Contents lists available at GrowingScience

Management Science Letters

homepage: www.GrowingScience.com/msl

Validation of enterprise architecture through colored Petri nets

Somayeh Toghyani[a*] and Ali Harounabadi[b]

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

C H R O N I C L E

Article history:
Received September 28, 2014
Accepted 15 January 2015
Available online
January 21 2015

Keywords:
Enterprise architecture
Architecture behavior accuracy check
Petri nets – executable model

A B S T R A C T

Enterprise architecture procedure contains some instructions for conversion of enterprise architecture from the current state to the desirable state. This procedure generally contains 3 phases each of which is the basis and prerequisite of the next phase. These phases are: Strategic information technology planning, enterprise architecture planning and enterprise architecture execution. As each phase is a prerequisite of the next one, any fault in each phase causes bigger faults in final results. Therefore, each phase should be double checked to make sure no fault has occurred. The second phase can greatly influence the final results. Therefore the preparation of an executable enterprise architecture model and checking it with functional and non-functional requirements can prevent many faults and lead to execution of a perfect model of the enterprise. The primary objective of this research is to check the accuracy of EA behavior in achieving an appropriate architecture. In this research, official models have been used to propose a solution to transform the products of C4ISR framework to executable Petri nets. Finally, a method is proposed to check the accuracy of the mentioned model. The proposed solution makes the EA semi-automatically check the correctness of the enterprise architecture behavior and increase its accuracy.

© 2015 Growing Science Ltd. All rights reserved.

1. Introduction

EA is a complicated process, which encompasses all parts of an enterprise and involves a large number of individuals with a variety of specializations. Doubtlessly, conducting such an extensive process is not possible without following a premeditated and integrated model. One of the successful approaches in such processes is to apply an EA framework. EA framework enables the architect to control the complexities by using it to regulate/adjust the structure. So far, various architecture frameworks such as Zachman, C4ISR, FEAF, etc. have been introduced. The primary objective of this paper is to focus on the correctness of the behavior of architectural products and considers this framework. Among current architecture frameworks, C4ISR is preferred as it allows demonstration of its products by formal models. C4ISR, which was originally designed to describe military systems, is a comprehensive framework and describes architecture through documents called products.

*Corresponding author.
E-mail addresses: s.toghiyani62@gmail.com (S. Toghyani)

© 2015 Growing Science Ltd. All rights reserved.
doi: 10.5267/j.msl.2015.1.007

The second part of the article is associated with the requirements for this task, while the third part will focus on the procedures taken in line with the subject of the article. The proposed method is explained in the fourth part. The fifth section will offer a case study and a conclusion will be drawn in the sixth part.

2. Requirements

The requirements for this process include an integrated modelling language, Petri Nets, timed Colored Petri Nets and clichés based on service quality, which will be examined briefly due to the significance of Petri Nets.

2.1. Petri Nets

The Petri network theory was first proposed by Carl Adam Petri, and perfectly suits to explain the behavior of systems with concurrent and interactive elements. Petri Nets provide a clear, graphic view of the system with a mathematical technique, which shows communication, control and information trend models. Furthermore, these Nets provide a framework for analysis, validation and evaluation of performance. Petri Nets are graph-based and can be informally described as a directed digraph comprised of the two elements of place and transition.

2.1.1. Colored Petri Nets (CPNs)

Colored Petri Nets were introduced by Kurt Jensen (Jensen, 1994) as a developed model of Petri Nets. In addition to places, transitions and tokens, concepts of color, guard, arc, and code segment are introduced in these nets. The input variables in these nets are carried by tokens. Petri Nets provide more accurate models of non-synchronized complex processing systems where these nets, unlike Petri Nets, are differentiable as each token has a quality called color. Unlike in Petri Nets, tokens can be distinguished in CPNs.

2.1.2. Marked Petri Nets

Petri Nets, which include tokens are called marked Petri Nets. These tokens are represented by (•) go to places. Every marking in a Petri Net is a graph, which a non-negative integer (number of tokens) attributes to each of the places in the net. In a marked Petri Net when each place *T* in transition includes as many tokens as the weight of the arc that connects it to transition *T*, transition *T* is said to have been activated. An activated transition can fire, where the transition in question will remove as many tokens as the weight of the each input arcs in each input place and create as many tokens as the weight of each of the output arcs in each output place. When a transition fires, the marking of the net may change.

2.1.3. Analysis of CPNs

These nets are supported by different tools, among them CPN Tools and Artifex ADesign/CPN. CPN tools software was presented by the University of Aarhus, Demark, and was first released in 2001. The language used to define and modify inputs in this tool is Standard ML. IN this article, we used this tool for simulation and drawing CPNs.

