Contents lists available at GrowingScience

Management Science Letters

homepage: www.GrowingScience.com/msl

Information systems validation using formal models

Azadeh Sarram[a*] **and Ali Harounabadi**[b]

[a]*Islamic Azad University, Damavand Science and Research Branch, Iran*
[b]*Department of Computer Engineering, Islamic Azad University Central Tehran Branch, Iran*

C H R O N I C L E

Article history:
Received October 28, 2013
Received in revised format 20 November 2013
Accepted 10 January 2014
Available online January 14 2014

Keywords:
Unified Modeling Language (UML)
Activity diagram
Validation
Executable model
Colored Petri nets

A B S T R A C T

During the past few years, there has been growing interest to use unified modeling language (UML) to consider the functional requirements. However, lacking a tool to detect the accuracy and the logic of diagrams in this language makes a formal model indispensable. In this study, conversion of primary UML model of a system to a colored Petri net has been accomplished in order to examine the precision of the model. For this purpose, first the definition of priority and implementation tags for UML activity diagram are provided; then it is turned into colored Petri net. Second, the proposed model provides translated tags in terms of net transitions and some monitors are used to control the system characteristics. Finally, an executable model of UML activity diagram is provided so that the designer could simulate the model by using the simulation results to detect and to refine the problems of the model. In addition, by checking the results, we find out the proposed method enhances authenticity and accuracy of early models and the ratio of system validation increases compared with previous methods.

© 2014 Growing Science Ltd. All rights reserved.

1. Introduction

Development of systems and engineering is a discipline, which includes various activities, such as identifying requirements, designs, implementation, testing and supporting efforts. Since designing is the first phase of a system development process, it is important to consider all required features at this stage. One way to confront this challenge is to construct a system model. Modeling is a global technique applied in development of system operations. Using various models is inevitable in development process since they are capable of concealing communication system in a particular stage of development and reduce its complexity. As software systems become more complicated, it is getting difficult to ensure the accuracy of their behavior (Berardi et al., 2005). As building a model becomes definitely necessary, we require a modeling language.

*Corresponding author.
E-mail addresses: azadeh.sarram@gmail.com (A. Sarram)

© 2014 Growing Science Ltd. All rights reserved.
doi: 10.5267/j.msl.2014.1.012

Nowadays, Unified Modeling Language (UML) is an advanced and powerful technique for complex systems analysis implemented by many designers and engineers. However, lacking a diagnosis tool to define the precision and logic of systems can be considered as a blind spot for UML. In other words, UML is not yet a formal model and we should convert it into a formal language such as Petri net to verify the correctness of the designed system (Fu et al., 2010). There are several studies to dissolve the problem of semi-formal feature of the UML, where in many of them transition algorithm is created to convert a UML model to corresponding Petri net until validation operation is pursued most strongly (Gogolla et al., 2005; Storrle, 2004; Hu et al., 2004).

In this paper, UML activity diagrams are used to verify the system validity. For validation, first, activity diagram has been annotate by two tags and they are turned into colored Petri net. The paper also provides tags, which are translated in terms of net transitions then monitors are used to control the system characteristics. Finally, an executable model of UML activity diagram is provided so that we could find out the correctness and incorrectness of the primary UML model.

2. Literature review

Since most information systems are written with UML, it is important to make sure the application field validation can be performed directly on the UML diagrams, and by using this approach, we can decrease the expenses of the implementation. For validation, diagrams should be converted into a formal model, whereas Petri nets as a formal model with a strong mathematical background as well, is implemented to perform the validation in this research. Among behavioral diagrams of the UML, activity diagrams are applied in this study.

2.1. Colored Petri net

Petri nets can be used in modeling, describing and analyzing the nature of a distributed, parallel, or occasional system (Jensen et al., 2007). An important feature of Petri nets is that they are executable. Unlike UML, in Petri Nets, analysis and implementation are carried out simultaneously. Colored Petri nets as an advanced technique have been presented by Jensen et al. (2007), and are being used for modeling and validation of systems in which concurrency, communication and synchronization play essential role. Petri nets generally can be divided into two types of networks; high and low level network. Colored Petri net belongs to the high-level networks and is a combination of Petri net and standard modeling language (SML) (Kristensen & Christensen, 2004). Besides, places, transitions and tokens, concepts of color, guard, arc expressions and code segments are presented in this network.

