cannot be validated until it is implemented. In this thesis, an approach for transforming UML Interaction Overview Diagram (IOD) to Colored Petri Net (CPN) models is proposed. This transformation is used to bridge the gap between informal notation (UML diagrams) and more formal notation (CPN models) for analysis purpose. CPN model is validated by CPN tool. CPN tool is executable, and it is possible to inspect the behavior of the system by simulating CPN model. An interaction overview diagram has been designed for the different operation of an Automatic Teller Machine (ATM) using Magic Draw. Later, this diagram is transformed to CPN model. The specification of the proposed system has been analyzed by simulating the CPN model on CPN tool.

Model checking is an important technique for ensuring the correctness of any system. This thesis presents a case study for model checking through an example of verifying ATM with Simple PROMELA Interpreter (SPIN). The ATM system was modeled in Process or Protocol Meta Language (PROMELA) for business flow of an ATM system. It is then checked for deadlock and unreachable code with SPIN model checker. Here the SPIN model checker is used to apply Linear Temporal Logic (LTL) formula on the ATM system and check for liveness and safety properties. The results showed that the ATM model did not have deadlock and unreachable code, and also satisfied the liveness and safety properties.

Keywords: *Automatic Teller Machine; Colored Petri Net; Formal Specification; Linear Temporal Logic; Model Checking; PROMELA; SPIN*

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Chapter 1

Introduction

1.1 Introduction

Software system model specification plays a critical role in software development process. In a software development life cycle, after completing requirements analysis and getting a requirements specification document, the model for the software system need to be designed. Modeling and design activities play a significant role in bridging a gap between a requirements specification and an implementation of the system. If a designer adopts the system model that is not appropriate for the requirements then the system with proper functionality (quality) would not be produced and it would lead to wasting huge amount of cost and development time because he or she has to re-develop the system of a certain quality. To avoid this, describing an architectural design formally and validating its appropriateness at the architectural design step, before starting the implementation of the system, is very helpful [1] [2].

As software specification and behavioral analysis play a critical role in software development process, this thesis addresses a technique for describing behavioral aspects of software architecture. In this thesis, a process for deriving system specification combining the UML [3] as object-oriented method and Colored Petri Net [4] (CPN) as a formal method has been proposed. CPN is an extended version of Petri Net [5] and has several powerful analysis and simulation techniques associated with tools such as Design/CPN are being used. At the beginning of the process; Interaction Overview Diagram (IOD) of UML 2.x is developed using magic draw and later the diagram is transformed to CPN model by applying the proposed mapping rules [6] and considering states as places and user interactions as guard conditions of transitions. The use of CPN tools may allow for verification and simulation of the resulting specifications. Simulation tools can detect which parts of the design would be the inappropriate ones i.e. not designed according to the given requirement. So, by simulating the particular architectural CPN model on CPN tool [7], the fault with the system can be identified before the implementation of the model.

The objective of this thesis is to employ model checking technique to validate important properties of Automated Teller Machine (ATM). The book Object-Oriented Modeling and Design by James R Rumbaugh [3] provides a good description of ATM system. Based on the requirements given in the book, the PROMELA (Process Meta Language) model is develop to validate the properties of an ATM. The ATM state diagram from Object-Oriented Modeling and Design [8] is used as the primary source for modeling the ATM with PROMELA language [9] [10]. The closed system of the ATM model consists of three active entities: Card-holder, ATM, and Bank-server. From the state diagram, It is possible to extract various actions of the three active entities [11]. The state diagram also helps to identify the different system properties which need to be validate. Once the ATM has been modeled, SPIN (Simple PROMELA Interpreter) [10] model checker is used to validate the ATM model against deadlock, un-executed code and properties that are define according to state diagram of an ATM system. For efficiency, used jSPIN , a graphical user interface for SPIN model checker. To validate against deadlock and un-executed code, need to set the verification mode to safety according to the manual [12] if the validation is successful, the output should report no errors and zero unreachable state.

1.2 Motivation of Our Work

The client and developer has different point of views towards the requirement of any software system. Hence, the design and implementation of the system is not so easy. To design an error free system, various constraints need to be considered and these constraints should be validated to meet the customer requirement. Several reasons exists to specify and to validate the UML diagrams using CPN. UML diagram lacks support for strong simulation and analysis techniques. CPN's are graphical formalisms, which can validate, verify and simulate the UML in order to check the correctness of the model. CPN's have sound mathematical properties and are suitable for visualization and provides executable model which helps in analyzing different properties through state space analysis of CPN model. The Model checking methodology with the help of SPIN and PROMELA has been done to validate the safety and liveness property of the system model.

1.3 Objective of Our Work

The objective of our research work is to analyze, evaluate, design an error free model; at the design level to avoid wasting huge amount of cost and development time require for the re-development of the system of a certain quality. The research work carried out is based upon following objectives:

- Identified the major functionalities of the application system and modeled the system using interaction overview diagram.
- Later the diagram is converted to a CPN model and simulated with the help of CPN tool. The use of CPN tools allow for verification and simulation of the resulting specifications.
- Simulation tools can detect which parts of the model would be the inappropriate ones i.e. not designed according to the given requirement. So, by simulating the particular architectural CPN model on CPN tool [7], the fault with the system can be identified before the implementation of the model.

- Later the verification of the different system properties and linear temporal logic formula's is done with the help of PROMELA and SPIN.

1.4 Problem Statement

The system developer must analyze the different views of the system to build complex systems. Models should be built by using precise notations, should be verified to satisfy the requirements of the system, and then adding detail gradually to transform the models using validation tool. The main aim of this thesis is to analyze and design the models which should be unambiguous, precise and verifiable using CPN and SPIN. In this thesis, the UML models are built by using an analytical tool i.e., Magic Draw, to give a precise meaning to the UML model and CPN has been used to remove the ambiguity present in the model. After precise development of the model, CPN is used to check the model and make it executable as the UML model is not directly executable. When the model is developed successfully, CPN state space diagram should be analyzed to generate the different properties of the model.

SPIN is a generic verification system that supports the design and verification of asynchronous process systems. SPIN accepts the design specification written in the verification language PROMELA and it accepts the correctness claims specified in the syntax of standard Linear Temporal Logic (LTL). To illustrate the process of model checking, example of different operation of an Automatic Teller Machine (ATM) System is considered.

1.5 Thesis Organization

The thesis is organized as follows: Chapter 2 describes the literature review done for this thesis. Chapter 3 discusses the related concepts used in this thesis. Chapter 4 proposed work of Validating Interaction Overview Diagram using CPN through the case study of Automatic Teller Machine. Chapter 5 explains the model checking with SPIN and PROMELA through the case study of ATM. Finally, conclude with summary of contributions and discuss the future work in Chapter 6.

Chapter 2

Chapter 2

Literature Review

In spite of informal semantics and some of the ambiguities, UML is a widespread modeling language used in both academic and industry. On the other hand, Petri nets are mathematical modeling language with a formal semantics [5]. Petri Nets are well suited for formal verification. However, although there is a growing interest in model checking techniques from industry, the software engineers continue to be unfamiliar with such a formalism. It is convenient to supply formal verification techniques of UML diagrams that are completely automatic and transparent to the designer.

In [13] Bowles and Dulani have defined a formal strongly consistent transformation from UML 2 sequence diagrams to colored petri net. They define the language of sequence diagram and show how this is mapped onto an equivalent language of CPNs through formal transformation rules. Researches on UML 2 activity diagrams are carried out by Storrle [14] who analyzed the semantics of these diagrams and proposed an approach to their formalization.

Staines [15] proposed a suitable formalism to achieve the transformation from UML2 activity diagrams to Petri nets. The transformation of UML 2 activity diagrams into petri net semantics have been researched by various reasons. Translating UML activities into petri net creates new problems. The petri net diagrams are

- More complex
- Contain more node and edges.

- Unsuitable for visualization by stakeholders.

A solution to this problem is to translate the UML activity diagram into a fundamental modeling concepts petri net diagram compact notation. This can be converted to CPN for execution and validation. The petri net semantics are represented using reduced FMC-PND which are easier to comprehend than a complete petri net. The FMC-PND is converted into CPN for more detail verification and simulation.

In [16] Ribeiro and Fernandes presents a set of rules that allows software engineers to transform the behavior described by UML 2.0 sequence diagram into colored petri net. The main purpose of the transformation is to allow the development team to construct animations based on the CPN that can be shown to the users or the client in order to reproduced the expected scenario and thus validate them. Thus, nontechnical stakeholders are able to discuss and validate the captured requirement. They shows a set of rules to transform sequence diagram into equivalent CPN's for animation proposes.

Elkoutbi and Rodulf [17] have given the procedure for transformation of the simple use case structure to colored petri nets and Kamandi et al. [18] have transformed the use case to Object Stochastic Activity Network (OSAN). There are different approaches developed for the transformation of different UML diagrams to Petri nets. Bernardi et al. [19] define transformation of sequence diagram to a Generalized Stochastic Petri Nets, the transformations are based on mapping messages as well as conveying them and in Ourdani approach [20] the transformation is based on message sender and receiver component.

There have been several proposals for semantics of UML2 activity diagrams so far, but very little for the Interaction Overview Diagram (IOD)and anyway none using Hierarchical Colored Petri Net (HCPN). This thesis proposes a formalization of IOD's using HCPN's, for this purpose, an adequate mapping between the IOD and HCPN structure is presented.

Model checking is an important method for formal verification of state transition systems because it allows the automatic analysis of the designed system. The model checking of an system model consists of several steps. The first step is modeling investigated system, i.e., convert the system to a formalism which can be accepted by a model checking tool. In the second step it is important to state the properties that the system must satisfy. The temporal logic formula are used for the specifications of system properties. The third step is the verification. In the verification the system is model in PROMELA and validated using SPIN model checker. So this way system model is specified and verified using SPIN model checker.

In [21], the Dynamic Host Configuration Protocol (DHCP) is studied according to the concept of modeling and verification. In [22], the paper proposed a formal method for the verification of ebXML based e-commerce system. And the approach allowed to highlight some weakness of the protocol mainly due to the lack of a clear and complete set of specifications. In [23], the paper shows how finite model-checking based approach can be applied to analyze properties of ad-hoc sensor networks. It proves that SPIN and finite model checking are appropriate for studying properties of ad-hoc sensor networks specifications. In [24], the paper gives an approach to verify the object model of rCOS using model checker Spin.

Chapter 3

Chapter 3

Related Concepts

3.1 Unified Modeling Language

Modeling is the central part of all activities performed in the system which leads up to the development of a good software. UML is a well-defined, expressive and powerful to a wide spectrum of problem domains [25]. It describes the software system both structurally and behaviorally. It describes the static as well as dynamic behavior of the software system. The UML is used to specify, visualize, modify, and document the artifacts of an object-oriented software-intensive system under development. It does not provide a process for developing software but it gives a syntactic representation, to describe all parts of a system i.e. data, function and behavior through a number of diagrams. Here in this thesis UML is used to design the interaction overview diagram for different operation of an ATM systems [3].

3.1.1 Aims of Modeling

- Models help in visualizing the system.
- Models specify the structure or behavior of the system.
- Models gives the template which helps in constructing the system.
- Models document the decisions that has been made by the developer.

3.1.2 Necessity of Building Model

- Communicating with the desired structure and behavior of the system.
- Visualizing and controlling the system's architecture.
- Better understanding of the system which is building and exposing opportunities for re-usability and simplification.
- Managing risk.

