SPIN MODEL CHECKER FOR MACT

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Murali Krishna Somavarapu

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

October 2013

Fargo, North Dakota

North Dakota State University
Graduate School

Title

SPIN MODEL CHECKER FOR MACT

By

MURALI KRISHNA SOMAVARAPU

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Kendall Nygard
Chair

Changhui Yan

Simone Ludwig

Mohamed Khan

Approved:

11/13/2013	Brian M. Slator
Date	Department Chair

ABSTRACT

The SPIN (Simple PROMELA Interpreter) model checker is a verification system used to check high-level models of concurrent systems and improve the overall system's performance. The SPIN model checker takes the PROMELA model as input and verifies the properties or system requirements to check for any errors in the system's design based on the system requirements. The MACT (Model-based Aspect/Class Checking and Testing tool) has an excellent testing approach. The SPIN is added to the MACT to strengthen the results delivered by the MACT for a given aspect-oriented model.

This paper discusses SPIN model implementation for the MACT tool and how SPIN technology is embedded into the MACT tool to check the syntax correctness of the state models. This paper also discusses the conversion of class models into PROMELA models and checks the design-level correctness of the PROMELA model in the MACT tool with respect to the system's requirements.

ACKNOWLEDGMENTS

I would like to express special thanks to my adviser, Dr. Kendall Nygard. His guidance and support greatly helped in completing this work. I would also like to sincerely thank Dr. Changhui Yan, Dr. Simone Ludwig, Dr. Mohamed Khan, and Dr. Dianxiang Xu for their suggestions that helped improve the quality of this paper.

I would like to thank my friends who gave me a lot of encouragement and support during my studies. Special thanks to Ashok, Ganesh, Rajani, and Vinay who gave me lot of support. Also, I would like to thank Dr. Kenneth Magel, Associate Chair, for his kind help during my study at NDSU as a graduate student.

Finally, I would like to thank my family for their support, without which it would have been impossible for me to pursue my graduate program.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

The SPIN model checker is a verification system that can be used to test high-level models of concurrent systems. A PROMELA model is an abstract model of a group of dynamic concurrent processes that explain the life cycle of a software program. It is easy to convert the actual software design to the abstract PROMELA model, which in turn makes it easy to pass the PROMELA model and the properties or system requirements through the SPIN model checker and find out if there are inconsistencies in the given system's behavior.

Given a model specified in PROMELA and the properties for that PROMELA model, SPIN can perform random simulations of the system's execution as well as generate a C program that performs a fast, exhaustive verification of the system state space and matches it against the properties or the system requirements. The verifier can check, for instance, if user-specified system invariants may be violated during a protocol's execution [1].

A PROMELA model consists of data types, arrays, process types, and message passing. Process types will contain states with transitions and events triggering those transitions. The movement from one state to another with provided guard conditions and events is known as transition. It is also possible that guard conditions may be applicable for a particular event to take place. From the given current state and event (guard condition may or may not be applicable), we can predict the next possible state. The guard condition is optional and may not be applicable for all events. In the PROMELA model, the event with a specific guard condition is declared a new event with a unique name.

The MACT tool can handle complex class models and so can the SPIN model checker, which allows for flexibility and is an easier way to test the state models. This paper explores a unique way to convert class models and the syntax of the representing PROMELA models, which will be the class models.

PROMELA Model Semantics

The syntax correctness is handled by the SPIN model checker, which comes in-built and is easy to merge into the MACT tool. Any PROMELA model starts with the keyword "mtype," which is the message type definition and contains the list of events in brackets. It is followed by the channel declaration and the protocols, and main and FileName are listed with all the transactions.

The PROMELA model will be generated from the class model by the MACT tool. By selecting the "Generate PROMELA file" from the menu, the class model will be taken as input, and the PROMELA model file of the specific class model will be generated and later update the list of files in the MACT tool with the newly-generated file. The property files contain the system requirement that needs to be met by the PROMELA model. There are specific notations to represent the system requirements in the properties file, which is a text file with an extension, PRP. As soon as the "Check PROMELA model" option is selected, MACT automatically takes the given PROMELA model file name and checks the properties file in the same directory that has the extension PRP. The semantics for the PROMELA model are explained in detail below.

This synchronizes the MACT tool with the new PROMELA file generated and automatically opens the PROMELA file in the MACT editor for viewing/editing purposes. The conversion algorithm has a flexible technique that helps convert any complex class model into a SPIN executable PROMELA model file and, in turn, can be used to check the inconsistencies in the design. The newly-designed system can complete the following three tasks:

1. Generate the PROMELA file
2. Check the PROMELA file syntax
3. Check the PROMELA model's design using SPIN

If a user selects "Generate PROMELA file" and the currently active file in the editor is a class model file, a file with *.cm extension, the MACT tool automatically generates the PROMELA file with the same name as the class model file but with a different extension, PML. Using the conversion algorithm semantics, it generates the PROMELA file and later opens it up in the MACT editor for viewing and editing purposes. The MACT tool also gives the user the freedom to select different options regarding the SPIN so that the user can perform one of the three tasks listed above. The task will be accomplished based on the file format selected in the MACT tool. This paper covers the SPIN model checking feature for the MACT tool. Even though there is a wide range of file formats accepted by MACT, the file type that we considered for the generation of the PROMELA model file is the class model.

When a user opens a class model file and selects the option "Generate PROMELA file," the MACT editor gets the extension automatically and checks whether it is "*.cm" or not. Later, the conversion algorithm reads all the states, events, and transitions into arrays and objects and sends the collected data to the respective class instances. If there are any syntax errors in the class model file that do not follow the general class model semantics, an error will be displayed as a pop-up message saying that the class model file is designed with a few errors in it and the conversion program aborts.

The MACT tool used to allow the user to open/edit only the class model, aspect model, and project model files dynamically, which means only a file of CM, AM, or PM extensions could be opened. The modifications are made in such a way that it also allows the PROMELA model file to be opened in the MACT editor for viewing/editing purposes, which means it allows opening a file with a PML extension. For example, it used to allow a user to only open the file with extensions CM, AM, and PM. There is a Java method that contains a list of acceptable file formats and the actions to be done based on the file type selected. The modification is done to ensure that files with a PML extension are also allowed to be opened in the MACT editor.

A user can create a PROMELA model file that can include all the required lists of “mtype,” which is the message type definition, the list of events in the brackets, and the channel declaration and protocols, main and file name listed with all the transactions. Now, the file needs to be checked for syntax as a PROMELA model with the SPIN in-built syntax checker for the PROMELA model files. After checking for syntax, this PROMELA model will be used to check the consistency in the design using the SPIN verification process. By opening a class model file, the tool allows the selection of one of three options available with respect to the SPIN features, and by opening a PROMELA model file, the tool allows the selection of two of three options available with respect to the SPIN features. These can be simply attained by going to the OPEN menu and selecting the class or PROMELA model file, and it opens the selected file in the MACT editor.

A user can select either the class or PROMELA file from the tree available on the left side of the MACT tool, which contains a list of all previously opened files. After selecting the class model, creating the PROMELA file, and checking for syntax, the PROMELA model file will be tested for possible inconsistencies in the design using the SPIN model checker. The MACT tool is enhanced with the additional functionality of the ability to test the class model using the SPIN model checker by converting it into the PROMELA model file. This ensures that the MACT tool can test the state models with not just one approach but two approaches.

The MACT tool can also test the class model itself by generating test cases for the class models. The added functionality of checking the class model through the SPIN model checker makes the MACT tool more reliable. The PROMELA model generator for the MACT tool is designed in such a manner that it is more general and applicable to any complex class model and can be converted to a PROMELA model file.

This paper describes the conversion algorithm used in the MACT tool and, more precisely, the technique used to take the class model as input and generate the PROMELA model as output. The conversion algorithm is a unique technique to convert the class model file to a PROMELA model file with the same name as the class model file. The extension given for the PROMELA model file is PML (PROMELA model file). The MACT tool is based on Java and JSpin (a Java version of the SPIN model checker). JSpin is based on Java and also uses a few other executables to attain the task of checking the PROMELA model file for errors with the SPIN model checker. The JSpin (Java version of SPIN) is used to check the syntax correctness as well as the inconsistencies in the PROMELA model file design.

JSpin

The JSpin [6] is a graphical user interface for the SPIN model checker that is used for verifying concurrent and distributed programs. It is a completely Java-based application GUI (Graphical User Interface) and also has the ability to do the functionality from the command line, which will later be incorporated into the MACT tool to achieve the tasks done by JSpin. The user interface of JSpin is simple and consists of a single window with menus, a toolbar, and three adjustable text areas and is not as complex as the MACT tool. SPIN option strings are automatically supplied and the SPIN output is filtered and presented in tabular form. Each class model file has its own syntactic semantics and uses some keywords alone. Similarly, each class model file has its own syntactic semantics. Here are a few keywords used by the PROMELA model file: mtype, chan, if, do, fi, and od. The aspect model will be weaved with the class model using the weaving mechanism to form a woven model in the MACT tool that will be a crucial step for testing state models. The syntax can be checked for correctness using JCC and, more precisely, the syntax of the class, aspect, and project models can be checked.

The PROMELA models in the MACT tool are designed in a text document with the extension PML. The PROMELA model file generated looks different from the respective class model, but it is easier to figure out how the states, events, and transitions are designed in the PROMELA file. It is possible to type in a PROMELA model if the user has design alone and does not have the class model for it. That is, it is possible to create the PROMELA model files with or without the MACT tool, and the MACT tool can still use the same PROMELA model file for testing purposes. The MACT tool has an editor to open the class model, aspect model, and project model files. It is modified to make it possible to open the PROMELA model file to edit it if the user wants to. This makes the MACT tool flexible to edit the PROMELA model file with additional functionalities that are not in the class model and still test the PROMELA model file for inconsistencies.

CHAPTER 2. LITERATURE REVIEW

This chapter explains the JSpin mechanism for the SPIN model checker and how the pairing of JSpin with the MACT tool allows tasks done by JSpin to also be done by the MACT tool. This chapter describes the features of JSpin, which can be used in designing a PROMELA model file from the given class model, underlying Java classes that are used to convert the class model file to a PROMELA model file and the semantics for the PROMELA model file. A list of keywords is used as the look-up values in the PROMELA model file, which helps the SPIN model checker identify the states, events, and transitions of a given PROMELA model file.

The MACT tool is based on Java and the conversion program is also implemented in Java. The PROMELA model files have the unique pseudo code used by the SPIN model checker, and research has been done to convert the class model to its relative PROMELA model exactly without any changes from a design perspective. Each PROMELA model has its own syntax but still follows one generalized semantic, so it is easy to check the expected semantics versus the actual semantics in the PROMELA model file.

PROMELA Model

Converting the class model file to a PROMELA model file is a crucial step to ensure that the SPIN model checker is testing what it is supposed to test in the design model. Representing the PROMELA model from the actual design with the help of a list of JSpin keywords is critical in the design of a PROMELA model. A class model consists of a name, events, lists of states, and the transitions occurring in it.

A PROMELA model file is similar to a class model file but is completely different in syntax and semantics. The mtype values in the PROMELA model can be encountered as events in the class model file, the escape sequences in the PROMELA model file can be encountered as the states in the class model file, and the selection in the PROMELA model file can be encountered as the events firing the transitions in the class model file. A generic PROMELA model can have compound statements, deterministic steps, inline definitions, and reading input, but the PROMELA model designed for the class model does not contain the entire list above. It only contains message types, channels, escape sequences, and process types. The discussion of all the features available in the PROMELA language is beyond the scope of this paper.

In the PROMELA model file, all the escape sequences should be listed in the process types and the message types. If one is missing, then the SPIN model checker will report an error saying that the escape sequence is not defined in the message types. Each PROMELA model file has its own list of message types and escape sequences, which will be considered as constants by the SPIN model checker, but the keywords like message types and channels will be checked for syntax. If there is any syntax error then the user will be notified of the error. The SPIN model checker will be used to check the syntax of the PROMELA model file based on the in-built functionality to check the semantics of any PROMELA model file. The JSpin is capable of checking the syntax of the PROMELA model file as well as checking the design level flaws in the given PROMELA model file.

The JSpin contains RunSpin class, which executes the executable spin.exe file located in the local drive to check the syntax of the PROMELA file. The output after the execution of the spin.exe file, which may be either a success message or an error message, will be returned to the MACT tool, which will be displayed as a message.

JSpin

JSpin (Java version of the SPIN model checker) supports an easy way to use the functions so that the functionality can be replicated by any Java application if necessary. JSpin was developed uniquely to test the PROMELA model files using the Java platform as a core feature. It makes the system calls to execute the SPIN files to receive the output.

A PROMELA model is constructed mainly from basic types of objects [3]: Processes, Data Objects, and Message Channels. The definitions and semantics of these are described below with examples.

Processes

Processes are instantiations of the proctypes in the PROMELA model file. Even though multiple instantiations of the proctypes are possible by simply adding the number in square brackets after the keyword, active, in this paper we only instantiate once. That will be enough to modify the class model to a PROMELA model file. A simple example of how the processes can be coded is shown below:

```
active proctype bankAccount()
{
……
}
```

This will create a process with a name, bankAccount, which is the file name of the class model provided.

Data Objects

The Data Objects are the data variables declared and used throughout the processes. The data objects can be bit, Boolean, byte, channel, mtype, pid, short, int, or unsigned.

Though there are many types of data objects, we use only mtype and channel in the creation of the PROMELA model file. A sample of how to code the data objects is as follows:

```
Init {
mtype n = sample; /* initialize n to sample */
printf(“the value of n is : ”);
printm(n);
}
```

Here the data object “n” is of mtype and is initialized to the sample. The output of the process would be printing the value “sample.”

Message Channels

Message channels are used to model the exchange of data between processes. We use these to declare the events that trigger the transitions from one state to another. The guard conditions themselves are coded into a new event and listed in the events. The sample of how the message channels are coded is given below:

```
mtype = {new, getBalance, deposit, withdraw, close, freeze, unfreeze};
chan tpc = [0] of { mtype };
active proctype main()
{
….
}
```

Here, the declaration mtype has seven types of messages. The channel is declared in the next line with the message type, mtype.

Semantics of JSpin

There are a few keywords that help JSpin recognize the design of the PROMELA model file and parse the PROMELA model file to get the list of states, events, and transitions, and finally test the design. These keywords work as a framework to build the PROMELA model file. The reserved keywords supported by JSpin are listed below, and the function of each reserved keyword is also explained in detail later in this section.

i. Declare the events in the class model for the PROMELA model file using the message types.

ii. Declare the message channels, which will simply be the list of the message types mentioned above.

iii. Declare the process types: one with the name “main” and another with the class model name. This will be explained later in this section.

iv. List all the escape sequences based on the rules of executables. These are basically the source states with the target states, and the rules are nothing but the events that are firing the transition to take place.

Following are some of the reserved words that are recognized by JSpin to check the functional semantics for the PROMELA model file. There are more keywords that JSpin recognizes, but the explanation of those keywords is beyond the scope of this paper.

Mtype

This keyword declares the message types for the PROMELA model file, which will be the list of events from the class model file. This will be used later in the PROMELA model file to declare the channels that contain all the transitions mentioned in the class model. These are case-sensitive, and the SPIN model checker recommends using the lower case for all events declared in the message types. An example is given below to illustrate the basic syntax for this command and how it is used later in the PROMELA model file:

```
/*BankAccount.pml */
mtype = {new};
chan tpc = [0] of { mtype };
active proctype main()
{
ALPHA:        tpc!new;
…
}
active proctype bankAccount()
{
ALPHA:
if
:: tpc?new -> goto OPEN
fi;
}
```

The keyword “mtype” is used to declare the message types that contain the list of events separated by commas and enclosed in curly brackets. The event names should not contain any special characters like brackets or empty spaces. They should also not contain reserved words like init, end, and start. If any special characters or reserved words are found in the mtype declaration, JSpin will report an error saying that these characters are not allowed.