In colored Petri nets, data values have been carried by distinguishable tokens and unlike Petri net tokens; they are distinguishable from one another because each token has an attribute called color. Different colors in the colored Petri net make it possible to transfer the tokens, which carry various data values and are recognizable from each other. In this network, firing requires tokens on the input places to be coincide with arc expressions. Because of colored tokens, colored Petri nets have high capability in the concept of concurrency. A formal description of Colored Petri Nets is as following:

Colored Petri net is a six-arranged in the form of (P, T, C, I +, I-, M0):

•P is a finite and non-empty set of places.

•T is a finite and non-empty set of transitions.

•Share of two P and T sets is empty.

•C is a colored function that is mapping of P□T set to non-empty set.

•I+ and I- are respectively the forward and backward cross functions defined on the P * T , that we have for each (p, t) belongs to C (p):

I-(p, t) I+(p, t) : C(t)→ C(p)

• M0 is a function defined on P, which describes the initial marking in a way that per each p belonging to P, M0 (p) € C (p) relation is satisfied.

2.2 UML activity diagram

One of the UML behavioral diagrams is the activity diagram, which represents the workflow, performs operations during an activity and offers a view of the system behavior. In this diagram, sequence and conditions as well as order of performing an operation are indicated. Sequence of operations from start to end is called an activity. These diagrams are usually used to express the behavior of objects of classes, which are associate with an activity.

2.2.1 Activity diagram validation criteria

Validation criteria of the activity diagram are as follows,

• Completeness: whether or not the main events of the designed systems take place through special circumstances.

• Deadlock: It describes a state where a process is waiting for some event that will never happen. It could be waiting for a resource to be available before continuing its execution while another process is holding indefinitely this resource. In this situation, the system would not have any progress.

• Priority: It specifies the order between events in the system with respect to time, for instance, a certain event should not occur unless a particular event or a sequence of events were finished.

• Reachability: Each node must be reachable from the initial node. This property ensures that each node is not superfluous.

3. Related work

Since UML is a semi-formal language, it is not possible to apply mathematical methods directly on its model to check the system validation. For this purpose, there is a significant efforts to consider mappings from UML to the other (mostly formal) modeling techniques to validate UML models (e.g. using B, CSP, SPIN, and PVS, petri nets, Z / Eves and object-Z).

López-Grao et al. (2004) used the sequence and state diagrams for validation and system efficiency evaluation. According to Fu et al. (2010) UML architecture is converted into a formal model called SAM. According to Knapp and Merz (2002) a component named HUGO is presented where the focus is on the consistency of UML state. This tool can also detect the presence or absence of deadlocks.

4. The proposed method

In this paper, first, activity diagram is annotated by priority and implementation tags. Then, it is turned into colored Petri net. Second, defined tags have been translated in terms of net transitions then monitors are used to control the system’s characteristics. Finally, an executable model of UML activity diagram is provided so that the designers would be able to simulate the designed model, to identify and to refine the model problems by using the results of simulation and its analysis.

4.1 Conversion of the activity diagram components to CPN

Components of activity diagram convert into CPN is follows,

• The initial node is converted into a place in the CPN.
• The final node is converted into a place in the CPN.
• The conditions of the activity diagram are CPN guards on the arcs.
• Transmissions are converted into transitions in the CPN.
• Actions are the same places in the CPN.
• Fork in the activity diagram is converted into a network as Fig. 1.
• Join in the activity diagram is converted into a network as Fig. 2.
• Decision is converted into multiple transitions as Fig. 3.
• Merge is converted into a network as Fig. 4.

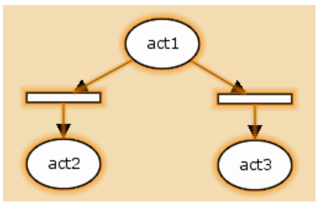


Fig. 1. Fork in the CPN

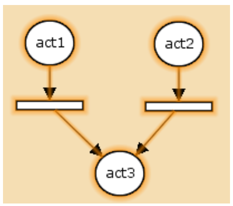


Fig. 2. Join in the CPN

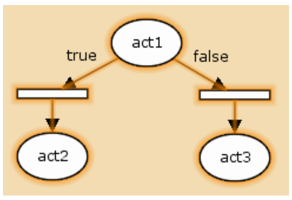


Fig. 3. Decision in the CPN

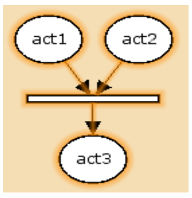


Fig. 4. Merge in the CPN

• If sub-activity exists in the diagram, after the desired action, which involves a sub-action, a substitution transition and a place is considered to transfer the result of the transition and in a separate page (the page related to substitution transition), the related network will be drawn for that.