The Graphical user interface of this software allows an easy drawing of CPNs. It also allows animation of CPNs and creation of a state space based on CPNs, while it allows defining a user's ability to question the model's behavior, and creating output files to demonstrate the results of the simulation of models.

2.1.4. Timed CPNs

The concept of CPNs is introduced through the Global Clock. The variables taken by this clock indicate the time of the model. The time can be an integer, which indicates discrete time intervals or a real number, which indicates a continuous time domain. In addition, a variable can be assigned to each token which is called a timestamp. A timestamp refers to the first model where a token can be used.

3. Related work

Evaluation of software architecture based on CPNs were investigated by Shin et al. (2003). Among formal models, generalized versions of Petri Net (Emadi & Shams, 2009) or transforming a sequence diagram to a Petri Net (Mozaffari et al., 2011) into enterprise architecture using formal models of a software system, greatly influence obtaining non-function requirements of a system. Rezai (2006) looks into methods for evaluation of enterprise architecture process program. Raouf (2009) focuses on evaluation and analysis of enterprise architecture and recommends simulation and pre-modelling for evaluation of productions by enterprise architecture program. In order to review and to describe behavioral aspects, the technical article presents software architectures based on CPNs and a technique for its quality assessment using CPNs. In Levis's approach, creating an executable evaluation method does not essentially require creation of architectural productions with a particular modeling language. Saldhana and Shatz (2000) also work on evaluation of architecture from a functionalist point of view using the Archimate model.

4. Proposed strategy to convert UML activity graph to Petri Net

4.1. How to convert activity graph's structures to Petri Net

Converting structures of sequence, decision, repetition, merge, Join and Fork into Petri Nets is carried out as following:

- Sequence structure: in this structure, a number of activities are performed sequentially or in an order (See Fig. 1).

- Fork and Join structures: these structures in activity graph appear as a horizontal or vertical line which has a number of input arcs and an output arc or has one input arc and several output arcs.

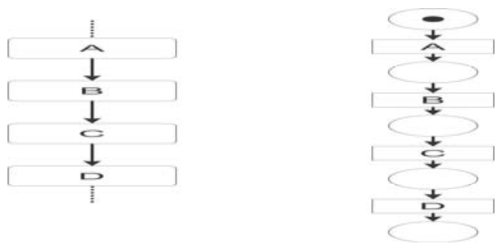


Fig. 1. Sequence structure and its equivalent in Petri Nets

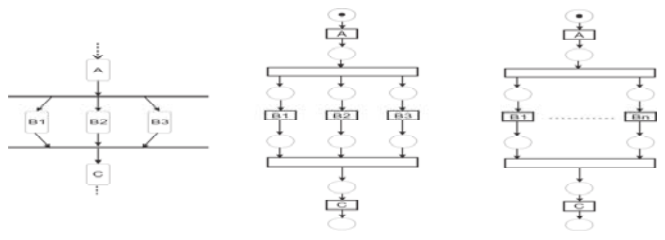


Fig. 2. Fork and Join structures and their equivalents in Petri Nets

- Decision structure: this structure is shown like a rhombus. The decision structure indicates the choice of an operand from among a collection of operands. The operation must implicitly or explicitly involve a protection expression so that it can be correctly evaluated for selection at this point of interaction (Fig. 3).

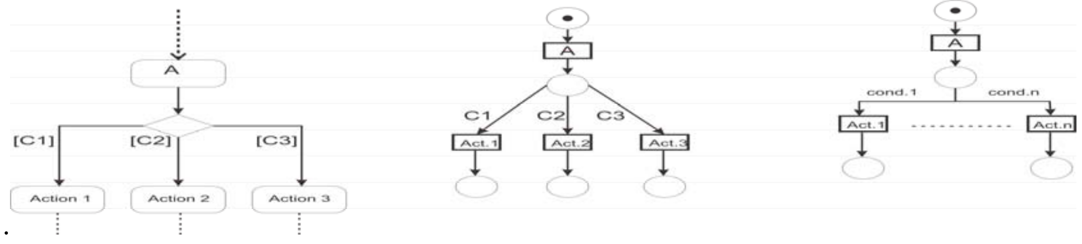


Fig. 3. Decision structure its CPN equivalent

- Iterative structure: this structure shows one or several actions. In this structure several actions are repeatedly executed (Fig. 4).

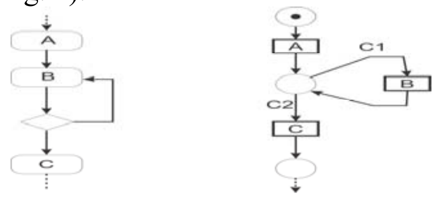


Fig. 4. Iterative structure and its CPN equivalent

- Merge structure: this structure is used to show a merge control node in a UML activity graph. This node assembles alternative flows. A control node has several input edge and several output edges. This node does not apply to concurrent flows but selects only one from among several input flows.

