World Applied Sciences Journal 14 (6): 831-841, 2011
ISSN 1818-4952
© IDOSI Publications, 2011

A Method for Validating the Behavior of Enterprise Architecture

[1]*M. Mozaffari*, [2]*A. Harounabadi and* [2]*S.J. Mirabedini*

[1]Department of Computer, Arak Branch, Islamic Azad University, Arak, Iran
[2]Department of Computer, Central Tehran Branch, Islamic Azad University, Tehran, Iran

Abstract: Enterprise architecture process includes Enterprise architecture transition from current state architecture into target state architecture.This process contains three phases namely information technology strategic planning, enterprise architecture planning and execution of the enterprise architecture. In this process, each phase is pre-requisite for another one. Any errors in each phase can lead to an error in the whole architecture, so it cause a great cost in time and economy to the enterprise. This is very important in the second phase (enterprise architecture planning). If executable model of enterprise architecture products is created in the second phase and this model is used to validating the behavior and evaluating non-functional requirement of enterprise architecture, any possible errors during execution phase can be avoided. In this paper, we use formal models and create an executable model from enterprise architecture products using Colored Petri Nets that is used for validating the behavior of the architecture. Then we propose a method for validation of created model.

Key words: Enterprise architecture · Executable model · Validating the behavior · Colored Petri Nets

INTRODUCTION

In order to recognize and manage the chaotic nature of enterprise-wide IT system in the real world, the discipline of enterprise architecture (EA) has been emerged since a few years ago. In general, EA provide a knowledge base and supports for making proper decisions on the overarching IT related issues within the underlying enterprise. Enterprise architecture is a comprehensive description of an enterprise contains a huge collection of models, diagrams and documents as it is needed for applying in the enterprise's business. Such a huge jungle of models may not be organized without the proper use of a logical structure. Framework presents a logical structure for categorizing architecture blueprints. So, framework is the most important concept in the enterprise architecture and enterprise architecture depends on it to achieve its goals. Many frameworks have been proposed for the enterprise architecture. Among them, C4ISR enterprise architecture framework is a suitable and comprehensive framework for military organizations that has been published by Department of Defense (DoD).

Enterprise architecture process contains three phases namely information technology strategic planning, enterprise architecture planning and execution of the enterprise architecture. In this process, each phase is pre-requisite for another one. Any errors in each phase can lead to an error in the whole architecture, so it cause a great cost in time and economy to the enterprise. This is very important in the second phase (EA planning). If executable model of EA products is created in the second phase and this model is used to validating the behavior and evaluating non-functional requirement of enterprise architecture, any possible errors during execution phase can be avoided.

During the recent years, many techniques have been proposed for Enterprise Architecture Assessment [1-10]. [1, 2] describe the main line of a methodology / approach in use by several organizations to review the activities and results of enterprise architects. [1] is version 2.1 of this approach and will be continuously refined based on practical experience. [2] is version 2.2 of this approach and will be continuously refined based on practical experience. [3] presents an approach for quantitative analysis of layered, service-based enterprise architecture models, which consists of two phases: a top-down propagation of workload parameters and a bottom-up propagation of performance or cost measures. [4] provides a general description of an architecting process based on object orientation and UML and It then provides a rationale for style constraints on the use of UML artifacts for

Corresponding Author: M. Mozaffari, Department of Computer, Arak Branch, Islamic Azad University, Arak, Iran.

representing DoD Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance (C4ISR) architectures. Finally it describes a mapping between the UML artifacts and an executable model based on colored Petri nets that can be used for logical, behavioral and performance evaluation of the architecture.[5] proposes an integrated process for developing data architecture views in Zachman framework and [6] presents a formal language based on Petri nets in order to obtain verifiable models for all cells in Zachman framework. The presented method helps developers to validate and verify completely integrated business and IT systems which results in improve the effectiveness or efficiency of the enterprise itself. [7] suggests a meta model derived specifically with a set of theory-based system quality analyses in mind. [8] introduces mentioned about determine EA qualification and its qualitative characteristics more clearly that can be used as a reference to investigate EA qualification and its models. [9] applies some kinds of views of UML to describe the relative C4ISR (Command, Control, Communication, Computer, Intelligence, Surveillance and Reconnaissance) architecture products. Considering the relationship among the products, the architecture products modeling in UML diagram is transformed to the executable OPN (Object-based Petri Nets) models based on the transformation rules. It shows the process of OPN mechanisms application for the C4ISR system design validation. [10] provides a conceptual framework for information security management in such convergent environment among federated and related organizations.

In this paper, we want to present a method for validating the behavior of C4ISR Enterprise Architecture products. For this purpose, we use formal models and create an executable model from enterprise architecture products using Colored Petri Nets that is used for validating the behavior of the architecture. Since C4ISR enterprise architecture framework use UML (Unified Modeling Language) for modeling and EA products can be created by UML, so at first We describe transformation algorithm which transforms one type of UML diagrams: Sequence Diagrams, used in the architecture, into Colored Petri Nets and create an executable model base on Colored Petri Nets. Then we propose a method for validation of created model. In The proposed method we use the sequence diagram elements such as: messages, send/receive events and source/destination of messages and write properties in terms of boolean expression over the elements. Finally, we check if created model provides properties correctly. In this study, we use CPN Tools to simulate the execution and validation of architecture. CPN Tools is a CASE Tool for editing, simulating and analyzing Colored Petri Nets. It uses the CPN ML language for declarations and net inscriptions.

