Journal of Communication and Computer 12 (2015) 184-190
doi: 10.17265/1548-7709/2015.04.004

Towards a New Framework for Building a Whole User-Defined System from a Colored Petri Networks

Zeddari Abderrazzak and Ettalbi Ahmed

1. Models and Systems Engineering Team, Mobile and Embedded Information Systems Laboratory, National Higher School of Computer Science and Systems Analysis, Mohammed-V University, PB 713, Rabat , Morocco

2. Models and Systems Engineering Team, Mobile and Embedded Information Systems Laboratory, Software Engineering Department, National High School of Computer Science and Systems Analysis, Mohammed-V University, PB 713, Rabat , Morocco

Abstract: In this paper we provide a discussion of designing a new Framework for building a whole system from a colored Petri Network. This framework will integrate the translation process of a multiview class to a colored Petri network developed in in our last research. With two examples, we will extend this process by using fields, methods and access control. The advantages of using colored Petri networks compared to ordinary ones are also discussed as well as our new framework design.

Key words: UML, view and viewpoint, modelling, Petri networks, colored Petri nets, validation rules.

1. Introduction

The concept of a multiview system comes with model that can design solutions to problems based on the needs and skills of users, not otherwise. This will help the end-users to be more effective and learn the new system faster, because the used tools are based on how they do their work. This new approach allows engineers and clients to implement new kinds of design methods and techniques that adapt to the specific needs of the workers.

The multiview system concept offers many features such as the multiview class [1], whose goal is to store and deliver information according to users' viewpoint. It supports the dynamic change of viewpoint and offers mechanism to describe view dependencies.

In the next sections of this paper, we will discuss the following points that allow us to add more features to our approach developed in Ref. [2]:

- Including methods with the access right of Read/Update in the translation process;
- Discussing the Petri Nets properties that are used during the analysis and validation phases;
- Discussing our new Framework which allows generating a whole system from the obtained colored Petri Networks.

Corresponding author: *Ahmed Ettalbi*, professor, research fields: web services, software architecture, business modeling, models and services, cloud computing. E-mail: ettalbi1000@gmail.com.

2. CPN Translation Appraoch

2.1 Multiview Class Diagram

First, we begin by defining a class diagram and its component. In Ref. [3], a class diagram denotes a set of objects with common features. A class is a simple rectangle divided into three parts. The first part contains the name of the class, which has to be unique in the whole diagram. The second part contains the attributes of the class, each denoted by a name, possibly followed by the multiplicity, and with an associated type for the attribute values. The third part contains the operations of the class, namely, the methods associated to the objects of the class. The second and the third part are optional.

2.2 Approach

In Ref. [2], we defined an approach of translating a

multi-view class to a colored Petri Networks based on three steps. In this paper, we will develop more this procedure. We will translate a whole class containing fields and methods. Also, we will focus on the dynamic behavior of this process by implementing the access control (Read/Update) and Authorization on our new Petri networks.

Step 1: Defining Domain Color and Places

At every viewpoint, we associate a colored token and two places:

- VPDA: deactivated state of Viewpoint I;
- VPAC: activated state of Viewpoint I.

At each view, we associate two colored tokens:

- Vread corresponds to a View in the read state;
- Vwrite corresponds to a View in the write state.

Step 2: Arcs and Functions

At each arc, we associate a function to determine instances of tokens (views) necessary, activated and deactivated in crossing a transition.

Step 3: TRANSITIONS

At each ViewPoint, we associate two transitions:

- AVP: the event Activate Viewpoint I;
- DVP: the event Deactivate Viewpoint i.

For access controls, we define a transition:

- AUTH: the Authorization event check.

INITIAL STATE

Initially, all viewpoints are deactivated. Place VPDA contains all colors; the other places do not contain any color.

Below, we apply our approach on two examples. The first one is used in Ref. [1] and corresponds to a multiview class car while the second example corresponds to a multiview class Software.

2.3 Application Examples

Multiview Class Car

In Ref. [2], an example of modelling a multiview class Car using Colored Petri Network has been developed.

This class includes the following fields:

Ref: String, represent car reference

Brand: String, represent the car brand

Color: String, represent car color

Fuel: String, represent the car fuel

Consumption: Real, represent the consumption of the car

Discount: real, the discount in the price of the car

SellingPrice: real, represent the selling price of the car

Recommended Price: Real, recommended price and the following methods:

ShowInfo():Void, to show car information's.