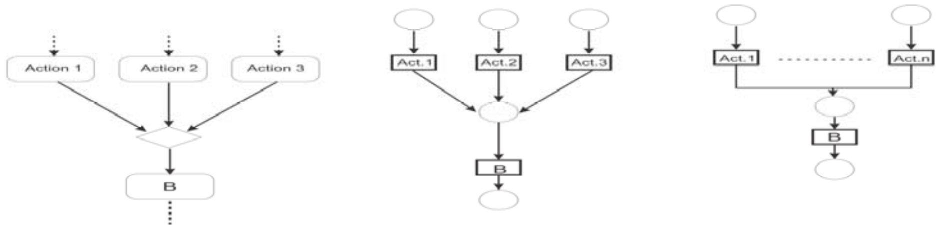


Fig. 5. Merge structure and its CPN equivalent

4.2. Algorithm for converting activity graph into Petri Nets

After it is determined how to convert different structures can be converted into Petri Nets, an algorithm must be offered to convert the various elements in the activity diagram. Elements including edges, activity nodes, control nodes (such as Merge, Join, Fork, etc.), and start and final nodes have to be converted into Petri Net elements such as place, transition and arc. The proposed algorithm is as follows:

1. Start node, final node, control nodes and activity edges are converted into/linked to places,
2. Activity nodes, control nodes and activity edges are converted into/linked to transitions,
3. Convert activity edges into/linked to input/output flows (arcs),
4. Convert Edge expressions into CPN arc inscriptions,
5. Convert conditions on Join, Fork and Merge nodes into arc inscriptions.

4.3. Proposed method to probe correctness of enterprise architecture behavior

This section provides a new method for investigating the correctness of architecture production behavior (UML diagrams). Activity graphs were among the UML diagrams used, thus this proposal is presented on activity diagrams.

Step 1: In the proposed method, activity graphs, that is activity nodes, send and receive events and the existing structures, are used.

Step 2: Flags are set for each activity node. If the value of this flag is true, this indicates that the activity has been executed. Boolean variables are initialized with the value false.

For instance, the msg_b_s flag indicates sending message (b) and msg_b_r indicates that message (b) has been received. The value of this flag indicates whether or not this action has been fulfilled. This way, if the msg_b_s flag value is false, this action has not been done and vice versa.

Send= False, Receive=False,proc1_user = False, proc1_bank = False, proc2_user = False, proc2_bank = False, msg_a_s= False, msg_a_r= False, msg_b_s= False, msg_b_r= False

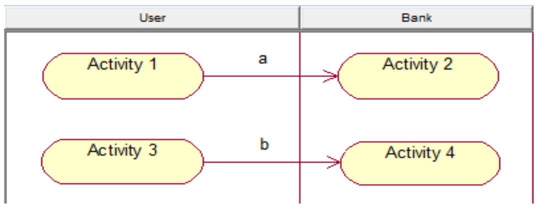


Fig. 6. Activity diagram

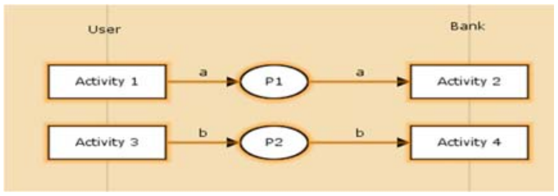


Fig. 7. Equivalent Petri Net

Step 3: The values of flags change when an activity (transition) occurs, that is a place takes a colored token from input, uses it and creates one or several tokens in the output. The value of flags is replaced by the value of true when an activity takes place on the diagram.

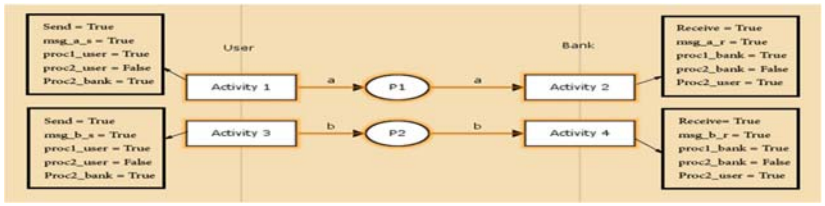


Fig. 8. Activity flags before and after execution

Step 4: The specifications and requirements of the system are expressed as Boolean expressions on defined flags. The operators in use in Boolean expressions include main and complementary operators.