Definitions And Primary Concepts

C4isr Architecture Framework Version 2.0: The C4ISR Architecture Framework document issued by the Department of Defense specifies four views of an information architecture and defines a set of products that describe each view. These architecture views are to serve as the basis for C4ISR system development and acquisition. The four views are All View, Operational Architecture View, Systems Architecture View and the Technical Architecture View. Each view describes a particular characterization of the architecture using a set of products that are graphical, tabular, or textual. The *All View* is comprised of two products. The Overview and Summary Information Product, AV-1, is like an executive summary and it contains summary textual information that will allow quick reference and comparison among architectures. This information includes the name of the architecture and the architect, its purpose, scope and context. It also describes major findings and recommendation that are based on the architecture. The Integrated Dictionary is AV-2 [4]. The *Operational Architecture View* is a description of the tasks and activities, operational elements and information flows required to accomplish or support a military operation. It describes the architecture from the operators' conceptual viewpoint. Thus it does not reflect systems or technology, but rather operational activities that are performed at notional operational nodes and the interchanges of messages and data that take place between those nodes. It is composed of seven products, OV-1 through OV-7 [4]. While the OV products focused on the operators' perspective of the architecture the products of the *System Architecture View* focus on the systems and their interconnections and interfaces that will enable operators to carry out their mission. Thus, the System Architecture View is a description, including graphics, of systems and interconnections providing for, or supporting, warfighting functions. It is composed of 11 products, SV-1 through SV-11. Each product highlights a different aspect of the architecture from the systems perspective. The *Technical Architecture View* conveys the set of rules that governs system implementation. This view is composed of two products: The Technical Architecture Profile (TV-1) references the technical standards that apply to the architecture and how they need to be implemented. TV-2 is the Standards Technology Forecast. It extends the information contained in TV-1 to list anticipated updates and changes in applicable standards for the architecture [4].

The Unified Modeling Language: The Unified Modeling Language (UML) [11] is a semi formal language developed by the OMG to specify, visualize and document models of software systems and non-software systems too. UML defines twelve types of diagrams, divided into three categories: static diagrams, behavioral diagrams and diagrams to organize and manage application modules. Behavioral diagrams, i.e., activity, collaboration, sequence, state chart and use case diagrams portray the dynamic behavior of the system. When used to specify a system, each of these diagrams represents a specific aspect of the same system. UML is a rich language that can be used to represent architectures of information systems, including C4ISR systems, using multiple views. Several approaches can be used to generate the executable model. For example, executable models can be derived from various behavioral diagrams (activity diagrams, state chart diagrams, sequence diagrams, etc.) or structural diagrams (class, object, implementation diagram, etc.). Our approach uses the sequence diagram to generate the executable model. Sequence Diagrams are used to represent the life cycle of an object or the sequence of interactions between objects by message passing. They are used to get a better grip of an interaction situation for an individual designer or for a group that needs to achieve a common understanding of the situation [12]. basic elements of sequence diagram are: life lines (the sender and receiver components), messages, connector, send and receive events and a variety of structures like sequence, alternation and option, loop, parallel and etc which are presented as combined fragments. In this diagram, communications are of two kinds: asynchronous, synchronous.in an asynchronous communication, action just ends with sending of massage and the sender isn't waiting for the completion of the action and the receiving of the answer. In a synchronous communication, when the message is sent, sender is waiting for the answer and the completion of the action. Synchronous/asynchronous messages and answers are visualized by arrows with solid and hollow heads, respectively and answers are visualized by dotted arrows.

Petri Nets: Petri Nets was first coined by Carl Adam Petri at early 60's. Here the basic aspects of distributed systems are represented both mathematically and conceptually. Petri Nets is a graphical oriented language for specification, design, simulation and verification of systems.a Petri Net consists of *places*, *transitions* and *arcs* [13]. Places are represented as circle, transition as rectangle and arcs as a directional line connecting from place(s) to transition(s) or vice versa but adding an arc between two places or transitions is not the correct way of representation.

A place can contain tokens. A Petri Net model with different number of tokens in each place is a marking which represents particular instance of the designed system (state of a system).Transitions are active elements that are fired when they are enabled (i.e. when precondition associated with the transition is fulfilled). Firing of transition results in decrement of some tokens from all input places of that transition and increment of some tokens at all output places. The number of tokens added or removed from associated place is based on the expression associated with the arc connected from these places to the firing transition.

Formally Petri Nets can be defined as a 4-tuple
PN = (P, T, F, M0)

Where
P is a finite set of places
T is a finite set of Transitions
$F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs called flow relation
$M0: P \rightarrow \{0, 1, 2, \ldots\}$ is the initial marking
$P \cap T = \emptyset$ and $P \cup T \neq \emptyset$

Colored Petri Nets (CPN) is a high level Petri Nets which describes complex systems in a manageable way [14]. In this model each token is attached with a data type called *color*. A color might be as simple as an *integer* type representing the number of tokens. A complex type can be collection of simpler data types and/or complex data types.

CPN Tools is a CASE tool for editing, simulating and analyzing Colored Petri Nets. The tool features incremental syntax checking and code generation that take place while a net is being constructed. A fast simulator efficiently handles both untimed and timed nets. It uses the CPN ML language for declarations and net inscriptions [15]. In this study, we have used CPN Tools to simulate the execution and verification of architecture.

Transformation Algorithm: The first step for the creation of an executable model is transformation of UML sequence d iagram into Colored Petri Nets.

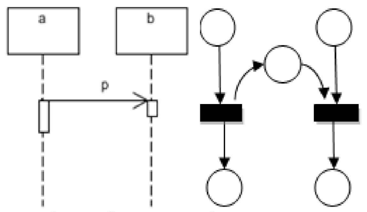


Fig. 1: Mapping of an asynchronous message [16]

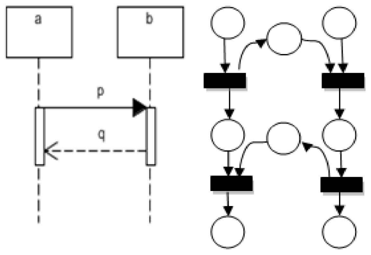


Fig. 2: Mapping of a synchronous message [16]

For this purpose, there are a variety of methods. In the paper we use proposed approach in [16] and [17]. In [16] transformation concentrates on an message sender/receiver object and a variety of messages are transformed into Petri Nets. In [17] a variety of structures in the sequence diagram have been transformed into Colored Petri Nets. In continuation, we explain that.

