A Case Study of Model Checking Retail Banking System with SPIN

Huiling Shi
Shandong Provincial Key Laboratory of Computer Network
Shandong Computer Science Center, Jinan 250014, P. R. China
Email:shihl@keylab.net

Wenke Ma, Meihong Yang and Xinchang Zhang
Shandong Provincial Key Laboratory of Computer Network
Shandong Computer Science Center, Jinan 250014, P. R. China

***Abstract*—Model checking is an important technique for ensuring the correctness of investigated 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 paper presents a case study of model checking the business flow of retail banking System, through an example of verifying automatic teller machine (ATM) with SPIN. We present the specific approach to effectively abstract the related part of ATM system, and give our experiment results. The verification results show that model checking is feasible technique for verifying the ATM system.**

***Index Terms*—model checking, spin, verification, automatic teller machine, retail banking**

I. INTRODUCTION

Retail bank is an important bank business, which offers a range of services to individual customers and small businesses, rather than to large companies and other banks. The services usually include current accounts, savings accounts, investment advice and broking, and loans and mortgages. Retail banks must enable customers to securely and reliably conduct transactions. Traditionally, retail banks have provided these services directly to the customer via branches. At present, retail banks also offer their services by telephone, the internet and automatic teller machine (ATM) as well. In particular, some operate solely via the internet and do not have facilities to serve customers at physical outlets. Additionally, some other organizations, such as supermarkets, have now entered the banking sector and also offer a wide range of banking services. Clearly, it is very necessary to ensure the correctness of services of retail banking system.

Testing is an indispensable step to try 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 [1-10] which report the successful instances of using model checking to provide the validate system verifications.

SPIN(Simple Promela Interpreter) [5] [6] [7] 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) [8] [9] [10], and it accepts correctness claims specified in the syntax of standard Linear Temporal Logic (LTL) [11]. 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.

Model checking seems to be a promising approach for ensuring the correctness of retail 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 paper presents a case study of model checking the business flow of retail banking system, through an example of verifying automatic teller machine (ATM) with SPIN. In the case study, we will present the specific approach to effectively abstract the related part of ATM system, and give our experiment results.

The rest of the paper is organized as follows. We introduce the related work in Section II. Section III introduces the model checking tool SPIN and extended finite state machine (EFSM), which is used to modeling the ATM system. In Section IV, we present the detail approach for modeling the ATM system. The experiment

Manuscript received July 30, 2011; revised September 5, 2011; accepted December 19, 2011.
Project number: 61070039, and BS2011DX033
Contact author: huiling shi.

doi:10.4304/jcp.7.10.2503-2510

results are given in Section V. Finally, we summarize the paper in Section VI.

II. RELATED WORK

Model checking is an important method for formally verifying state transition systems because it allows the fully automatic analysis of investigated system. The application of model checking consists of several tasks. The first step is modeling investigated system, i.e., the system to be verified should be converted in a formalism accepted by a model checking tool. In the second step it is necessary to state the properties that the system must satisfy. The specifications are expressed in a logical formalism. It is common to use a temporal logic that can describe how the system evolves over time. The third step is the verification. Often it involves some human assistance for example to perform the analysis of verification results. In case of a negative result, the user is provided with an error trace. This can be assumed as a counterexample for the checked property and means that the system does not verify the property. Hence the system design can be modified and checked again.

So far the model checking has been largely implied, a complete state of the art can be found in [1-13].

In [3], The Dynamic Host Configuration Protocol (DHCP) is studied according to the concept of modeling and verification. In [11], 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 [12], 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 [13], the paper gives an approach to verify the object model of rCOS using model checker Spin. And a case study presents to show how the approach works.

The troublesome problem of using model checking technique is the state-space explosion. There are several approaches to combat this problem, which can be classified into two categories, i.e. simplifying the system model by higher abstraction (e.g. [9]), and reducing the resource consumption in the process of model checking (e.g. [14])

III. PRELIMINARIES

A. SPIN

Model checking is used, for example, for the verification of protocols and hardware circuits [1] [2]. Many tools, called model checkers, have been developed to this aim. The most famous one is SPIN [7].Trying all possible interleaving to see which ones can lead to failure would be astoundingly complex. To avoid this SPIN uses a theory of partial order reduction [14] to group process executions into equivalence classes.