Chan

This keyword declares the message channels for the PROMELA model file, which is the simple declaration of channels with zero in square brackets followed by the keyword “of,” followed by the message type in curly brackets. This will be used later in the PROMELA model file to do all the transitions mentioned in the class model. These are case-sensitive, and the SPIN model checker recommends using the lower case for the channel declared for the message types. An example is given below to understand the basic syntax for this command and how it is used later in the PROMELA model file:

```
/*BankAccount.pml */
mtype = {new};
chan tpc = [0] of { mtype };
active proctype main()
{
ALPHA:      tpc!new;
…
}
active proctype bankAccount()
{
ALPHA:
if
:: tpc?new -> goto OPEN
fi;
}
```

The keyword “chan” is used to declare the message channel that contains the message type enclosed in curly brackets. The message type used by the PROMELA model file should be declared prior to its usage. We generally use just one instance of the message channel, and it is sufficient to do a PROMELA model file from any given complex class model file. If any error is found in the message channel declaration, the JSpin will report an error saying that the message channel declaration has a few errors, and it also gives the line number to fix them more easily.

Processes

The process declaration is the most crucial of all the steps since it contains the core transitions of the class model. This keyword helps declare the processes for the PROMELA model file, which will be a total of two. The first one will be the main process that instantiates the process, and the second one will contain all the transitions. This will be used by JSpin to test the whole PROMELA model file for inconsistencies in the design level model. These are case-sensitive and the SPIN model checker recommends using the lower case for all the events but upper case for all the states in the PROMELA model. By following this, it will be much easier to differentiate states and events in the PROMELA model file. An example is given below to show the basic syntax for this command and how it is used later in the PROMELA model file.

```
/*BankAccount.pml */
mtype = {new};
chan tpc = [0] of { mtype };
active proctype main()
{
```

```
ALPHA:        tpc!new;
…
…
…
}
active proctype bankAccount()
{
ALPHA:
if
:: tpc?new -> goto OPEN
fi;
}
```

The keyword “proctype” is used to declare the processes. In this example there are two processes declared, which is generalized in all the PROMELA model files that are designed by the MACT tool.

The first process has the name “main(),” which is common in all the MACT designed PROMELA model files. It contains all the transitions without the conditional or events in it. The second process has the name “bankAccount,” which is the name of the provided class model file name. It contains all the transitions with the events firing the transitions. It is not possible to contain reserved words like init, end, and start as the file names since the process will be declared with the name of the reserved keyword and throws an error at the run-time syntax check of the PROMELA model file. The JSpin will report an error saying that the process type with the keyword is not allowed.

Escape Sequence

The escape sequence is the simple IF condition used to make it possible for the transitions in the class model to be converted into the PROMELA model. The IF condition is used to let the source state undergo transition to the target state for the given event. This keyword helps to follow the flow of the class model in the PROMELA model file. The IF condition starts with the keyword “IF” and will continue the conditional statements on every proceeding line until the ending keyword “FI” is encountered. This will be used by JSpin to code the transitions from the class model to the PROMELA model file in a more generalized manner. This IF condition is not case-sensitive, but the SPIN model checker recommends using the lower case for all IF conditions to better understand the PROMELA model file. An example is given below to explain the basic syntax of the IF condition and how it is used later in the PROMELA model file.

```
/*BankAccount.pml */
mtype = {new, getBalance, deposit};
chan tpc = [0] of { mtype };
active proctype main()
{
ALPHA:      tpc!new;


OPEN:
if
:: tpc!getBalance -> goto OPEN
:: tpc!deposit -> goto OPEN
fi;
}
active proctype bankAccount()
{
ALPHA:
if
:: tpc?new -> goto OPEN
fi;
OPEN:
if
:: tpc?getBalance -> goto OPEN
:: tpc?deposit -> goto OPEN
fi;
}
```

In the above example, the IF condition starts with the keyword “if.” The next line (:: tpc?getBalance -> goto OPEN) is the conditional statement to check the tpc. If it is equal to getBalance, then the transition is done to the target state “OPEN” from the source state “OPEN.” This is the same with the deposit event. The IF condition ends with the keyword “fi” and the condition starts with the double colons.

CHAPTER 3. PROMELA MODEL

This chapter explains the syntax of the PROMELA-based state models, the BNF (Backus-Naur Form) notation [3] developed for the PROMELA models. It also covers how JSpin is incorporated into the MACT tool to perform the functionalities of JSpin from the MACT tool itself. This chapter also provides instructions for the user to generate a PROMELA model file from the class model file using the MACT tool.

This paper describes the BNF notation for the PROMELA models to handle all the generic class models so that it can be used by the MACT tool to generate the PROMELA model file by giving any class model file as input. Designing the semantics of the PROMELA model file for the MACT tool is a crucial part of this paper. The class model is a simple text file with the extension CM, and it will be converted to its respective PROMELA model file, which is basically a text file generated by the MACT tool with the extension PML but with the same name as the class model file. The keywords will help the conversion program get the list of states, events, and transitions in the class model file. This list of values will be passed to the FileToArray class object in the MACT tool, which in turn will complete a few calculations based on the conversion algorithm and finally produce the PROMELA model file.

Using the conversion program to generate a PROMELA model file automatically is efficient, as it is very easy to generate the PROMELA model file from the user's perspective. It is as simple as clicking one button after selecting a class model file into the MACT tool's editor. The testing users do not need to learn the syntax of the PROMELA language at all, and even the errors in the given PROMELA model file are clearly and easily understandable by the users. This increases the overall productivity of the system.

This paper explains the development of the PROMELA generation algorithm, which will be generic so it can be used for any class model file. The paper also discusses how the modifications are done with the MACT tool as well as the JSpin to incorporate the functionality of the JSpin in the MACT tool itself. The in-built Java class files are used to do the same thing. Below is the sample PROMELA model file that contains a few events and the transitions of the states.

```
/*BankAccount.pml */
mtype = {new, getBalance, deposit, withdraw, close, freeze, unfreeze};
chan tpc = [0] of { mtype };
active proctype main()
{
ALPHA:      tpc!new;
OPEN:
        if
        :: tpc!getBalance -> goto OPEN
        :: tpc!deposit -> goto OPEN
        :: tpc!withdraw -> goto OPEN
        :: tpc!close -> goto CLOSED
        :: tpc!freeze -> goto FROZEN
        :: tpc!withdraw -> goto OVERDRAWN
        :: tpc!withdraw -> goto OPEN
        fi;
FROZEN:
```

```
            if
            :: tpc!unfreeze -> goto OPEN
            fi;
OVERDRAWN:
            if
            :: tpc!deposit -> goto OPEN
            :: tpc!deposit -> goto OVERDRAWN
            :: tpc!getBalance -> goto OVERDRAWN
            fi
;
CLOSED:  tpc!close;
}
active proctype bankAccount()
{
ALPHA:
            if
            :: tpc?new -> goto OPEN
            fi;
OPEN:
            if
            :: tpc?getBalance -> goto OPEN
            :: tpc?deposit -> goto OPEN
            :: tpc?withdraw -> goto OPEN
```

```
        :: tpc?close -> goto CLOSED
        :: tpc?freeze -> goto FROZEN
        :: tpc?withdraw -> goto OVERDRAWN
        :: tpc?withdraw -> goto OPEN
        fi;
FROZEN:
        if
        :: tpc?unfreeze -> goto OPEN
        fi;
OVERDRAWN:
        if
        :: tpc?deposit -> goto OPEN
        :: tpc?deposit -> goto OVERDRAWN
        :: tpc?getBalance -> goto OVERDRAWN
        fi
;
CLOSED:  tpc?close;
}
```

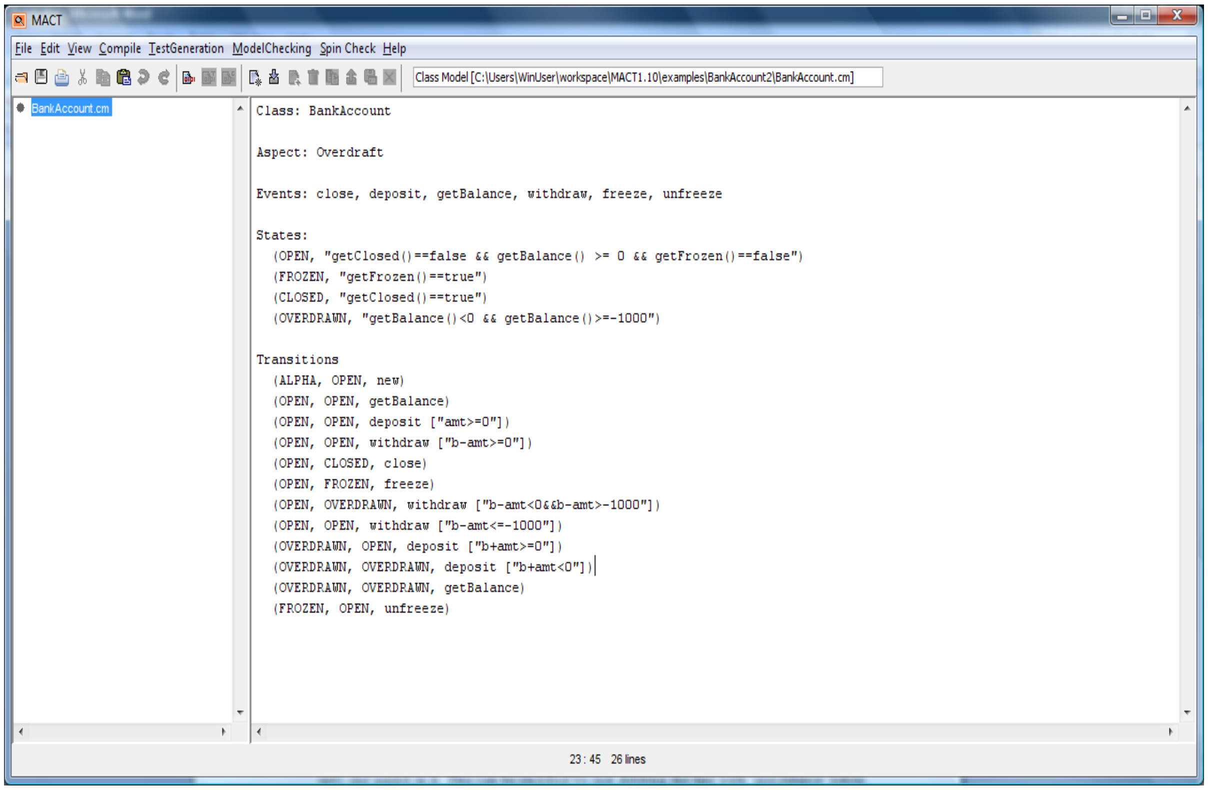
When a user wants to generate the PROMELA model file for the class model selected in the MACT editor, the user needs to select the file and select the “Generate PROMELA file” option from the “SPIN Check” menu. The generatePromelaFile() method is invoked for the specified action. This method will automatically generate the PROMELA model file based on the current class model file selected in the MACT editor. If the file currently active in the MACT editor is not a class model file, the code is written to report the error. This checks the complete list of expected keywords in the class model file and finally passes the values to the FileToArray Java class.

As the class model is selected in the MACT editor and the “Generate PROMELA file” option is selected, the MACT tool automatically generates the PROMELA model file with the same name as the class model file selected, and it also updates the list of previously opened files with the current PROMELA file generated. If the file selected is not a class model then the MACT tool will be able to inform the user that the selected file is not a class model file and ask to select the class model file. This can be achieved by writing the fragment of code in generatePromelaFile() method.

```
if (file == null || !FileUtil.isClassModelFile(file.getName())) {

JOptionPane.showMessageDialog(mactFrame, "Select a class model
file");
return;

}
```

The Java method “generatePromelaFile()” will check whether the file extension is CM. If it is, then it goes ahead with the conversion algorithm to convert the class model file to a PROMELA model file. If there is some syntactic error with the class model it will report the error to the user and give some useful suggestions like the excepted keyword, etc. The MACT tool has a more flexible and sophisticated approach to check the syntax of the class model file, but discussion of that technique is beyond the scope of this paper. Interested readers can read more about it in another paper [5]. If the class model itself is not selected by the user, the active file in the MACT editor is not a class model file, the MACT tool notifies the user of the error with a pop-up message that indicates that the file selected for the generation of a PROMELA model file is not a class model file.

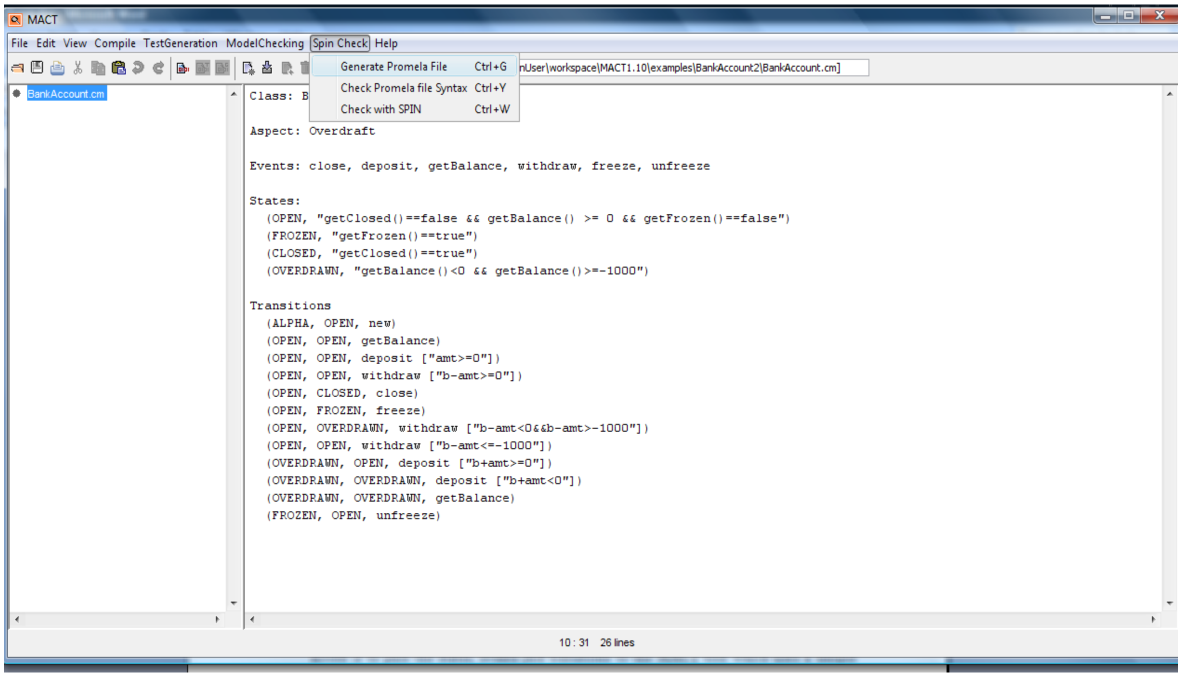
The class model file is simply parsed into states, events, and transitions, and is stored in an ArrayList of strings and sent to the FileToArray Java class to do the conversion from the class model to the respective PROMELA model file. Figure 1 is a screenshot of a simple class model example that has six events, four states, and twelve transitions taking place. The class model file selected does not have any syntactic errors. We can also see the list containing the presently active class model file in the tree on the left side of the MACT tool.