Translation of Asynchronous Messages into Colored Petri Nets: Figure 1 shows translation of an asynchronous message into Colored Petri Net. Such a communication is made up by means of a shared place that is seeing as an outcome place from the sender object and an income place from the receiver object. The sender and the receiver are represented each one as Place-Transition-Place [16].

Translation of Synchronous Messages into Colored Petri Nets: Figure 2 shows translation of a synchronous message into Colored Petri Net. Such a communication is made up by two shared place that one for the call and the second for the return. the sender and the receiver are represented each one as P-T-P-T-P (Place-Transition-Place-Transition-Place).The centric P of the P-T-P-T-P sequence plays the part of waiting place for the sender and provided method place for the receiver [16]. The second shared place is equivalent to the acknowledge return or result.

Translation of UML 2.0 Combined Fragments into Colored Petri Nets: A variety of structures like sequence, alternation and option, loop, parallel and etc are presented as combined fragments. In continuation, we explain

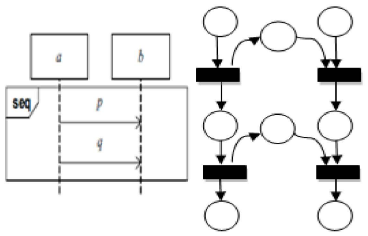


Fig. 3: Weak sequencing operator and Colored Petri Net of its equivalent [17]

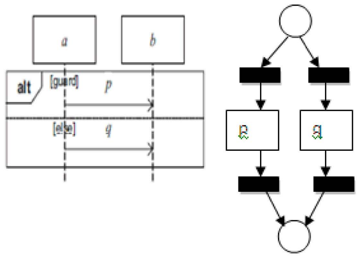


Fig. 4: Alt operator and Colored Petri Net of its equivalent [17]

translation of the most popular combined fragments into Colored Petri Nets.

- Weak sequencing combined fragment: Figure 3 shows translation of weak sequencing combined fragment into Colored Petri Net.
- Alternation and option combined fragments: alternation and option combined fragments represent a choice of behavior in sequence diagrams. Alternative and Option operators are denoted as **alt** and **opt**, respectively. Figure 4 shows alt operator and Colored Petri Net of its equivalent.
- Parallel combined fragments: A parallel combined fragments, denoted by **par** operator, represents a parallel merge between the behaviours of the operands. Figure 5 shows parallel operator and Colored Petri Net of its equivalent.
- Loop combined fragments: The operator **loop** indicates that the combined fragment represents a repetition structure. The loop operand will be repeated a certain number of times according to the values defined by the designer. Figure 6 shows loop operator and Colored Petri Net of its equivalent.
- Break combined fragments: The interaction operator **break** shows a combined fragment representing a breaking scenario. If the guard condition is true, the operand scenario is performed instead of the remainder of the enclosing interaction fragment. Figure 7 shows parallel operator and Colored Petri Net of its equivalent.

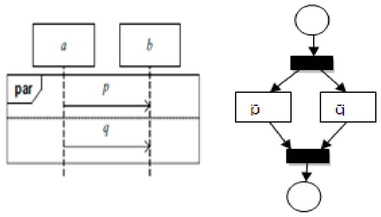


Fig. 5: Parallel operator and Colored Petri Net of its equivalent [17]

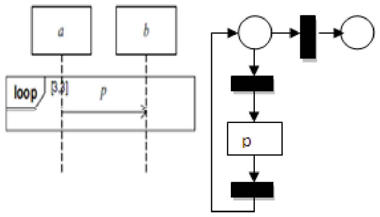


Fig. 6: Loop operator and Colored Petri Net of its equivalent [17]

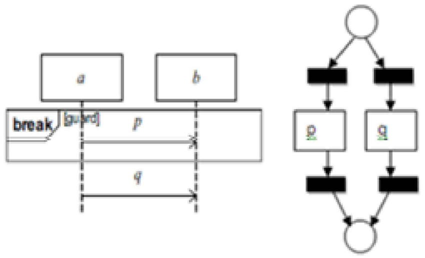


Fig. 7: Break operator and Colored Petri Net of its equivalent

An Executable Model Based on Colored Petri Nets: In this section, we create an executable model base on Colored Petri Nets and propose a method for validation of created model. An executable model from sequence diagram for validating the behavior of the architecture is created as follows:

Step 1: A hierarchical method is used for making of an executable model by Colored Petri Nets. First, we have to consider a substitution transition instead of every combined fragment and create a subpage for every substitution transition.

Step 2: For transforming of the sequence diagram into Colored Petri Net use the presented algorithm in the previous section.

Step 3: For each lifeline (sender/receiver component of message), each send/receive events and each message in sequence diagram, a variable is declared. The values of these variables are updated together with each send/receive event. In Colored Petri Net of its equivalent, in the code segments of transition that do the send/receive events of messages, these variables are taken amount. For example, in the code segment of transitions of figure 8, these variables are taken amount as follows:

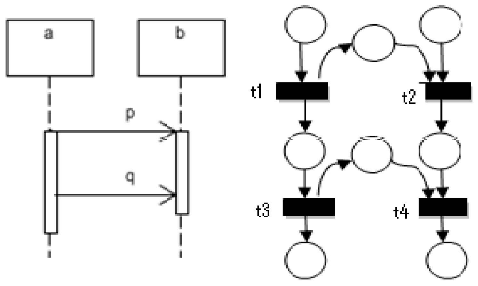


Fig. 8: A sequence diagram and Colored Petri Net of its equivalent

t1 transition:
{send=true, msg_p_s=true, proc1_a=true, proc2_a=false, proc2_b=true}
t2 transition:
{receive=true, msg_p_r =true, proc1_b=true, proc2_b=false, proc2_a=true}
t3 transition:
{send=true, msg_q_s=true, proc1_a=true, proc2_a=false, proc2_b=true}
t4 transition:
{receive=true, msg_q_r=true, proc1_b=true, proc2_b=false, proc2_a:=true}

