A Method Based on Petri Nets for Identification of Aspects*

Vahdat Abdelzad
Islamic Azad University Science and Research Branch, Tehran, Iran
v.abdelzad@srbiau.ac.ir

Fereidoon Shams Aliee
Electrical and Computer Engineering Faculty
Shahid Beheshti University, GC, Evin, Tehran, Iran
f_shams@sbu.ac.ir

Abstract

One of the important factors in creating complexity in software systems is the existence of crosscutting concerns. The concept of aspect orientation with presentation of a method could modulate crosscutting concerns into the single unit that is called aspect, and solve many problems which are created such as tangling and scattering. However, identification and specification of crosscutting concerns and regarding them as aspects is not easy. For this reason, various methods are presented but such methods are informal. In this paper, we propose a formal method based on Petri Nets for identification of aspects. In the method, a software system is expressed in terms of a number of concerns. A concern is composed of one or several requirements which realization of them cause realization of that concern. The proposed method defines requirements and concerns in the formal form by Petri Nets and named them as requirement nets and concern nets. Concern nets with dependencies which there are between requirement nets, model the final system. The execution of final modeled software system based on Petri Nets and monitoring its transitions, shows crosscutting concerns which are candidate aspects.

Categories and Subject Descriptors

D.2.10 [**Software Engineering**]: Design; D.2.1 [**Software Engineering**]: Requirements/Specifications—*Elicitation methods*

Keywords

Crosscutting concerns, requirement nets, concern nets, Petri nets, Aspects, CPN/Tools, aspect-oriented requirements engineering

1. Introduction

Separation of concerns [5] is one of the important principles in software systems development. The goal of separation of concerns is to break a software system into several modules which have minimum overlapping with each other. However, there is specific kind of concerns that cannot be placed into a single module, these concerns are called crosscutting concerns. In computer science, crosscutting concerns are facets of a program which affect (crosscut) other concerns. Crosscutting concerns have two important characteristics [11]:

- Lack of decomposition from other sections (design and implementation)
- Placing their implementation code among several components

This kind of concerns when applied in the software system may cause tangling and scattering problems. Aspect-Oriented Programming (AOP) [12] through encapsulation of crosscutting concerns into module called aspect could prevent implementation level problems. Aspect-Oriented Software development (AOSD) [1] express that aspect orientation in implementation phase is not adequate, therefore, this concept has to be applied in other development phases too. One of these phases is Aspect-Oriented Requirements Engineering (AORE) [8]. The goal of AORE is separation of crosscutting concerns and identification of aspects. Many suitable methods for identification of aspects are offered in [2, 14, 17, 4, 11]. These methods are informal or concerns are regarded as non-functional requirements.

Regarding the importance of formal methods in acceptance and application of a new method, it is necessary to offer formal methods for identification and definition of aspect-oriented primitive concepts. At this domain, in [21] a formal definition of aspect using Petri Nets is presented. In [9] author(s) proposed an approach for solving conflict result from applying various aspects in the same join-point. However, none of them present a formal method for identification of aspects.

In this paper a formal method based on Petri Nets to identify crosscutting concerns is proposed. In the method main axis of activity is the notion of concern. A concern is one or several functional or non-functional requirements

*An earlier version of this paper was presented at the Early Aspects 2010 workshop at the 9th International Conference on Aspect-Oriented Software Development. The paper is recommended by Ruzanna Chitchyan and Steffen Zschaler.

Abdelzad, V., and Aliee, F. S. A Method Based on Petri Nets for Identification of Aspects. Information Sciences and Technologies Bulletin of the ACM Slovakia, Special Section on Early Aspects, R.Chitchyan, S. Zsachaler (Eds.), Vol. 2, No. 1 (2010) 43-49

that can be seen as candidate for aspect. The system that we want to identify its aspects is constructed as series of *concern nets* and extant dependencies between *requirement nets*. The execution of resulting Petri Net for the system will give an output that it is main factor for identification of aspects. The reason of using Petri Nets as a formal method for identification of aspect is that:

- Petri Nets is an executable modeling language, so can identify aspects with executing the model that is constructed for a system.
- Understandability of Petri Nets is more than other formal methods such as matrix.
- Petri Nets has useful tools for modeling and execution itself, so we can use these tools in order to implement the method without design new tools.

In this paper, in Section 2 we have an introduction of Petri Nets. In Section 3 we study concerns and crosscutting concerns then we present formal definitions based on Petri Nets for them. In Section 4, proposed method for obtaining aspects will be described. In Section 5 a case study according to proposed method is stated. Section 6 present related works and finally we have conclusions.