To check the compliance of a system with a logic system property specified in linear temporal logic, SPIN first converts the formula into a test automaton that works much like an observer or monitor of the system executions. While building the system executions, the monitor is consulted at every step to see if violations occurred. If a violation is detected, SPIN displays the exact interleaving sequence leading from the initial system state to the state where the violation was detected. This serves as a counter-example to the correctness claims and facilitates diagnosis of the detected violation.

SPIN accepts design specifications written in the verification language PROMELA, and it accepts correctness claims specified in Linear Temporal Logic (LTL).

PROMELA is a language for building verification models that represent an abstract of a system, which contains only those aspects that are relevant to the properties one wants to verify [9]. A PROMELA program consists of processes, message channels, and variables. Processes are defined globally; while message channels and variables can be declared either globally or locally within a process. Processes are used to specify system behaviors, and channels and global variables are used to define the environment in which the processes run. Examples and further details about the PROMELA language can be found in references [8][9].

LTL is a modal logic aimed at encoding how states evolve over time. It has been proven to have good expressivity and more natural language like statements for verification. LTL has three unary modal operators (X,F,and G) and three binary modal operators (U,R,W)[11]. A formula Xφ is true in particular state if and only if the formula φ is true in the next state; Gφ is true if and only φ is true from now on; Fφ is true if φ is or will be true at some time in the future; φUψ is true if ψ will eventually become true and φ stays true until then.

B. EFSM Model

In a finite state machine (FSM), the transition is associated with a set of input boolean conditions and a set of output boolean functions. Different from FSM, the transition of EFSM can be expressed by an "if statement" consisting of a set of trigger conditions. If trigger conditions are all satisfied, the transition is fired, bringing the machine from the current state to the next state and performing the specified data operations.

Formally, EFSM $\boldsymbol{M}$ is defined as the tuple $(S,s_0,V,M_V,P,M_P,I,O,T)$, in which:

- S is a finite set of states, s_0 is the initial state, $s_0 \in S$;
- V is the finite set of the internal variable (environment variable), and the range of the internal variable is DV;
- M_V is the set of the initial (or default) value of variables in V, in which any element can be expressed as a tuple (s, v), $s \in S$, $v \in D_V$;
- P is the input and output parameters;
- M_P is the set of the initial (or default) value of variables in P, in which any element can be expressed as a tuple (p, u), $p \in I \cup O$, $u \in Dp$, Dp is the range of the input and output parameters;

- I is a set of the input symbols;
- O is a set of the output symbols;
- T is a finite set of state transition.

A state transition $t(t \in T)$ is defined as the tuple (s,x,y,g_P,g_E,op,e), where:

- s and e are the start (head) state and the end (tail) state;
- g_P is the input and output conditions to determine;
- g_E is the conditions to determine of the variable required for migration;
- x and y are the input and output symbols;
- op is output operations.

IV. MODELING THE BUSINESS FLOW OF ATM

ATMs are the most immediately visible type of retail banking technology. The main operations of ATMs include balance and transaction enquiries, withdrawals, deposits and accounts transfer. In this section, we will present the model (including EFSM and Promela models) of the main business flow of the ATMs. The model has been simplified in order to obtain a minor number of states to manage in the formal verification. Particularly, encryption has been omitted.

A. EFSM Model

ATM can be used to login with a card and a pin, perform transactions against the account (deposit, withdraw, inquire balance), and logoff after desired transactions. A user gets 3 chances to login with a valid pin, after which the card is locked until reset by an official. There are usual restrictions on amounts that can be withdrawn and the number of withdrawals (6) that can be made.

ATM in the actual course of three parties are involved: the cardholder, the terminal, the bank sever. Cardholder interacts with the banking system through a terminal, for withdrawals, transfers, inquiries. Terminal receives a request from the cardholder to handle all the business logic from the terminal, and forwards the request to bank server, while waiting for the bank server's response, and forwards 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.