ModifyInfo():Object Car, return the modified car.

RegisterFailure():Object, return the saved record.

RepairFailure():Object, return the saved record.

AnswerProposal():Object, return the saved record.

OfferPrice(): Object, return the saved record.

This class supports three viewpoints: that of the Client, another associated with the Commercial and the third is related to the Mechanic.

Table 1 shows the fields accessible for each user according to his views.

In Table 2, we present the views of the class Car. Five views can be distinguished: V1, V2, V3, V4 and V5. Each view contains fields. The views are then grouped to give point of views.

In Table 3, we present for each viewpoint, the views composing it.

Vw: View with Write state.

Vr: View with Read state only.

Multiview Class Software

Table 1 Accessible fields for each viewpoint.

Client	***Commercial***	***Mechanic***
Ref	Ref	Ref
brand	brand	brand
Color	color	color
Fuel	fuel	fuel
consumption	consumption	consumption
discount	discount	ShowInfo()
Selling Price	Selling Price	ModifyInfo()
ShowInfo()	Recommended Price	Register Failure()
Offer Price()	ShowInfo()	Repair Failure()
	ModifyInfo()	
	Answer Proposal()	

Table 2 Views related to the class Car.

V1	*V2*	*V3*	*V4*	*V5*
Ref	discount	Recomm_endedPrice	Modify_Info()	Register_Failure()
brand	SellingPrice	Answer_Proposal()		RepairFailure()
color	ShowInfo()			
fuel				
consumption				
ShowInfo()				

Table 3 Composition of viewpoints in terms of views.

VP1: Mechanic	*VP2: Client*	*VP3: Commercial*
V1w+V5w	V1r+V2r	V1w+V2w+V3w+V4w

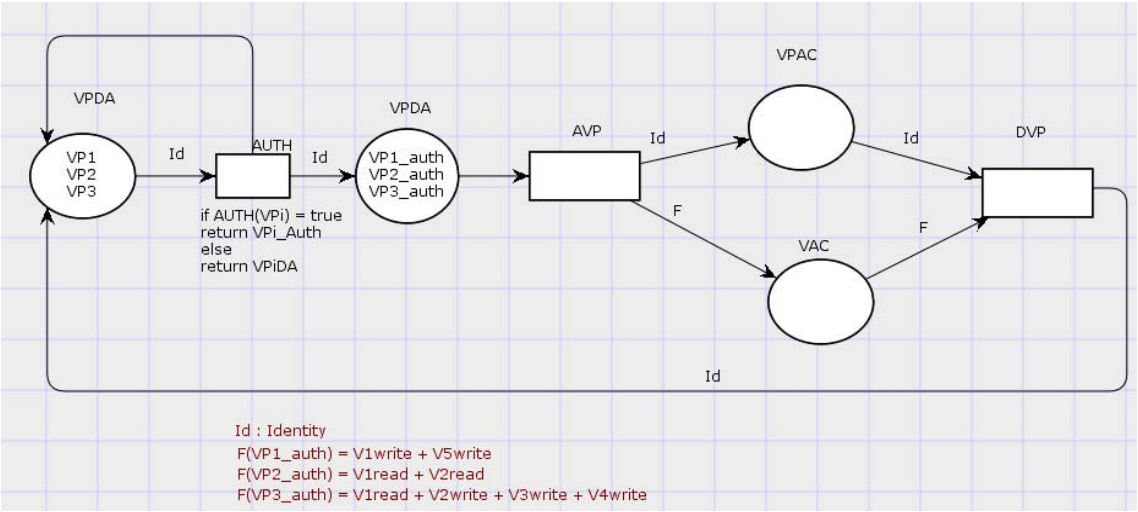

Fig. 1 Colored Petri network associated with the multiview class Car using CPNTool 4.