MACT
File Edit View Compile TestGeneration ModelChecking Spin Check Help
Class Model [C:\Users\WinUser\workspace\MACT1.10\examples\BankAccount2\BankAccount.cm]
BankAccount.cm
23 : 45   26 lines

```
Class: BankAccount

Aspect: Overdraft

Events: close, deposit, getBalance, withdraw, freeze, unfreeze

States:
  (OPEN, "getClosed()==false && getBalance() >= 0 && getFrozen()==false")
  (FROZEN, "getFrozen()==true")
  (CLOSED, "getClosed()==true")
  (OVERDRAWN, "getBalance()<0 && getBalance()>=-1000")

Transitions
  (ALPHA, OPEN, new)
  (OPEN, OPEN, getBalance)
  (OPEN, OPEN, deposit ["amt>=0"])
  (OPEN, OPEN, withdraw ["b-amt>=0"])
  (OPEN, CLOSED, close)
  (OPEN, FROZEN, freeze)
  (OPEN, OVERDRAWN, withdraw ["b-amt<0&&b-amt>-1000"])
  (OPEN, OPEN, withdraw ["b-amt<=-1000"])
  (OVERDRAWN, OPEN, deposit ["b+amt>=0"])
  (OVERDRAWN, OVERDRAWN, deposit ["b+amt<0"])
  (OVERDRAWN, OVERDRAWN, getBalance)
  (FROZEN, OPEN, unfreeze)
```

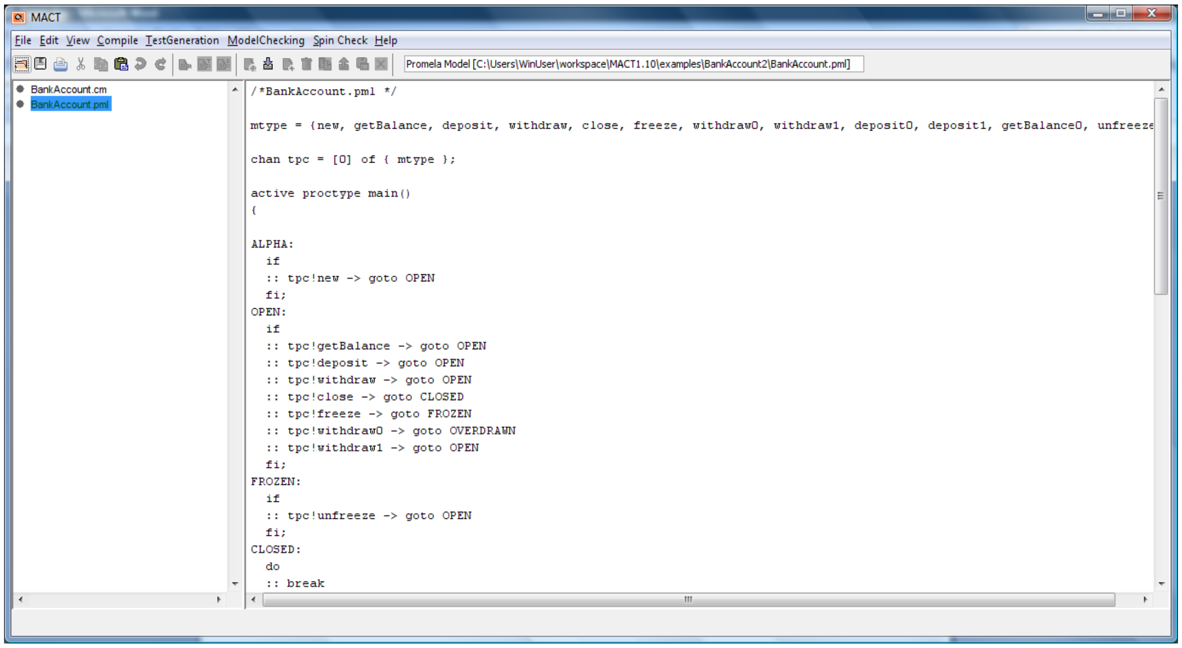
Figure 1. Selecting a Class Model for Generating a PROMELA Model File.

Figure 2 shows how to select the “Generate PROMELA File” option in the MACT tool. The “SPIN Check” menu contains the “Generate PROMELA File” option. Figure 3 shows the newly generated PROMELA model file using the class model as input. The PROMELA model file contains message types, message channels, and escape sequences.


MACT
File Edit View Compile TestGeneration ModelChecking Spin Check Help
Generate Promela File Ctrl+G
Check Promela file Syntax Ctrl+Y
Check with SPIN Ctrl+W
nUser\workspace\MACT1.10\examples\BankAccount2\BankAccount.cm]
BankAccount.cm
10 : 31 26 lines

```
Class: B

Aspect: Overdraft

Events: close, deposit, getBalance, withdraw, freeze, unfreeze

States:
   (OPEN, "getClosed()==false && getBalance() >= 0 && getFrozen()==false")
   (FROZEN, "getFrozen()==true")
   (CLOSED, "getClosed()==true")
   (OVERDRAWN, "getBalance()<0 && getBalance()>=-1000")

Transitions
   (ALPHA, OPEN, new)
   (OPEN, OPEN, getBalance)
   (OPEN, OPEN, deposit ["amt>=0"])
   (OPEN, OPEN, withdraw ["b-amt>=0"])
   (OPEN, CLOSED, close)
   (OPEN, FROZEN, freeze)
   (OPEN, OVERDRAWN, withdraw ["b-amt<0&&b-amt>-1000"])
   (OPEN, OPEN, withdraw ["b-amt<=-1000"])
   (OVERDRAWN, OPEN, deposit ["b+amt>=0"])
   (OVERDRAWN, OVERDRAWN, deposit ["b+amt<0"])
   (OVERDRAWN, OVERDRAWN, getBalance)
   (FROZEN, OPEN, unfreeze)
```

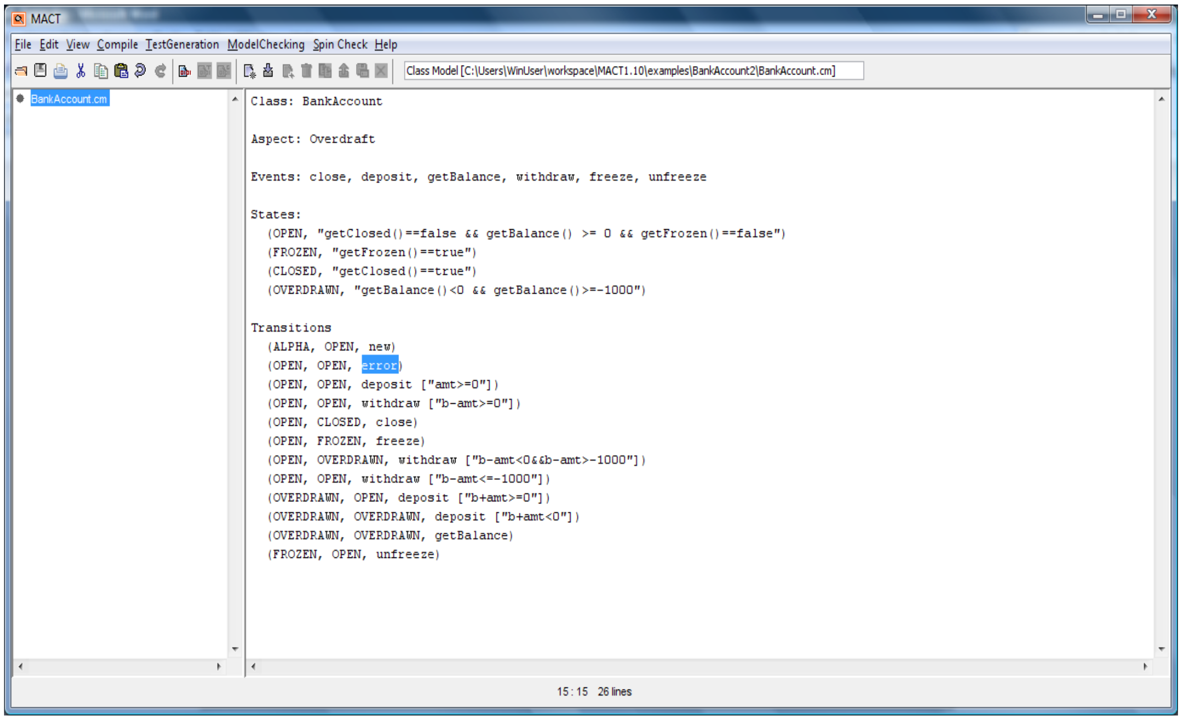
Figure 2. Selecting "Generate PROMELA File Option."


MACT
File Edit View Compile TestGeneration ModelChecking Spin Check Help
Promela Model [C:\Users\WinUser\workspace\MACT1.10\examples\BankAccount2\BankAccount.pml]
BankAccount.cm
BankAccount.pml

```
/*BankAccount.pml */

mtype = {new, getBalance, deposit, withdraw, close, freeze, withdraw0, withdraw1, deposit0, deposit1, getBalance0, unfreeze

chan tpc = [0] of { mtype };

active proctype main()
{

ALPHA:
  if
  :: tpc!new -> goto OPEN
  fi;
OPEN:
  if
  :: tpc!getBalance -> goto OPEN
  :: tpc!deposit -> goto OPEN
  :: tpc!withdraw -> goto OPEN
  :: tpc!close -> goto CLOSED
  :: tpc!freeze -> goto FROZEN
  :: tpc!withdraw0 -> goto OVERDRAWN
  :: tpc!withdraw1 -> goto OPEN
  fi;
FROZEN:
  if
  :: tpc!unfreeze -> goto OPEN
  fi;
CLOSED:
  do
  :: break
```

Figure 3. PROMELA Model File Generated from the Class Model.

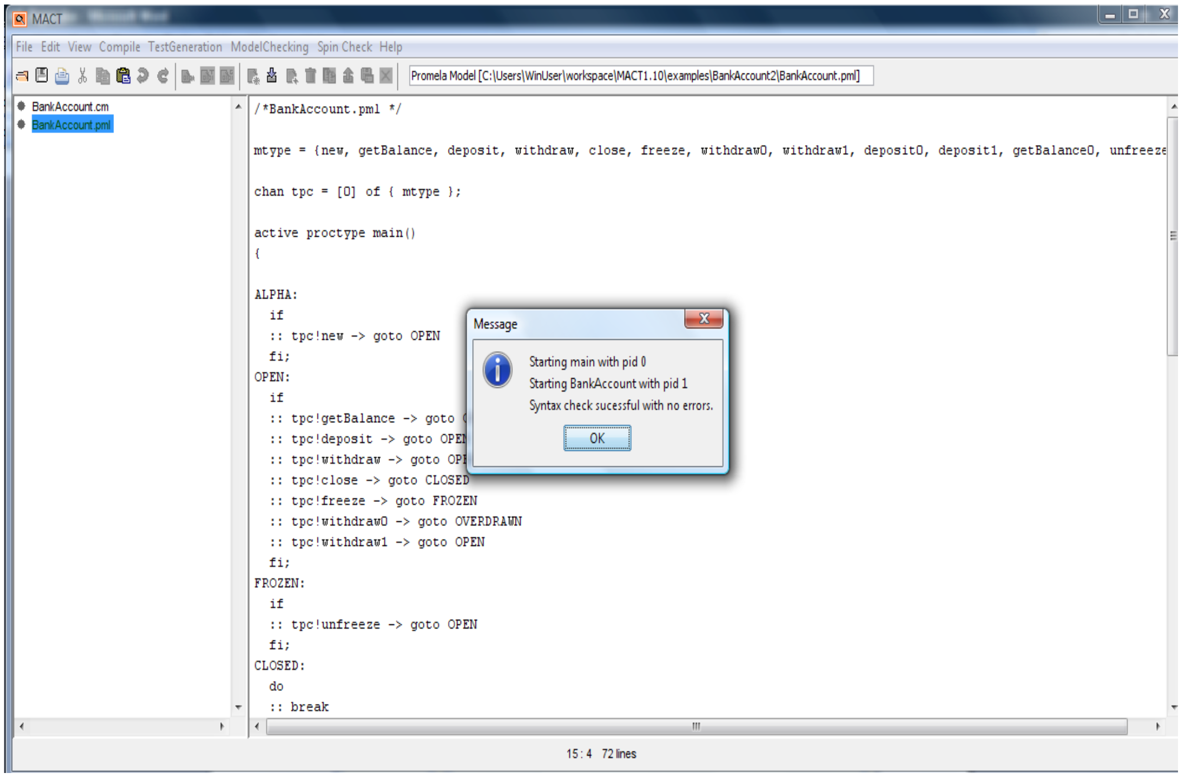
Figure 4 is a screenshot of the syntactically erroneous class model that has the additional state in transitions but is not declared in the list of states. Syntax checking is not done by the JSpin, and hence it will not report any error. The MACT tool has sophisticated syntax checking technology, although a discussion of it is beyond the scope of this paper.


MACT
File Edit View Compile TestGeneration ModelChecking Spin Check Help
Class Model [C:\Users\WinUser\workspace\MACT1.10\examples\BankAccount2\BankAccount.cm]
BankAccount.cm
15 : 15 26 lines

```
Class: BankAccount

Aspect: Overdraft

Events: close, deposit, getBalance, withdraw, freeze, unfreeze

States:
  (OPEN, "getClosed()==false && getBalance() >= 0 && getFrozen()==false")
  (FROZEN, "getFrozen()==true")
  (CLOSED, "getClosed()==true")
  (OVERDRAWN, "getBalance()<0 && getBalance()>=-1000")

Transitions
  (ALPHA, OPEN, new)
  (OPEN, OPEN, error)
  (OPEN, OPEN, deposit ["amt>=0"])
  (OPEN, OPEN, withdraw ["b-amt>=0"])
  (OPEN, CLOSED, close)
  (OPEN, FROZEN, freeze)
  (OPEN, OVERDRAWN, withdraw ["b-amt<0&&b-amt>-1000"])
  (OPEN, OPEN, withdraw ["b-amt<=-1000"])
  (OVERDRAWN, OPEN, deposit ["b+amt>=0"])
  (OVERDRAWN, OVERDRAWN, deposit ["b+amt<0"])
  (OVERDRAWN, OVERDRAWN, getBalance)
  (FROZEN, OPEN, unfreeze)
```

Figure 4. Class Model with Errors.

In this way, all the class models can be converted to their respective PROMELA model files and can be checked for the syntactical correctness of the PROMELA model file. They can also test the PROMELA model file for inconsistencies from a design perspective. The final goal of generating the PROMELA model file is to replicate all the functionalities of the class model to the PROMELA model and test the resultant PROMELA file for design or functional errors. Figure 5 shows the output of the PROMELA model file tested for syntax errors, and Figure 6 shows the output tested for the design inconsistencies, if any.