In order to use model checker SPIN, we have to describe the specifications of ATM in PROMELA language. But before describing the specifications in PROMELA, we can model the specification in Extended Finite State Machine (EFSM) that can be readily expressed in PROMELA. Fig. 1 shows specifications for ATM expressed in EFSM. Variable CA is said that the number of unsuccessful logon attempts, if CA is greater than or equal 3, then the card is locked. Variable CW is said that the number of successful withdrawals, if CW is greater than 6, then emerges an error.

Fig. 1 shows the EFSM model of Automated Teller Machine, containing 4 states and 13 state transitions. The label of transition "PIN/rOk[CA<=2,CW<6],Logon" means that, when in the implementation of "Logon" operation and the input symbol of the state PIN satisfying " CA<=2"and"CW<6", the state will be converted from the state "Init" to the state "Withdraw/deposite/balinq", and with the outputting of "rOk". Similarly, The label of transition "PIN/Invalid[CA=2],Logon" means that, when in the implementation of *"Logon"* operation and the input symbol of the state *PIN* satisfying " *CA=2"*, the state will be converted from the state "Re_Logon" to the state "CardLocked" with outputting of "Invalid"

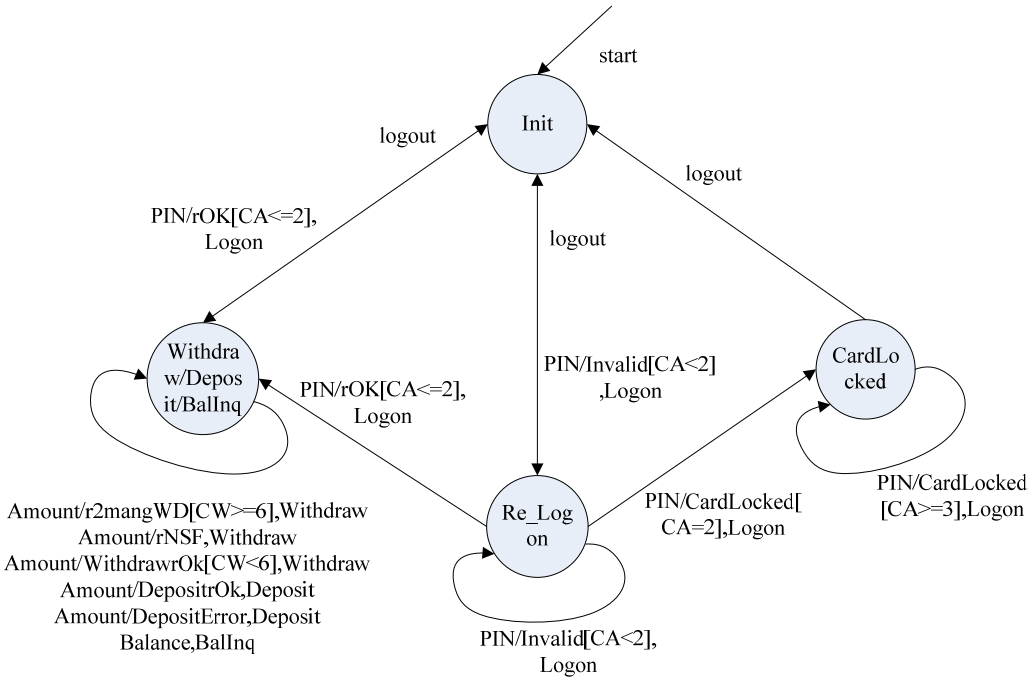

Figure 1. EFSM of automated teller machine

B. PROMELA Model

PROMELA program consists of asynchronous processes, message channels, and data objects. Process Characterizes the behavior of systems, channels and global variables used to define the process execution environment. PROMELA is similar to the C programming language that allows dynamic creation of parallel processes, and processes can be synchronized via a message channel (using the meeting point (rendezvousPort) and asynchronous (buffered) communication.

Though EFSM model is expressed, we need to decide how to model the exchange of messages among three parties.The message flow is shown in Fig. 2. Message is defined as "mtype={PIN,InvalidPIN,CardLocked,OkPIN, Operate_Logon,Operate_Withdraw,Operate_Deposit,Ope rate_BalInq,Operate_Logout,WithdrawrOk,DepositrOk,r Nsf,r2ManyWD, DepositError, WithdrawError, SmbolRequest, SmbolResponse }".