3.1.3 Advantages of UML

- UML is process independent but it should be used in a process which is use case driven, incremental, architecture-centric and iterative.
- UML is a standard language used for writing software blueprints.
- UML provides a visual expression of the software system which is achieved by using various UML notations and diagrams.
- UML omits the irrelevant details but allows concise expression of the essential aspects of the software system being developed.

3.1.4 Roles of Interaction Overview Diagram in UML

Interaction Overview Diagram (IOD) is one of the fourteen types of diagrams of the Unified Modeling Language (UML), which can picture a control flow with nodes that can contain interaction diagram. IOD is found in version 2.0 due to the need to overcome the faults and disadvantages of activity and sequence diagrams. The nature of the IOD is to show the interaction of the different system components within the system at the higher level of abstraction. The stronger side of the IOD is the fact that it can show dependence between the important sequences of a business system, which can be presented by a flow chart or by an activity diagram. A single sequence within the IOD can be sequence diagram or use case diagram connected with if then else structure which enables decomposition to lower levels of abstraction.

3.2 Petri Net and Colored Petri Net

A Petri net is one of several mathematical modeling languages for the description of distributed system [5]. It is a formal, graphical, executable technique for the specification and analysis of concurrent and dynamic systems. There has been considerable research on petri net to support model properties such as synchronization, communication and concurrency. It has two serious drawbacks i.e. there are no concepts of data; therefore the models often became excessively large and also there are no hierarchy concepts; therefore this was not possible to build a large model having a set of sub models with well-defined interfaces.

An extension of Petri Net, known as Colored Petri Net (CPN) [4] was introduced by K. Jensen in 1987. In addition to existing concepts like places, tokens and transitions, CPN includes the concept of colors, expressions and guards [26]. Colored Petri Nets are one of the most well-known high-level Petri nets. CP-nets include both data structuring and hierarchical decomposition. The graphical representation makes it easy to see the basic structure of a complex CPN model i.e., to understand how the individual processes interact with each other [26].

Definition: A Colored Petri Net is a nine tuple CPN = (Σ, P, T, A, N, C, G, E,I) satisfying the following requirements:

1. Σ is a finite set of non-empty types, called color sets.
2. P is a finite set of places.
3. T is a finite set of transitions.
4. A is a finite set of arcs such that:
   - $P \cap T = P \cap A = T \cap A = \phi$
5. N is a node function. It is defined from A into P X T $\cup$ T X P.
6. C is a color function. It is defined from P into Σ.
7. G is a guard function. It is defined from T into expressions such that:

- $\forall t \epsilon T : [Type(G(t)) = Bool \wedge Type(var(G(t))) \sqsubseteq \Sigma]$

8. E is an arc expression function. It is defined from A into expressions such that:

    - $\forall a \epsilon T : [Type(E(a)) = C(p(a))_{MS} \wedge Type(var(E(a))) \sqsubseteq \Sigma]$

9. I is an initialization function. It is defined from P into closed expressions such that:

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

3.2.1 CPN Tool :

In practice, a CPN model can be created using CPN tool [7] developed by University of Aarhus, Denmark. It is a graphical tool that allows one to create a visual representation of a CPN model. It is based on state machine theory and is an extension of place-transition petri net [27].

3.2.2 State Space Analysis :

All the states of a system model constitute its state space. The state spaces are calculated fully automatically by the CPN state space tool using a state space construction algorithm [26]. The CPN tool supports a number of stop and branching options that makes it possible for the user to control the state space generation [27]. The state space is a tuple $<$ S0,S,T $>$ such that:

- S0 ϵ S is the initial state which is composed of all the initial markings in the CPN.
- S is the set of all states in state spaces. All the elements of S will be reached by the transition fire sequence from S0.
- T is the set of transitions. All the elements of T will be enabled if the associated arc expressions of all incoming arcs can be evaluated to a multi-set, compatible with the current tokens in their respective input places, and its guard is satisfied.

3.3 Model Checking

Model checking is an important method for formally verifying state transition systems because it allows the fully automatic analysis of investigated system. There are many model checking tools, have been developed to this aim.

3.3.1 PROMELA

PROMELA is a language for building verification models that represent an abstract model of a system, which contains only an important aspects that are relevant to the properties which users wants to verify. The SPIN is used to verify the behavior of processes which interact with each other in the system model [11]. The relevant behavior of a system is modeled in PROMELA and verified. In PROMELA program consist of process, message channels, and variables. Processes are global objects. Message channels and variables can be declared either globally or locally inside a process. Processes specify behavior, channels and global variables define the environment in which the processes run [12].

3.3.2 Linear Temporal Logic

Temporal logic is a formal system for specifying and reasoning about concurrent programs. It provides a uniform framework for describing system at any level of abstraction, thereby supporting hierarchical specification and verification. Due to it's temporal quantifiers temporal logic is convenient and appropriate means to reason with time related proposition. It mainly used to express the time dependent correctness property "the program never deadlocks" or "an interrupt is eventually handled" uses temporal logic. Temporal logic is well suited for the formal specification, verification and requirements description of the system. Different operators which are used for the declaration of linear temporal logic formula are listed in Table- 3.1.

Table 3.1: Linear Temporal Logic Operators

Temporal Operator	**Pictorial Notation**	**Textual Notation**
Next Operator	$\bigcirc$	X
Eventually Operator	$\Diamond$	F
Globally or Alaways Operator	$\Box$	G
Until Operator	$\cup$	U

3.3.3 SPIN

SPIN (Simple Promela Interpreter) is a generic verification system that supports the design and verification of asynchronous process systems [10]. This model checker accepts design specifications written in the verification language PROMELA [28] (Process Meta Language) and it accepts correctness claims specified in the syntax of standard Linear Temporal Logic (LTL) [9]. SPIN helps in finding unreachable codes or deadlocks also verifies LTL properties. SPIN can be used as a full LTL model checking system, supporting all correctness requirements expressible in linear time temporal logic, but it can also be used as an efficient verifier for more basic safety and liveness properties [29].

Chapter 4

Chapter 4

Specification and Verification of Interaction Overview Diagram using Colored Petri Net

4.1 Introduction

To build complex systems, the developer must abstract different views of the system, build models using precise notations, verify models which satisfy the requirements of the system, and gradually add detail to transform the models into an implementation level. Analysis of software requirements using UML-based system model are not executable in nature; so the behavior of the model cannot be validated until it is implemented. In this thesis, a technique for describing behavioral aspects of software architecture formally based on Colored Petri Nets (CPNs) has been used. In this thesis the behavioral model of an ATM withdraw operation has been developed as an interaction overview diagram of UML and it is proposed to transform the diagram to a CPN model. CPN tool is used for validation purpose because it not only shows the CPN model's behavior but also checks several properties such as boundedness, home, liveliness and fairness properties. The validation using CPN tool and ultimately depicting the validation through state space diagram justifies the choice of the particular software behavioral model. CPN tool is used for editing, simulating and analyzing CPN model.

4.2 Interaction Overview Diagram Of ATM Withdraw Operation

Interaction Overview Diagram (IOD) is useful design tool since it provide a dynamic view of the system behavior, which can be difficult to extract from static diagrams or specification [2]. IOD combines the power of sequence diagram and activity diagram together. It can be used to describe an overview of a complex system by embedding the objects of activity diagram, inline interaction or interaction occurrences inside a control flow structure. IOD provides high level structuring mechanism for sequence diagrams. IOD illustrates dependence between the important sequences of a system, which can be presented by an activity diagram. The notations used in IOD incorporate constructs from sequence diagrams with fork, join, decision and merge nodes from activity diagrams [30]. Even though UML 2.x brings more accuracy and precision than UML 1.x, it remains informal and lacks tools for automatic analysis and validation [6].

In this thesis the specification and validation of IOD is done with the help of CPN. For this the transformation of the IOD into CPN model is done based on some mapping rules. The CPN model is simulated for the validation purpose. The Following Figure-4.1 shows a UML interaction overview diagram which models the behavior of the different operation of an ATM system. To Perform a normal ATM withdrawal, Card-holder must acquire an authorization for their pin number and ensure that the amount requested must be in some proper format and also withdrawn amount should be equal to or less than the balance amount. Customers can successfully withdraw money only after both of the conditions are satisfied. For their validation, here manually transform the interaction overview diagram shown in Figure- 4.1 into a CPN model by applying the proposed mapping rules and then shows the behavioral correctness of the a normal withdrawal and abnormal withdrawals due to an invalid PIN number and the request of an excessive amount. To prove the behavioral correctness of the CPN model the simulation function provided by CPN tool is used. The IOD shows the interaction of the different sequences diagrams, Figure-4.2, Figure-4.3, Figure-4.4 and Figure-4.5 shows some of the sequence diagram which are refereed in the IOD of an ATM system.

4.3 Rules for Transformation of IOD to CPN Model

1. Rules for a control node:
    - An initial node is mapped into the place in the CPN.
    - A final node is mapped into the place in the CPN.
    - A fork node is mapped into the transition in the CPN.
    - A join node is mapped into the transition in the CPN.
    - A merge node is mapped into the place in the CPN.
    - A decision node is mapped into the place in the CPN.
2. Rules for an executable node:
    - An action is mapped into the transition in the CPN.
    - Each action of the conditional node is mapped into the transition that finally merges into a single input place in the CPN, and each condition of the conditional node is mapped into the guard condition of the arc derived from a single output place.
    - Each action of the setup, body, and test section of the loop node is mapped into the transition in the CPN, and each condition of the test section is mapped into the guard condition of the arc in the CPN.
    - Each action of the sequence node is mapped into the transition connected to the place through arc in the CPN.
3. Rules for an interaction edge:
    - A control flow is mapped into the arc in the CPN.
    - An object flow is mapped into the arc in the CPN.
4. Exceptional Rules for a connection between two Transitions:

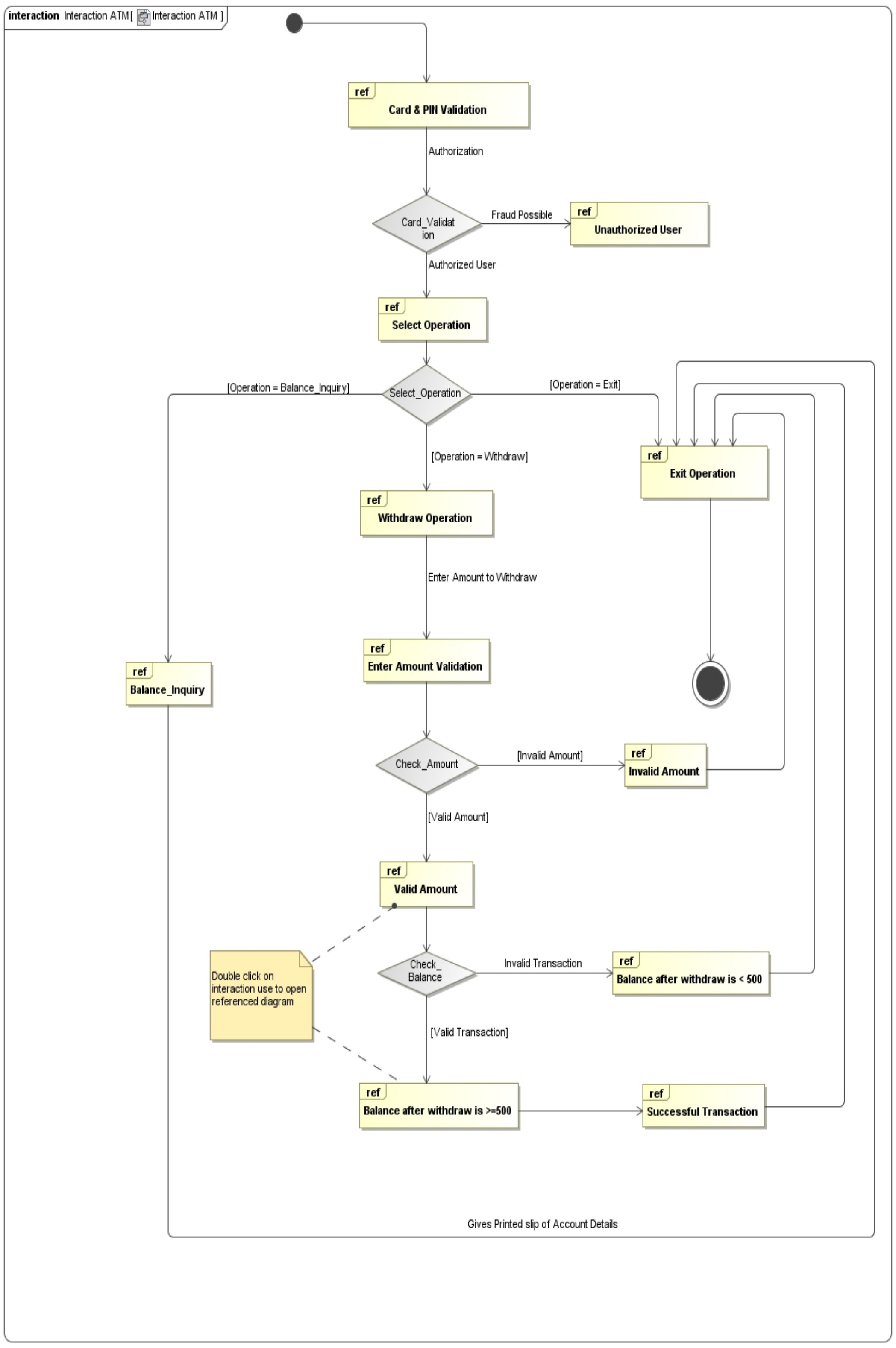


Figure 4.1: Interaction Overview Diagram

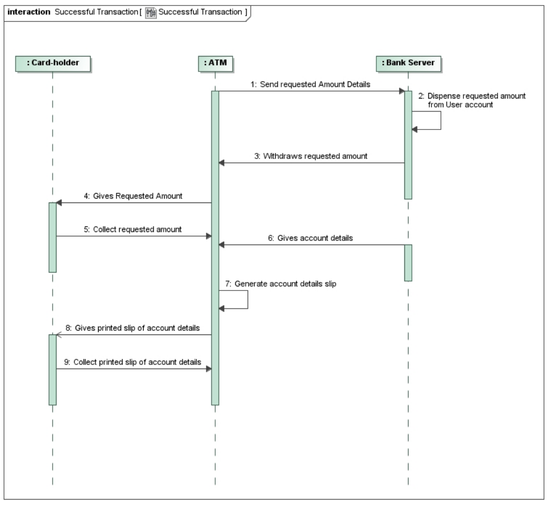

interaction Successful Transaction [Successful Transaction]
: Card-holder
: ATM
: Bank Server
1: Send requested Amount Details
2: Dispense requested amount from User account
3: Withdraws requested amount
4: Gives Requested Amount
5: Collect requested amount
6: Gives account details
7: Generate account details slip
8: Gives printed slip of account details
9: Collect printed slip of account details

Figure 4.2: Sequence Diagram of Successful Transaction

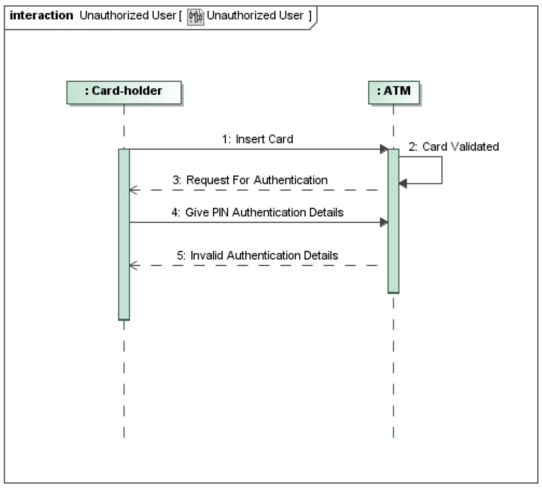

interaction Unauthorized User [Unauthorized User]
: Card-holder
: ATM
1: Insert Card
2: Card Validated
3: Request For Authentication
4: Give PIN Authentication Details
5: Invalid Authentication Details

Figure 4.3: Sequence Diagram for Unauthorized User

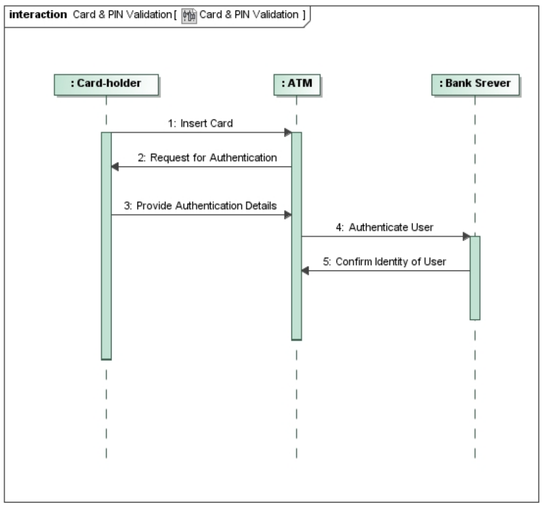

interaction Card & PIN Validation [Card & PIN Validation]
: Card-holder
: ATM
: Bank Srever
1: Insert Card
2: Request for Authentication
3: Provide Authentication Details
4: Authenticate User
5: Confirm Identity of User

Figure 4.4: Sequence Diagram of Card and PIN Validation

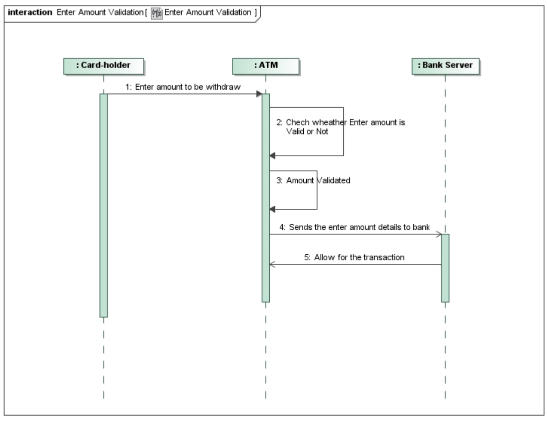

interaction Enter Amount Validation [Enter Amount Validation]
: Card-holder
: ATM
: Bank Server
1: Enter amount to be withdraw
2: Chech wheather Enter amount is Valid or Not
3: Amount Validated
4: Sends the enter amount details to bank
5: Allow for the transaction

Figure 4.5: Sequence Diagram for Amount Validation

- The connection (arc) between two transitions is not possible in the CPN. So place with incoming and outgoing arcs is used to connect two transitions.

5. Exception rules for a connection between two places:

- The connection (arc) between two places is not possible in the CPN. So transition with incoming and outgoing arcs is used to connect two places.

4.4 CPN Model of an Interaction Overview Diagram

The interaction overview diagram depicted in Figure-4.1 is transformed into the CPN model as shown in Figure-4.6 by applying the proposed transformation rules. As mentioned earlier, for proving the behavior correctness of the CPN model, the simulation function can be used. Prior to utilization of the simulation function, it is necessary to define some information, such as color sets, functions, constant values, and so on, in the declaration, if needed. CPN ML language supports the net inscriptions and declaration of color sets, functions, constant values, and so on.

In the validation process, all the interaction used for the ATM banking system are validated whether the Personal Identification Number (PIN) is valid or not. If it is valid then the execution will flow towards “authorization” otherwise it will give a message “Invalid PIN”. The transition enter authorization and match the password with the database. If it matches then the execution will flow towards “Entering the amount” otherwise it will give a comment as “Invalid PIN. The transition “Enter the amount will again check whether the amount to be withdrawn is less than any specified limit (hundred) or not. If it is less than the specified limit then a STRING type statement “INVALID AMOUNT is displayed. In the next transition, it will check for total balance after withdrawal should be greater than the pre-specified limit (500). If it is not true then the transaction will come to end

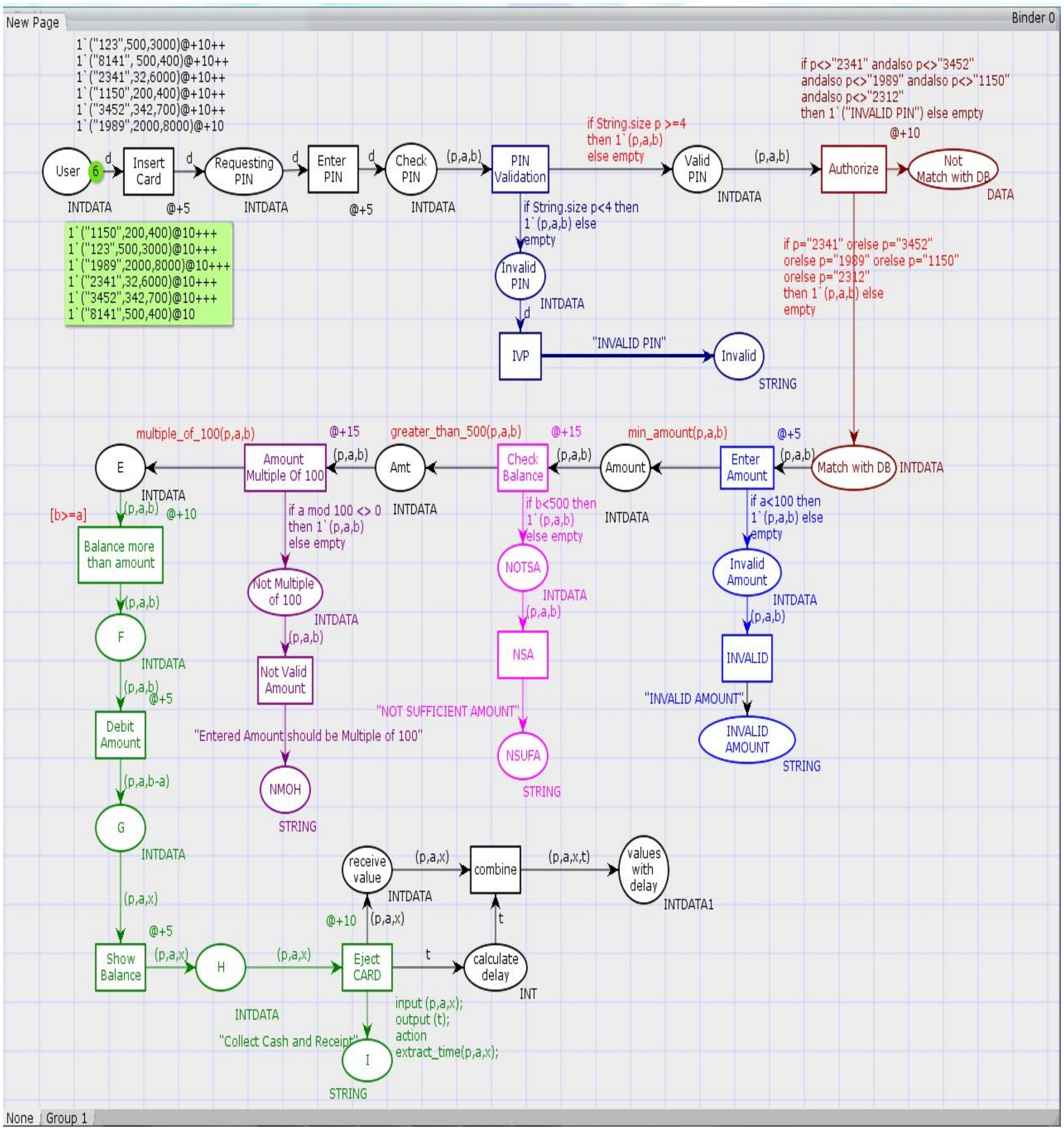


Figure 4.6: CPN Model for Interaction Overview Diagram

and display a STRING type statement “NOT SUFFICIENT AMOUNT. Now it will check for whether entered amount is multiple of pre-specified limit (100) or not. If it is multiple of pre-specified limit then a withdraw operation is executed successfully. After successfully completing all these transactions a STRING type statement “collect Cash and Receipt is displayed and lastly the time delay with output value is displayed.