4.2 Definition of tags to annotate the activity diagram

• APriority tag: To assign the value of this tag, we start from the beginning of the diagram and move to its end. The amount of this tag will measure for the first action, No. 1, and for second action, No. 2 in this manner.
• AExecution tag: This tag will have false value for all the actions in the activity diagram and will change while running the corresponding CPN.
• AAtomic tag: If there is a sub-activity for an action in the activity diagram, the value of this tag will be true, otherwise it will be false.

4.3 Separation of how to assign APriority tags

• In case there is a simple activity diagram shown in Fig. 5, one should begin from the first part of the diagram and the APriority value for the first action will be assigned to number 1 and moving down toward the diagram the next numbers are assigned to the subsequent actions, as shown in Figs. (5-6).

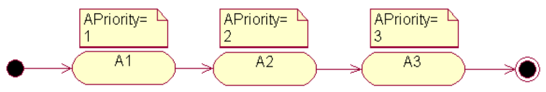


Fig. 5. Assigning APriority tag value for a simple activity diagram

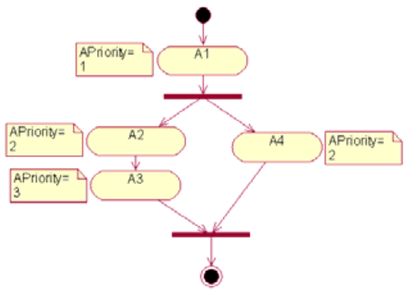


Fig. 6. Assigning APriority tag value for an activity diagram with fork or join

• In case there is any fork or join in the activity diagram, for each diagram branches, same numbers will be assigned for the APriority tag as shown in Figs. (7-8) and the last number used in the branches, will be our criteria for the other actions APriority value.

• If there is any decision and merge in the activity diagram respectively shown in Figs. 7-8, for the APriority tag, first, the lower number will be assigned to the subsidiary path, then for the other paths we will assign the numbers from continuation of former preference path.

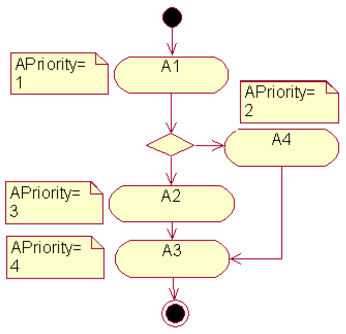


Fig. 7. Assigning APriority tag value for an activity diagram with decision

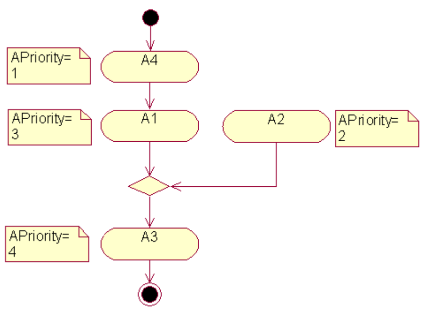


Fig. 8. Assigning APriority tag value for an activity diagram with merge

• In case an action involves sub-activities, normal priority number will be assigned to APriority tag according to diagram motion from top to bottom and a substitution transition will be located after location of mentioned action and we will carry out numbering as a new diagram on the page relevant to the substitution transition of the mentioned action.

• In case, we see return to an action in the diagram, APriority tag value must be set to the APriority tag value of the same action as the action is entered.

4.4 Initializes the defined tags

In this paper, we choose CPN Tools to work with Colored Petri nets. Activity diagram tags have been initialized as Fig. 9, before the corresponding net implementation in the CPN Tools.