In order to avoid the state explosion, the type of amount is "Byte", and "Byte" type can also simulate the amount of transfer between the three parties and changes.

Message channels are used to model the exchange of data between processes. Channels are declared as shown in Fig. 3. "MAX" represents the number of Terminals. The number of Cardholders is equal to terminals .The same to say, one bank servers MAX terminals. "CusToATM[Max]" is an array of type channel. "CusToATM" sends messages from cardholder to terminal, and each message is said to consist of four fields: the first is declared to be of type byte that represents the cardholder No, the second is of type mtype that represents operation from cardholder to terminal, the third is of type byte that represents amount relates and the last is of type mtype that represents messages related to operations. Similarly, "ATMToCus" sends messages from terminal to cardholder. "ATMToBank" is declared to be capable of storing up to MAX messages from terminal to bank, and each message is said to consist of four fields: the first is declared to be of type byte, the second is of type mtype, the third is of type byte and the last is of type mtype. "BankToATM" sends messages from bank to terminal similarly to "ATMToBank".

According to the analysis above, we create models in PROMELA as shown in Fig. 4, Fig.5 and Fig.6. We instantiate processes as shown in Fig. 7 by using a predefined operator called "run". In addition to, some global variables are defined, such as "AccountBalance" that indicates the current account balance and changes with successful deposit or withdrawal operations.

Because the model checker runs in a closed condition, the user can not participate in the process of running. Therefore, in order to traverse all cases, the initial account balance, and the amount of each deposit and withdrawal will be careful to design. For example, if the initial account balance is "20" and amount of withdrawal is 40, then the model checker may only traverse a path with outputting " rNSF", while the other states are not reachable. In this case, we can not decide that the model does not meet properties. In addition to, For example, given the initial value of" *CW=0*", to covers the transition "*Amount/r2manyWD [CW>=6],withdraw*" in the EFSM of Fig. 1, we need to invoke the transition labeled" *Amount/rok[CW<6], withdraw*" 5 times. As an example of an infeasible transition, given the initial balance of 100 and the amount withdrawal of 5, the transition labeled" *Amount/rNSF, Withdraw*" is not feasible. In order to model more realistic simulation of the situation, the amount of deposit and withdrawal required some changes. In Fig. 4, the amount of deposit is determined concurrently between the fixed values by using "::".

In Fig. 7, we instantiate processes with the order from Bank Server, ATM to Cardholder

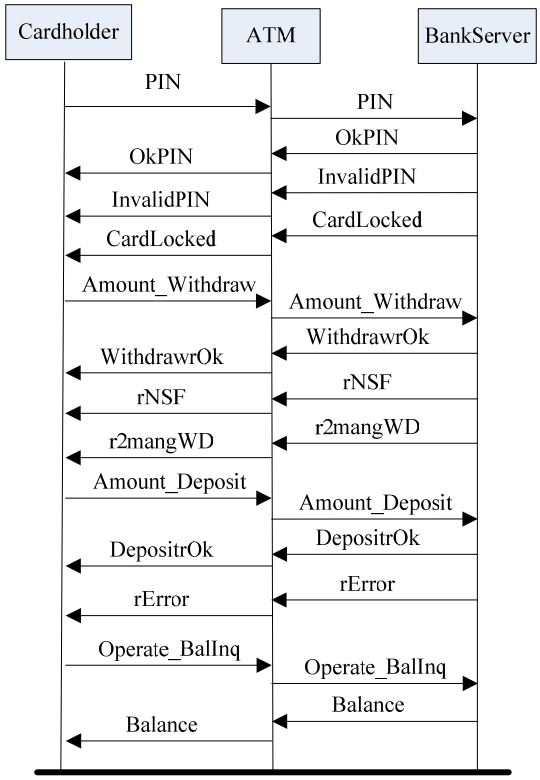

Figure 2. Massage flow of ATM

```
Chan CusToATM[MAX]=[0]of{byte, mtype, byte, mtype};
/*message from customer to ATM*/
Chan ATMToCus[MAX]= [0]of{byte, mtype, byte, mtype};
/* message from ATM to customer*/
Chan ATMToBank=[MAX]of{byte, mtype, byte, mtype};
/*message from ATM to bank */
Chan BankToATM=[MAX]of{byte, mtype, byte, mtype};
/*message from bank to ATM */
```