MACT
File Edit View Compile TestGeneration ModelChecking Spin Check Help
Promela Model [C:\Users\WinUser\workspace\MACT1.10\examples\BankAccount2\BankAccount.pml]
BankAccount.cm
BankAccount.pml
Message
OK
15 : 4 72 lines

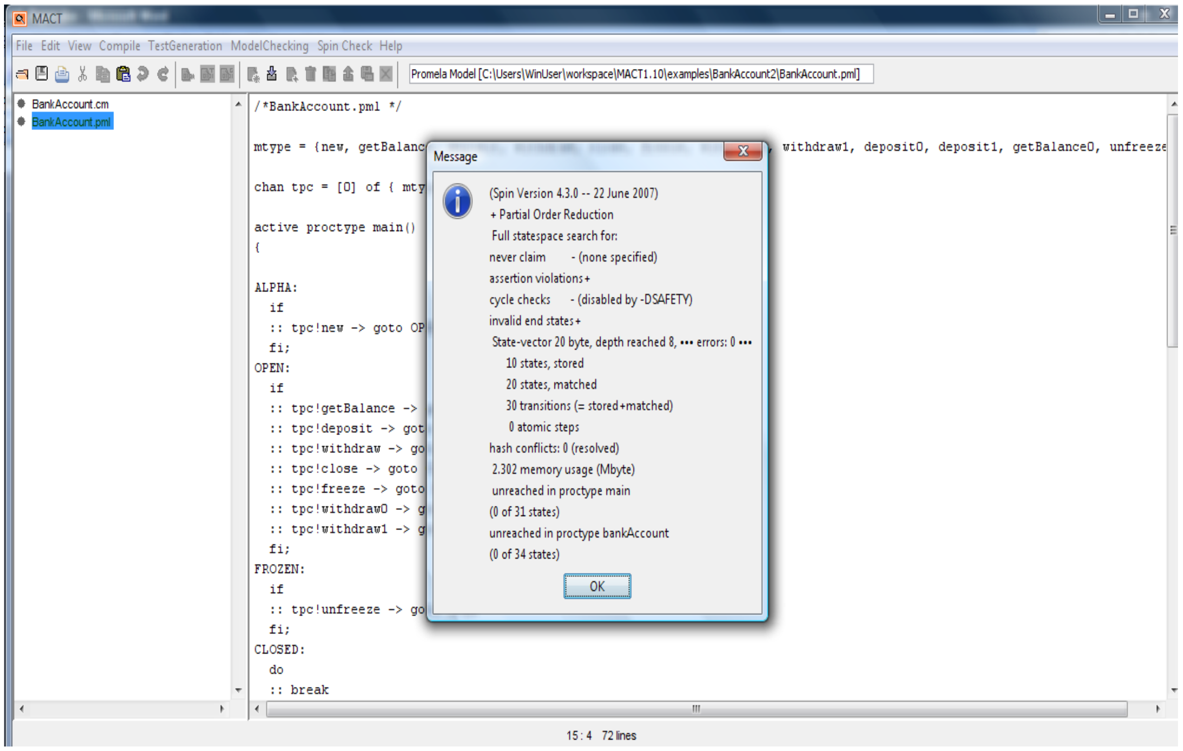
```
/*BankAccount.pml */

mtype = {new, getBalance, deposit, withdraw, close, freeze, withdraw0, withdraw1, deposit0, deposit1, getBalance0, unfreeze

chan tpc = [0] of { mtype };

active proctype main()
{

ALPHA:
  if
  :: tpc!new -> goto OPEN
  fi;
OPEN:
  if
  :: tpc!getBalance -> goto 
  :: tpc!deposit -> goto OPE
  :: tpc!withdraw -> goto OP
  :: tpc!close -> goto CLOSED
  :: tpc!freeze -> goto FROZEN
  :: tpc!withdraw0 -> goto OVERDRAWN
  :: tpc!withdraw1 -> goto OPEN
  fi;
FROZEN:
  if
  :: tpc!unfreeze -> goto OPEN
  fi;
CLOSED:
  do
  :: break
```

```
Starting main with pid 0
Starting BankAccount with pid 1
Syntax check sucessful with no errors.
```

Figure 5. Output of the PROMELA Model File after Testing for Syntax Errors.

The JSpin is the Java version of the SPIN model checker that is developed as a GUI (Graphical User Interface) for Java users. The PROMELA model file can be opened from the JSpin GUI and checked for syntax correctness as well as design correctness of the PROMELA model. There are many additional features, but they are not incorporated into the MACT tool in this paper. The JSpin is completely written in Java language and can be easily modified to embed the JSpin with the MACT tool because the MACT tool is also completely based on Java.

The JSpin also makes system calls to execute a spin.exe file, which is a crucial step for all SPIN-related actions from JSpin. For example, if a PROMELA model file is selected and the “Check for Syntax” option is selected, the syntaxCheckWithSpin() method will be executed from the SpinActionListener class and a system call will be made from the


MACT
File Edit View Compile TestGeneration ModelChecking Spin Check Help
Promela Model [C:\Users\WinUser\workspace\MACT1.10\examples\BankAccount2\BankAccount.pml]
BankAccount.cm
BankAccount.pml
Message
OK
15 : 4 72 lines

```
/*BankAccount.pml */

mtype = {new, getBalanc                                    , withdraw1, deposit0, deposit1, getBalance0, unfreeze

chan tpc = [0] of { mty

active proctype main()
{

ALPHA:
  if
  :: tpc!new -> goto OP
  fi;
OPEN:
  if
  :: tpc!getBalance ->
  :: tpc!deposit -> got
  :: tpc!withdraw -> go
  :: tpc!close -> goto
  :: tpc!freeze -> goto
  :: tpc!withdraw0 -> g
  :: tpc!withdraw1 -> g
  fi;
FROZEN:
  if
  :: tpc!unfreeze -> go
  fi;
CLOSED:
  do
  :: break
```

```
(Spin Version 4.3.0 -- 22 June 2007)
+ Partial Order Reduction
 Full statespace search for:
never claim        - (none specified)
assertion violations +
cycle checks       - (disabled by -DSAFETY)
invalid end states +
 State-vector 20 byte, depth reached 8, ••• errors: 0 •••
    10 states, stored
    20 states, matched
    30 transitions (= stored+matched)
     0 atomic steps
hash conflicts: 0 (resolved)
 2.302 memory usage (Mbyte)
 unreached in proctype main
(0 of 31 states)
unreached in proctype bankAccount
(0 of 34 states)
```

Figure 6. Output of the PROMELA Model File after Testing for Design Errors.

RunSpin object to execute the spin.exe and get whatever data is returned to the JSpin and finally to the MACT tool. This can be achieved by the following code:

```
String[] sa = stringToArray(command, parameters);
ProcessBuilder pb = new ProcessBuilder(sa);
File pf = currentDirectoryAsFile;
if (pf != null) pb.directory(pf.getCanonicalFile());
pb.redirectErrorStream(true);
p = pb.start();
```

Here the ProcessBuilder class is the delivered Java API to execute the system calls, which take the arrays of commands and parameters as input and return the data after the system call. This will be returned later from the JSpin to the MACT tool, and the MACT tool displays the output. The output may be a success message or an error message with details of the error. This is the same method that will be used for syntax checking as well as the SPIN model checking approach. The syntax check will be done at both times of execution, which ensures the syntax correctness at all times.

SPIN Model Checker

SPIN is an acronym for Simple PROMELA Interpreter. SPIN can be used to get a quick impression of the types of behavior that are captured by a system model as it is being built [4]. The aspect-oriented approach increases the reusability of the aspects on different base classes with less complexity and few modifications to the design. This aspect-oriented approach is explained in a paper by Gerard J. Holzmann. SPIN uses the simulation mode to check the syntax correctness for the given PROMELA model file. SPIN can also provide visualized simulation runs to give an overview of the entire PROMELA model.

A SPIN model is used to describe the behavior of systems of potentially interacting processes: multiple, asynchronous threads of execution [4]. The primary unit of execution is a process that will be declared by the “proctype” keyword. To execute a process, the proctype must be instantiated explicitly and, hence, we will use the main() process to instantiate the actual process.

A SPIN model basically contains the mtype declaration or the message type declaration. The variables can be taken from the mtype declaration as the events in the class model. The process types should be active by using the “active” keyword before the “proctype” keyword. The control flow can be altered by using IF statements with expected semantics for the IF conditional statement. The keyword “FI” is the ending token for the IF conditional statement. All the lines within the IF condition are each a condition and an ending state. This helps the PROMELA model file encounter the transitions mentioned in the class model file to be typed into the PROMELA model file. The syntax of how the PROMELA model file should look is explained in the next section.

BNF for PROMELA Model File

The Backus-Naur Form for the PROMELA model file will be explained is this section. This BNF for the PROMELA model file mentioned here is only related to the one created using the class model file as input. Even though there is a way to form a BNF for a more generic PROMELA model file, it is beyond the scope of this paper, and the BNF for the PROMELA model that is covered in this section will be very generic. The BNF for the PROMELA model file considers all the possible combinations of words that also contain the numbers to accept all possible ways of forming the names for the states, events, and the transitions in the class model file.

The BNF for the PROMELA model file also considers all the keywords that cannot be the names of states or events. There are some reserved words for the PROMELA model file that can be used only at specific sections of the PROMELA model files. For example, the keyword “init” cannot be the name of the PROMELA model file since it is used to initialize the process types in the PROMELA language. All the keywords listed below cannot be the names of the states or events. These will be used later by the conversion program to convert the class model file to its respective PROMELA model file.

The BNF grammar for the PROMELA model file will be designed using all the permutations and combinations of the alphabets and numbers but also will consider special cases, like making sure the reserved keywords are not used after the BNF notation formation. A few keywords considered are “init,” “mtype,” “chan,” “proctype,” “if,” “fi,” “do,” and “od.” The complete BNF grammar for the PROMELA model file based on the class model file is given below:

PROMELAMODEL := COMMENTSTART STRING COMMENTEND

MTYPEDECLARATION CHANDECLARATION PROCTYPEMAINDECLARATION ACTUALPROCTYPEDECLARATION EOF

MTYPEDECLARATION := MTYPE CURLYBRACKETSTART NEW (COMMA STRING)+ CURLYBRACKETEND SEMICOLON

CHANDECLARATION := CHAN TPC SQUAREBRACKETSTART ZERO SQUAREBRACKETEND OF CURLYBRACKETSTART MTYPE CURLYBRACKETEND SEMICOLON

PROCTYPEMAINDECLARATION := ACTIVE PROCTYPE MAIN BRACKETSTART BRACKETEND CURLYBRACKETSTART MAINPROCTYPECONTENTS CURLYBRACKETEND

ACTUALPROCTYPEDECLARATION := ACTIVE PROCTYPE STRING BRACKETSTART BRACKETEND CURLYBRACKETSTART ACTUALPROCTYPECONTENTS CURLYBRACKETEND

MAINPROCTYPECONTENTS := ALPHA COLON NONCONDITIONALTRANSITIONDECLARATION (STRING COLON NONCONDITIONALTRANSITIONDECLARATION)+

ACTUALPROCTYPECONTENTS := ALPHA COLON CONDITIONALTRANSITIONDECLARATION (STRING COLON CONDITIONALTRANSITIONDECLARATION)+

NONCONDITIONALTRANSITIONDECLARATION := (STRING COLON NONCONDITIONALTRANSITION) | (STRING COLON IF (NONCONDITIONALTRANSITION)+ FI SEMICOLON)

NONCONDITIONALTRANSITION := DOUBLECOLON TPC EXCLAMATION

STRING IMPLIES GOTO UPPERCASESTRING SEMICOLON

CONDITIONALTRANSITIONDECLARATION := STRING COLON IF (CONDITIONALTRANSITION)+ FI SEMICOLON

CONDITIONALTRANSITION := DOUBLECOLON TPC QUESTIONMARK STRING IMPLIES GOTO UPPERCASESTRING SEMICOLON

UPPERCASESTRING := UPPERCASELETTER (UPPERCASELETTER | NUMBER)* && !INIT && !END && !ALPHA

STRING := LETTER (UPPERCASELETTER | LETTER | NUMBER)*

ALPHA = ‘ALPHA’

BRACKETSTART := ' ('

BRACKETEND := ')'

CHAN := ‘chan’

COMMA := ','

COLON := ':'

COMMENTSTART := ‘/*’

COMMENTEND := ‘*/’

CURLYBRACKETSTART := '{'

CURLYBRACKETSTART := '}'

DOUBLECOLON := ‘::’

EQUAL := ‘=’

END := ‘END’

EXCLAMATION := ‘!’

FI := ‘fi’

GOTO := ‘goto’

IF := ‘if’

IMPLIES := ‘->’

INIT := ‘INIT’

LETTER := 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x' | 'y' | 'z'

MAIN := ‘main’

MTYPE := 'mtype'

NEW := 'new'

NUMBER := '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

OF := ‘of’

QUESTIONMARK := ‘?’

SQUAREBRACKETSTART := '['

SQUAREBRACKETEND := ']'

SEMICOLON := ';'

TPC := ‘tpc’

UPPERCASELETTER := 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H' | 'I' | 'J' | 'K' | 'L' | 'M' | 'N' | 'O' | 'P' | 'Q' | 'R' | 'S' | 'T' | 'U' | 'V' | 'W' | 'X' | 'Y' | 'Z'

Any PROMELA model file should follow the above syntax so that the JSpin will not report any errors. All the states are uppercase and the events can contain a combination of uppercase as well as lowercase letters, but should start with a lowercase character. The STRING here contains the combination of uppercase letters, lowercase letters, and numbers also, as there is a possibility that the events can contain numbers. Similarly, UPPERCASESTRING contains uppercase letters as well as numbers. This finalizes the BNF for the PROMELA model file.

Keyword Handling

There are many keywords that need to be checked in the BNF for the PROMELA model file itself, as the JSpin reports errors if any of them are declared as the names of the states. The BNF for the PROMELA model file above is designed in a very efficient manner to handle these keywords. However, the keywords that are reserved for the PROMELA compiler should not be encountered when the names of states are being sorted and assigned. The PROMELA model file syntax is different from the class model file in this aspect. Even though there are keywords like class, events, states, and transitions, these can also be the name of a class or event or state or transition. For example, in the class model file, “class” is the keyword that the MACT tool expects it to be in the class model file. But there can be an event with the name “class” in the same class model. This is not the case with the PROMELA model file. Keywords like “init” and “end” cannot be the names of the states.

Exception Handling

The PROMELA compiler designed in the JSpin expects all the keywords to be in the PROMELA model file as mentioned in the BNF in the previous section. But the actual values in the events section of the class model will not be checked for consistency in the events in the transition section of the class model. This has one positive impact and one negative impact. The positive impact is it allows the system to handle the events with and without guard conditions. The events with guard conditions are renamed with a unique name that contains the event name and is prefixed by an integer. On the other hand, there may be orphan events in the class model file that may want to be included in it, but the PROMELA model file generator will ignore them and only consider the events within the transitions. There is a limitation for using the keywords in the PROMELA model file. Keywords such as “mtype,” “chan,” and “main()” can only be used once in the PROMELA model file in their respective sections.

CHAPTER 4. DESIGN AND IMPLEMENTATION

This chapter explains the implementation of the conversion program developed using the Java in-built APIs and considering the BNF for the PROMELA model file in the previous section. The explanation consists of all the steps and sub-steps during the implementation of the conversion program for the MACT tool. There are three core features covered in this paper regarding the SPIN model checker: generate the PROMELA model file from the class model file, check the syntax correctness of the PROMELA model file using JSpin, and check the design correctness of the PROMELA model file using JSpin.

The assumptions are that the class model file will have the file extension “CM” and the PROMELA model file will have the file extension “PML.” The file extensions are not case sensitive.

Generating the PROMELA Model File

The control flow of how the class model file is converted into the PROMELA model file is given below in detail:

i. Get the input file from the MACT tool, which is the current active file in the MACT tool.

ii. Check the extension of the input file. If the file extension is not CM, then write code to report an error to the user saying that the file needs to be a class model file or, more precisely, the extension of the input file, should be CM only and exit to the final step.

iii. If the file extension is CM then proceed to the next step, taking the file name and file location.

iv. Now, start line by line processing, ignoring the commented line. The commented lines start with two backslashes. All the remaining lines will be in one of the sections—class section, aspect section, events section, or transition section. The values in these sections will be passed to the next step.

v. Now, the class name is also set as the target PROMELA model file but with the PML file extension. The target location is set to the location where the source input class model file is present.

vi. Using the BNF for the PROMELA model file, start writing the complete states, events, and transitions into JSpin in understandable PROMELA language. As the BNF for the PROMELA model file explains, there will be two process types declared: one with the name main() and one with the class name itself.

vii. All the data in the previous step is passed into an array. That array is passed to this step in which we start writing line by line to the output file, which is the target PROMELA model file.

Coding done in the MACT tool to achieve this:

i. A new action listener is created for the selection of the “Generate PROMELA file” option. It creates an instance of FileToArray class and passes the file name to it. The instantiation of this class will take the object of the MactFrame class so that the current instance of the MACT tool is in sync.

ii. Now, a call to the fileToArray() method, which is in the FileToArray class, is made to initiate the first step of the core conversion process. It reads the data line-by-line for the provided file, which is the current active file in the MACT editor, and sends it into an array of strings.

iii. After that, a call to arrayProcess() is made, which is the second step of the conversion process. This process splits all the data in the class section, events section, and transitions section and sends them into their specific class objects, which are again a combination of ArrayList with string data type.

iv. The last step is called the evaluteStates and evaluateTransitions method to attain the task. This will calculate all the events and update the events list and also evaluate the transitions from the source state to the target state and the events firing the transitions. Everything is evaluated and saved in the ArrayList of strings. This makes it easy to retrieve the values in the future steps of evaluation.

v. Now, an instance of ArrayToFile class is created and the instantiation uses the values of events and transitions to create an ArrayList of strings that is built based on the BNF for the PROMELA model file mentioned before.

vi. The resultant final ArrayList of strings is written to the target PROMELA model file line-by-line.