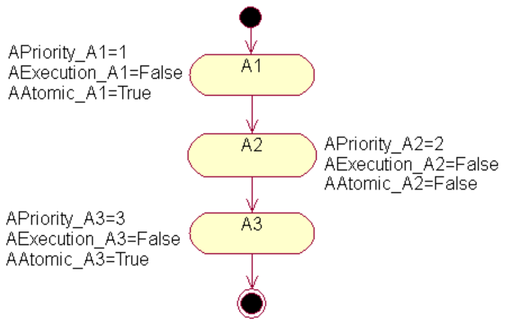


Fig. 9. Tags initialization before implementing the corresponding CPN

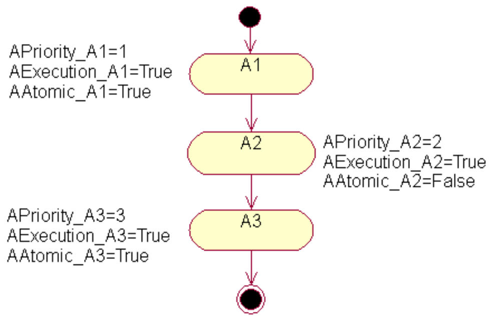


Fig. 10. Tags initialization after running the corresponding CPN

Activity diagram tags have been initialized as Fig. 10, after the corresponding net implementation in the CPN Tools. Summary of steps to check the UML models correctness created from a system is as follows:

- Draw the activity diagram of the desired system,
- Add the defined tags to the activity diagram,

- Convert activity diagram into an equivalent CPN based on defined transformations for each component of the activity diagram,
- Define the APriority tags by variables and AExecution tags in frame of global variables in the CPN Tools,
- Convert the transition after the actions which have AAtomic = False tags in the diagram, to a substitution transition,
- In CPN, for each input transition to a place (places are same as activity diagram actions) testing that prior action APriority tag value is smaller than the latter action,
- In the transition code segment prior to any action, AExecution tag value of that action should be equal to true,
- Defining monitors in the network in order to check whether the created model has the characteristics of the system or not,
- In order to examine existence or nonexistence of a deadlock in the model, it should be controlled through running Petri net whether all paths reach to the final place or not.

5. Case study

A case study is considered in order to demonstrate the method's applicability and to evaluate the correctness of suggested method. For the case study, we look for comprehensive and the simplest structure, which is also a small sample of other actual and large examples. Bank ATM System example is a simple example with various elements of activity diagrams. Then, by using the proposed method and executable model simulation in CPN tools, we verify the behavior of created model through mentioned system.

5.1 Description of the bank ATM system

An ATM interacts with two other entities as Customer (User) and the Bank. One of the main functions of the ATM is "Withdrawal". For this case, customer puts his/her card into the ATM system first. Then, the system verifies the validity of the card and issues a password request and the customer enters the password. If the card is invalid, ATM returns it to the customer; otherwise, it investigates the credibility and validity of the customer card password. If the password is invalid, system returns the customer card. Otherwise, system shows the options of "Withdrawal" and "transfer". If the customer selects "Withdrawal" option, then the system will ask for the amount. Then the customer enters the withdrawal amount and system checks whether there is enough money in the account or not. In addition, system will check whether the ATM has the requested funds or not. In both cases, if there is not sufficient fund, the system gives a warning to the consumer; otherwise, the system deducts withdrawal amount from customer's account and pays the money to the customer. Finally, the system returns the customer's card.

5.2 Withdrawal Activity diagram annotated with tags

Withdrawal activity diagram is shown in Fig. 11. In order to verify correctness of the created model, first, we have added the tags to the diagram. Since, all actions other than "Check Banks Card" action in the diagram have An AAtomic tag with true value, it has been prevented to use tags for all the actions to maintain diagram readability. The activity diagram is converted to an equivalent Petri net as Fig. 12.