Figure 3. .Channels' definition in PROMELA

```
proctype Cardholder(chan cusToATM;chan atmToCus;byte j)
{  ...
 Cus_Logon_Request:
    if
     ::cusToATM!j,Operate_Logon,0,PIN->
       if
         ::atmToCus?eval(j),Operate_Logon,0,OkPIN->
                printf(" You are welcome!");
                 goto SelectOperator
        ::atmToCus?eval(j),Operate_Logon,0,InvalidPIN->
                printf(" PIN is Invalid!");
                goto Cus_Logon_Request
        ::atmToCus?eval(j),Operate_Logon,0,CardLocked  ->
                printf(" Card is Locked! ");
                goto Cus_CardIsLocked
       fi;
 Cus_CardIsLocked:
    if
     ::CA=0 ->
         goto Cus_Logon_Request/*simulate next day,*/
     ::cusToATM!j,Operate_Logout,0,SmbolRequest->
                /* logout*/
       ...
    fi;
SelectOperator:
    if
     :: printf("CUS_SelectOperator:Deposit \n")->
        if
           ::Amount_Deposit=20
           ::Amount_Deposit=30
        fi;
        if
         ::cusToATM!j,Operate_Deposit,Amount_Deposit
           ,SmbolRequest->
            if
            ::atmToCus?eval(j),Operate_Deposit,0, DepositError->
                    printf(" Error, deposit is not ok  \n");
                    goto SelectOperator
           ...
           fi;
        fi;
       :: printf("CUS_SelectOperator:WithDraw \n")->
          if
           ::Amount_Withdraw=20
           ::Amount_Withdraw=30
          fi;
          if
           ::cusToATM!j,Operate_Withdraw,Amount Withdraw
             , SmbolRequest ->
               if
                 ::atmToCus?eval(j),Operate_Withdraw,0
                    ,WithdrawrOk->
                  printf("withDraw is ok  \n");
                  goto SelectOperator
                  ...
               fi;
          fi;
          ...
}
```

Figure 4. Fragment of cardholder process in PROMELA

```
proctype ATM(chan cusToATM;chan atmToCus;
chan atmToBank;chan bankToATM;byte i)
{   ...
 ATM_Begin:
   if
     ::cusToATM?eval(i),Operate_Logon,0,PIN->
        atmToBank!i,Operate_Logon,0,PIN;
        if
         :: bankToATM?eval(i),Operate_Logon,0,OkPIN ->
              atmToCus!i,Operate_Logon,0,OkPIN;
              goto ATM_Begin
         :: bankToATM?eval(i),Operate_Logon,0,InvalidPIN->
              atmToCus!i,Operate_Logon,0,InvalidPIN;
              goto ATM_Begin
        :: bankToATM?eval(i),Operate_Logon,0,CardLocked->
              atmToCus!i,Operate_Logon,0,CardLocked;
              goto ATM_Begin
        fi
      :: cusToATM?eval(i),Operate_Deposit
          ,ATM_Amount_Deposit,  SmbolRequest->
                ::atmToBank!i,Operate_Deposit,ATM_Amount
                  _Deposit,SmbolRequest->
        if
         ::bankToATM?eval(i),Operate_Deposit,0,DepositrOk->
                  atmToCus!i,Operate_Deposit,0,DepositrOk;
                   goto ATM_Begin
         ::bankToATM?eval(i),Operate_Deposit,0,DepositError->
                  atmToCus!i,Operate_Deposit,0,DepositError;
                    goto ATM_Begin
        fi
       :: cusToATM?eval(i),Operate_Withdraw,
          ATM_Amount_Withdraw,SmbolRequest->
             ::atmToBank!i,Operate_Withdraw,
               ATM_Amount_With  draw,SmbolRequest->
                if
                 ::bankToATM?eval(i),Operate_Withdraw,0,
                    WithdrawrOk->
                  atmToCus!i,Operate_Withdraw,0,WithdrawrOk;
                    goto ATM_Begin
                 ::bankToATM?eval(i),Operate_Withdraw,0,rNsf->
                   atmToCus!i,Operate_Withdraw,0,rNsf;
                    goto ATM_Begin
             ...
}
```