Operator	Example
AND	X AND Y
OR	X OR Y
NOT	NOT X

Main Boolean operators

Operator	Example	Mapping of main operators
XOR	X XOR Y	(NOT(X) AND Y) OR (X AND (NOT Y)
NOR	X NOR Y	NOT (X OR Y)
NAND	X AND Y	NOT(X AND Y)
→	X→ Y	NOT (X) OR Y

Compelementary Boolean operators

Boolean operator	ML Bool
AND	Andalso
OR	Orelse
NOT	Not

Boolean operators in Standard ML

For instance, take the evaluation of these attributes into account:

- ✓ Attribute No.1: Imagine in an activity diagram we want to check "The element User does not send message b as long as the element Bank has not receive message a".

The Boolean expression for attribute No.1:

NOT (X) OR Y

X = (proc1_user AND Send AND msg_b_s),Y = (proc1_bank AND Receive AND msg_a_r)

- ✓ Attribute No.2: imagine in the activity diagram we want to check "First the element DB receives message a and then the element User receives message b"

The Boolean expression for attribute No.2:

NOT (X) OR Y

X= (proc1_user AND Receive AND msg_b_r) , Y = (proc1_DB AND Receive AND msg_a_r)

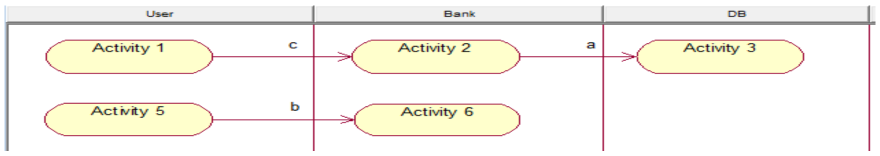


Fig. 9. Activity diagram including a user, a bank and a data base

- ✓ Attribute No.3: Imagine an online shopping center's system, where a user enters the wrong username or bank account password, he/she should not be allowed to finalize purchase and should be allowed to finalize his/her order only after both username and bank account password are entered, correctly. If we call the situation with the wrong username Act 1, with the right username Act 2, with the wrong bank account password Act 3 and with the correct password Act 4, the Boolean expression used is as follows,

Boolean expression for attribute No. 1:

NOT(X) AND Y, X= (Act1 OR Act3), Y= (Act2 AND Act4)

If the output of the first Boolean expression is 1, the activity diagram is correctly drawn and user will be allowed to finalize purchase. If the output is 0, the activity diagram is problematic. The output of this Boolean expression will be 1 only when user has entered their username and bank account password without any mistakes.

To use this Boolean expression in Petri Net, it must be converted into Standard ML. the Boolean expression for attribute No. 1 in Standard ML is as follows:

Not (X) and also Y, X= (Act1 or else Act3), Y= (Act2 and also Act4)

Step 5: One of the specifications of Petri Nets is the possibility of multiple, which enables us to examine the net in all possible situations in scenarios with a number of different modes. For this, the Petri Net needs to be designed so that for each parameter in the activity diagram a unique timestamp is produced.

Step 6**:** After inventing diverse modes for the net execution, all possible execution paths in the diagram are explored and behavioral correctness of the attributes in each path is examined and the architectural paths where the correctness of the system behavior is risked are identified to be removed.

5. Case study

An online purchase model is offered in the form of an activity model. All of the elements in the activity diagram are then be converted into CPNs based on the proposed solution and consequently the behavioral correctness of the proposed model is examined through a facility called the Monitor.

Some of the possible conditions that can compromise the behavioral correctness of this model include:

1. User who has entered the wrong information, such as username or password, on the online purchase model website can still continue the proceedings,
2. User's request is finalized despite wrong account information,
3. User's request is finalized despite insufficient account balance.