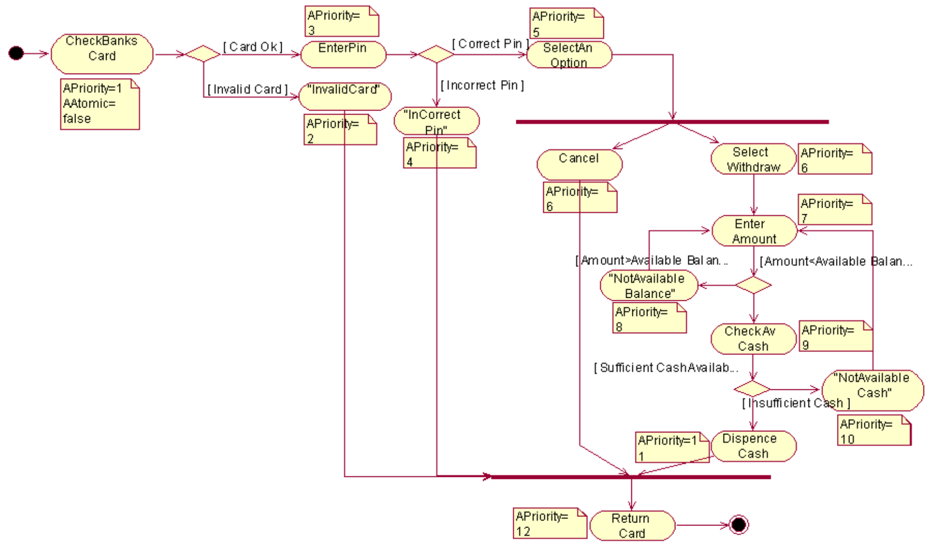


Fig. 11. withdrawal activity diagram annotated with tags

As can be seen in Fig. 12, a substitution transition called sub and an assistance place called p is considered for “Check Banks Card” action when activity diagram converts to the CPN. Page related to sub transition can be seen in Fig. 13. To simulate different card numbers entered to the system, a function called rand () is defined which generates a random number between 1 and 100, and enters as third input component to the "Check Banks Card" action.

At the end of the mentioned page, it is considered that if the card number is valid, number 1 otherwise 0 as the third component is sent to the p output place.

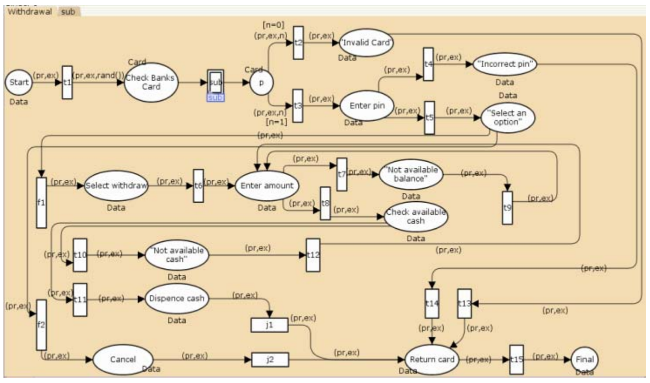


Fig. 12. Equivalent CPN with withdrawals activity diagram

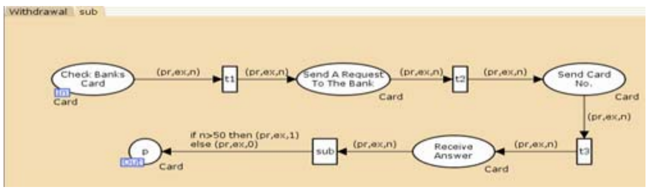


Fig. 13. Page related to sub transition

In order to control two defined tags during the simulation, two variables named pr and ex are considered as network inputs. In the input transition guard to every action, two conditions are controlled as follows:

First, is the amount of AExecution tag true?

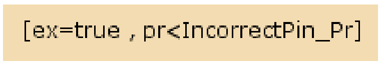
Second, is the Apriority tag value of the previous action less than the next action?

If both conditions hold, the token will pass from transition and go to the next place; it means the next action will perform. A sample of transition guard for “Incorrect Pin” action can be seen in Fig. 14. The APriority tag value of every action on its entrance arc will be placed instead of former amount of pr variable, so that with implementing any action, the APriority tag value of the same action considered as the input variable for the next transition. In Fig. 15, all relevant guards have been assigned to the transitions.

By tracking the simulation, we can easily control the different paths in the network and by moving in the network and change in the value of pr variable, different paths will be diagnosable and considerable by transitions which are able to fire. By automatically running the model, the existence or nonexistence of a deadlock will be recognizable.

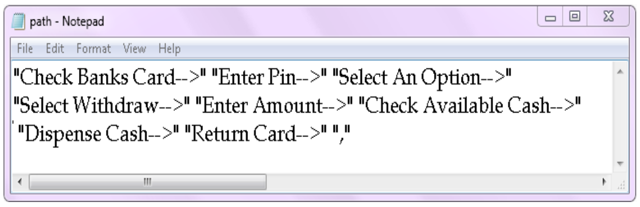
5.3 Code segments definition

In order to check the AExecution tag, in the code segment of each transition, we equalize relevant global variable value to true. To store the token motion path in the network, in the same section of code, we create a new file named path and passing through each action, we will save the name of that action in the related file.