Steps to generate the PROMELA model file for the given class model:

i. Run the MACT tool by executing MactFrame.java class. The main() method builds the GUI for the MACT tool.

ii. Select the existing class model file, which can be attained in the following way: The user should go to the FILE menu and select the OPEN option from the dropdown menu. Now, locate the class model file for which the user wants to generate the PROMELA model file.

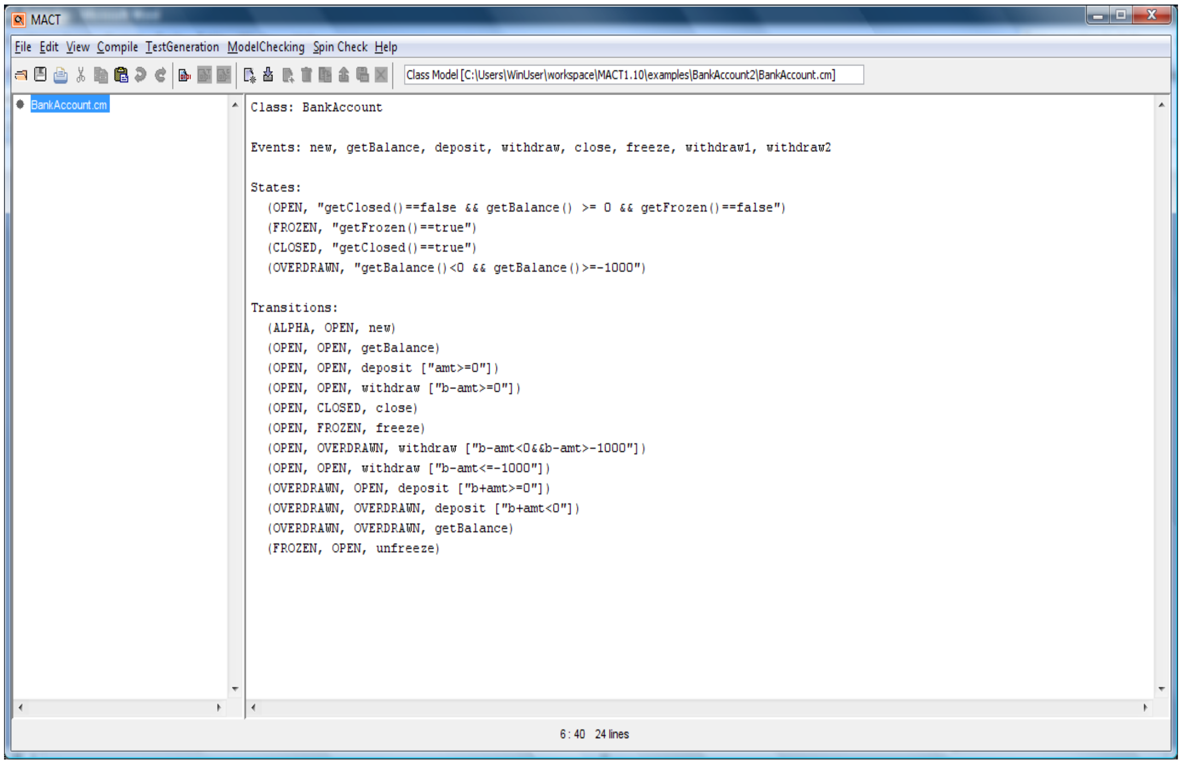
iii. This will open the class model file for which the PROMELA model file needs to be generated into the MACT editor.

iv. Now, go to the SPIN CHECK menu and simply select the GENERATE PROMELA FILE option from the dropdown menu, which is first in the list.

v. This will create the PROMELA model file at the same location where the source class model file is and with the same name but with the PML extension.

vi. It also updates the file list on the left side of the MACT tool with the current PROMELA model file that is generated, and it opens the file in the MACT editor for viewing and editing purposes.

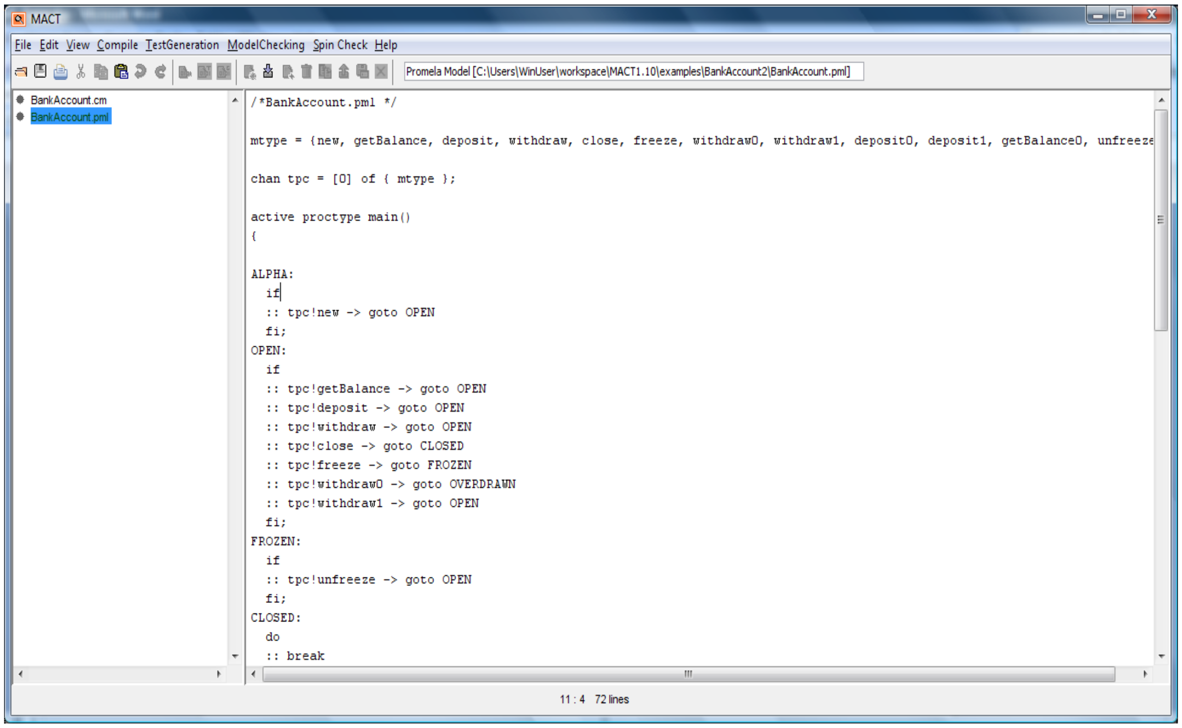
Figure 7 is a screenshot of a sample BankAccount.cm class model file opened in the MACT tool. It contains the class section with the class name as BankAccount. The events section has eight events and the transition section has twelve transitions in which a few contain guard conditions and a few do not contain guard conditions.


MACT
File Edit View Compile TestGeneration ModelChecking Spin Check Help
Class Model [C:\Users\WinUser\workspace\MACT1.10\examples\BankAccount2\BankAccount.cm]
BankAccount.cm
6 : 40 24 lines

```
Class: BankAccount

Events: new, getBalance, deposit, withdraw, close, freeze, withdraw1, withdraw2

States:
  (OPEN, "getClosed()==false && getBalance() >= 0 && getFrozen()==false")
  (FROZEN, "getFrozen()==true")
  (CLOSED, "getClosed()==true")
  (OVERDRAWN, "getBalance()<0 && getBalance()>=-1000")

Transitions:
  (ALPHA, OPEN, new)
  (OPEN, OPEN, getBalance)
  (OPEN, OPEN, deposit ["amt>=0"])
  (OPEN, OPEN, withdraw ["b-amt>=0"])
  (OPEN, CLOSED, close)
  (OPEN, FROZEN, freeze)
  (OPEN, OVERDRAWN, withdraw ["b-amt<0&&b-amt>-1000"])
  (OPEN, OPEN, withdraw ["b-amt<=-1000"])
  (OVERDRAWN, OPEN, deposit ["b+amt>=0"])
  (OVERDRAWN, OVERDRAWN, deposit ["b+amt<0"])
  (OVERDRAWN, OVERDRAWN, getBalance)
  (FROZEN, OPEN, unfreeze)
```

Figure 7. BankAccount Class Model.

Figure 8 is a screenshot of the PROMELA model file generated from the class model file selected in Figure 7. This is the final step of the “Generate PROMELA file” action. The PROMELA model file will be opened in the MACT editor so that it can viewed or edited by the user if needed.


MACT
File Edit View Compile TestGeneration ModelChecking Spin Check Help
Promela Model [C:\Users\WinUser\workspace\MACT1.10\examples\BankAccount2\BankAccount.pml]
BankAccount.cm
BankAccount.pml
11 : 4 72 lines

```
/*BankAccount.pml */

mtype = {new, getBalance, deposit, withdraw, close, freeze, withdraw0, withdraw1, deposit0, deposit1, getBalance0, unfreeze

chan tpc = [0] of { mtype };

active proctype main()
{

ALPHA:
  if
  :: tpc!new -> goto OPEN
  fi;
OPEN:
  if
  :: tpc!getBalance -> goto OPEN
  :: tpc!deposit -> goto OPEN
  :: tpc!withdraw -> goto OPEN
  :: tpc!close -> goto CLOSED
  :: tpc!freeze -> goto FROZEN
  :: tpc!withdraw0 -> goto OVERDRAWN
  :: tpc!withdraw1 -> goto OPEN
  fi;
FROZEN:
  if
  :: tpc!unfreeze -> goto OPEN
  fi;
CLOSED:
  do
  :: break
```

Figure 8. BankAccount PROMELA Model.

In the newly generated PROMELA model file, we can see that there are a few new events added to the events or the mtype section of the file. The new events added are withdraw0, withdraw1, deposit0, deposit1, and getBalance0. The reason for the inclusion of these new events is the possible guard condition attached to those specific parent events. The events that have the guard conditions attached to them are withdraw, deposit, and getBalance, and in some cases there are more than one guard condition to be considered, so the incremental number is suffixed to the actual parent events like deposit0, deposit1, and so on.

The uppercase words in the PROMELA file followed by a colon are generally the states, and the IF conditional allows the control flow to follow the transitions that are mentioned in the class model file. As we can see, the keywords in the class model file and the PROMELA model file are completely different, but the functional flow are one and the same. It is not mandatory that the states be in uppercase and events should be of initial lowercase letters, but it may provide better understanding and can be easily edited by the user if needed.

Checking the PROLEMA File for Syntax

The next step that the user will do is to test the syntax of the current PROMELA model file. Here are the steps to test the PROMELA model file:

vii. Run the MACT tool by executing MactFrame.java class. The main() method builds the GUI for the MACT tool.

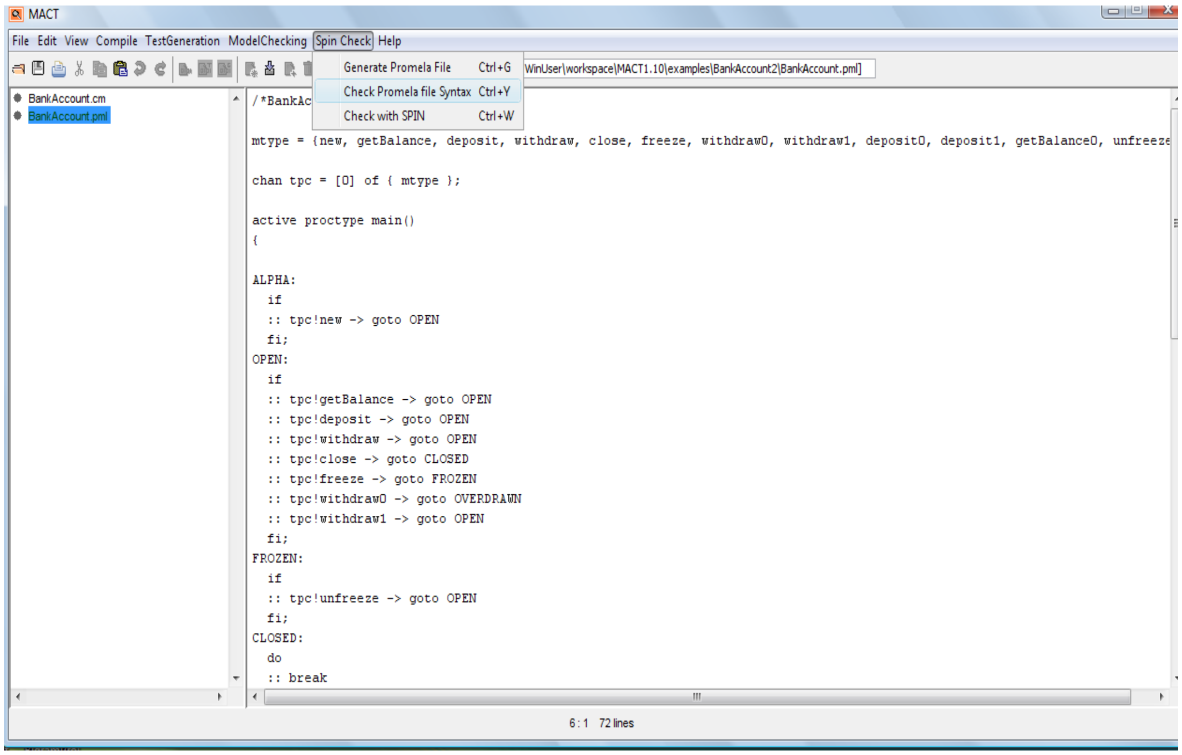
viii. Select the existing PROMELA model file in the following way: The user should go to the FILE menu and select the OPEN option from the dropdown menu. Now, locate the PROMELA model file.

ix. This will open the PROMELA model file that needs to check the syntax in the MACT editor.

x. Now, go to the SPIN CHECK menu and simply select the CHECK PROMELA FILE SYNTAX option from the dropdown menu, which is first in the list.

xi. This will show the pop-up message of the result from JSpin. It may display one of the two messages below. It may say starting two processes and in the last line say “Syntax check successful with no errors.” It may also say that a few errors were found and explain the exact expected keyword.

Figure 9 through Figure 12 are screenshots of how the PROMELA model file is successfully tested without any errors and also how the PROMELA model file with errors is tested, as well as how the errors are displayed so that the user can manually fix the errors and retest the updated PROMELA model file.