Hierarchical Colored Petri Net is used to describe the large complex complicated systems (larger nets) into a set of smaller ones using sub-pages. In Figure-4.7, the hierarchical CPN model of the same interaction overview diagram has been shown that is similar to normal CPN model as shown in Figure-4.6. Separate sub-pages are developed for each referred sequence diagram in the interaction overview diagram.

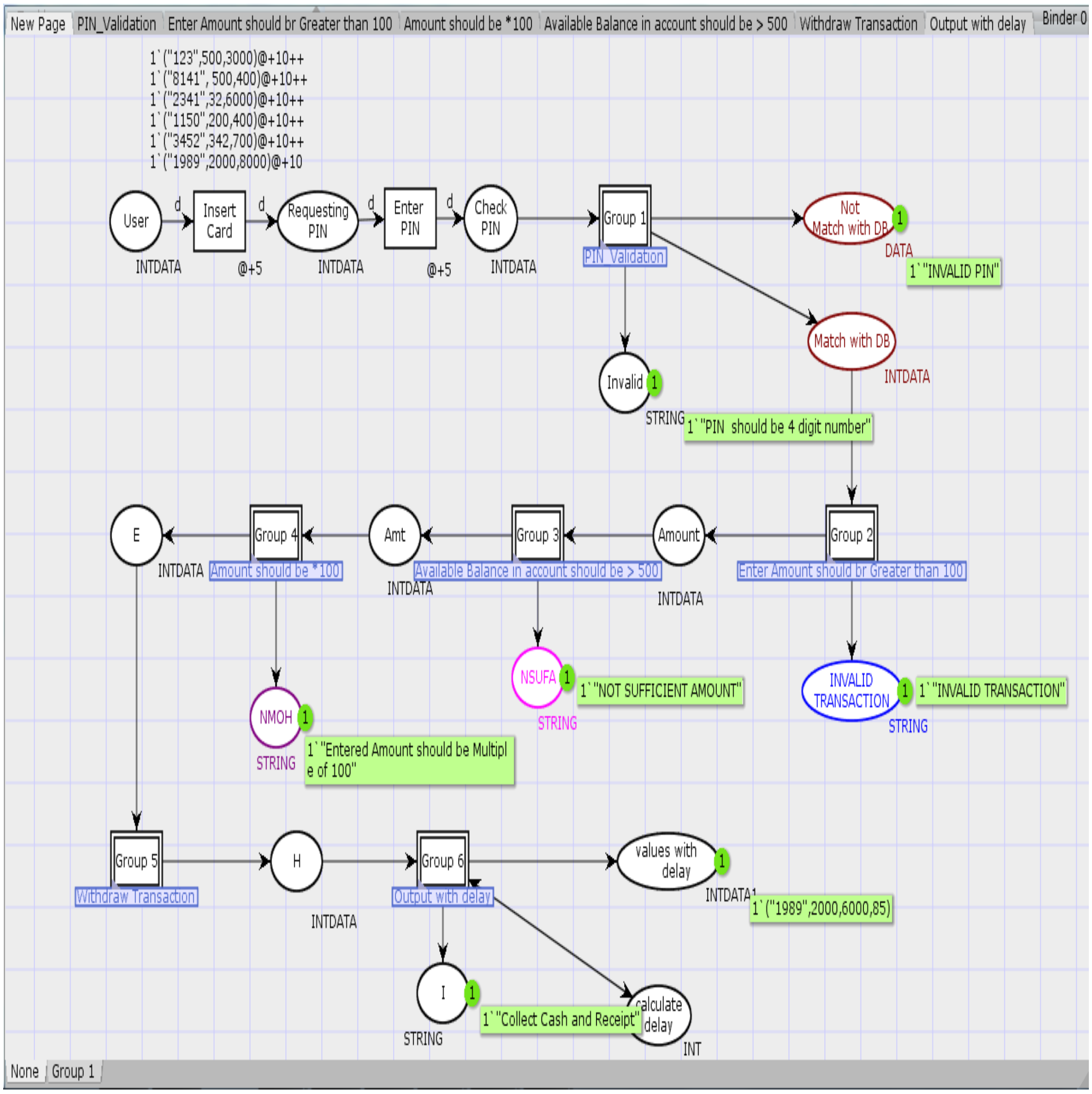


Figure 4.7: CPN Model for Interaction Overview Diagram after simulation

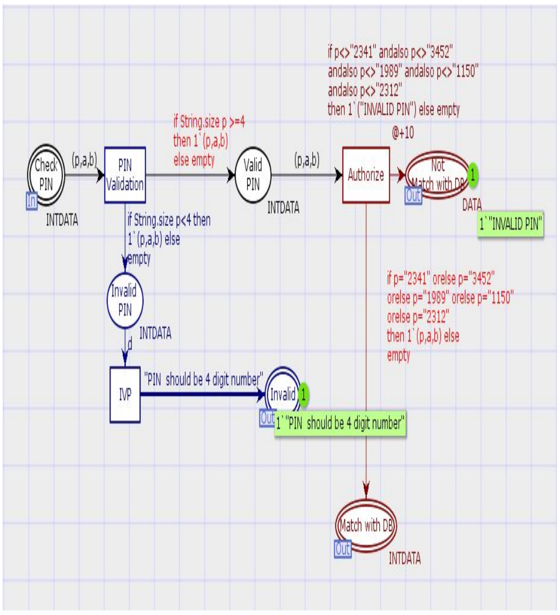


Figure 4.8: Subpage showing Unauthorized User

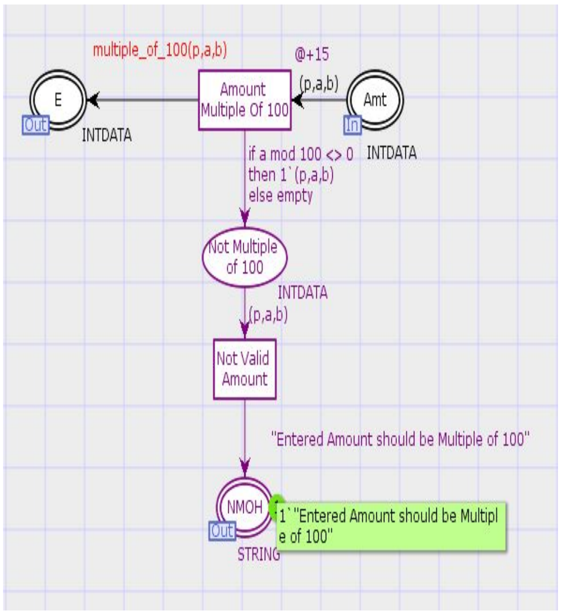


Figure 4.9: Subpage to check enter amount is multiple of 100 or not

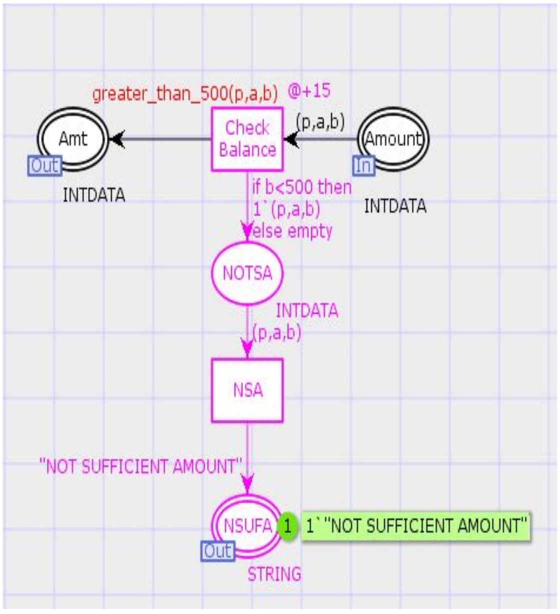


Figure 4.10: Subpage to check whether Balance after withdraw is greater than 500 or not