Step 4: After binding values with variables, properties are written in terms of boolean expression over variables. Below, we present two examples of properties written using variables:

Example 1: Suppose one wants to verify in the sequence diagram from Figure 8 whether "a will not send message *q* until *b* receives message p". The boolean expression corresponding to this property is:
(not X) orelse Y
x= (proc1_a andalso send andalso msg_q_s)
Y= (proc1_b andalso receive andalso msg_p_r)

Example 2: Suppose one wants to verify in the sequence diagram from Figure 9 whether "a will not receive message *q* until *c* receives message p". The Boolean expression corresponding to this property is:
(not X) orelse Y
X= (proc1_a andalso receive andalso msg_q_r)
Y= (proc1_c andalso receive andalso msg_p_r)

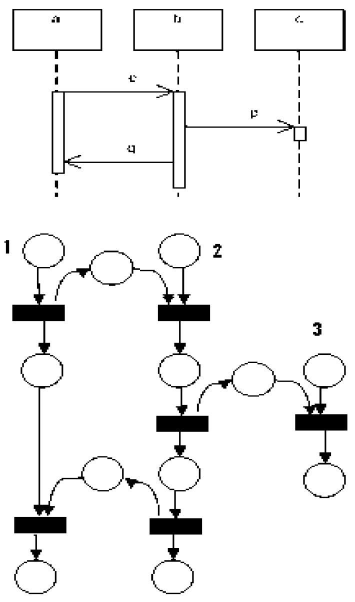


Fig. 9: A sequence diagram and Colored Petri Net of its equivalent

Step 5: Does really the sequence diagram present correctly expressed properties in step 4 or not? We should check each of the properties for the all of the execution paths in sequence diagram. For this purpose, we create tokens in number of lifelines in sequence diagram and lay them at the first place each lifeline (in the Colored Petri Net its equivalent). For example in the Colored Petri Net from Figure 9, three token are created and laid on the places 1, 2 and 3.

In every stage of simulation, one of the executed paths in sequence diagram is surveyed and the results of simulation show if execution paths provide expressed properties correctly or not?

Step 6: There is a mechanism in CPN Tools named monitor that is used to observe, inspect, control, or modify a simulation of a CP-net. Monitors can inspect both the markings of places and the occurring binding elements during a simulation and they can take appropriate actions based on the observations [15]. In order to survey expressed properties, for each property a monitor is written and associated with group of transitions in every subpage. The start/stop functions of the monitor return none and predicate function returns true if and only if one of the transitions occurs. Observation function is called when predicate function returns true. Then observation function check

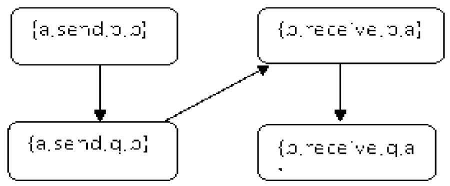
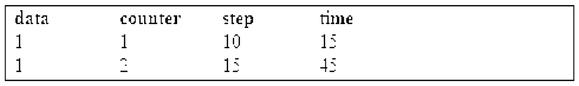


Fig. 10: An execution path of sequence diagram in Fig. 8



```
data        counter     step       time
1           1           10         15
1           2           15         45
```

Fig. 11: Obtained data in a log file

the verification of the property. If its respective boolean expressions is true, observation function returns one and otherwise it returns zero. These values save in a log file.

Executable Model Analysis and Simulation Results: After the executable model is created by Colored Petri Nets, we should assign properties to Colored Petri Net and execute the model and check the Validation of UML sequence diagrams. For example, after assign property 1(example 1) to Colored Petri Net in figure 8 and execute it, the results show that there is at least an execution path on which the property 1 doesn't occur (Figure 10). Figure 11 shows the obtained data in a log file after the execution of the model. Values that is written at the first column is the returned values of the observation function. Because one of the returned values is zero, therefore the sequence diagram in Fig. 8 doesn't provide the property 1 correctly and its validation is refused.

Case Study: In this case study we are going to analyze an Automated Teller Machine (ATM). The ATM interacts with two other entities: The Customer (User) and the bank. Figure 12 describes a use case where the user starts a cash withdrawal request by inserting his/her card. The ATM must verify the card and the personal identification number (PIN) to proceed. If the verification fails the card should be ejected. Otherwise, the user chooses cash withdrawal operations and enters the amount to be withdrawn. The first and second combined fragments are dealing with the authentication of the card and the PIN, respectively. The third checks the account balance.

At first, a hierarchical method is used for making of an executable model by Colored Petri Nets. In the hierarchical method, we have to consider a substitution transition instead of every combined fragment (Figure 13) and create a subpage for every substitution transition(Figure 14, Figure 15, Figure 16, Figure 17). Figure 18 shows Screenshot of the CPN Tools workspace for executable model created by Colored Petri Net.