[ex=true , pr<IncorrectPin_Pr]

Fig. 14. Input transition guard to "Incorrect Pin" action



```
"Check Banks Card-->" "Enter Pin-->" "Select An Option-->"
"Select Withdraw-->" "Enter Amount-->" "Check Available Cash-->"
"Dispense Cash-->" "Return Card-->" ","
```

Fig. 15. Path file for successful withdrawal track

To ensure there is no deadlock in the created model; path file can be controlled for different paths. Since, "Return Card" action is the last action on the above system; all paths should end to it.

5.4 Monitors to control the system characteristics

Now, we define certain monitors for some system features. The first characteristic: after the operation is canceled by the customer, the card should be returned to her

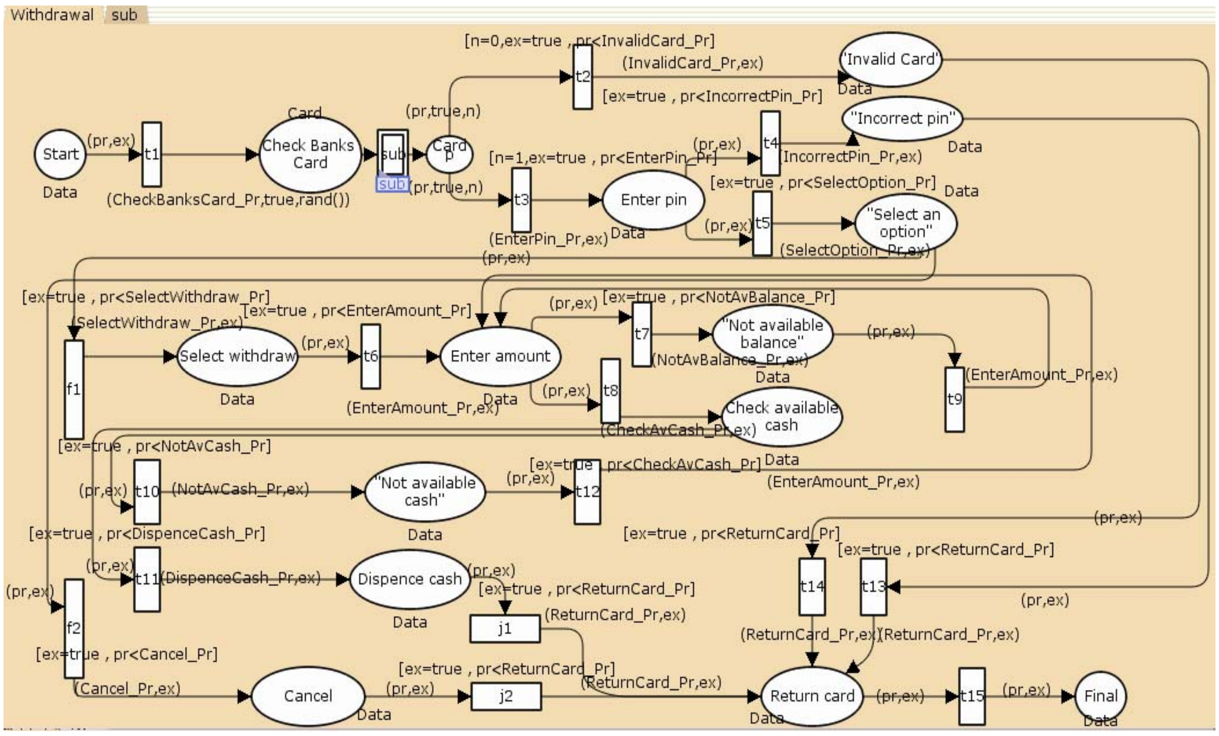
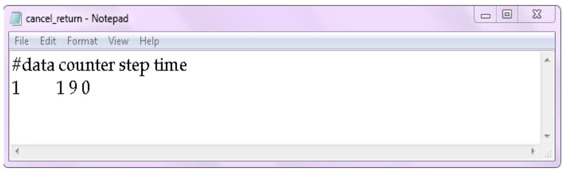


Fig. 16. CPN with transition guards

Whenever the transition after Cancel action named j2 fires, the monitor associated with this feature considers the correctness of Cancel action AExecution tag; if the tag has true value number 1 and otherwise 0 in the first left column of the output file will be printed.