2. Petri Nets

Petri Net is a mathematical based method for modeling and verifying software artifacts that for first time in 1962 by Carl Adam Petri was introduced. Petri Net provides clear and precise semantics, an intuitive graphical notation, and many techniques and tools for their analysis, simulation and execution. A formal definition for Petri Net is following [20]:

Definition of Petri Net: A Petri Net is a 3-tuple PN= (P, T, F) where:

- P is a finite set of places
- T is a finite set of transitions, $P \bigcap T = \emptyset$
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs

In [13] T. Murata gave some typical interpretations of transitions and places. A transition (an event) has a certain number of input and output places representing the pre-condition and post-condition of the event respectively. The presence of a token in a place is interpreted as holding the truth of the condition associated with the place, therefore, every software system can modeled with Petri Nets. For example, take personnel management system in consideration [9]. One of the system concerns is increasing employee salary. For realization of the concern, a system manager should enter user name and password for entering to the system, then should read employee salary and increase amount of his/her salary. Finally, the manager exit from system. Sequence operations of increasing employee salary concern are specified by the Petri Net CN in Figure 1.

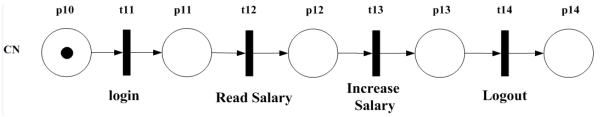

Figure 1: A Petri net for personnel management system.

3. Concerns and Crosscutting Concerns

If aspect-oriented software development is to be fully realized, concerns must be treated as first–class entities throughout the life cycle [17]. Therefore, the systems that want to develop with aspect-oriented software development have to express their specifications and documentations in terms of concerns. Although the concept of concern is well-understood intuitively but expressing a good definition of concern is too hard. Many definitions of concern are offered in [2, 7, 18, 10] which each of them have different dimensions. We offer a comprehensive definition of concern that includes these definitions.

One or several requirements depending on stakeholders and system development that is able to implement by a code structure, is called concern. In this definition, the "one or several" indicates that one or several requirements may constitute a concern. The "requirements" mentions to expectation behaviors in a system or program [16]. The "stakeholders" indicates which requirements include both system requirements and stakeholder requirements (e.g. developers). The "development" indicates that the definition is not limited to a certain phase of development process, such as implementation phase. The "able to implement by a code structure" enhances the application of concern concept in many developing methods, such as object-oriented, structured and any developing methods which have structures related to implementation. So, we can utilize concern concept for quality and quantity characteristics of systems.

Crosscutting concerns are main reason for causing tangling problem. The tangling problem is an obstacle for understandability and maintainability of systems [15]. According to the above definition of concern, we can define a crosscutting concern in the following: a crosscutting concern is a type of concern and has requirements that used to realization of other concerns, or entities of these requirements realize other concerns. Also, we can relate the following definition to tangling problem: If the requirement of a concern is applied to realization of other concerns then the requirement has tangling problem. The definition for tangling is high level since a requirement can constitute from several fine-granularity requirements (entity) and tangling problem is occurred in one of them.

Now, we offer formal definitions based on Petri Nets for concern and requirement. These definitions are necessary for proposed method. In the definitions, requirements and concerns are defined as *requirement nets* and *concern nets* respectively.

Definition of Concern Net (CN): A concern net is a 2-tuple CN= (SoR, SoE) where

- SoR= (RN_1, RN_2, ..., RN_n) (n>0), it is a finite set of requirement nets.
- SoE= (EO_1, EO_2, ..., EO_n) (n>0), it is a finite set of execution orders

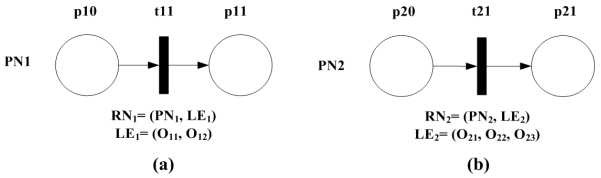

Figure 2: Requirement nets RN_1, RN_2.

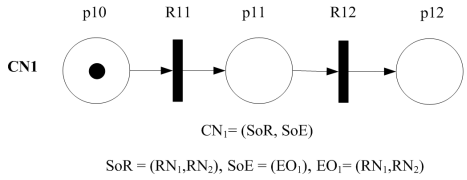

Figure 3: Concern net CN_1 with two requirement nets and one execution order.