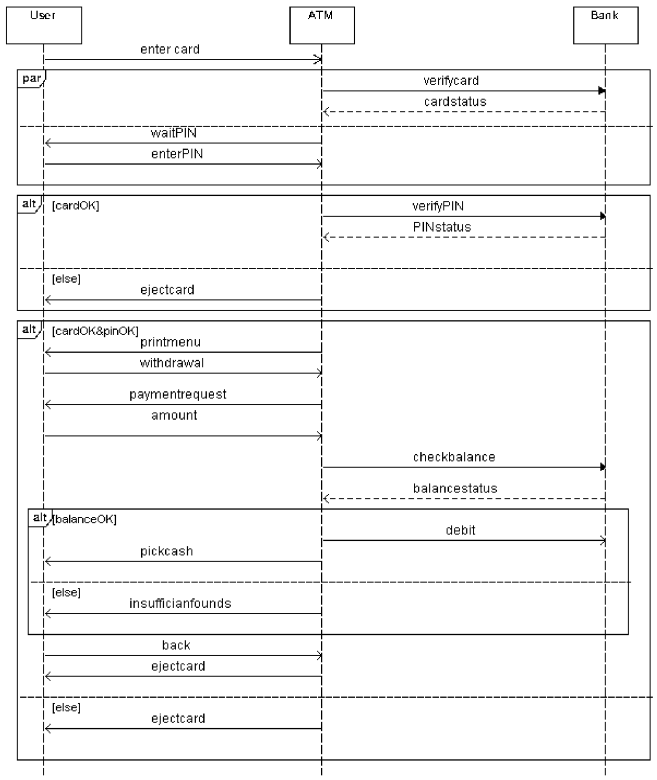


Fig. 12: ATM sequence diagram

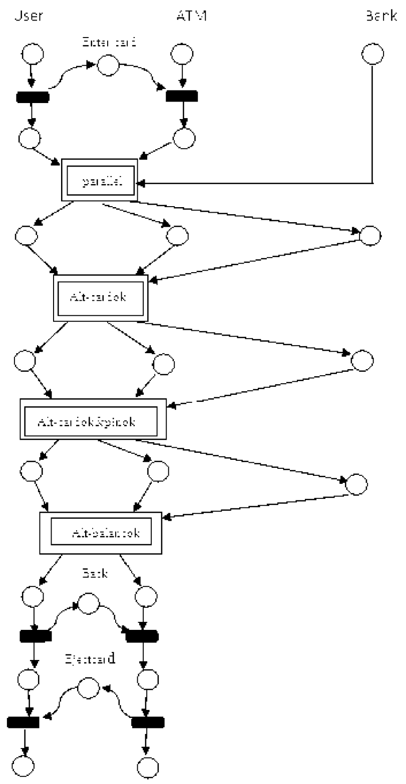


Fig. 13: Hierarchical net for ATM sequence diagram

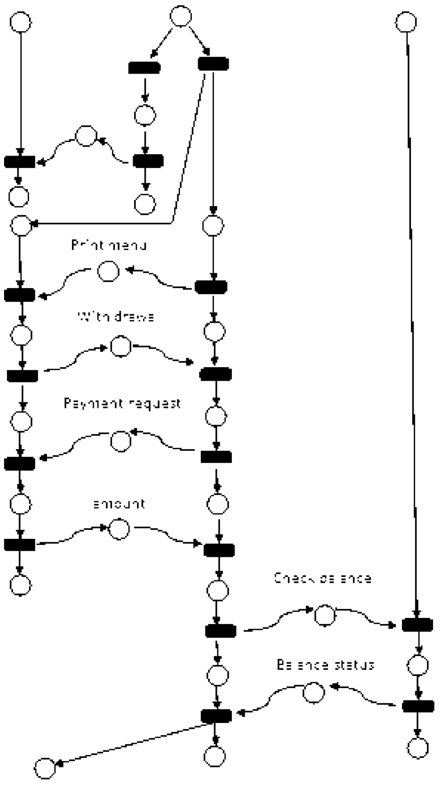


Fig. 14: Substitution transition **cardok&pinok**

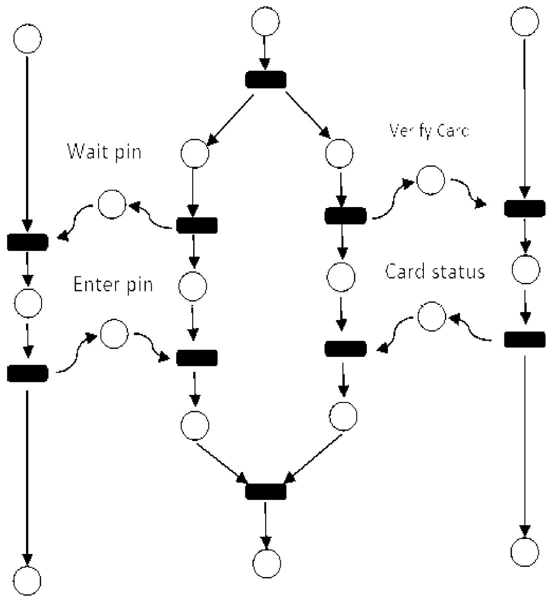


Fig. 15: Substitution transition **parallel**

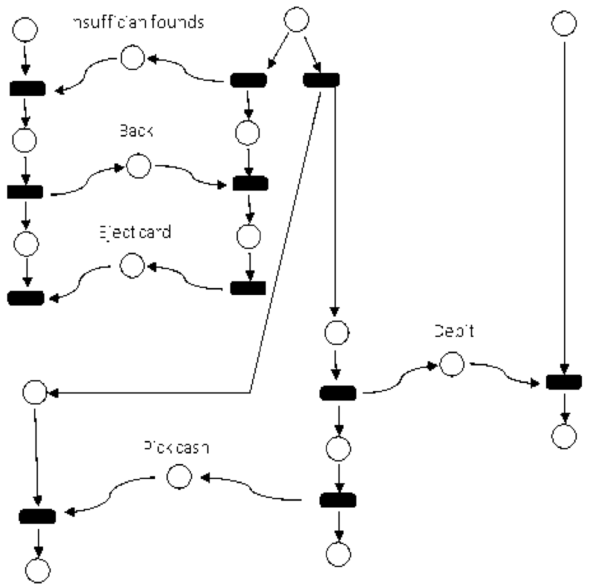


Fig. 17: Substitution transition **balanceok**

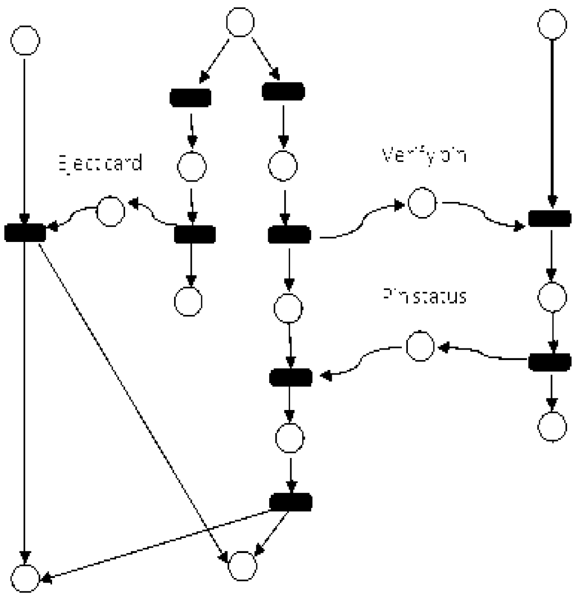


Fig. 16: Substitution transition **cardok**

ML Properties:

- The first property states that the ATM cannot allow the user to request an operation if either the card or the PIN is not valid:

$$(x \rightarrow \text{not } y) = (\text{not } x \text{ orelse not } y)$$

Where x = (not (cardOK) orelse not (pinOK)) and y=(Proc1_User andalso receive andalso msg_printmenu_r)

- The second property is needed to avoid inconsistencies between the money given to the user and the amount debited in the bank. It asserts that the ATM must first debit the amount in the bank and then give the money to the user. In other words, the user does not receive pickCash until the bank receives debit:

$$(\text{not } x \text{ orelse } y)$$

where x= (proc1_User andalso receive andalso msg_pickcash_r) and
y= (proc1_Bank andalso receive andalso msg_debit_r)

- The third property is to ensure the correct end of the session between the ATM and the user. It says that, after the user receives ejectCard, the ATM cannot send anything to the user:

$$(x \rightarrow \text{not } y) = (\text{not } x \text{ orelse not } y)$$

where x= (proc1_User andalso receive andalso msg_ejectcard_r) and
y= (proc1_ATM andalso send andalso proc2_User)

ATM Case Study Results: After the executable model is created by Colored Petri Nets, we should assign properties to Colored Petri Net and execute the model and check the Validation of UML sequence diagrams. Using CPN Tools to verify the properties described previously,

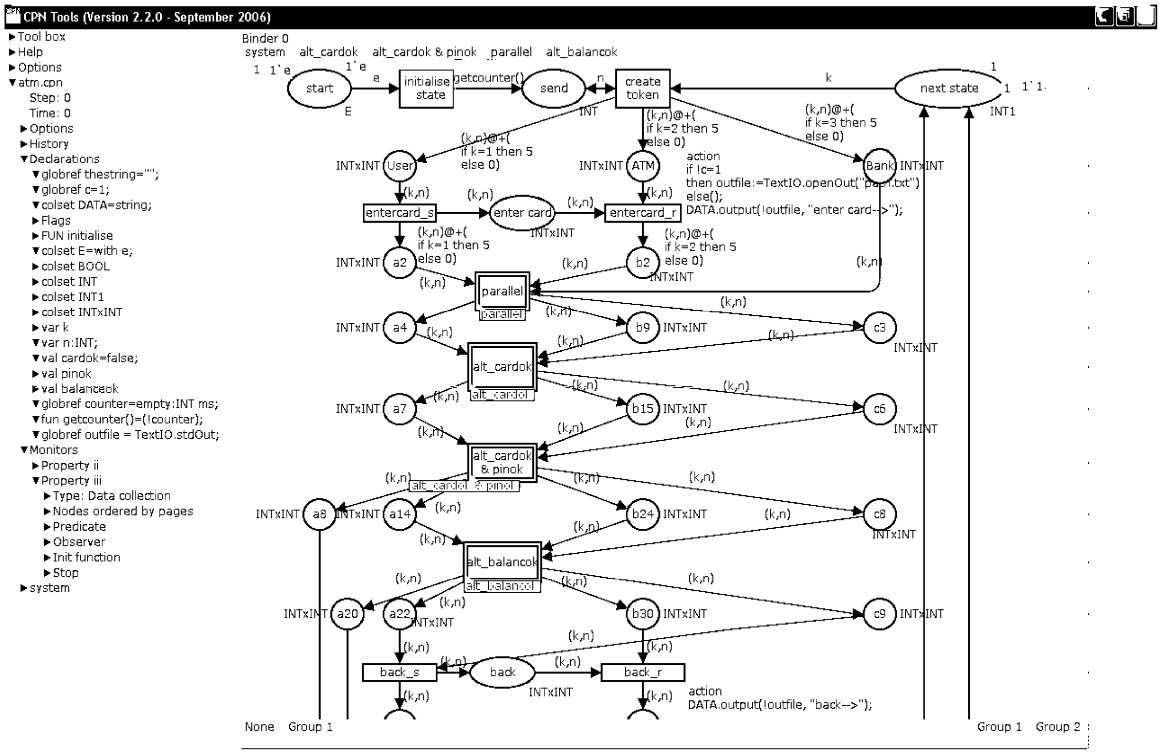
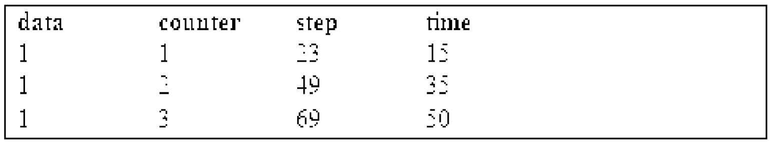
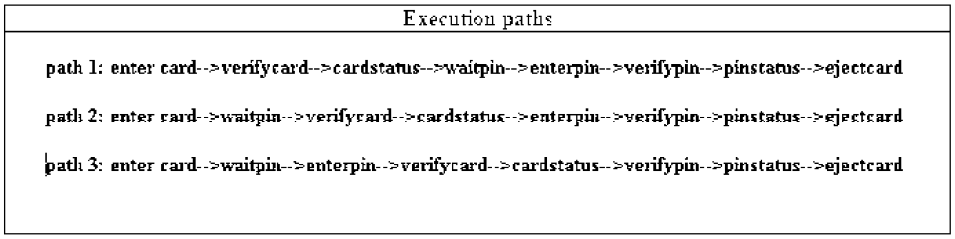

```
CPN Tools (Version 2.2.0 - September 2006)
▸Tool box
▸Help
▸Options
▾atm.cpn
   Step: 0
   Time: 0
  ▸Options
  ▸History
  ▾Declarations
    ▾globref thestring="";
    ▾globref c=1;
    ▾colset DATA=string;
    ▸Flags
    ▸FUN initialise
    ▾colset E=with e;
    ▸colset BOOL
    ▸colset INT
    ▸colset INT1
    ▸colset INTxINT
    ▸var k
    ▾var n:INT;
    ▾val cardok=false;
    ▸val pinok
    ▸val balanceok
    ▾globref counter=empty:INT ms;
    ▾fun getcounter()=(!counter);
    ▾globref outfile = TextIO.stdOut;
  ▾Monitors
    ▸Property ii
    ▾Property iii
      ▸Type: Data collection
      ▸Nodes ordered by pages
      ▸Predicate
      ▸Observer
      ▸Init function
      ▸Stop
  ▸system
```
Binder 0
system alt_cardok alt_cardok & pinok parallel alt_balancok
start
initialise state
getcounter()
send
create token
next state
User
ATM
Bank
entercard_s
enter card
entercard_r
a2
b2
parallel
a4
b9
c3
alt_cardok
a7
b15
c6
alt_cardok & pinok
a8
a14
b24
c8
alt_balancok
a20
a22
b30
c9
back_s
back
back_r
(k,n)@+(if k=1 then 5 else 0)
(k,n)@+(if k=2 then 5 else 0)
(k,n)@+(if k=3 then 5 else 0)
action if !c=1 then outfile:=TextIO.openOut("path.txt") else(); DATA.output(!outfile, "enter card-->");
action DATA.output(!outfile, "back-->");
INTxINT
INT1
None Group 1
Group 1 Group 2