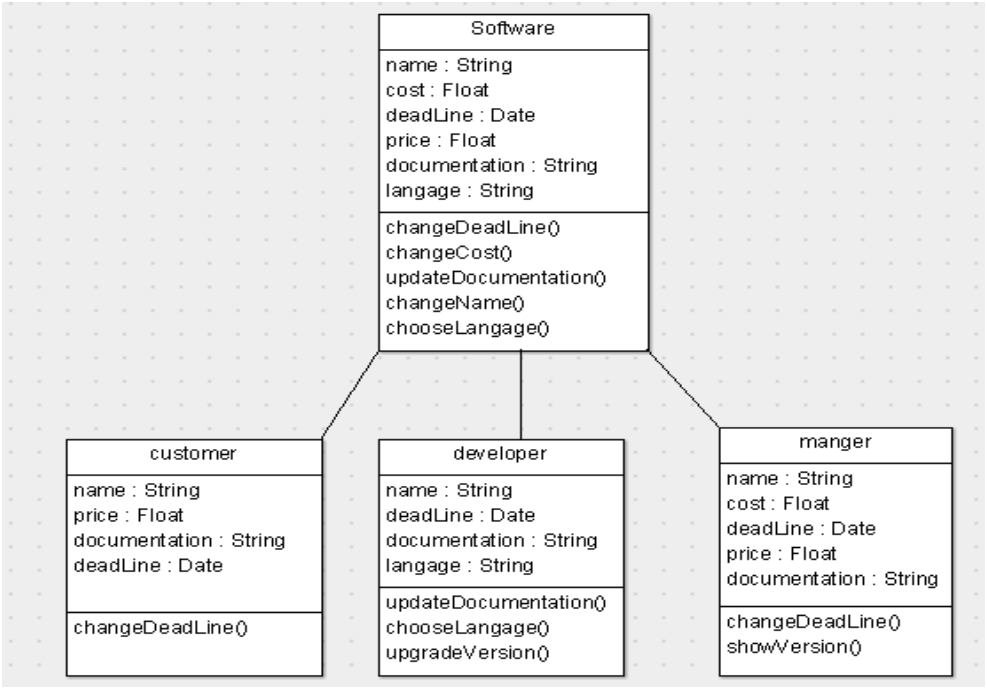

Fig. 2 Multiview class software.

The next figure shows the fields and the method of the class software:

This class supports three viewpoints: that of the customer, another associated with the Developer and the third is related to the Manager.

Table 4 shows the fields accessible for each user according to his views.

In Table 5, we present the views of the class Software. Four views can be distinguished: V1, V2, V3 and V4. Each view contains fields. The views are then grouped to give point of views.

In Table 6, we present for each viewpoint, the views composing it.

Vw: View with Write state.

Vr: View with Read state only.

Table 4 Accessible fields for each viewpoint.

Customer	***Developer***	***Manager***
name	name	name
deadLine	deadLine	deadLine
documentation	documentation	documentation
price	langage	cost
changeDeadLine()	updateDocumentation()	price
	chooseLangage()	changeDeadLine()
		changeCost()

Table 5 Views related to the class software.

V1	***V2***	***V3***	***V4***	***V5***
name	price	langage	cost	Documentation
deadLine	changeDeadLine()	chooseLangage()	changeCost()	
		upgradeVersion()	showVersion()	
		updateDocmentation()		

Table 6 Composition of viewpoints in terms of views.

VP1: Customer	***VP2: Developer***	***VP3: Manager***
V1w+V2w+V5r	V1r+V3w+V5w	V1w+V2w+V4w+V5r