Figure 5. Fragment of ATM process in PROMELA

```
proctype BankServer(chan atmToBank;chan bankToATM )
{  ...
 Bank_Begin:
   if
    ::atmToBank?k,Operate_Logon,0,PIN->
      if
       ::(CA==0)->
        if
          ::bankToATM!k,Operate_Logon,0,OkPIN->
                goto Bank_Begin
          ::bankToATM!k,Operate_Logon,0,InvalidPIN->
              CA=CA+1;  goto Bank_Begin
        fi
       ::(CA==1)->
          if
            ::bankToATM!k,Operate_Logon,0,OkPIN->
                    goto Bank_Begin
            ::bankToATM!k,Operate_Logon,0,InvalidPIN->
                    CA=CA+1;  goto Bank_Begin
          fi
       ::(CA==2)->
          if
            ::bankToATM!k,Operate_Logon,0,OkPIN->
                    goto Bank_Begin
            ::bankToATM!k,Operate_Logon,0,CardLocked->
                    CA=CA+1;    goto Bank_Begin
          fi
       fi
    :: atmToBank?p,Operate_Withdraw,Bank_Amount_
       Withdraw, SmbolRequest->
     if
       ::(Bank_Amount_Withdraw>Balance)->
            bankToATM!p,Operate_Withdraw,0,rNsf;
             goto Bank_Begin
        ::(Bank_Amount_Withdraw<=Balance)->
            if
              ::(CW<6)->
                 if
     :: bankToATM!p,Operate_Withdraw,0,WithdrawrOk->
              Balance=Balance-Bank_Amount_Withdraw;
              CW=CW+1;  goto Bank_Begin
     ::bankToATM!p,Operate_Withdraw,0,WithdrawError->
                    goto Bank_Begin
                 fi
               ::(CW==6)->
        bankToATM!p,Operate_Withdraw,0,r2ManyWD;
                        goto Bank_Begin
            fi
     fi
    …
}
```

Figure 6. Fragment of bankserver process in PROMELA

```
init {
        atomic{
         run BankServer(ATMToBank,BankToATM);
         run ATM(CusToATM[0],ATMToCus[0],ATMToBank,BankToATM,0);
         run ATM(CusToATM[1],ATMToCus[1],ATMToBank,BankToATM,1);
         run ATM(CusToATM[2],ATMToCus[2],ATMToBank,BankToATM,2);
         run Customer(CusToATM[0],ATMToCus[0],0);
         run Customer(CusToATM[1],ATMToCus[1],1);
         run Customer(CusToATM[2],ATMToCus[2],2);
         …
              }
      }
```

Figure 7. Fragment of init process in PROMELA

V. PROPERTIES DEFINITION AND VERIFICATION

A. Properties Definition

SPIN assists users in finding unreachable codes or deadlocks. In addition, SPIN also verifies LTL properties that we are interested in against PROMELA models. LTL allows expressing temporal properties we expect the system behavior will conform to during the system lifetime.

About ATMs, the first property ensures that deposits and withdrawals are mutually exclusive operations forever. It can be expressed in LTL formula as :"!(<>(withdrawing &&depositing))". And PROMELA Model also needs insert few control statements, as in shown in Fig. 8.

```
proctype BankServer(chan atmToBank;chan bankToATM )
{
  ...
   if
    ::bankToATM!p,Operate_Withdraw,0,rNsf->
          goto Bank_Begin
     ::(withdrawing==false && depositing==false)->
          atomic{
                  withdrawing=true;
              bankToATM!p,Operate_Withdraw,0,WithdrawrOk;
                    withdrawing=false;
                      goto Bank_Begin
                    }
       ...
   fi
  ...
}
```