Fig. 18: Screenshot of the CPN tools workspace



data	counter	step	time
1	1	23	15
1	2	49	35
1	3	69	50

Fig. 19: Obtained data in a log file of property i



Execution paths

path 1: enter card-->verifycard-->cardstatus-->waitpin-->enterpin-->verifypin-->pinstatus-->ejectcard

path 2: enter card-->waitpin-->verifycard-->cardstatus-->enterpin-->verifypin-->pinstatus-->ejectcard

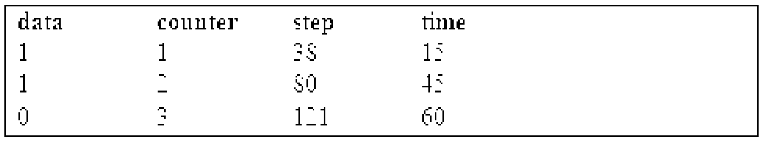
path 3: enter card-->waitpin-->enterpin-->verifycard-->cardstatus-->verifypin-->pinstatus-->ejectcard

Fig. 20: Execution paths of sequence diagram in fig. 12 after the execution of model for property i

we found that only the first property is satisfied. Figure 19 shows the obtained data in a log file and figure 20 shows execution paths of sequence diagram in figure 12 after the execution of the model for property **i**. In the following, we present the failing trace related to the verification of each property:

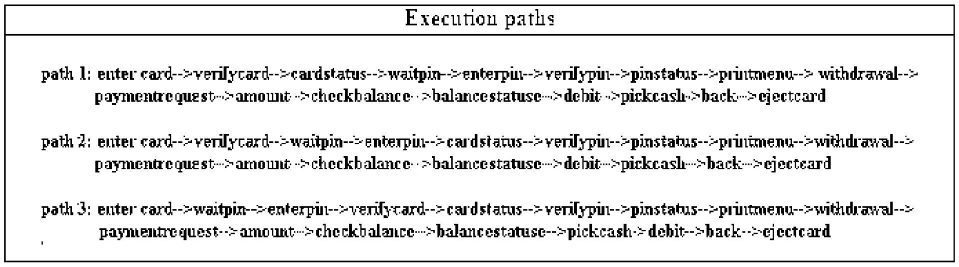
Property: In the results shown in Figure 21, it possible to see that there is at least an execution path on which the property **ii** doesn't occur. Values that is written at the first column is the returned values of the observation function. Because one of the returned values is zero therefore the sequence diagram in Fig. 12 doesn't provide the property **ii** correctly and its verification is refused. Figure 22 shows execution paths of sequence diagram in Figure 12 after the execution of the model for property **ii.**

Property iii: In Figure 23, at least one of the returned values is zero (execution path on which the ATM tries to eject the card twice), therefore



data	counter	step	time
1	1	38	15
1	2	80	45
0	3	121	60

Fig. 21: Obtained data in a log file of property ii



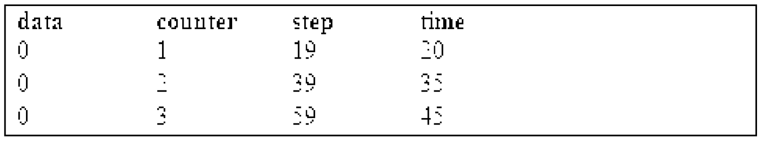
Execution paths

path 1: enter card-->verifycard-->cardstatus-->waitpin-->enterpin-->verifypin-->pinstatus-->printmenu--> withdrawal-->paymentrequest-->amount-->checkbalance-->balancestatuse-->debit-->pickcash->back-->ejectcard

path 2: enter card-->verifycard-->waitpin-->enterpin-->cardstatus-->verifypin-->pinstatus-->printmenu-->withdrawal-->paymentrequest-->amount-->checkbalance-->balancestatuse-->debit-->pickcash-->back-->ejectcard