MACT
File Edit View Compile TestGeneration ModelChecking Spin Check Help
Generate Promela File Ctrl+G
Check Promela file Syntax Ctrl+Y
Check with SPIN Ctrl+W
WinUser\workspace\MACT1.10\examples\BankAccount2\BankAccount.pml]
BankAccount.cm
BankAccount.pml
6 : 1 72 lines

```
/*BankAc

mtype = {new, getBalance, deposit, withdraw, close, freeze, withdraw0, withdraw1, deposit0, deposit1, getBalance0, unfreeze

chan tpc = [0] of { mtype };

active proctype main()
{

ALPHA:
  if
  :: tpc!new -> goto OPEN
  fi;
OPEN:
  if
  :: tpc!getBalance -> goto OPEN
  :: tpc!deposit -> goto OPEN
  :: tpc!withdraw -> goto OPEN
  :: tpc!close -> goto CLOSED
  :: tpc!freeze -> goto FROZEN
  :: tpc!withdraw0 -> goto OVERDRAWN
  :: tpc!withdraw1 -> goto OPEN
  fi;
FROZEN:
  if
  :: tpc!unfreeze -> goto OPEN
  fi;
CLOSED:
  do
  :: break
```

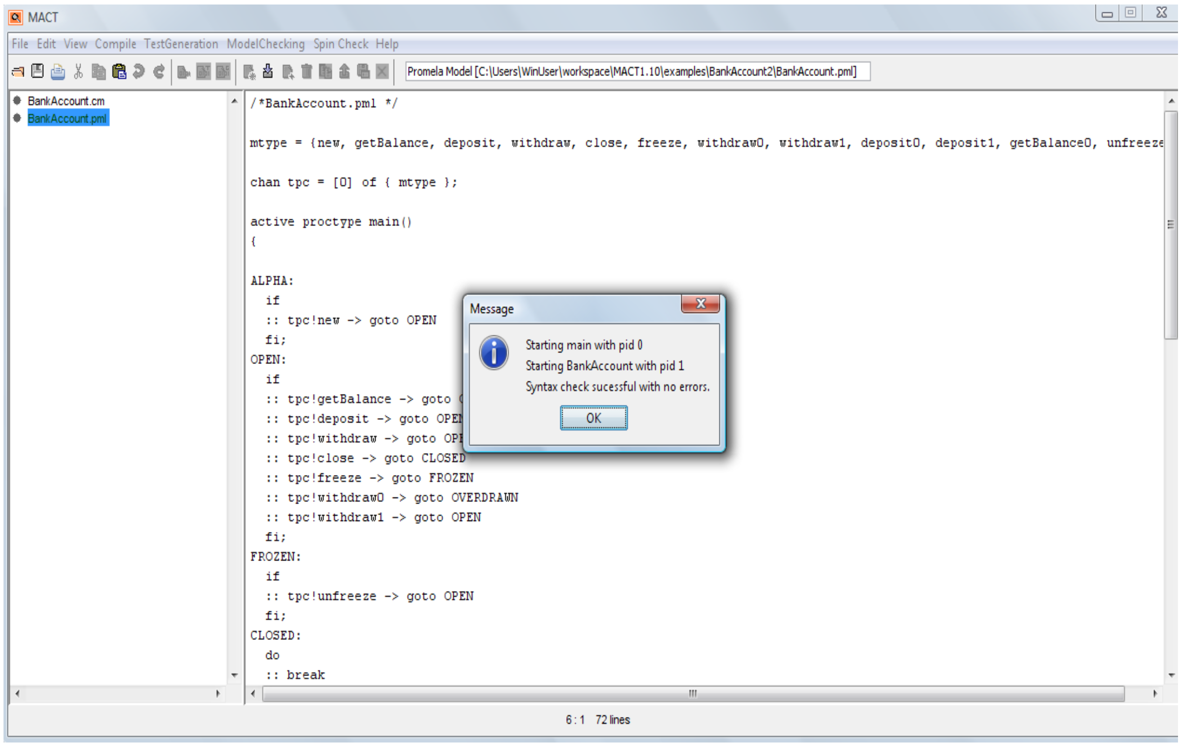
Figure 9. Sample PROMELA Model File Opened.

The error message contains a very detailed message saying which line is causing errors and what keyword will be accepted.

Checking the PROMELA File for Design

The next step that the user will do is to test the actual design correctness of the current PROMELA model file. Here are the steps to test the PROMELA model file:

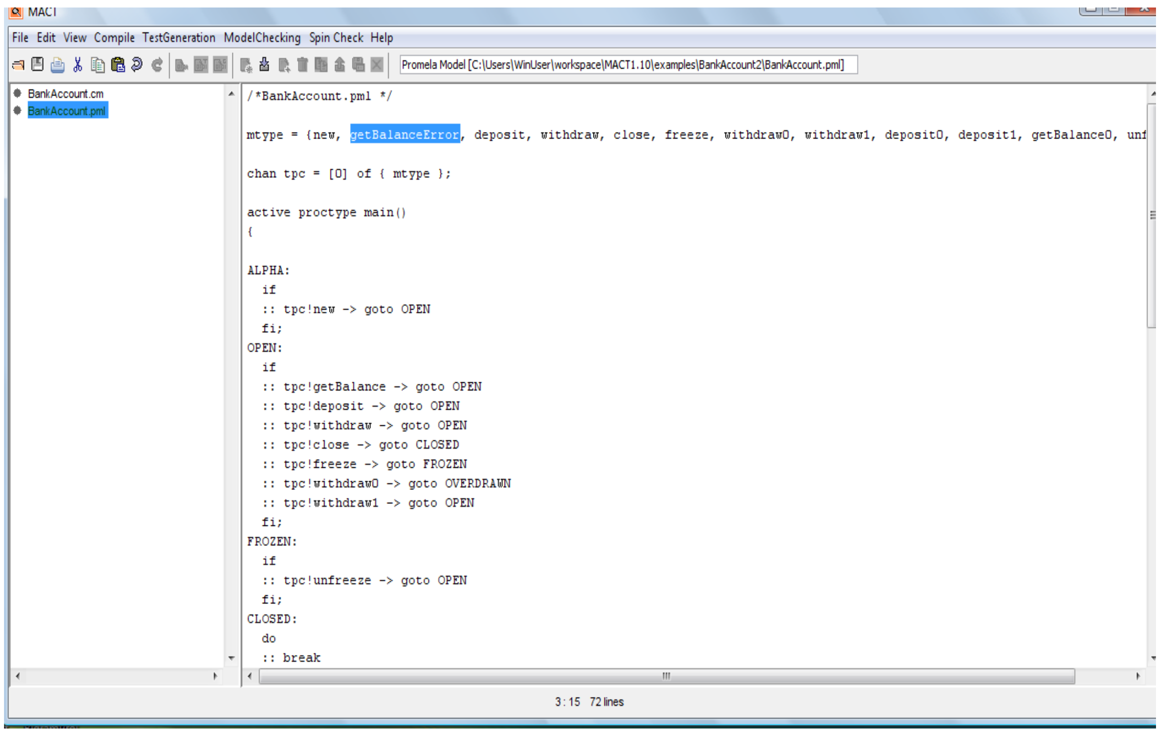
xii. Run the MACT tool by executing MactFrame.java class. The main() method builds the GUI for the MACT tool.


MACT
File Edit View Compile TestGeneration ModelChecking Spin Check Help
Promela Model [C:\Users\WinUser\workspace\MACT1.10\examples\BankAccount2\BankAccount.pml]
BankAccount.cm
BankAccount.pml
Message
OK
6 : 1 72 lines

```
/*BankAccount.pml */

mtype = {new, getBalance, deposit, withdraw, close, freeze, withdraw0, withdraw1, deposit0, deposit1, getBalance0, unfreeze

chan tpc = [0] of { mtype };

active proctype main()
{

ALPHA:
  if
  :: tpc!new -> goto OPEN
  fi;
OPEN:
  if
  :: tpc!getBalance -> goto (
  :: tpc!deposit -> goto OPEN
  :: tpc!withdraw -> goto OPI
  :: tpc!close -> goto CLOSED
  :: tpc!freeze -> goto FROZEN
  :: tpc!withdraw0 -> goto OVERDRAWN
  :: tpc!withdraw1 -> goto OPEN
  fi;
FROZEN:
  if
  :: tpc!unfreeze -> goto OPEN
  fi;
CLOSED:
  do
  :: break
```

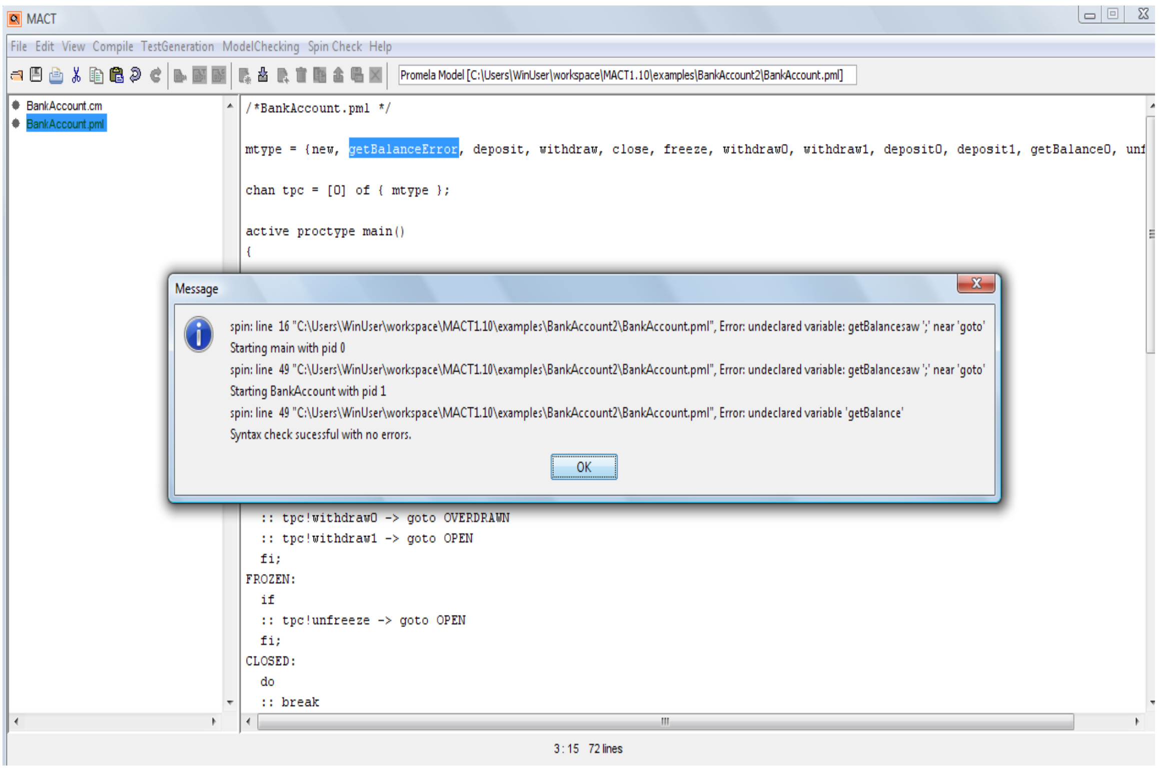
```
Starting main with pid 0
Starting BankAccount with pid 1
Syntax check sucessful with no errors.
```

Figure 10. PROMELA Model File Successfully Checked.


MACT
File Edit View Compile TestGeneration ModelChecking Spin Check Help
Promela Model [C:\Users\WinUser\workspace\MACT1.10\examples\BankAccount2\BankAccount.pml]
BankAccount.cm
BankAccount.pml
3 : 15 72 lines

```
/*BankAccount.pml */

mtype = {new, getBalanceError, deposit, withdraw, close, freeze, withdraw0, withdraw1, deposit0, deposit1, getBalance0, unf

chan tpc = [0] of { mtype };

active proctype main()
{

ALPHA:
  if
  :: tpc!new -> goto OPEN
  fi;
OPEN:
  if
  :: tpc!getBalance -> goto OPEN
  :: tpc!deposit -> goto OPEN
  :: tpc!withdraw -> goto OPEN
  :: tpc!close -> goto CLOSED
  :: tpc!freeze -> goto FROZEN
  :: tpc!withdraw0 -> goto OVERDRAWN
  :: tpc!withdraw1 -> goto OPEN
  fi;
FROZEN:
  if
  :: tpc!unfreeze -> goto OPEN
  fi;
CLOSED:
  do
  :: break
```

Figure 11. Sample Error PROMELA Model File Opened.


MACT
File Edit View Compile TestGeneration ModelChecking Spin Check Help
Promela Model [C:\Users\WinUser\workspace\MACT1.10\examples\BankAccount2\BankAccount.pml]
BankAccount.cm
BankAccount.pml
Message
OK
3 : 15   72 lines

```
/*BankAccount.pml */

mtype = {new, getBalanceError, deposit, withdraw, close, freeze, withdraw0, withdraw1, deposit0, deposit1, getBalance0, unf

chan tpc = [0] of { mtype };

active proctype main()
{
```

```
spin: line  16 "C:\Users\WinUser\workspace\MACT1.10\examples\BankAccount2\BankAccount.pml", Error: undeclared variable: getBalancesaw ';' near 'goto'
Starting main with pid 0
spin: line  49 "C:\Users\WinUser\workspace\MACT1.10\examples\BankAccount2\BankAccount.pml", Error: undeclared variable: getBalancesaw ';' near 'goto'
Starting BankAccount with pid 1
spin: line  49 "C:\Users\WinUser\workspace\MACT1.10\examples\BankAccount2\BankAccount.pml", Error: undeclared variable 'getBalance'
Syntax check sucessful with no errors.
```

```
  :: tpc!withdraw0 -> goto OVERDRAWN
  :: tpc!withdraw1 -> goto OPEN
  fi;
FROZEN:
  if
  :: tpc!unfreeze -> goto OPEN
  fi;
CLOSED:
  do
  :: break
```

Figure 12. PROMELA Model File Check Displaying Errors.

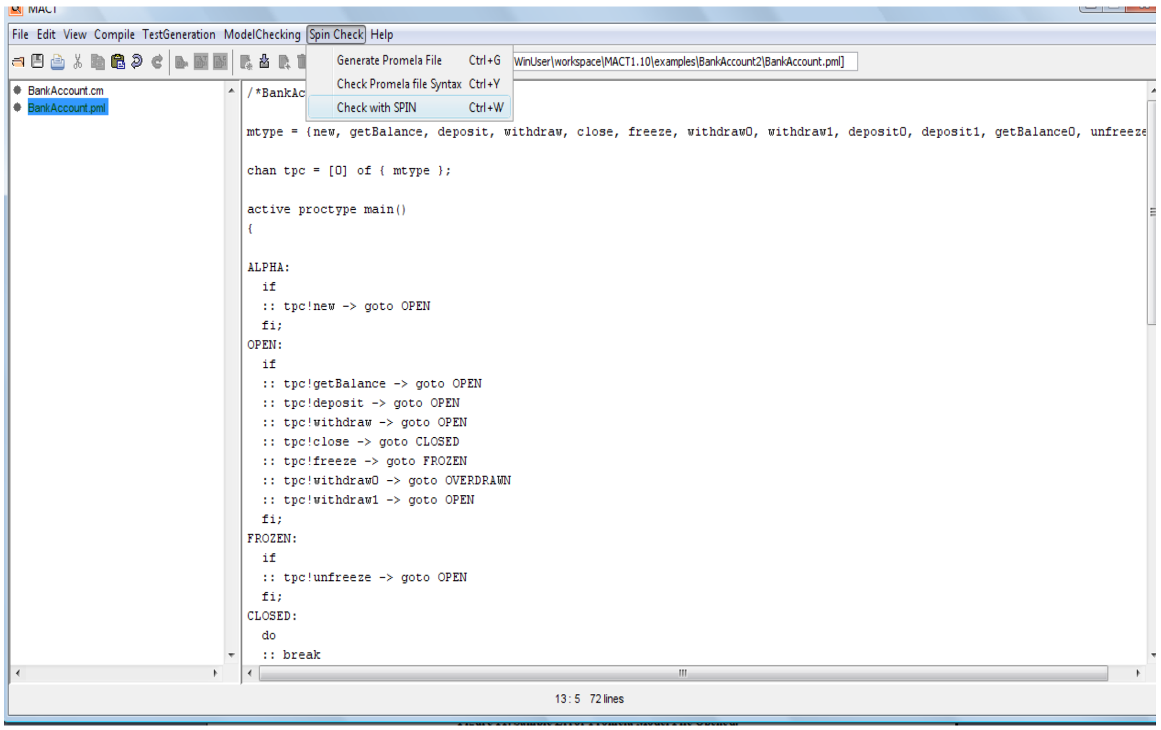
xiii. Select the existing PROMELA model file in one of two ways. The first is to go to the FILE menu and select the OPEN option from the dropdown menu. Now, locate the PROMELA model file.

xiv. This will open the PROMELA model file to check the syntax in the MACT editor.

xv. Now, go to the SPIN CHECK menu and simply select the CHECK WITH SPIN option from the dropdown menu, which is first in the list.