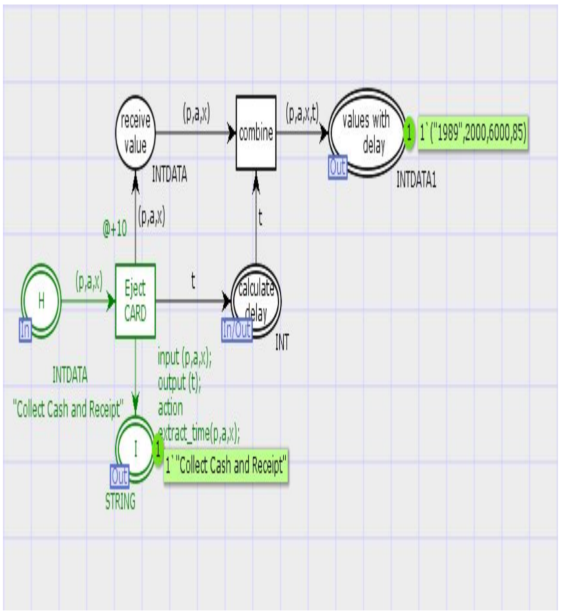
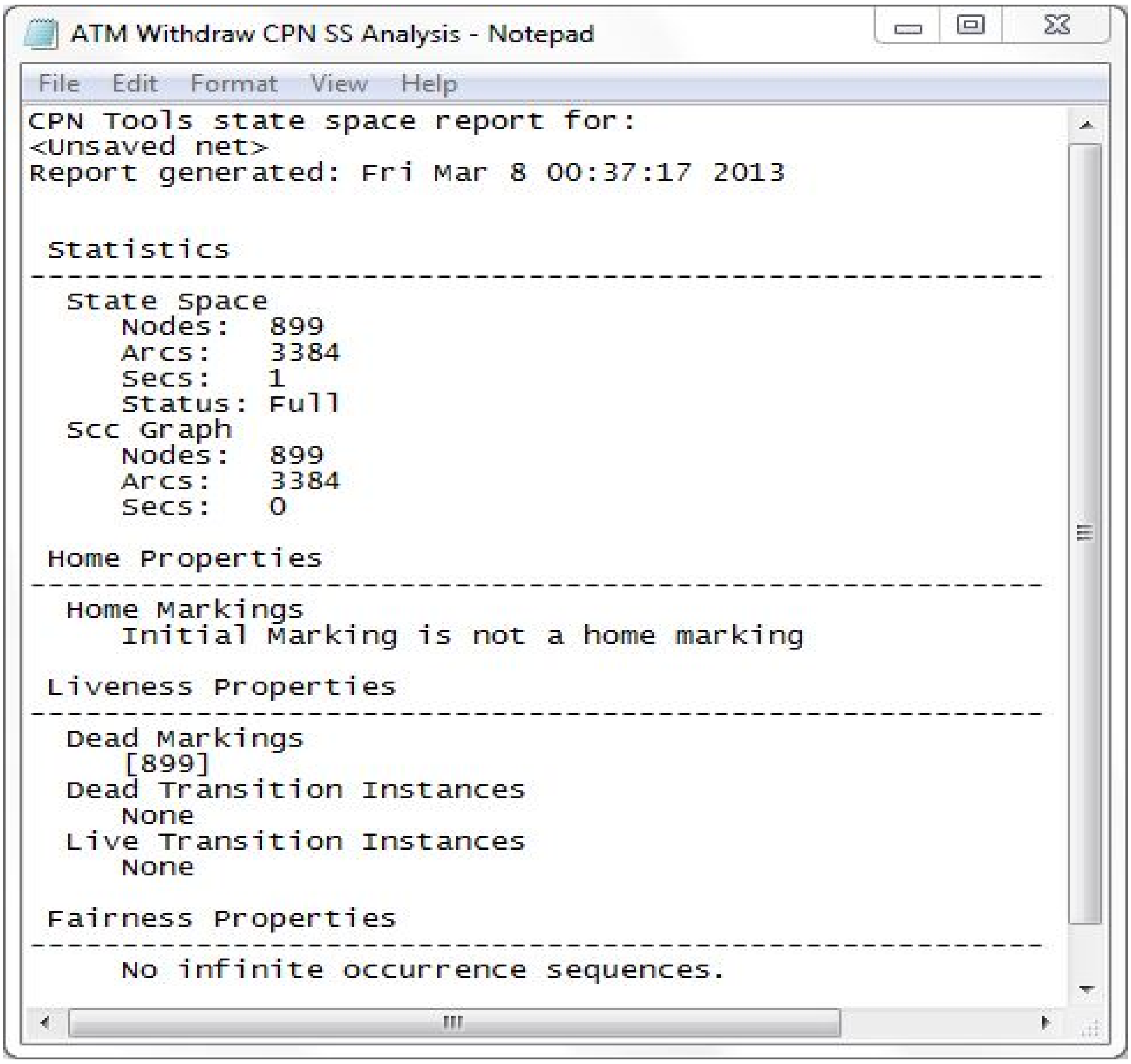


Figure 4.11: Subpage Showing Successful Transaction

The transformed CPN model of each class is analyzed using CPN tool. The state space analysis tool [31] provided by CPN tools produces a result showing properties such as Bounded-ness, Home, Liveness and Fairness etc are shown in Figure-4.12. In this state space analysis, the presence of dead transition instance signifies that there exists some transition which is never activated throughout the system run. This means the presence of such transition is not desirable. But the transformed model in CPN has well defined tool for analysis which covers every path of execution.


ATM Withdraw CPN SS Analysis - Notepad
File Edit Format View Help

```
CPN Tools state space report for:
<Unsaved net>
Report generated: Fri Mar 8 00:37:17 2013


 Statistics
------------------------------------------------------------------------
  State Space
     Nodes:  899
     Arcs:   3384
     Secs:   1
     Status: Full
  Scc Graph
     Nodes:  899
     Arcs:   3384
     Secs:   0

 Home Properties
------------------------------------------------------------------------
  Home Markings
     Initial Marking is not a home marking

 Liveness Properties
------------------------------------------------------------------------
  Dead Markings
     [899]
  Dead Transition Instances
     None
  Live Transition Instances
     None

 Fairness Properties
------------------------------------------------------------------------
     No infinite occurrence sequences.
```

Figure 4.12: State Space Analysis of CPN Model

4.5 Summary

In this chapter, an approach for transforming UML interaction overview diagram to Colored Petri Net models is proposed. This transformation is used to bridge the gap between informal notation (UML diagrams) and more formal notation (CPN models) for analysis purpose. CPN model is validated by CPN tool. The state space method of CP-nets makes it possible to validate and verify the functional correctness of systems.

Chapter 5

Chapter 5

Model Checking using PROMELA and SPIN

5.1 Introduction

Testing is an important step to ensure the correctness of a system. However, testing can never completely identify all the defects within an investigated system. Model checking is a method for formally verifying finite-state concurrent systems. In model checking, properties about the system under verification are usually expressed as temporal logic formulas, and efficient algorithms are used to traverse the system model to check whether the properties hold or not. Model checking is attractive for the system in which problems of concurrency and distribution make traditional testing challenging. In recent years, there have been many papers which report the successful instances of using model checking to provide the validate system verifications. Model checking is a promising approach for ensuring the correctness of ATM banking system. However, the model checking tools subject to the state space explosion problem, which is an ignored hurdle to the practical application of the technique.

This chapter presents a case study of model checking through an example of verifying automatic teller machine (ATM) with SPIN. In the case study, it has been present the specific approach to effectively verify the related part of ATM system. Here the objective is to employ model checking technique to validate important properties of ATM System.

5.1.1 PROMELA

PROMELA is a language for building verification models that represent an abstract model of a system, which contains only an important aspects that are relevant to the properties which users wants to verify. The SPIN is used to verify the behavior of processes which interact with each other in the system model [11]. The relevant behavior of a system is modeled in PROMELA and verified. In PROMELA program consist of process, message channels, and variables. Processes are global objects. Message channels and variables can be declared either globally or locally inside a process. Processes specify behavior, channels and global variables define the environment in which the processes run [12].

5.1.2 SPIN

SPIN(Simple PROMELA Interpreter) [11, 12, 32] is a generic verification system that supports the design and verification of asynchronous process systems. This model checker accepts design specifications written in the verification language PROMELA (a Process Meta Language) [28] and it accepts correctness claims specified in the syntax of standard Linear Temporal Logic (LTL) [9, 33]. The input language of the model checker SPIN allows us to build high-level models of distributed systems from three basic components: asynchronous processes, message channels, and data objects.

5.1.3 Objective

The objective of this chapter is to employ model checking technique to validate important properties of Automated Teller Machine (ATM). The book Object-Oriented Modeling and Design by James R Rumbaugh, [3] provides a good description of ATM system. Based on the requirements from the book, the state model of an ATM system is designed, later the PROMELA code is constructed to validate the properties of an ATM on the basis of system model.

5.2 Model Checking

For the model checking of an ATM need to design the ATM state diagrams on the basis of Object-Oriented Modeling and Design method [34]. The state diagrams are the primary source for modeling the ATM with PROMELA language. The closed system of the ATM model consists of three active entities: Card-holder, ATM, Bank Server. From the state diagram, it is possible to extract various possible actions of the three active entities. The state diagram also helps us to identify properties that want to validate [11]. Once the ATM has been modeled, SPIN Model Checker is used to validate the ATM model against deadlock, unexecuted code and the different properties of the system model. Here the objective is to employ model checking technique to validate important properties of ATM System. The application of model checking follows following steps:

1. The system to be verified is converted to a formal model.
2. After that it is necessary to state the properties that the system must satisfy.
3. Finally the model is verified using SPIN tool.

5.2.1 Formal Model of an ATM System

ATM can be used to login with a card and a pin, after authentication it perform certain operation according to the user requirement. There are usual restriction on amount to be withdraw and format of amount and also on number of transaction. In ATM system in actual three parties communicate with each other i.e. the card-holder, the terminal and the bank server. Card-holder interacts with the banking system through a terminal, for withdrawals, transfer and inquiries. Terminal receives a request from the card-holder to handle all the action from the terminal, and forwards the request to bank server, while waiting for the bank server's response, and forward the response to the terminal. Bank server receives a request from the terminal to make the approval or rejection of the response and make the appropriate accounting treatment. Here the model for the ATM system is designed which shows the main business flow of the ATM system. The model has been simplified in order to obtain a minor number of states to manage in the formal verification.

Here SPIN model checker is used to check the ATM system. So there is a need to describe the specification of the system using PROMELA language. Before describing the specification in PROMELA, there is a need to model system specification in Extended Finite State Machine (EFSM). This EFSM model is easy to express in PROMELA. The EFSM model with six states is developed for the ATM system which is shown in Figure-5.1.

In EFSM model six states are there at the beginning the system is in idle state, after that if user gives the proper authentication detail it moves to authorization state. If user not gives the proper authentication details it moves to the error state. After authentication it is possible to move in any operation state i.e. Withdraw, Deposit and Balance Inquiry according to card-holder response. If there is any problem with any state or not satisfy the requirement require for the state to move to the next then it moves to an error state. If the operation is perform successfully then it again moves in an idle state.

5.2.2 Linear Temporal Logic Properties Definition

SPIN model checker is used to find the unreachable codes or deadlocks in the system model. In addition SPIN also verifies Linear Temporal Logical (LTL) properties with the help of PROMELA models. LTL allows expressing temporal properties which express the system behavior.

1. **Safety Property:** In all path globally at any point of time both the process do not enter in the critical section. Some examples of safety property are

   - Deposits and withdrawals are mutually exclusive operations forever. It can be expressed in LTL formula as follows:
     $!(\diamond \text{ (withdrawing \&\& depositing)})$
   - If user enters incorrect password then the ATM will never dispense cash. It can be expressed in LTL formula as follows:
     $\Box \text{ (! Authorization} \rightarrow \text{! withdraw)}$