Figure 8. Fragment of process with control statements

The second property to be verified should ensure that card holders should be allowed access to deposit, withdraw and query after entering the correct password. It can be expressed in LTL formula as: *[](pinIsOk-><>(depositPermit ||withdrawPermit||balinqPermit||logoutPermit))*

The third property to be verified should ensure that As long as the withdrawal amount is not greater than the account balance, finally withdrawal will be successful. It can be expressed in LTL formula as:

[](amountNoMoreThanBalance-><>withdrawIsOk)

B. Verfication

We stated some properties that an ATM should verify. Let us now observe whether they are or not verified in the analyzed PROMELA model. We performed the experiments with SPIN version 6.1.0 on a cloud computing platform. The allocated resources from the platform are as follows: Intel xeon E7540 2GHz; 8GB of RAM; Linux version 2.6.18.

We verified successfully that PROMELA models meet all the properties with 3 or less cardholders, and No deadlock or non-progress cycle were found. With 3 cardholders, PROMELA models are sufficient to simulate the business of ATM, and do not need to verify with more than 3 cardholders. The Figs below summarize some of the performance measures of the verification. Fig. 9 shows number of states with deferent cardholders. Fig. 10 shows number of transitions with deferent cardholders. Fig.11 shows total memory usages with deferent cardholders. Fig.12 shows elapsed times with deferent cardholders. Fig.13 shows DFS-search with deferent cardholders. All Figs match index movement tendency.