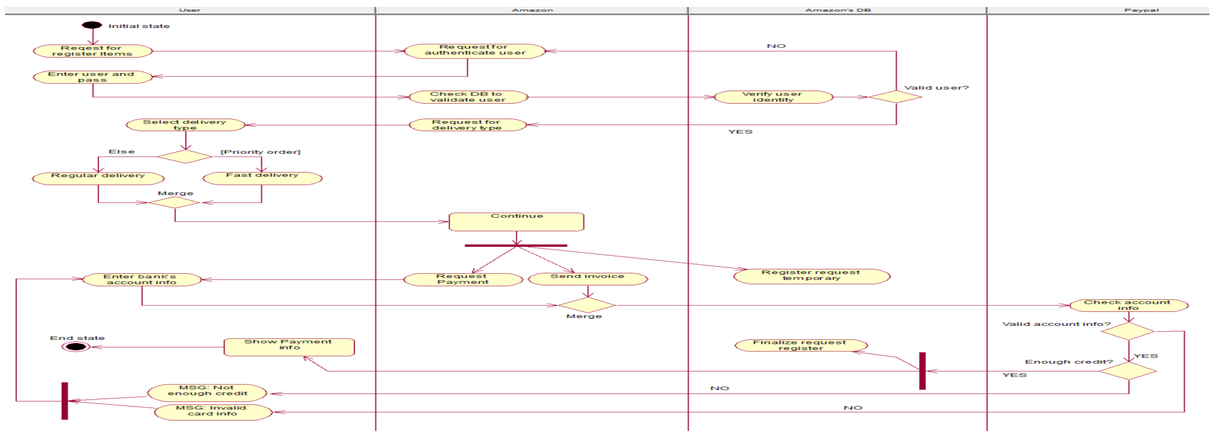


Fig. 10. Activity diagram of a purchase process

Given the limited number of elements in PNCs, demonstration of an activity diagram based on the proposed solution in form of Petri Nets requires huge space. Thus, a Petri Net facility known as the Alternative Sub-transition is used. The page TOP shows a general view of the net.

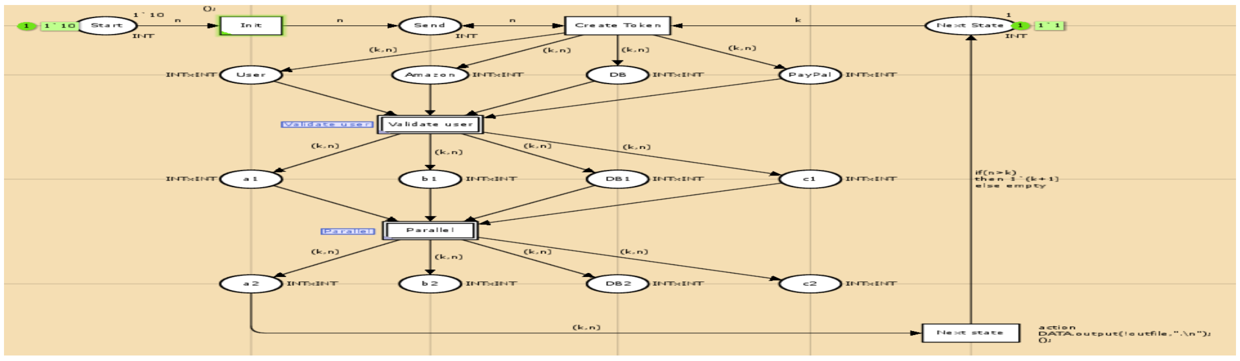


Fig. 11. Page TOP

The alternative sub-transitions Validate User and Parallel, as illustrated in the middle of Fig. 12, are the functions that perform a number of operations on the inputs (input arcs) and send the result to the output (output arcs).