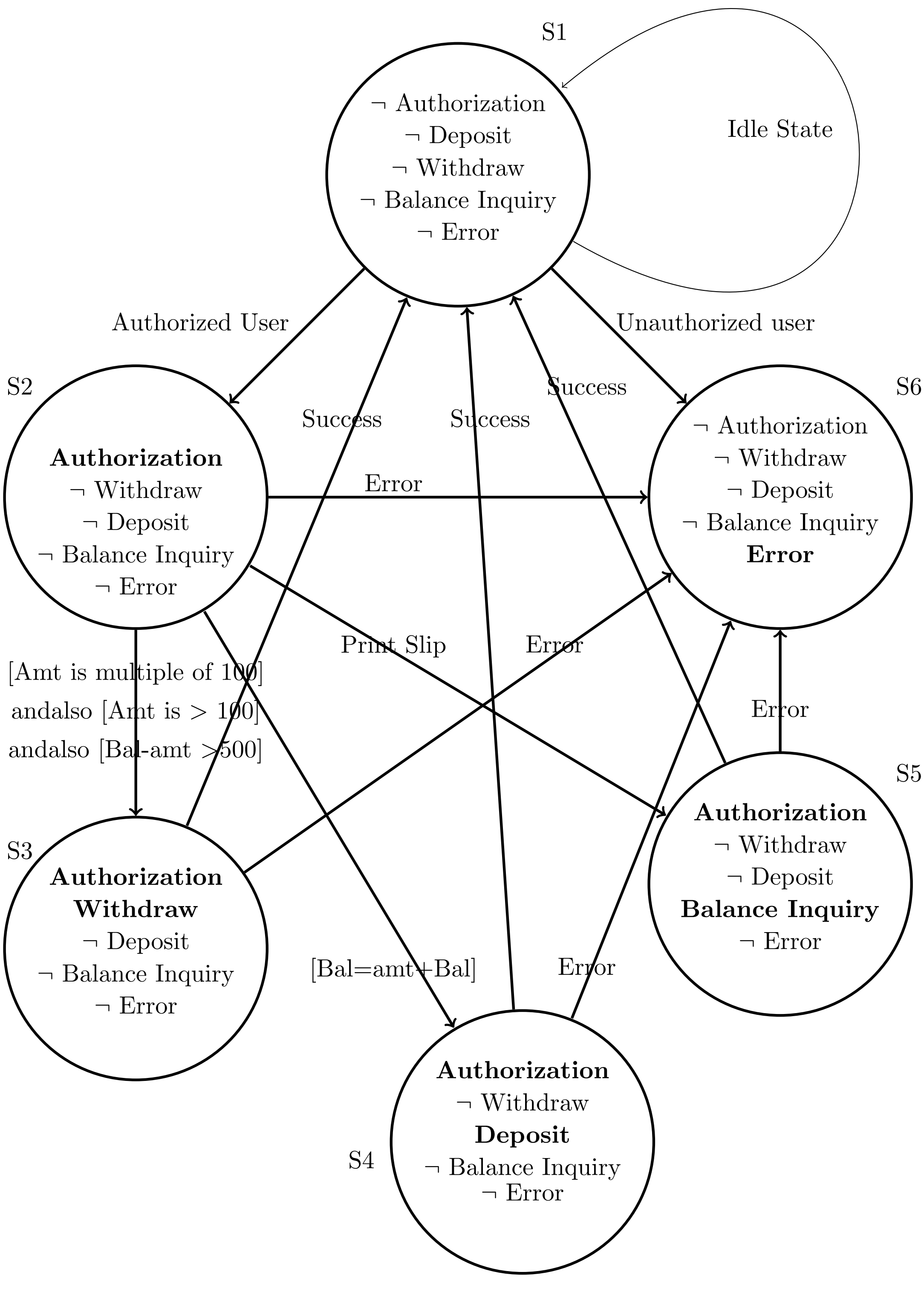


Figure 5.1: State Model of Automated Teller Machine

2. **Liveness Property:** If any process wants to enter into the critical section that process eventually must get the chance to enter into the critical section. Some of the examples of liveness property are

   - If user chooses withdrawal and transaction is successful then ATM will eventually dispense cash.

     $\Box$(withdraw selected && transaction success $\rightarrow \diamond$ cash dispensed)
   - If the ATM dispense cash, it will eventually print receipt.

     $\Box$(cash dispense $\rightarrow \diamond$ receipt printed)

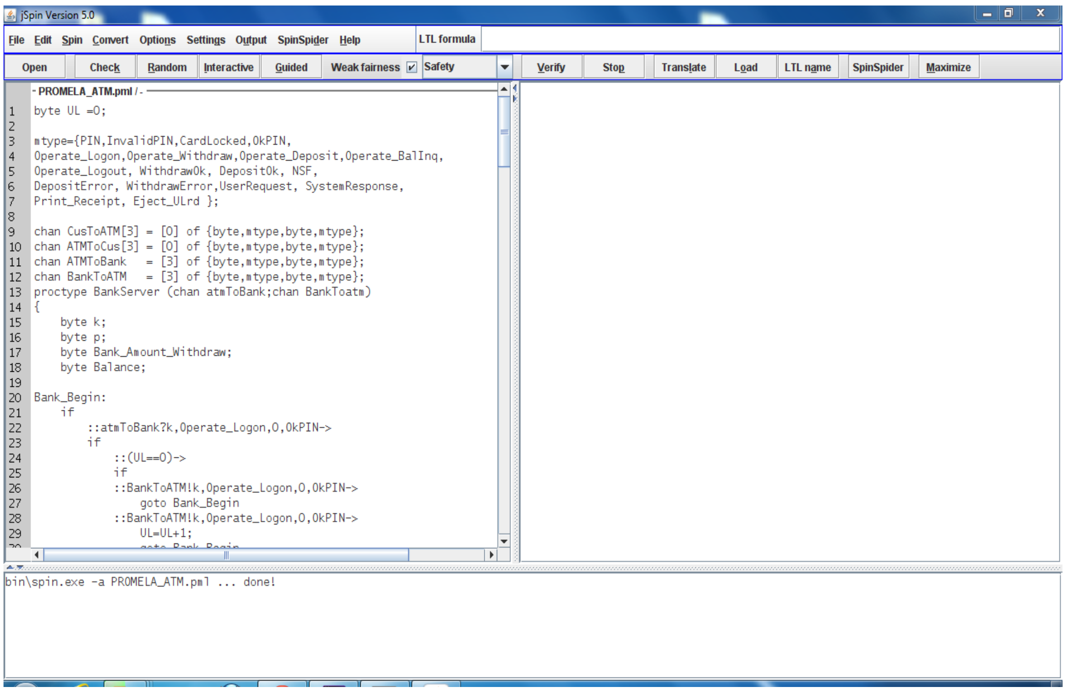
5.2.3 Verification of Model using jSPIN

To verify the system model using jSPIN, PROMELA based model is designed according to system specification. Later this model is verified using jSPIN version 5.0 for verification of the important properties of a system and different LTL formula's which are defined for the specification of the system. Different option are there in jSPIN which are used while verifying system properties and LTL formulae.

1. ***check:*** It is possible to check the error in the PROMELA code using this option of the jSPIN tool. Figure-5.2 shows that there is no error in the code written for the ATM system in PROMELA language.

2. ***Translator*** option is used to translate the LTL formula into the PROMELA format. Here the LTL formula for the liveness property of an ATM system has been defined. This formula shows that the withdrawing and depositing is not possible at the same time in system. By using the translator option that formula is converted to PROMELA format as shown in Figure-5.3.

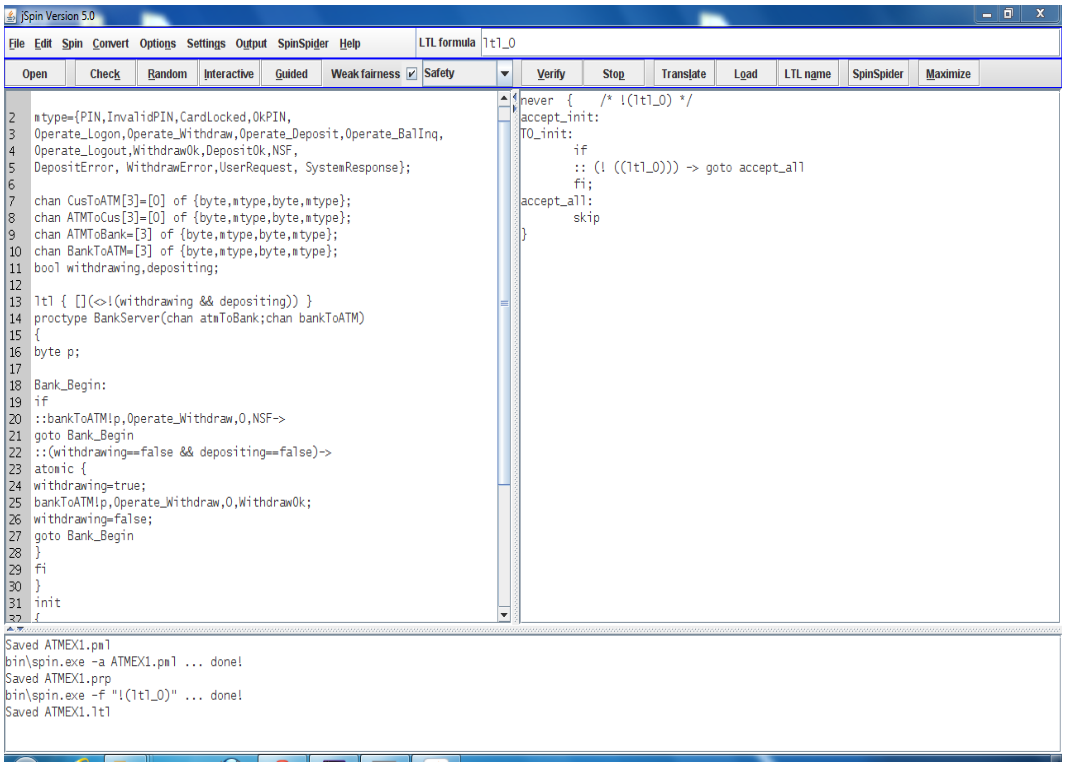
3. ***Random*** option gives the details about interaction of different channels present in the system. Here four channels and different message types are used for the specification of ATM system in PROMELA language which are shown in Figure-5.4. This option shows the random interaction between the mentioned channels and messages as shown in Figure-5.5.


jSpin Version 5.0
File Edit Spin Convert Options Settings Output SpinSpider Help
LTL formula
Open Check Random Interactive Guided Weak fairness ✓ Safety
Verify Stop Translate Load LTL name SpinSpider Maximize
- PROMELA_ATM.pml / -

```
byte UL =0;

mtype={PIN,InvalidPIN,CardLocked,OkPIN,
Operate_Logon,Operate_Withdraw,Operate_Deposit,Operate_BalInq,
Operate_Logout, WithdrawOk, DepositOk, NSF,
DepositError, WithdrawError,UserRequest, SystemResponse,
Print_Receipt, Eject_ULrd };

chan CusToATM[3] = [0] of {byte,mtype,byte,mtype};
chan ATMToCus[3] = [0] of {byte,mtype,byte,mtype};
chan ATMToBank   = [3] of {byte,mtype,byte,mtype};
chan BankToATM   = [3] of {byte,mtype,byte,mtype};
proctype BankServer (chan atmToBank;chan BankToatm)
{
    byte k;
    byte p;
    byte Bank_Amount_Withdraw;
    byte Balance;

Bank_Begin:
    if
        ::atmToBank?k,Operate_Logon,0,OkPIN->
        if
            ::(UL==0)->
            if
            ::BankToATM!k,Operate_Logon,0,OkPIN->
                goto Bank_Begin
            ::BankToATM!k,Operate_Logon,0,OkPIN->
                UL=UL+1;
```

```
bin\spin.exe -a PROMELA_ATM.pml ... done!
```

Figure 5.2: Check Option Showing No Error in PROMELA Code


jSpin Version 5.0
File Edit Spin Convert Options Settings Output SpinSpider Help
LTL formula ltl_0
Open Check Random Interactive Guided Weak fairness ✓ Safety
Verify Stop Translate Load LTL name SpinSpider Maximize

```
mtype={PIN,InvalidPIN,CardLocked,OkPIN,
Operate_Logon,Operate_Withdraw,Operate_Deposit,Operate_BalInq,
Operate_Logout,WithdrawOk,DepositOk,NSF,
DepositError, WithdrawError,UserRequest, SystemResponse};

chan CusToATM[3]=[0] of {byte,mtype,byte,mtype};
chan ATMToCus[3]=[0] of {byte,mtype,byte,mtype};
chan ATMToBank=[3] of {byte,mtype,byte,mtype};
chan BankToATM=[3] of {byte,mtype,byte,mtype};
bool withdrawing,depositing;

ltl { [](<>!(withdrawing && depositing)) }
proctype BankServer(chan atmToBank;chan bankToATM)
{
byte p;

Bank_Begin:
if
::bankToATM!p,Operate_Withdraw,0,NSF->
goto Bank_Begin
::(withdrawing==false && depositing==false)->
atomic {
withdrawing=true;
bankToATM!p,Operate_Withdraw,0,WithdrawOk;
withdrawing=false;
goto Bank_Begin
}
fi
}
init
{
```