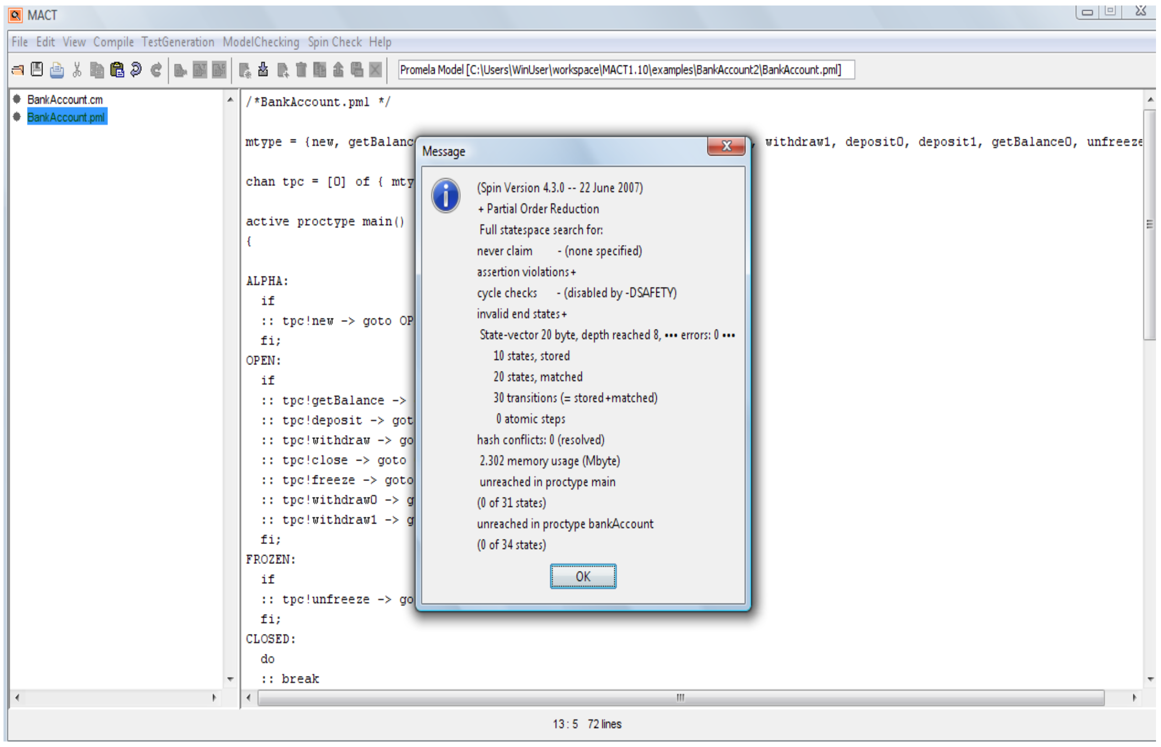
xvi. This will get the pop-up message of the result from JSpin. It may display either the errors, which means that some state is unreachable and includes the line numbers in the PROMELA model file that are not reachable. The SPIN model checker considers that all the lines in the PROMELA model file should be reached at least once. If not, the error will be reported.

Figures 13 and 14 are screenshots of how the PROMELA model file is successfully tested without any errors.


MACT
File Edit View Compile TestGeneration ModelChecking Spin Check Help
Generate Promela File Ctrl+G
Check Promela file Syntax Ctrl+Y
Check with SPIN Ctrl+W
WinUser\workspace\MACT1.10\examples\BankAccount2\BankAccount.pml]
BankAccount.cm
BankAccount.pml
13 : 5 72 lines

```
/*BankAc

mtype = {new, getBalance, deposit, withdraw, close, freeze, withdraw0, withdraw1, deposit0, deposit1, getBalance0, unfreeze

chan tpc = [0] of { mtype };

active proctype main()
{

ALPHA:
  if
  :: tpc!new -> goto OPEN
  fi;
OPEN:
  if
  :: tpc!getBalance -> goto OPEN
  :: tpc!deposit -> goto OPEN
  :: tpc!withdraw -> goto OPEN
  :: tpc!close -> goto CLOSED
  :: tpc!freeze -> goto FROZEN
  :: tpc!withdraw0 -> goto OVERDRAWN
  :: tpc!withdraw1 -> goto OPEN
  fi;
FROZEN:
  if
  :: tpc!unfreeze -> goto OPEN
  fi;
CLOSED:
  do
  :: break
```

Figure 13. Sample PROMELA Model File Opened for Design Check.


MACT
File Edit View Compile TestGeneration ModelChecking Spin Check Help
BankAccount.cm
BankAccount.pml
Message
OK
13 : 5 72 lines

Promela Model [C:\Users\WinUser\workspace\MACT1.10\examples\BankAccount2\BankAccount.pml]

```
/*BankAccount.pml */

mtype = {new, getBalanc                        , withdraw1, deposit0, deposit1, getBalance0, unfreeze

chan tpc = [0] of { mty

active proctype main()
{

ALPHA:
  if
  :: tpc!new -> goto OP
  fi;
OPEN:
  if
  :: tpc!getBalance ->
  :: tpc!deposit -> got
  :: tpc!withdraw -> go
  :: tpc!close -> goto
  :: tpc!freeze -> goto
  :: tpc!withdraw0 -> g
  :: tpc!withdraw1 -> g
  fi;
FROZEN:
  if
  :: tpc!unfreeze -> go
  fi;
CLOSED:
  do
  :: break
```

```
(Spin Version 4.3.0 -- 22 June 2007)
+ Partial Order Reduction
 Full statespace search for:
never claim        - (none specified)
assertion violations+
cycle checks       - (disabled by -DSAFETY)
invalid end states+
 State-vector 20 byte, depth reached 8, ••• errors: 0 •••
    10 states, stored
    20 states, matched
    30 transitions (= stored+matched)
     0 atomic steps
hash conflicts: 0 (resolved)
 2.302 memory usage (Mbyte)
 unreached in proctype main
(0 of 31 states)
unreached in proctype bankAccount
(0 of 34 states)
```

Figure 14. PROMELA Model File Checked Successfully with No Design Errors.

CHAPTER 5. EVALUATION OF THE SYSTEM

This chapter provides three real-time examples of how the PROMELA model file is generated from the source class model file and checked for both syntax errors and design errors in the PROMELA model file.

AirLineReservation Example

This example explains how the AirLineReservation class model file is converted to its respective PROMELA model file and tested in the later steps. Basically, the AirLineReservation is an abstract design model which is incorporated in the software and is used to book air tickets online.

AirLineReservation Class Model Design

There are seven events, including a "new" event, six states that include the initial dummy state "ALPHA," and twelve transitions. Below is a diagram of the control flow of the application. This is an abstract model and will not consider all the transactions done in the real-time airline reservation system.

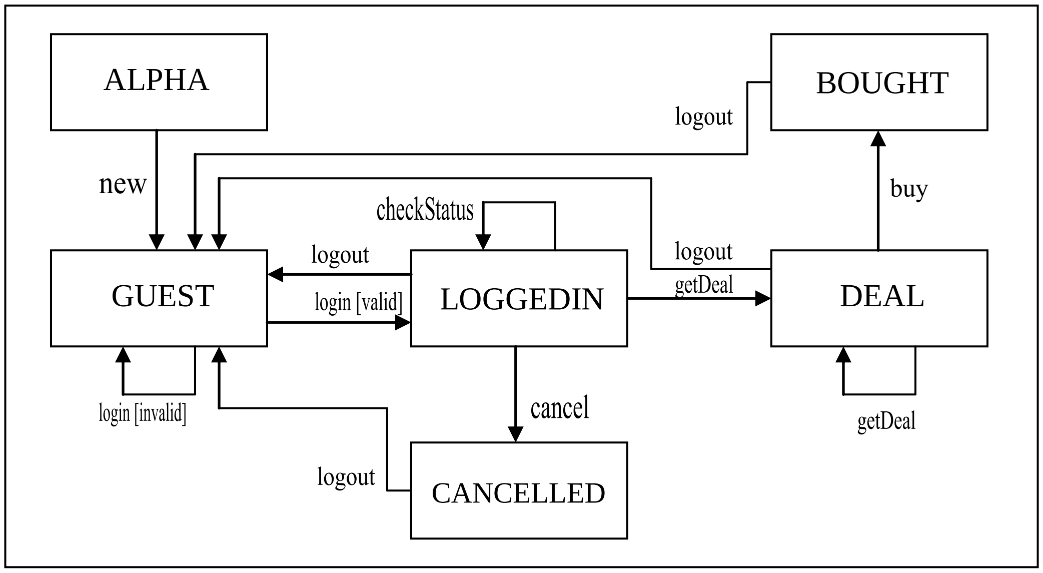


Figure 15. Control Flow of the AirReservation Design.

AirLineReservation Class Model Code

Here is the class model that is extracted from the above control flow diagram:

```
Class: AirLineReservation

Events: new, login, checkStatus, getDeal, buy, cancel, logout

States:

(GUEST,"isActive()==false || isValidUser()==false")

(LOGGEDIN, "isActive()==true && isValidUser()==true")

(DEAL, "isActive()==true && isValidUser()==true && getDealsLength()>0")

(BOUGHT, "isActive()==true && isValidUser()==true && isValidCreditCard()==true")

(CANCELLED, "isActive()==true && isValidUser()==true && remainingDays()>2")

Transitions:

(ALPHA    , GUEST    , new)

(GUEST    , LOGGEDIN , login ["isValidUser()==true && isActive()==true"])

(GUEST    , GUEST    , login ["isValidUser()==false || isActive()==false"])

(LOGGEDIN , LOGGEDIN , checkStatus)

(LOGGEDIN , DEAL     , getDeal)

(LOGGEDIN , CANCELLED, cancel)

(DEAL     , DEAL     , getDeal)

(DEAL     , BOUGHT   , buy)

(LOGGEDIN , GUEST    , logout)

(DEAL     , GUEST    , logout)

(BOUGHT   , GUEST    , logout)

(CANCELLED, GUEST,  logout)
```

AirLineReservation PROMELA Model File

Here is the code that is generated by selecting the “Generate PROMELA file” option in the MACT tool for the AirLineReservation class model file:

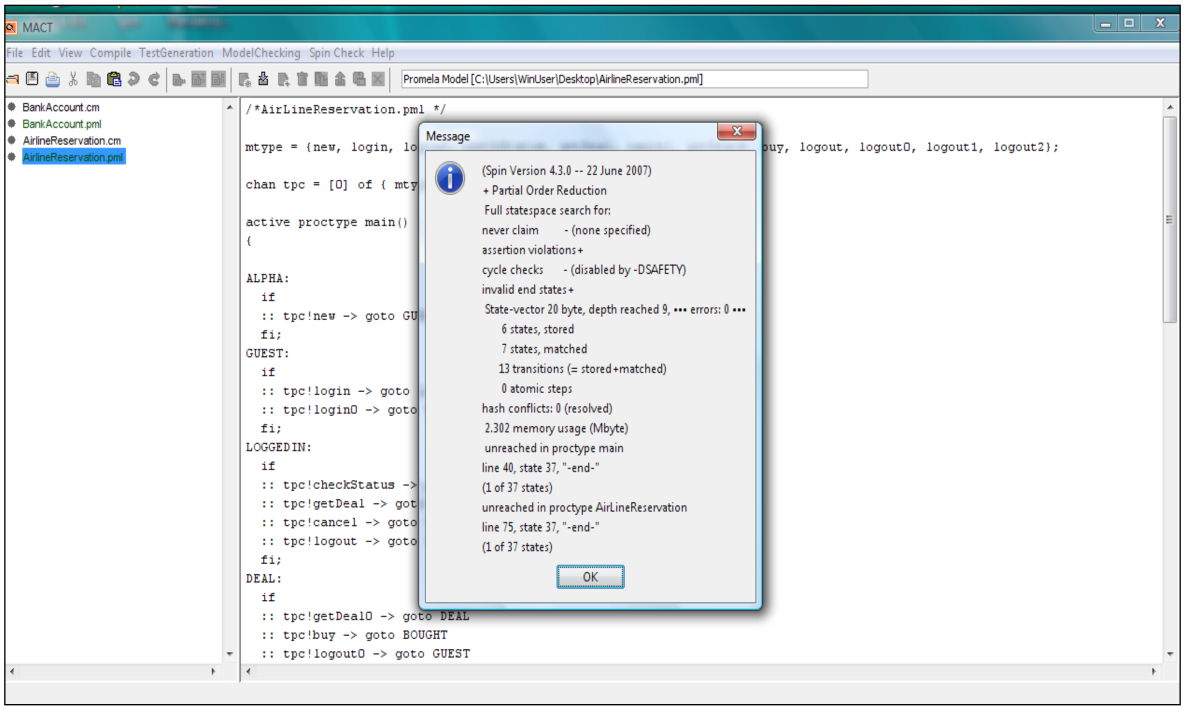
```
/*AirLineReservation.pml */
mtype = {new, login, login0, checkStatus, getDeal, cancel, getDeal0, buy, logout, logout0,
logout1, logout2};


chan tpc = [0] of { mtype };


active proctype main()
{


ALPHA:
	if
	:: tpc!new -> goto GUEST
	fi;
GUEST:
	if
	:: tpc!login -> goto LOGGEDIN
	:: tpc!login0 -> goto GUEST
	fi;
LOGGEDIN:
	if
```

```
        :: tpc!checkStatus -> goto LOGGEDIN
        :: tpc!getDeal -> goto DEAL
        :: tpc!cancel -> goto CANCELLED
        :: tpc!logout -> goto GUEST
        fi;
DEAL:
        if
        :: tpc!getDeal0 -> goto DEAL
        :: tpc!buy -> goto BOUGHT
        :: tpc!logout0 -> goto GUEST
        fi;
BOUGHT:
        if
        :: tpc!logout1 -> goto GUEST
        fi;
CANCELLED:
        if
        :: tpc!logout2 -> goto GUEST
        fi;
}

active proctype AirLineReservation()
{
```

```
ALPHA:
        if
        :: tpc?new -> goto GUEST
        fi;
GUEST:
        if
        :: tpc?login -> goto LOGGEDIN
        :: tpc?login0 -> goto GUEST
        fi;
LOGGEDIN:
        if
        :: tpc?checkStatus -> goto LOGGEDIN
        :: tpc?getDeal -> goto DEAL
        :: tpc?cancel -> goto CANCELLED
        :: tpc?logout -> goto GUEST
        fi;
DEAL:
        if
        :: tpc?getDeal0 -> goto DEAL
        :: tpc?buy -> goto BOUGHT
        :: tpc?logout0 -> goto GUEST
        fi;
```

```
BOUGHT:
        if
        :: tpc?logout1 -> goto GUEST
        fi;
CANCELLED:
        if
        :: tpc?logout2 -> goto GUEST
        fi;
}
```

This can be checked for syntax errors as well as design errors, and the output of the MACT tool is given in Figure 16 below.