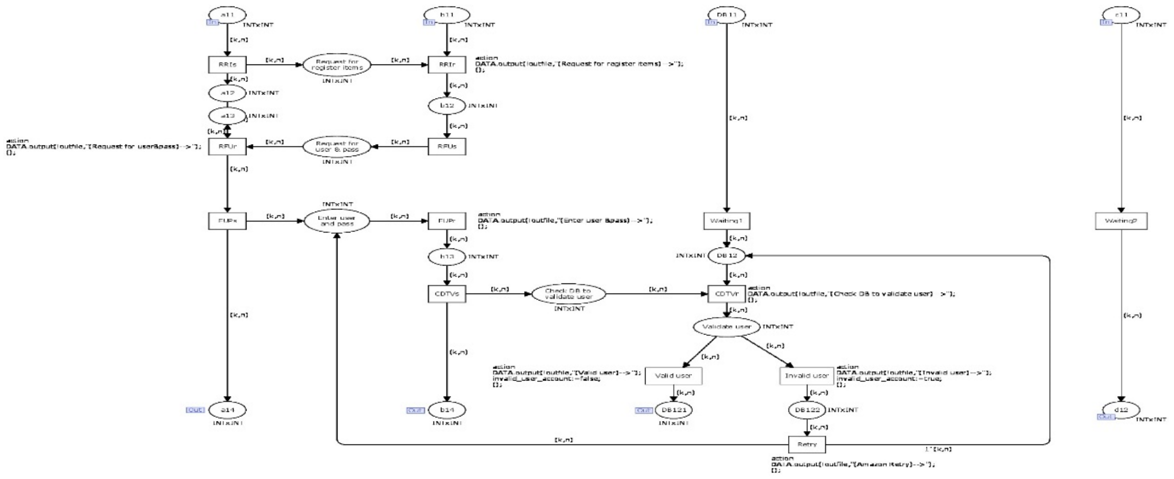

a11
b11
DB11
c11
INTxINT
[k,n]
RRIs
Request for register items
RRIr
action
DATA.output(outfile,"[Request for register items] -->");
();
a12
a13
b12
RFUr
Request for User & pass
RFUs
action
DATA.output(outfile,"[Request for user&pass]-->");
();
EUPs
Enter user and pass
EUPr
action
DATA.output(outfile,"[Enter user &pass]-->");
();
Waiting1
Waiting2
b13
DB12
CDTVs
Check DB to validate user
CDTVr
action
DATA.output(outfile,"[Check DB to validate user] -->");
();
Validate user
Valid user
Invalid user
action
DATA.output(outfile,"[Valid user]-->");
invalid_user_account:=false;
();
action
DATA.output(outfile,"[Invalid user]-->");
invalid_user_account:=true;
();
a14
b14
DB121
DB122
d12
Retry
1`(k,n)
action
DATA.output(outfile,"[Amazon Retry]-->");
();

Fig. 12. Sub-transition “Validate User”

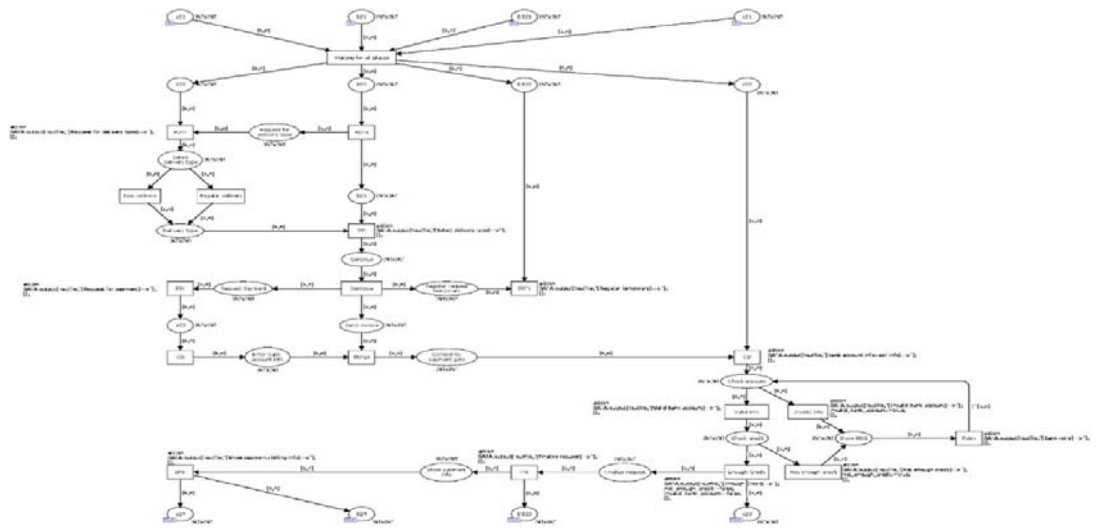
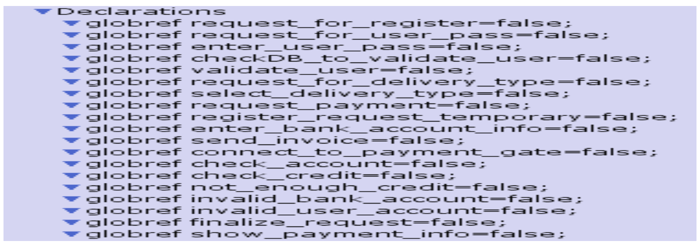


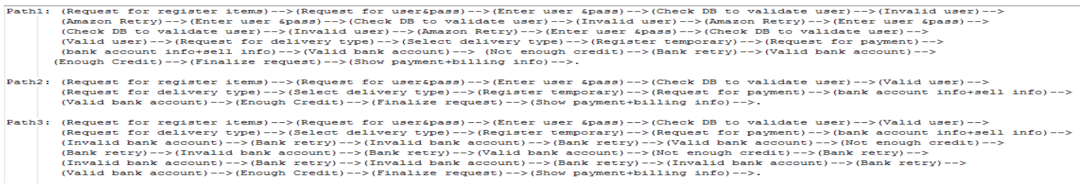
Fig. 13. Sub-transition “Parallel”

After the conversion of the activity diagram based on the proposed solution, it is time to produce a publication node. For this, a publication node is created for each event in the activity diagram under a corresponding name.



```
Declarations
  globref request_for_register=false;
  globref request_for_user_pass=false;
  globref enter_user_pass=false;
  globref checkDB_to_validate_user=false;
  globref validate_user=false;
  globref request_for_delivery_type=false;
  globref select_delivery_type=false;
  globref request_payment=false;
  globref register_request_temporary=false;
  globref enter_bank_account_info=false;
  globref send_invoice=false;
  globref connect_to_payment_gate=false;
  globref check_account=false;
  globref check_credit=false;
  globref not_enough_credit=false;
  globref invalid_bank_account=false;
  globref invalid_user_account=false;
  globref finalize_request=false;
  globref show_payment_info=false;
```

The initialization of each of the publication nodes is False. The occurrence of each activity changes the value to True.