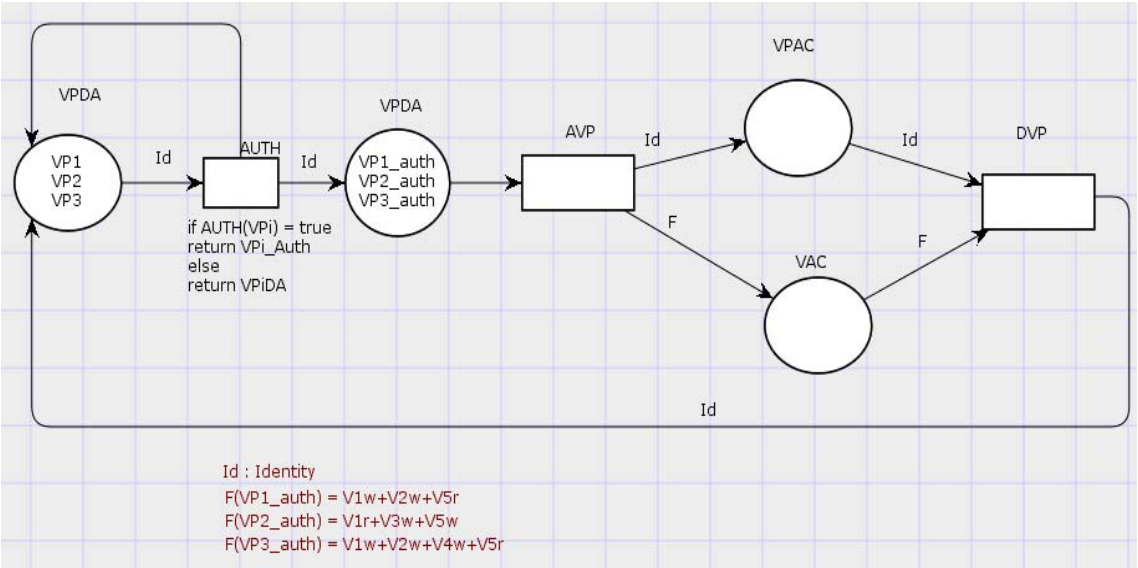

Fig. 3 Colored Petri network associated with the multiview class Software using CPNTool 4.

3. Colored Petri Networks Properties Validation

After translating our system model to a Petri Network, another step is necessary before we can begin its use, it is the properties analysis phase [4] and the problem associated with concurrent systems. Two types of properties can be found with a Petri-net model:

- Properties which depend on the initial marking;
- Properties which are independent of the initial marking.

The first type of properties is called behavioral properties whereas the latter type is called structural properties. In this paper, we will focus on the behavioral ones.

3.1 Behavioral Properties

(1) Reachability

The reachability problem for Petri nets is to decide, given a Petri net N and a marking M, whether, $M \in R(N)$. Clearly, this is a matter of walking the reachability graph, until either we reach the requested marking or we know it can no longer be found. So a marking M_n is said to be reachable from M_o if there exists a sequence of firings that transforms M_o to M_n, the reachability graph is generally infinite, and it is not easy to determine when it is safe to stop.

In fact, this problem was shown to be hard [5] years before it was shown to be decidable at all [6]. Researches continue to find methods to do it efficiently [7].

(2) Liveness

The concept of liveness [4] is closely related to the complete absence of deadlocks in operating systems. A Petri Net (N, M_0) is said to be live (or equivalent M_0 is said to be live marking for N) if, no matter what marking has been reached from M_0, it is possible to ultimately fire any transition of the net by progressing through some further firing sequence.

(3) Boundedness

A place in Petri net is called k-bounded if it does not contain more than k tokens in all reachable markings, including the initial marking; it is said to be safe if it is 1-bounded; it is bounded if it is *k*-bounded for some *k*.

A (marked) Petri net is called k-bounded, safe, or bounded when all of its places are. A Petri net (graph) is called (structurally) bounded if it is bounded for every possible initial marking.

Note that a Petri net is bounded if and only if its reachability graph is finite.

(4) Faireness

In Ref. [4], two different types of fairness have been proposed:

- Bounded-fairness;
- Global-fairness.

In Ref. [8], A PN is fair when the firing of any transition more than a given number of times is a sufficient condition for all the transitions in the net to have fired. When the PN is fair, no process in the system can be stared.

4. The New Framework

4.1 Framework Presentation

In Ref. [9], the MDA (model-driven architecture) is defined as an architecture that provides methods for software development that use models to describe the system to be built. These models provide a description that can be expressed at various levels of abstraction, with each level emphasizing certain viewpoints of the system.

The driving force behind the MDA is the fact that a software system will eventually be deployed to one or more platforms, used separately or together. Platforms are subject to change over time and they change at different, typically higher, rates than the higher-level models of the system, which in turn tend to grow increasingly independent of the target platforms.

Our aim is to design and develop a new framework that will follow the MDA architectures and will allow building a whole system (Web Apps, client Apps, Web Site ...) from a model.

Thus if our model changes the generated system will also change according to user needs.

This framework is within the scope of the agility, exactly in the scope of the rapid application development (RAD, Scrum), and will add more features to these methods exactly in a cloud computing context.

The innovation of this framework can be summarized in two ways:

- Including the notion of viewpoint in this process, to get a multiview system from one model;
- Including a new process of code generation from a Petri network to ensure application on all high level Petri networks not only colored ones.