MACT
File Edit View Compile TestGeneration ModelChecking Spin Check Help
Promela Model [C:\Users\WinUser\Desktop\AirlineReservation.pml]
BankAccount.cm
BankAccount.pml
AirlineReservation.cm
AirlineReservation.pml
Message
OK

```
/*AirLineReservation.pml */

mtype = {new, login, lo                                    buy, logout, logout0, logout1, logout2};

chan tpc = [0] of { mty

active proctype main()
{

ALPHA:
  if
  :: tpc!new -> goto GU
  fi;
GUEST:
  if
  :: tpc!login -> goto
  :: tpc!login0 -> goto
  fi;
LOGGEDIN:
  if
  :: tpc!checkStatus ->
  :: tpc!getDeal -> got
  :: tpc!cancel -> goto
  :: tpc!logout -> goto
  fi;
DEAL:
  if
  :: tpc!getDeal0 -> goto DEAL
  :: tpc!buy -> goto BOUGHT
  :: tpc!logout0 -> goto GUEST
```

```
(Spin Version 4.3.0 -- 22 June 2007)
+ Partial Order Reduction
 Full statespace search for:
never claim       - (none specified)
assertion violations+
cycle checks      - (disabled by -DSAFETY)
invalid end states+
 State-vector 20 byte, depth reached 9, ••• errors: 0 •••
     6 states, stored
     7 states, matched
    13 transitions (= stored+matched)
     0 atomic steps
hash conflicts: 0 (resolved)
 2.302 memory usage (Mbyte)
 unreached in proctype main
line 40, state 37, "-end-"
(1 of 37 states)
unreached in proctype AirLineReservation
line 75, state 37, "-end-"
(1 of 37 states)
```

Figure 16. AirLineReservation PROMELA Model Tested with no Design Errors.

StudentRegistration Example

This example explains how the StudentRegistration class model file is converted to its respective PROMELA model file and tested in the later steps. Basically, the StudentRegistration is an abstract design model that is incorporated in the software that is used to book the air tickets online.

StudentRegistration Class Model Design

There are seven events, including a "new" event, seven states that include the initial dummy state "ALPHA," and thirteen transitions. Below is a diagram of the control flow of the application. This is an abstract model and will not consider all the transactions done in the real-time Student Registration.

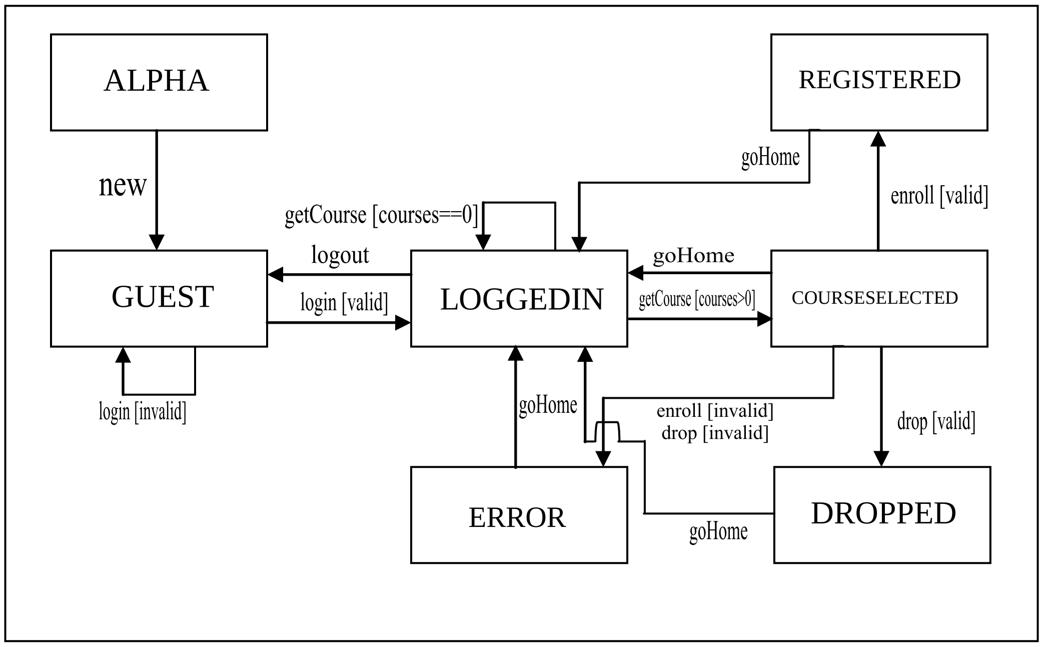


Figure 17. Control Flow of the StudentRegistration Design.

StudentRegistration Class Model Code

Here is the class model that is extracted from the above control flow diagram:

Class: StudentRegistration

Events: new, login, getCourse, enroll, drop, goHome, logout

States:

(GUEST,"isActive()==false || isValidUser()==false")

(LOGGEDIN, "isActive()==true && isValidUser()==true")

(COURSESELECTED, "haveValidCourses==true")

(REGISTERED, "validToRegister()==true")

(DROPPED, "currentlyEnrolled()==true")

(ERROR, "getErrorMessage!=null")

Transitions:

(ALPHA, GUEST, new)

(GUEST, LOGGEDIN, login ["isValidUser()==true"])

(GUEST, GUEST, login ["isValidUser()==false"])

(LOGGEDIN, COURSESELECTED, getCourse ["courseReturned()>0"])

(LOGGEDIN, LOGGEDIN, getCourse ["courseReturned()==0"])

(COURSESELECTED, REGISTERED, enroll ["validAction()==true"])

(COURSESELECTED, ERROR, enroll ["validAction()==false"])

(COURSESELECTED, DROP, drop ["validAction()==true"])

(COURSESELECTED, ERROR, drop ["validAction()==false"])

(COURSESELECTED, LOGGEDIN, goHome)

(ERROR, LOGGEDIN, goHome)

(REGISTERED, LOGGEDIN, goHome)

(DROPPED, LOGGEDIN, goHome)

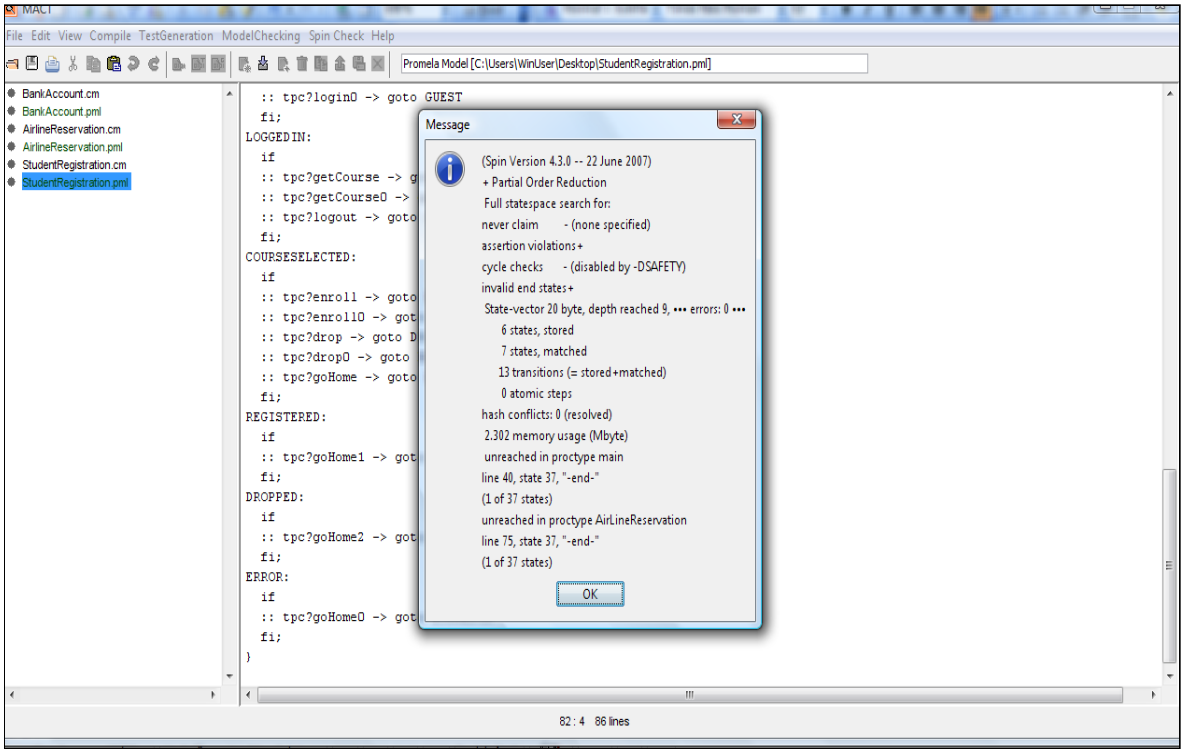
(LOGGEDIN, GUEST, logout)

StudentRegistration PROMELA Model File

Here is the code that is generated by selecting the “Generate PROMELA file” option in the MACT tool for the StudentRegistration class model file:

```
/*StudentRegistration.pml */


mtype = {new, login, login0, getCourse, getCourse0, enroll, enroll0, drop, drop0, goHome,
logout};


chan tpc = [0] of { mtype };


active proctype main()
{


ALPHA:
        if
        :: tpc!new -> goto GUEST
        fi;
GUEST:
        if
        :: tpc!login -> goto LOGGEDIN
        :: tpc!login0 -> goto GUEST
        fi;
LOGGEDIN:
        if
        :: tpc!getCourse -> goto COURSESELECTED
        :: tpc!getCourse0 -> goto LOGGEDIN
        :: tpc!logout -> goto GUEST
        fi;
COURSESELECTED:
        if
        :: tpc!enroll -> goto REGISTERED
        :: tpc!enroll0 -> goto ERROR
        :: tpc!drop -> goto DROP
        :: tpc!drop0 -> goto ERROR
        :: tpc!goHome -> goto LOGGEDIN
        fi;
REGISTERED:
        if
        :: tpc!goHome -> goto LOGGEDIN
        fi;
DROPPED:
        if
        :: tpc!goHome -> goto LOGGEDIN
        fi;
ERROR:
        if
        :: tpc!goHome -> goto LOGGEDIN
        fi;
}

active proctype StudentRegistration()
{

ALPHA:
        if
        :: tpc?new -> goto GUEST
        fi;
GUEST:
        if
        :: tpc?login -> goto LOGGEDIN
        :: tpc?login0 -> goto GUEST
        fi;
LOGGEDIN:
        if
        :: tpc?getCourse -> goto COURSESELECTED
        :: tpc?getCourse0 -> goto LOGGEDIN
        :: tpc?logout -> goto GUEST
        fi;
COURSESELECTED:
        if
        :: tpc?enroll -> goto REGISTERED
        :: tpc?enroll0 -> goto ERROR
        :: tpc?drop -> goto DROP
        :: tpc?drop0 -> goto ERROR
        :: tpc?goHome -> goto LOGGEDIN
        fi;
REGISTERED:
        if
        :: tpc?goHome -> goto LOGGEDIN
        fi;
DROPPED:
        if
        :: tpc?goHome -> goto LOGGEDIN
        fi;
ERROR:
        if
        :: tpc?goHome -> goto LOGGEDIN
        fi;
}
```

This can be checked for syntax errors as well as design errors, and the output of the MACT tool is shown in Figure 18 below.



```
  :: tpc?login0 -> goto GUEST
  fi;
LOGGEDIN:
  if
  :: tpc?getCourse -> g
  :: tpc?getCourse0 ->
  :: tpc?logout -> goto
  fi;
COURSESELECTED:
  if
  :: tpc?enroll -> goto
  :: tpc?enroll0 -> got
  :: tpc?drop -> goto D
  :: tpc?drop0 -> goto
  :: tpc?goHome -> goto
  fi;
REGISTERED:
  if
  :: tpc?goHome1 -> got
  fi;
DROPPED:
  if
  :: tpc?goHome2 -> got
  fi;
ERROR:
  if
  :: tpc?goHome0 -> got
  fi;
}
```

```
(Spin Version 4.3.0 -- 22 June 2007)
+ Partial Order Reduction
 Full statespace search for:
never claim         - (none specified)
assertion violations+
cycle checks        - (disabled by -DSAFETY)
invalid end states+
 State-vector 20 byte, depth reached 9, ••• errors: 0 •••
      6 states, stored
      7 states, matched
     13 transitions (= stored+matched)
      0 atomic steps
hash conflicts: 0 (resolved)
 2.302 memory usage (Mbyte)
 unreached in proctype main
line 40, state 37, "-end-"
(1 of 37 states)
unreached in proctype AirLineReservation
line 75, state 37, "-end-"
(1 of 37 states)
```

Figure 18. StudentRegistration PROMELA Model Tested with no Design Errors.

CHAPTER 6. CONCLUSIONS, LIMITATIONS, AND FUTURE WORK

The main goal of this paper is to incorporate the SPIN model checker for the MACT tool by generating the SPIN model checker to an understandable PROMELA model. The SPIN model checker has its own programming language that it can understand, and this paper successfully implements a way to convert the specified class model to a PROMELA file. There are various advantages of the approach to use JSpin and the SPIN model checker to generate the automated PROMELA model file for any given class model and test the design-level correctness in the MACT tool. A robust algorithm is designed and incorporated into the MACT tool so that it can convert any class model into a PROMELA model file based on the states in the class model. This paper also replicates the functionality of the JSpin (Java version of the SPIN model checker) to check the syntax errors as well as the design errors. The design of the algorithm is intended to be as flexible as possible.

Limitations

State Model

This paper only discusses a way to convert the class model to a PROMELA model file even though there is a woven model file that the MACT tool can read. The woven model is a combination of the class model and the aspect model woven together using a unique algorithm. Even though the woven model is beyond the scope of this paper, it is definitely a limitation of the current system and can be modified to incorporate this functionality in the MACT tool so that even the woven model can be tested using the SPIN model checker.

Future Work

This paper described the full implementation steps to add the functionality of the SPIN model checker for the MACT tool by generating the SPIN model checker to an understandable PROMELA model. However, the design is limited to the class model alone. The algorithm designed in this paper can be easily modified to add the SPIN model checker for additional model files, like the woven model, and can be made more generic. Also, the input format accepted by the code developed is only the text file, but it can be easily extended to different formats like XML files and other model file formats accepted by the MACT tool.

Depth Acceptance

There is functionality available in JSpin in which we can set the number of levels or the depth of the transitions that need to be checked. But it is not considered in the MACT tool assuming the depth can never reach more than 250, which is the default setting for JSpin and will be replicated in the MACT tool. This can be difficult when an application with high complexity is supposed to be tested. The SPIN model checker will stop testing the transitions beyond 250 steps and cannot be used for a few applications that have a high level of complexity. This is being tested by the MACT tool using the JSpin and the algorithm designed in this paper.

Spin Spider

JSpin has a functionality of generating graphical representation of the transitions in the PROMELA model file. This helps the user to actually see the picture of the tree and check whether all the transitions are valid or not. However, this functionality is not implemented in the MACT tool, which can be crucial in many cases. For example, in the implementation section of this paper, the control flow of the BankAccount class model is given, and later the actual BankAccount class model and from it the BankAccount PROMELA model file is designed. This JSpin has the functionality of creating a picture similar to the control flow diagram mentioned for the BankAccount design, which can be handy to check both the control flow diagram and the actual picture generated by the Spin Spider. This should be considered future work.

State Model

As mentioned in the Limitations section, the class model is the only model file that is accepted to generate the PROMELA model file. It is possible to modify the code to generate a PROMELA model file from other state model files as well. This can be an added functionality to the MACT tool to test both the class model files as well as the woven model files. This should be considered future work.

REFERENCES

[1] "Basic Spin Manual," 3 June 2007. [Online]. Available: http://spinroot.com/spin/Man/Manual.html. [Accessed 1 May 2009].

[2] "JavaCC Eclipse Plug-in and Headless Plug-in," GPL / Cecill, [Online]. Available: http://eclipse-javacc.sourceforge.net/. [Accessed 1 May 2009].

[3] D. Grune and C. J. Jacobs, Parsing Techniques: A Practical Guide, Amsterdam: Springer Science Business Media LLC, 2007.

[4] G. J. Holzmann, "IEEE TRANSACTIONS ON SOFTWARE ENGINEERING," *The Spin Model Checker*, vol. 23, no. 5, pp. 1 -17, 1997.

[5] D. Xu and X. Weifeng, "A Model-Based Approach to Test Generation for Aspect-Oriented Programs," Aosd'05 workshop on testing aspect-oriented programs, Chicago, 2005.

[6] B.-A. Mordechai, Principles of the Spin Model Checker, Springer, 2008.