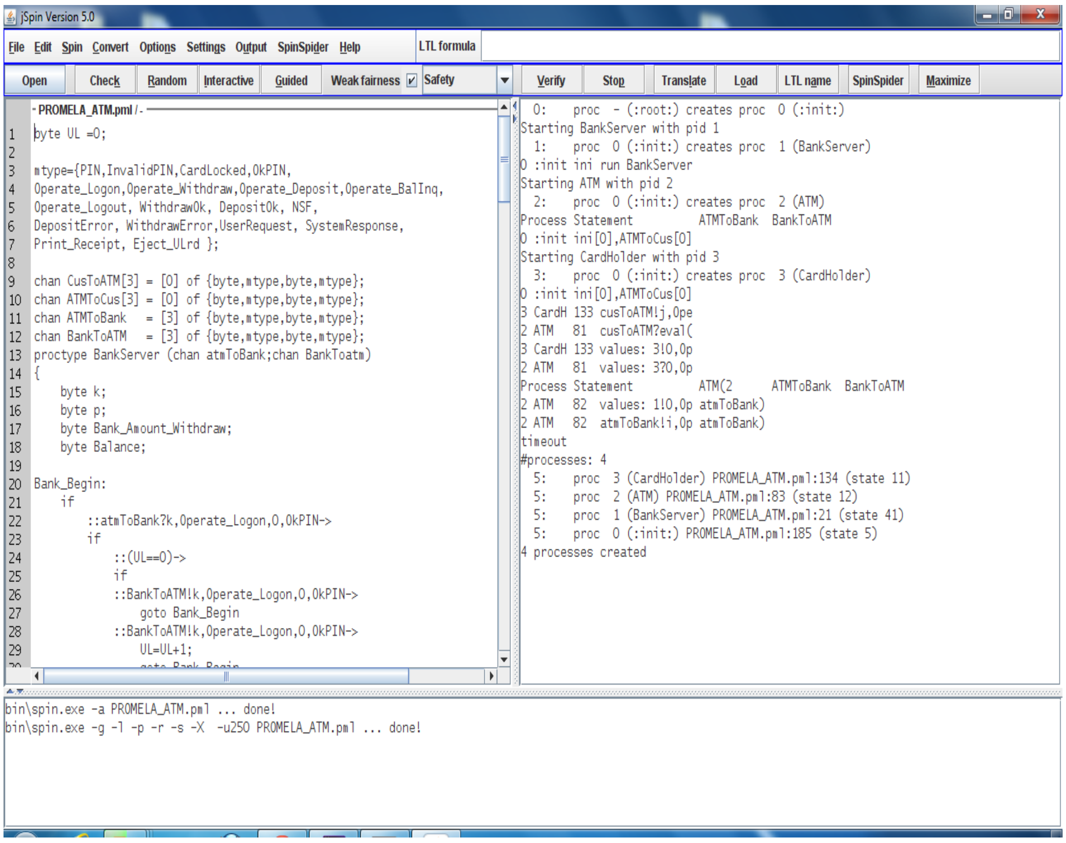
```
never  {    /* !(ltl_0) */
accept_init:
T0_init:
        if
        :: (! ((ltl_0))) -> goto accept_all
        fi;
accept_all:
        skip
}
```

```
Saved ATMEX1.pml
bin\spin.exe -a ATMEX1.pml ... done!
Saved ATMEX1.prp
bin\spin.exe -f "!(ltl_0)" ... done!
Saved ATMEX1.ltl
```

Figure 5.3: LTL Formula translated to PROMELA Format

```
mtype={PIN,InvalidPIN,CardLocked,OkPIN,Operate_Logon,Operate_Withdraw
,Operate_Deposit,Operate_BalInq,Operate_Logout, WithdrawOk,
DepositOk, NSF, DepositError, WithdrawError,UserRequest,
SystemResponse, Print_Receipt, Eject_ULrd };

chan CusToATM[3] = [0] of {byte,mtype,byte,mtype};
chan ATMToCus[3] = [0] of {byte,mtype,byte,mtype};
chan ATMToBank   = [3] of {byte,mtype,byte,mtype};
chan BankToATM   = [3] of {byte,mtype,byte,mtype};
```

Figure 5.4: Channels and Message types for ATM system


jSpin Version 5.0
File Edit Spin Convert Options Settings Output SpinSpider Help
LTL formula
Open Check Random Interactive Guided Weak fairness ✓ Safety
Verify Stop Translate Load LTL name SpinSpider Maximize
- PROMELA_ATM.pml / -

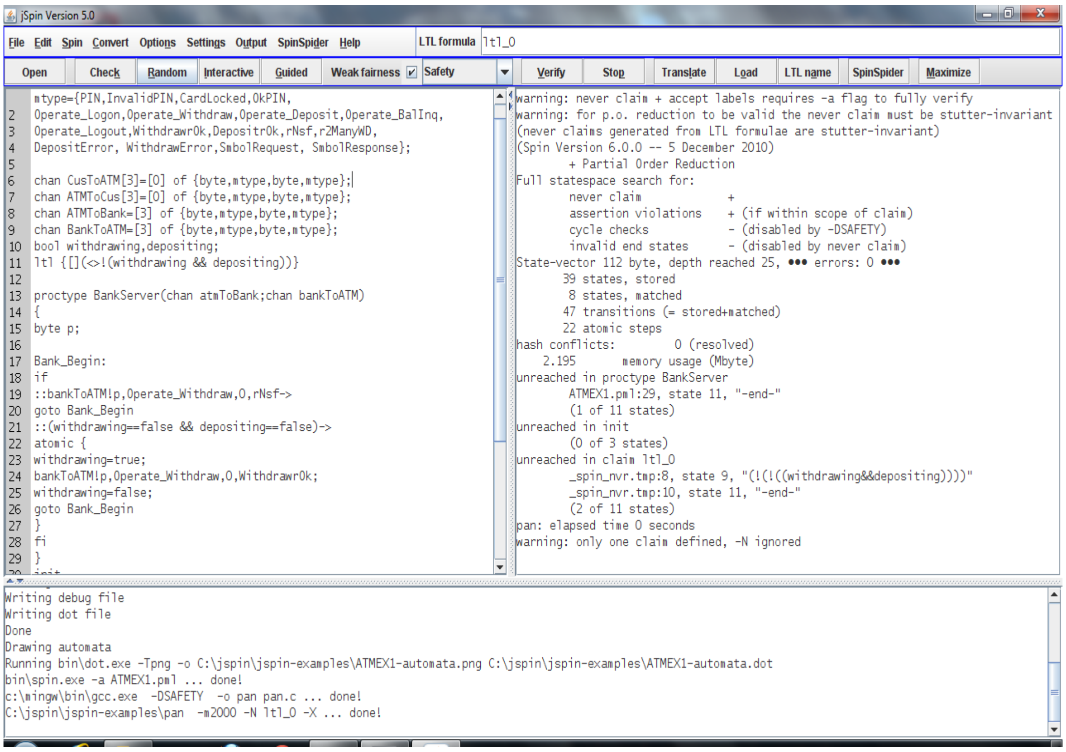
```
byte UL =0;

mtype={PIN,InvalidPIN,CardLocked,OkPIN,
Operate_Logon,Operate_Withdraw,Operate_Deposit,Operate_BalInq,
Operate_Logout, WithdrawOk, DepositOk, NSF,
DepositError, WithdrawError,UserRequest, SystemResponse,
Print_Receipt, Eject_ULrd };

chan CusToATM[3] = [0] of {byte,mtype,byte,mtype};
chan ATMToCus[3] = [0] of {byte,mtype,byte,mtype};
chan ATMToBank   = [3] of {byte,mtype,byte,mtype};
chan BankToATM   = [3] of {byte,mtype,byte,mtype};
proctype BankServer (chan atmToBank;chan BankToatm)
{
    byte k;
    byte p;
    byte Bank_Amount_Withdraw;
    byte Balance;

Bank_Begin:
    if
        ::atmToBank?k,Operate_Logon,0,OkPIN->
        if
            ::(UL==0)->
            if
            ::BankToATM!k,Operate_Logon,0,OkPIN->
                goto Bank_Begin
            ::BankToATM!k,Operate_Logon,0,OkPIN->
                UL=UL+1;
```

```
  0:    proc  - (:root:) creates proc  0 (:init:)
Starting BankServer with pid 1
  1:    proc  0 (:init:) creates proc  1 (BankServer)
0 :init ini run BankServer
Starting ATM with pid 2
  2:    proc  0 (:init:) creates proc  2 (ATM)
Process Statement          ATMToBank  BankToATM
0 :init ini[0],ATMToCus[0]
Starting CardHolder with pid 3
  3:    proc  0 (:init:) creates proc  3 (CardHolder)
0 :init ini[0],ATMToCus[0]
3 CardH 133 cusToATM!j,Ope
2 ATM   81  cusToATM?eval(
3 CardH 133 values: 3!0,0p
2 ATM   81  values: 3?0,0p
Process Statement          ATM(2      ATMToBank  BankToATM
2 ATM   82  values: 1!0,0p atmToBank)
2 ATM   82  atmToBank!i,0p atmToBank)
timeout
#processes: 4
  5:    proc  3 (CardHolder) PROMELA_ATM.pml:134 (state 11)
  5:    proc  2 (ATM) PROMELA_ATM.pml:83 (state 12)
  5:    proc  1 (BankServer) PROMELA_ATM.pml:21 (state 41)
  5:    proc  0 (:init:) PROMELA_ATM.pml:185 (state 5)
4 processes created
```

```
bin\spin.exe -a PROMELA_ATM.pml ... done!
bin\spin.exe -g -l -p -r -s -X  -u250 PROMELA_ATM.pml ... done!
```

Figure 5.5: Random Interaction between channels of ATM system

4. ***Verify*** option is used to verify the LTL define for system. The liveness property i.e withdrawal and deposit, mutually exclusive events is verified jSPIN tool as shown in Figure-5.6.


jSpin Version 5.0
File Edit Spin Convert Options Settings Output SpinSpider Help
LTL formula ltl_0
Open Check Random Interactive Guided Weak fairness ✓ Safety
Verify Stop Translate Load LTL name SpinSpider Maximize

```
   mtype={PIN,InvalidPIN,CardLocked,OkPIN,
2  Operate_Logon,Operate_Withdraw,Operate_Deposit,Operate_BalInq,
3  Operate_Logout,WithdrawrOk,DepositrOk,rNsf,r2ManyWD,
4  DepositError, WithdrawError,SmbolRequest, SmbolResponse};
5
6  chan CusToATM[3]=[0] of {byte,mtype,byte,mtype};
7  chan ATMToCus[3]=[0] of {byte,mtype,byte,mtype};
8  chan ATMToBank=[3] of {byte,mtype,byte,mtype};
9  chan BankToATM=[3] of {byte,mtype,byte,mtype};
10 bool withdrawing,depositing;
11 ltl {[](<>!(withdrawing && depositing))}
12
13 proctype BankServer(chan atmToBank;chan bankToATM)
14 {
15 byte p;
16
17 Bank_Begin:
18 if
19 ::bankToATM!p,Operate_Withdraw,0,rNsf->
20 goto Bank_Begin
21 ::(withdrawing==false && depositing==false)->
22 atomic {
23 withdrawing=true;
24 bankToATM!p,Operate_Withdraw,0,WithdrawrOk;
25 withdrawing=false;
26 goto Bank_Begin
27 }
28 fi
29 }
```

```
warning: never claim + accept labels requires -a flag to fully verify
warning: for p.o. reduction to be valid the never claim must be stutter-invariant
(never claims generated from LTL formulae are stutter-invariant)
(Spin Version 6.0.0 -- 5 December 2010)
        + Partial Order Reduction
Full statespace search for:
        never claim             +
        assertion violations    + (if within scope of claim)
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      - (disabled by never claim)
State-vector 112 byte, depth reached 25, ••• errors: 0 •••
       39 states, stored
        8 states, matched
       47 transitions (= stored+matched)
       22 atomic steps
hash conflicts:         0 (resolved)
    2.195       memory usage (Mbyte)
unreached in proctype BankServer
        ATMEX1.pml:29, state 11, "-end-"
        (1 of 11 states)
unreached in init
        (0 of 3 states)
unreached in claim ltl_0
        _spin_nvr.tmp:8, state 9, "(!(!((withdrawing&&depositing))))"
        _spin_nvr.tmp:10, state 11, "-end-"
        (2 of 11 states)
pan: elapsed time 0 seconds
warning: only one claim defined, -N ignored
```

```
Writing debug file
Writing dot file
Done
Drawing automata
Running bin\dot.exe -Tpng -o C:\jspin\jspin-examples\ATMEX1-automata.png C:\jspin\jspin-examples\ATMEX1-automata.dot
bin\spin.exe -a ATMEX1.pml ... done!
c:\mingw\bin\gcc.exe  -DSAFETY  -o pan pan.c ... done!
C:\jspin\jspin-examples\pan  -m2000 -N ltl_0 -X ... done!
```

Figure 5.6: Verification of Safety Property Using SPIN

5. ***Spin spider*** gives the automata for the system showing interaction between different processes. The spin spider automata generated for the system while verifying the above safety property is shown in the Figure-5.7. The node represents the line number of the code and edges shows the condition that are required to satisfy the move to the next state.