The design of this framework can be achieved by following defined steps. These steps will be detailed in the next section.

4.2 Framework Design

Our framework can be designed as follow:

1. Analysis and validation of the obtained Petri Nets;
2. Generating the PNML (Petri Network markup language) code;
3. Generating SQL Database script from the generated PNML code;
4. Reverse the relational database to get DAO layer;
5. Generating the whole system.

- Generating the service layer
- Generating the GUI layer

4.2.1 Analysis and validation

This step can be achieved by verifying that the obtained Petri Net complies with the properties defined above (Reachability, Liveness, Boundedness...). Similarly we can use the CPNTools to perform a complete simulation during the analysis and validation phases.

4.2.2 Generating Petri Network Markup Language

The PNML is a proposal of an XML-based interchange format for Petri Nets. Originally, the PNML was intended to serve as a file format for the Java version of the Petri Net Kernel. But, it turned out that currently several other groups are developing an XML-based interchange format too. So, the PNML is only one contribution to the ongoing discussion and to the standardization efforts of an XML-based format.

In Ref. [10], the author cited that the PNML is finally adopted as ISO/IEC 15909-2, it was a result of the annual ‘Petri Net Conference’ in Aarhus.

Once the PNML is defined and standardized, it will be adopted to generate our Petri Network as a list of instruction code that can inform the Sql statements for database creation.

4.2.3 PNML to SQL

As the title above suggests, this step allows getting a valid SQL script to create our database that will be used in the reverse engineering process; this work is still an issue for future research.

4.2.4 Generating the whole system

In this step, we will use a list of standards framework such as:

- Spring Row
- Hiberante
- Eclipse
- Etc …

The advantages of using this standards frameworks is to guarantee that there will be no bugs or probleme in the generated system. Adequate support and documentation is also available.

5. Conclusion and Perspectives

In this paper, we have presented a design for a new Framework, whose aim is to add more features in the Rapid Application development process by including user viewpoint and linking all the system architecture with the original user model. Thus the whole system will be driven by this model.

Our perspective is developing an eclipse plug-in for our framework design. This plug-in will help the user to get the whole generated system in a few minute.

References

[1] Gregut, X., Ebersold, S., Nassaret, M., and Coulette, B.

2005. “Design Pattern for Code Generation in VUML.” ENSEEIHT/Irit, Univ. Toulouse/GRIMM, Univ. Moulay Ismail/ENSAM.

[2] Abderrazzak, Z., and Ahmed, E. 2013. “Using Colred Petri Network in Modeling Multiview Class.” *IJCSNS International Journal of Computer Science and Network Security* 13 (August).

[3] Berardi, D., Calvanese, D., and Giacomoa, G. D. 2005. “Reasoning on UML Class Diagrams.” Dipartimento di Informatica e Sistemistica, Universita di Roma “La Sapienza”, Via Salaria 113, I-00198 Roma, Italy.

[4] Murata, T. 1989. “Petri Nets: Properties, Analysis and Application.” April.

[5] Lipton, R. 1976. *The Reachability Problem Requires Exponential Space*. Technical Report 62, Yale University.

[6] Mayr, E. W. 1984. “An Algorithm for the General Petri net Reachability Problem.” *SIAM Journal of Computing* 13 (3): 441-59.

[7] Kungas, P. 2005. “Petri Net Reachability Checking Is Polynomial with Optimal Abstraction Hierarchies.” In *Proceedings of the 6th International Symposium on Abstraction, Reformulation and Approximation*, SARA, Airth Castle, Scotland, UK, July 26-9.

[8] Kungas, P. 1989. “On Fairness and Conflicts in Petri Nets.” In *Proceedings of the 32nd Midwest Symposium on Circuits and Systems.*

[9] Mellor, S. J., Scott, K., Uhl, A., and Weise, D. 2002. “Advances in Object-Oriented Information Systems.” In *Proceedings of OOIS 2002 Workshops*, Montpellier, France.

[10] Hillah, L. M., Kindler, E., Kordon, F., Petrucci, L., and Treves, N. 2009. “A Primer on the Petri Net Markup Language and ISO/IEC 15909-2.” *Petri Net Newsletter* 76: 9-28, October (originally presented at the 10th International workshop on Practical Use of Colored Petri Nets and the CPN Tools –CPN’09.