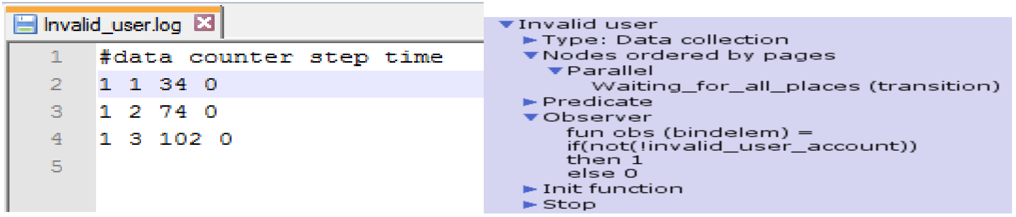
```
Path1: (Request for register items)-->(Request for user&pass)-->(Enter user &pass)-->(Check DB to validate user)-->(Invalid user)-->
       (Amazon Retry)-->(Enter user &pass)-->(Check DB to validate user)-->(Invalid user)-->(Amazon Retry)-->(Enter user &pass)-->
       (Check DB to validate user)-->(Invalid user)-->(Amazon Retry)-->(Enter user &pass)-->(Check DB to validate user)-->
       (Valid user)-->(Request for delivery type)-->(Select delivery type)-->(Register temporary)-->(Request for payment)-->
       (bank account info+sell info)-->(Valid bank account)--> (Not enough credit)-->(Bank retry)-->(Valid bank account)-->
      (Enough Credit)-->(Finalize request)-->(Show payment+billing info)-->.

Path2: (Request for register items)-->(Request for user&pass)-->(Enter user &pass)-->(Check DB to validate user)-->(Valid user)-->
       (Request for delivery type)-->(Select delivery type)-->(Register temporary)-->(Request for payment)-->(bank account info+sell info)-->
       (Valid bank account)-->(Enough Credit)-->(Finalize request)-->(Show payment+billing info)-->.

Path3: (Request for register items)-->(Request for user&pass)-->(Enter user &pass)-->(Check DB to validate user)-->(Valid user)-->
       (Request for delivery type)-->(Select delivery type)-->(Register temporary)-->(Request for payment)-->(bank account info+sell info)-->
       (Invalid bank account)-->(Bank retry)-->(Invalid bank account)-->(Bank retry)-->(Valid bank account)-->(Not enough credit)-->
       (Bank retry)-->(Invalid bank account)-->(Bank retry)-->(Valid bank account)-->(Not enough credit)-->(Bank retry)-->
       (Invalid bank account)-->(Bank retry)-->(Invalid bank account)-->(Bank retry)-->(Invalid bank account)-->(Bank retry)-->
       (Valid bank account)-->(Enough Credit)-->(Finalize request)-->(Show payment+billing info)-->.
```

Fig. 14. Output of tracking execution paths along the net after three executions

Monitor 1: **"**User is allowed to continue purchase proceedings only after username and password are correctly entered on the website." To activate this monitor, it must be applied on the right transition. This monitor is defined on waiting_for_all_places transition in the subpage "Parallel".



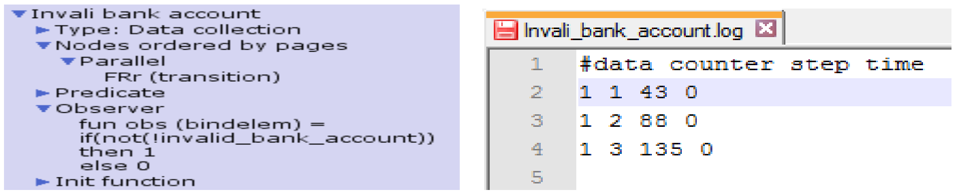
```
#data counter step time
1 1 34 0
1 2 74 0
1 3 102 0
```

```
▼Invalid user
  ►Type: Data collection
  ▼Nodes ordered by pages
    ▼Parallel
       Waiting_for_all_places (transition)
  ►Predicate
  ▼Observer
     fun obs (bindelem) =
     if(not(!invalid_user_account))
     then 1
     else 0
  ►Init function
  ►Stop
```

The output indicates that in none of the three paths taken in order of the occurrence of activities has the monitor's output been zero, which means the design of this part of the enterprise architecture and its relevant activity diagram are accurate in terms of behavior correctness.