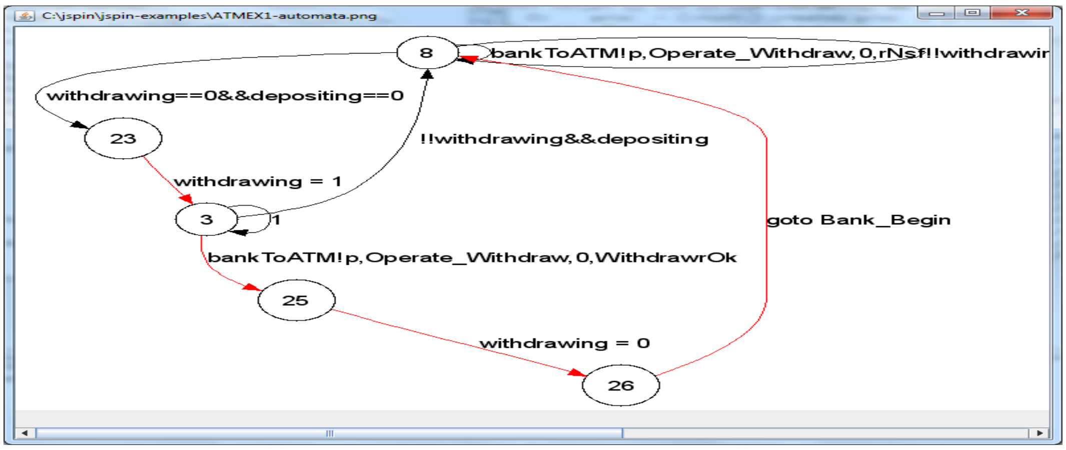

C:\jspin\jspin-examples\ATMEX1-automata.png
8
bankToATM!p,Operate_Withdraw,0,rNsf!!withdrawir
withdrawing==0&&depositing==0
23
!!withdrawing&&depositing
withdrawing = 1
3
1
goto Bank_Begin
bankToATM!p,Operate_Withdraw,0,WithdrawrOk
25
withdrawing = 0
26

Figure 5.7: Automata Generated using SPIN

5.3 Summary

This chapter, presents a case study for model checking through an example of verifying ATM with Simple PROMELA Interpreter (SPIN). The ATM system was modeled in Process or Protocol Meta Language (PROMELA) for business flow of an ATM system. It is then checked for deadlock and unreachable code with SPIN model checker. It also shows how properties of an ATM can be expressed in the form of linear temporal logic statements and then verified using SPIN model checker.

Chapter 6

Chapter 6

Conclusion and Future Work

6.1 Conclusion

- In this thesis an approach was explained to reduce the gap between informal and formal methods of loosely coupled software specification, verification, and validation methodologies.

- An approach for transforming UML interaction overview diagram to Colored Petri Net models is proposed. This transformation is used to bridge the gap between informal notation (UML diagrams) and more formal notation (CPN models) for analysis purpose. CPN model is validated by CPN tool. The state space method of CP-nets makes it possible to validate and verify the functional correctness of systems.

- Model checking methodology through an example of verifying ATM with Simple PROMELA Interpreter (SPIN) is explained. The ATM system was modeled in Process or Protocol Meta Language (PROMELA) for business flow of an ATM system. It is then checked for deadlock and unreachable code with SPIN model checker. It also shows how properties of an ATM can be expressed in the form of linear temporal logic statements and then verified using SPIN model checker.

6.2 Future Work

- Further, performance properties such as safety, liveness and fairness of the system model can be analyzed on the basis of time delay calculated using CPN and state space analysis.
- It is possible to use the CPN and SPIN model together to transform software architecture ‘description diagrams’ into an executable model. The timed concepts in petri net is used for performance analysis of software system. SPIN model checker is used to evaluate the non-functional requirements of system model.

Bibliography

[1] W. Wenxin and M. Saeki, "Specifying software architectures based on coloured petri nets," *IEICE TRANSACTIONS on Information and Systems*, vol. 83, no. 4, pp. 701–712, 2000.

[2] S. Emadi and F. Shams, "A new executable model for software architecture based on petri net," *Indian Journal of Science and Technology*, vol. 2, no. 9, pp. 15–25, 2009.

[3] J. Rumbaugh, I. Jacobson, and G. Booch, *Unified Modeling Language Reference Manual, The.* Pearson Higher Education, 2004.

[4] K. Jensen, "A brief introduction to coloured petri nets," in *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 203–208, Springer, 1997.

[5] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[6] H. T. Jung and S. H. Joo, "Transformation of an activity model into a colored petri net model," in *Trendz in Information Sciences & Computing (TISC), 2010*, pp. 32–37, IEEE, 2010.

[7] CPN Tools, "http://cpntools.org/."

[8] J. D. García, J. Carretero, J. M. Pérez, F. Garcia, and R. Filgueira, "Specifying use case behavior with interaction models," *Journal of Object Technology*, vol. 4, no. 9, pp. 143–159, 2005.

[9] S. Leue, *Specifying real-time requirements for SDL specifications: a temporal logic-based approach.* Bibliothek der Universität Konstanz, 1995.

[10] G. J. Holzmann, "The model checker spin," *Software Engineering, IEEE Transactions on*, vol. 23, no. 5, pp. 279–295, 1997.

[11] H. Shi, W. Ma, M. Yang, and X. Zhang, "A case study of model checking retail banking system with spin," *Journal of Computers*, vol. 7, no. 10, pp. 2503–2510, 2012.

[12] G. J. Holzmann, "Basic spin manual," 1980.

[13] J. Bowles and D. Meedeniya, "Formal transformation from sequence diagrams to coloured petri nets," in *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*, pp. 216–225, IEEE, 2010.

[14] H. Störrle and J. Hausmann, "semantics of uml 2.0 activities," in *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing*, 2004.

[15] T. S. Staines, "Intuitive mapping of uml 2 activity diagrams into fundamental modeling concept petri net diagrams and colored petri nets," in *Engineering of Computer Based Systems, 2008. ECBS 2008. 15th Annual IEEE International Conference and Workshop on the*, pp. 191–200, IEEE, 2008.

[16] O. R. Ribeiro and J. M. Fernandes, "Some rules to transform sequence diagrams into coloured petri nets," in *7th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools (CPN 2006)*, pp. 237–56, Citeseer, 2006.

[17] M. Elkoutbi and R. K. Keller, "Modeling interactive systems with hierarchical colored petri nets," in *Proc. of*, pp. 432–437, Citeseer, 1998.

[18] A. Kamandi, M. Abdollahi Azgomi, and A. Movaghar, "Transformation of uml models into analyzable osan models," *Electronic Notes in Theoretical Computer Science*, vol. 159, pp. 3–22, 2006.

[19] S. Bernardi, S. Donatelli, and J. Merseguer, "From uml sequence diagrams and statecharts to analysable petri net models," in *Proceedings of the 3rd international workshop on Software and performance*, pp. 35–45, ACM, 2002.

[20] A. Ouardani, P. Esteban, M. Paludetto, and J.-C. Pascal, "A meta-modeling approach for sequence diagrams to petri nets transformation within the requirements validation process," in *Proceedings of the European Simulation and Modeling Conference*, pp. 345–349, 2006.

[21] S. M. Islam, M. H. Sqalli, and S. Khan, "Modeling and formal verification of dhcp using spin," *International Journal of Computer Science & Applications*, vol. 6, no. 3, pp. 145–159, 2006.

[22] M. Mongiello, "Finite-state verification of the ebxml protocol," *Electronic Commerce Research and Applications*, vol. 5, no. 2, pp. 147–169, 2006.

[23] V. A. Oleshchuk, "Modeling, specification and verification of ad-hoc sensor networks using spin," *Computer Standards & Interfaces*, vol. 28, no. 2, pp. 159–165, 2005.

[24] X. Yu, Z. Wang, G. Pu, D. Mao, and J. Liu, "The verification of rcos using spin," *Electronic Notes in Theoretical Computer Science*, vol. 207, pp. 49–67, 2008.

[25] C. Choppy, K. Klai, and H. Zidani, "Formal verification of uml state diagrams: a petri net based approach," *ACM SIGSOFT Software Engineering Notes*, vol. 36, no. 1, pp. 1–8, 2011.

[26] K. Jensen and L. M. Kristensen, *Coloured petri nets*. Springer, 2009.

[27] L. Wells, "Performance analysis using coloured petri nets," in *Modeling, Analysis and Simulation of Computer and Telecommunications Systems, 2002. MASCOTS 2002. Proceedings. 10th IEEE International Symposium on*, pp. 217–221, IEEE, 2002.

[28] E. Mikk, Y. Lakhnech, M. Siegel, and G. J. Holzmann, "Implementing statecharts in promela/spin," in *Industrial Strength Formal Specification Techniques, 1998. Proceedings. 2nd IEEE Workshop on*, pp. 90–101, IEEE, 1998.

[29] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi, "On the temporal analysis of fairness," in *Proceedings of the 7th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pp. 163–173, ACM, 1980.

[30] L. Dai and K. Cooper, "A survey of modeling and analysis approaches for architecting secure software systems," *International Journal of Network Security*, vol. 5, no. 2, pp. 187–198, 2007.

[31] K. Jensen, L. M. Kristensen, and L. Wells, "Coloured petri nets and cpn tools for modelling and validation of concurrent systems," *International Journal on Software Tools for Technology Transfer*, vol. 9, no. 3-4, pp. 213–254, 2007.

[32] A. K. Zaidi, "On temporal logic programming using petri nets," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 29, no. 3, pp. 245–254, 1999.

[33] J. S. Ostroff, *Temporal logic for real-time systems*, vol. 40. Cambridge Univ Press, 1989.

[34] A. K. Shuja and J. Krebs, *IBM Rational Unified Process Reference and Certification Guide: Solution Designer (RUP).* IBM Press, 2007.