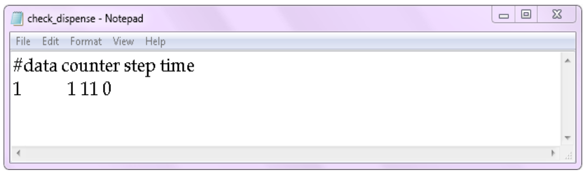
Second feature: If the cash in the ATM is enough, then the money would be given to the customer.

The second monitor, whenever transition named t11 fires, will check the correctness of "Check Available Cash" action AExecution tag value, if the tag has true value, number 1 and otherwise 0 in the first left column of the output file will be printed.



```
#data counter step time
1    1 9 0
```

Fig. 17. Monitor output of the first characteristic



```
#data counter step time
1     1 11 0
```

Fig. 18. Monitor output of the second characteristic

With regard to the gained output, one can conclude the created model meets the system's characteristics.

6. Conclusions and Future Work

In a few studies, in order to convert UML diagrams into formal models, the authors proposed a method based on on existing model checkers such as SPIN, COSPAN, Kronos and UPPAAL (Ober et al., 2006). To use these tools, first UML diagrams should be converted into the acceptable input language. To examine some specific characteristics, some individuals have presented their own special model evaluation language. Simulation has been applied vastly in these studies, although simulation-based analysis cannot provide strong official authentication results but even if this type of analysis cannot prove the correctness of the model, it is still worth to use it as a low cost analytical method, which can help us detect design errors.

These are tools for analyzing the state diagram by simulation or a formal model evaluation: Rhapsody, object GEODE and Rational rose. In VUML tool, the state diagram is considered for model validation. Above mentioned tools do not support a wide range of analytical operation.

In the recent research, there is no need to use any special language and only through adding tags to UML activity diagrams and convert them to a CPN model; one would be able to examine all characteristics of the created model. This method could prioritize the actions implementation and detect the deadlocks.

Furthermore, since the above model has ability to run, it is more than a model and many design errors will be detected by tracking down the network. Future research can be done on other UML behavioral diagrams for validation and one can also try to develop software, which can automatically convert the model into a colored Petri net according to proposed approach.

References

Berardi, D., Calvanese, D., & De Giacomo, G. (2005). Reasoning on UML class diagrams. *Artificial Intelligence*, *168*(1), 70-118.

Fu, Y., He, X., Li, S.H., Dong, Z., & Bording, P.H. (2010). A model-driven approach for runtime assurance of software architecture model. *International Journal of Computer and Network Security*, 2(5).

Gogolla, M., Bohling, J., & Richters, M. (2005). Validating UML and OCL models in USE by automatic snapshot generation. *Software & Systems Modeling*, *4*(4), 386-398.

Hu, Z., & Shatz, S. M. (2004, June). Mapping UML Diagrams to a Petri Net Notation for System Simulation. In *SEKE* (pp. 213-219).

Jensen, K., Kristensen, L. M., & Wells, L. (2007). Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer*, *9*(3-4), 213-254.

Knapp, A., & Merz, S. (2002, June). Model checking and code generation for UML state machines and collaborations. In *Proceedings of 5th Workshop on Tools for System Design and Verification, Technical Report* (Vol. 11, pp. 59-64).

Kristensen, L. M., & Christensen, S. (2004). Implementing coloured Petri nets using a functional programming language. *Higher-order and symbolic computation*, *17*(3), 207-243.

López-Grao, J. P., Merseguer, J., & Campos, J. (2004). From UML activity diagrams to Stochastic Petri nets: application to software performance engineering. *ACM SIGSOFT software engineering notes*, *29*(1), 25-36.

Ober, I., Graf, S., & Ober, I. (2006). Validating timed UML models by simulation and verification. *International Journal on Software Tools for Technology Transfer*, *8*(2), 128-145.

Storrle, H. (2004, September). Semantics of control-flow in UML 2.0 activities. In *Visual Languages and Human Centric Computing, 2004 IEEE Symposium on*(pp. 235-242). IEEE.