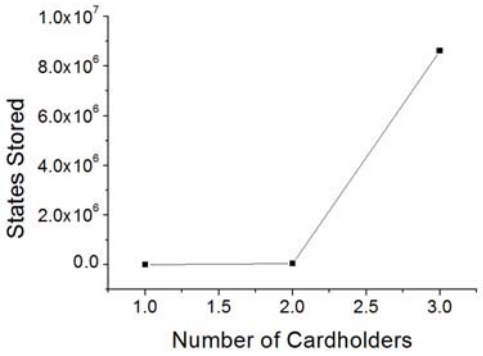

Figure 9. States stored of 1-3 cardholders

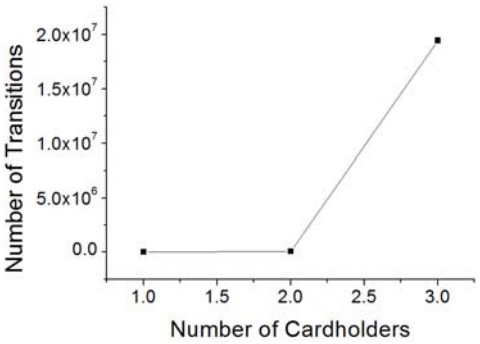

Figure 10. Transitions of 1-3 cardholders

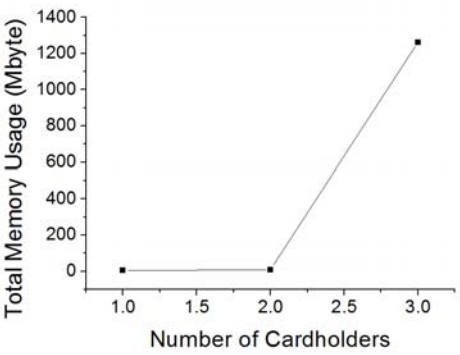

Figure 11. Total memory usage of 1-3 cardholders

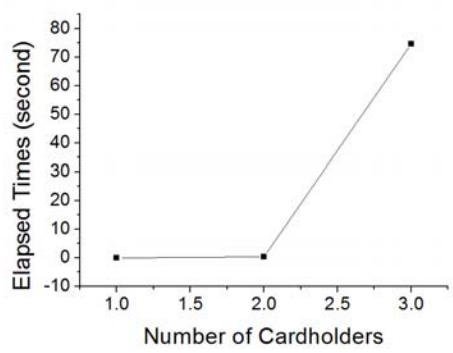

Figure 12. Elapsed times of 1-3 cardholders

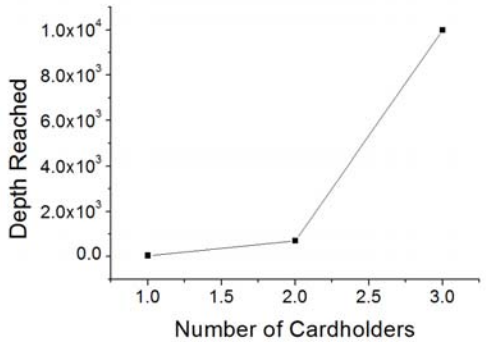

Figure 13. Depth reached of 1-3cardholders

VI. Conclusions and Future work

In this paper we give an approach to verify the business flow of retail banking system, through an example of verifying automatic teller machine (ATM) with SPIN. We consider ATM as extended finite state machines that can be presented in PROMELA. We show how properties of ATM can be expressed in the form of linear temporal logic statements and then verified by applying model checker SPIN. It proves that SPIN and model checking are appropriate for studying the business flow of ATMs. And in our future research, we will expand the field used by model checking with SPIN, for example e-banking system, mobile banking system. Also we will study approaches to solve the state explosion.

Acknowledgment

This work was supported by the National Natural Science Foundation of China under Grant No. 61070039, the Fund for the Doctoral Research of Shandong Academy of Sciences under Grant No. 2010-12, the Outstanding Young and Middle-aged Scholars Foundation of Shandong Province of China under Contract No. BS2011DX033.

References.

[1] Lu Simei, Zhang Jianlin, Luo Liming. "The Automatic Verification and Improvement of SET Protocol Model with SMV". *In: Proceedings of Information Engineering and Electronic Commerce*, 2009.

[2] Stylianos Basagiannis, Panagiotis Katsaros, Andrew Pombortsis. "An intruder model with message inspection for model checking security protocols".*Computers & Security*,29,pp:16-34,2010.

[3] Syed M.S. Islam, Mohammed H. Sqalli, Sohel Khan. "Modeling and Formal Verification of DHCP Using SPIN".*International Journal of Computer Science & Applications*. 6(3),pp:145-159, 2006.

[4] Andreas Both, Wolf Zimmermann and Rene Franke. "Model Checking of Component Protocol Conformance - Optimizations by Reducing False Negatives".*Electronic Notes in Theoretical Computer Science* ,263,pp:67-94,2010.

[5] Barry Long, Juergen Dingel, T.C. Nicholas Graham." Experience Applying the SPIN Model Checker to an Industrial Telecommunications System".*In.proc.of 30th International Conference on Software Engineering*, ACM 2008:693-702.

[6] Li Jing, Li Jinhua. "Model Checking the SET Purchasing Process Protocol with SPIN". *In: proc. Of 5th International Conference on Wireless Communications, Networking and Mobile Computing*, 2009.

[7] On-the-fly LTL model checking with SPIN. http://spinroot.com/spin/whatispin.html

[8] G.J. Holzmann, "The model checker SPIN", *IEEE Transactions on SE*, vol.23 (5), pp. 279–295, 1997

[9] G. J. Holzmann, "The SPIN Model Checker: Primer and Reference Manual", *Addison Wesley*, 2004.

[10] Michael Huth, Mark Ryan, "Logic in Computer Science: Modelling and Reasoning about System", Cambridge University Press, 1999.

[11] Marina Mongiello, "Finite-state verification of the ebXML protocol", *Electronic Commerce Research and Applications* pp: 147-169, 2006(5).

[12] Vladimir A. Oleshchuk, "Modeling,specification and verification od ad-hoc sensor networks using SPIN", *Computer Standards & Interfaces,pp* :159-165,2005(28).

[13] Xiao Yu, Zheng Wang,Geguang Pu, etc. "The Verification of rCOS Using Spin". *Electronic Notes in Theoretical Computer Science,* pp: 49-67, 2008(207).

[14] Flanagan C and Godefroid P. "Dynamic partial-order reduction for model checking software". *ACM SIGPLAN Notices*, 40(1),pp: 110-121,2005.

Huiling Shi(1978-),was born in Heze, China. She received her B.E degree in 2001 from Computer and Applications, Shandong University of Technology in Zibo, her M.E. degree in 2004 from Computer and Applications, Shandong University in Jinan. She works in Shandong Computer Science Center. Her main research field is in software testing, formal verification software quality assurance, and so on.