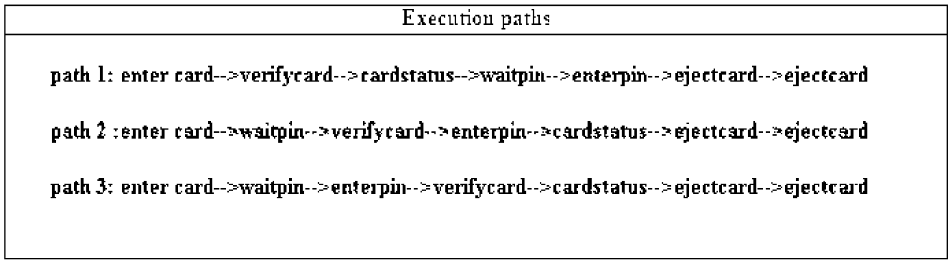
path 3: enter card-->waitpin-->enterpin-->verifycard-->cardstatus-->verifypin-->pinstatus-->printmenu-->withdrawal-->paymentrequest-->amount-->checkbalance-->balancestatuse-->pickcash->debit-->back-->ejectcard

Fig. 22: Execution paths of sequence diagram in Fig. 12 after the execution of the model for property ii.



data	counter	step	time
0	1	19	20
0	2	39	35
0	3	59	45

Fig. 23: Obtained data in a log file of property iii



Execution paths

path 1: enter card-->verifycard-->cardstatus-->waitpin-->enterpin-->ejectcard-->ejectcard

path 2 :enter card-->waitpin-->verifycard-->enterpin-->cardstatus-->ejectcard-->ejectcard

path 3: enter card-->waitpin-->enterpin-->verifycard-->cardstatus-->ejectcard-->ejectcard

Fig. 24: Execution paths of sequence diagram in Fig. 12 after the execution of the model for property iii

the sequence diagram in Figure 12 doesn't provide the property **iii** correctly and its verification is refused. Figure 24 shows execution paths of sequence diagram in Fig. 12 after the execution of the model for property **iii**.

CONCLUSION

In this paper, we present a method for validating the behavior of C4ISR Enterprise Architecture products. For this purpose, we use formal models and create an executable model from enterprise architecture products using Colored Petri Nets that be used for validating the behavior of the architecture. Since it takes into account the most popular UML combined fragments, this approach allows the developer to detect flaw in more completed complex sequence diagrams.

The mechanism introduced in this work to keep track of the execution state provides the information the developer needs to write ML properties. Moreover, the way it was implemented gives flexibility to write very expressive properties. This technique can provide a very useful framework to detect errors at the planning phase, as a result, the proposed method causes increasing validation and verification of behavior of system.

REFERENCES

1. Schekkerman, J., 2004. Enterprise Architecture Score Card Version 2.1, Institute For Enterprise Architecture Developments, The Netherlands.
2. Schekkerman, J., 2006. Enterprise Architecture Assessment Guide Version 2.2, Institute For Enterprise Architecture Developments, The Netherlands.

3. Locob, M.E. and H. Jonkers, 2005. Quantitative Analysis of Enterprise Architectures. In the Proceedings of the First International Conference on Interoperability of Enterprise Software and Applications (INTEROP-ESA'2005), Geneva, Switzerland, pp: 234-248.
4. Wagenhals, L.W., S. Haider and A.H. Levis, 2003. Synthesizing Executable Models of Object Oriented Architectures. Journal of Systems Engineering, Vol. 6, No.4, pp. 266-300.
5. Rezaei, R. and F. Shams, 2008. A Methodology to Create Data Architecture in Zachman Framework. World Applied Sci. J., 3(2): 43-49.
6. Ostadzadeh, sh., M.A Nekoui, 2009. A Petri-Nets Based Unified Modeling Approach for Zachman Framework Cells. In the Proceedings of SCSS'2009, pp:615-618.
7. Närman, P., P. Johnson and L. Nordström, 2007. Enterprise Architecture: A Framework Supporting System Quality Analysis. In Proceedings of the 11th International EDOC Conference, pp: 130-141.
8. Khayami, R., A. Towhidi and K. Ziarati, 2011. Evaluating Quality Characteristics of Enterprise Architecture. In the Proceedings of World Conference on Information Technology, 3: 1277-1282.
9. Hui Bai, X., 2008. An application with UML Object-based Petri Nets for C4ISR architecture simulation validation, In the Proceedings of International Conference on Machine Learning and Cybernetics, Kunming, pp: 2257-2263.
10. Elahi, S., A. Shayan and B. Abdi, 2008. Designing a Framework for Convergent Information Security Management among Federated Organizations. World Appl. Sci. J., 4(2): 21-32.
11. Booch, G., J. Rumbaugh and I. Jacobson, 1999. The Unified Modeling Language User Guide, Addison Wesley, Reading MA.
12. Haugen, Ø. and K. Stølen, 2003. STAIRS - steps to analyze interactions with refinement semantics, in: UML 2003 - The Unified Modeling Language. Model Languages and Applications. In the Proceedings of 6th International Conference, San Francisco, CA, USA, pp: 388-402.
13. Peterson, J.L., 1981. Petri Net Theory and the Modelling of Systems, Prentice-Hall, Englewood Cliffs, NJ.
14. Kristensen L.M., S. Christensen and K. Jensen, 1998. The Practitioner's Guide to Coloured Petri Nets. International Journal on Software Tools for Technology Transfer, Springer Verlag, pp: 98-132.
15. CPN tools help, http://wiki.daimi.au.dk/cpntool shelp/ cpntoolshelp.wiki.
16. Ourdani, A., P. Esteban, M. Paludetto and J.C. Pascal, 2006. A Meta Modeling Approach for Sequence Diagram to Petri Nets Transformation Within the Requirements Validation Process. In the Proceedings of the European Simulation and Modeling Confrence, Toulouse, France, pp: 345-349.
17. Emadi, S. and F. Shams, 2009. A new executable model for software architecture based on Petri Net. Indian J. Sci. and Technol., 2(9): 15-25.