Monitor 2:

"User cannot continue the proceedings if they entered the wrong bank account information in the payment step", shows the details for this monitor. This monitor is defined on the transition FRr in the subpage "Parallel".

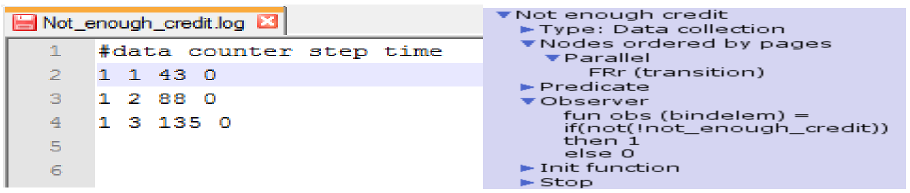


```
▼Invali bank account
  ►Type: Data collection
  ▼Nodes ordered by pages
    ▼Parallel
       FRr (transition)
  ►Predicate
  ▼Observer
     fun obs (bindelem) =
     if(not(!invalid_bank_account))
     then 1
     else 0
  ►Init function
```

```
#data counter step time
1 1 43 0
1 2 88 0
1 3 135 0
```

The output indicates that in none of the three paths taken has the monitor's output been zero, which means , which means the design of this part of the enterprise architecture and its relevant activity diagram are accurate in terms of behavior correctness.

Monitor 3: User's request is finalized only when the total cost of purchase is less than or equal to the balance of the credit card, whose information were entered by user in the previous step. Like Monitor 2, this monitor is defined on transition FRr on subpage "Parallel".



```
#data counter step time
1 1 43 0
1 2 88 0
1 3 135 0
```

```
▼Not enough credit
  ►Type: Data collection
  ▼Nodes ordered by pages
    ▼Parallel
       FRr (transition)
  ►Predicate
  ▼Observer
     fun obs (bindelem) =
     if(not(!not_enough_credit))
     then 1
     else 0
  ►Init function
  ►Stop
```

The output indicates that in none of the three paths taken in order of the occurrence of activities has the monitor's output been zero, which means the design of this part of the enterprise architecture and its relevant activity diagram are accurate in terms of behavior correctness.

6. Conclusion

In this paper, an attempt was made to convert a plan for enterprise architecture in an enterprise into an executable (simulated) model with the help of Petri Nets, before the plan is executed and implemented. In order to describe enterprise architecture, productions parameters in UML activity diagrams were used. This way, based on the proposed solution, elements and structures in the activity diagram were converted to Petri Net elements. The conversion of the activity diagrams to Petri Nets has allowed us to execute them. Petri Nets allow repeated execution of a process and examination of all the different conditions regarding the sequence and combination of various components in that process. Monitors are a practical tool in Petri Nets, which allow examination of different variables. By using monitors and initializing variables, conditions which undermine the accuracy of behavior can be identified and corrected.

References

Azgomi, M.A., Kamandi, A., Movaghar, A. (2004). Modelling and evaluation of software systems with object stochastic activity Nets. *International conference on software engineering advances*, 58-64.

C4ISR Architecture Working Group (AWG) (1997). C4ISR Architecture Framework Version 2.0, 72-74.

Emadi, S., & Shams, F. (2009). A new executable model for software architecture based on petri net. *Indian Journal of Science and Technology*, *2*(9), 15-25.

Iacob, M. E., & Jonkers, H. (2006). Quantitative analysis of enterprise architectures. In *Interoperability of Enterprise Software and Applications* (pp. 239-252). Springer London.

Jensen, K. (1994). Coloured Petri Nets. Springer-Verlag.

Mozaffari, M., Harounabadi, A., & Mirabedini, S.J. (2011). A method for validating the behavior of enterprise architecture. *World Applied Science Journal*, 14(6), 831-841.

Rezai, R. (2006). Presenting a method to evaluate enterprise architecture. *MS thesis, Science and research branch of Islamic Azad University*.

Raouf, R. (2009). Assessment and analysis of enterprise architecture. *Doctoral dissertation, Shiraz University*.

Shin, M. E., Levis, A. H., & Wagenhals, L. W. (2003, October). Transformation of UML-based system model to design/CPN model for validating system behavior. In *Proc. of the 6th Int. Conf. on the UML/Workshop on Compositional Verification of the UML Models*.

Saldhana, J., & Shatz, S. M. (2000, July). Uml diagrams to object petri net models: An approach for modeling and analysis. In *International Conference on Software Engineering and Knowledge Engineering* (pp. 103-110).

Technical committee of IT architecture, (2009). Acquaintance with frameworks of enterprise architecture. Development and Application of ICT (TKFA) monthly magazine, 2(3), 83-88.