Definition of Requirement Net (RN): A requirement net is a 2-tuple RN= (PN, LE) where:

- PN is a Petri Net which is following: |P|=2, |T|=1, |F|=2.
- LE= (O_1, O_2, ..., O_n), is a set of logical entities such as class.

Definition of execution order (EO): An execution order is a sequence of requirement nets which present following:

$$\text{EO}= (RN_1, RN_2, \ldots, RN_n)$$

For example, two *requirement nets* in the names of RN_1, RN_2 are shown in Figure 2. The *requirement net* RN_1 (cf. Figure 2.a) is constituted one Petri Net PN_1 and two logical entities O_{11}, O_{12}. Also *requirement net* RN_2 (cf. Figure 2.b) under one Petri Net PN_1 and three logical entities O_{21}, O_{22}, O_{23} is constituted. In Figure 3, a *concern net* in the name of CN_1 is shown. The *concern net* CN_1 is constituted two *requirement nets* RN_1, RN_2 and one execution order EO_1. The execution order EO_1 is (RN_1, RN_2). In the *concern net* CN_1 transitions R_{11}, R_{12} and *requirement nets* RN_1, RN_2 are face to face. Due to existence an execution order in Figure 3, one token in the first place of *concern net* CN_1 is placed.

4. Identification of Aspects Using Petri Nets

We consider eight stages for realization of the method. In order to identification of aspects, these stages should be satisfied respectively.

Stage 1: in this stage, system expresses in terms of concerns. The system concerns get from lexical analysis of the system text. Specifying the system via concerns is necessary for proposed method due to the fact that in our method concerns are first-class.

Stage 2: in this stage, we specify requirements which are associated to each concern. The requirements may obtain through any traditional requirements engineering approaches. The quality and quantity of specified requirements for every concern depend on interactions between requirements engineering and stakeholders [14]. However, specification of all requirements for each concern in first glance is not easy and some of them are usually specified with reviewing. In consideration of this method, a requirement is taken into account as a independent Petri Net. Therefore, it is possible that requirements gradually go into the concern.

Stage 3: in this stage, we should constitute a *requirement net* for each specified requirement in stage 2. According to the definition of *requirement net*, we have to identify logical entities for each *requirement net*. However, in this stage, identification of logical entities related to *requirement nets* is not necessary. This operation is postponed to stage 8, because it is not needed to decompose all requirements to logical entities for identifying aspects. The requirements that have dependencies with other concerns or requirements should be broken into logical entities.

Stage 4: in this stage, in order to constitute *concern nets*, we should specify execution orders for each concern which is identified in stage 1. Requirements engineers with analyzing purpose of a concern and associated requirements may elicit execution orders. Each execution order satisfies one of its purposes. Also, the number of execution orders has direct relation with requirements granularity (fine or coarse).

Stage 5: in this stage, according to definition in Section 3, we constitute a *concern net* for each concern that is specified in the stage 1. For constituting *concern nets*, we need to *requirement nets* and execution orders which are specified in stage 3, 4 respectively. The execution of each *concern net* implicates that the proper token is placed in the first place of *concern net*. Therefore, for any execution order that exist in a *concern net*, a token must be placed in the first place of *concern net*. If there is not enough token in first place, the *concern net* cannot be executed in the final Petri Nets model correctly. So we cannot identify aspects in the system.

Stage 6: in this stage, the dependencies, restrictions and relationships among *requirement nets* and *concern nets* must be identified. For example, restriction of execution order is a kind of dependency. The dependencies are the direct result of the business logic that system purpose to support [3]. The relationships is kind of logic that can be as co-process and co-data, also can take into account as interpretive relationship [17]. Interpretive relationships reflect interpreted semantics associations among concerns (logical).They depend primarily on the context-dependent interpretation of concern semantics and significance. In the applying of dependency between two *requirement nets*, one new place as temporary place is created. In the temporary place, a token of dependency is placed. This token is composed of *concern net* and *requirement net* names which causes complete execution of system. When these dependencies are imposed into the model, the Petri Nets model mentions to the final system. The model must be executed in proper form. Lacks of execution model indicates that dependencies and tokens of Petri Nets are not defined correctly.

Stage 7: in this stage, for identifying crosscutting concerns (aspects) following operations should be performed: first, we have to specify transitions of each *concern net* that have two or more than two entrances. Second, if

Table 1: Relating aspects with Logical Entities (LE).

	LE_1	LE_1	...	LE_n
$Aspect_1$				
$Aspect_2$				
...				
$Aspect_n$				

value of their entrances tokens are different, so entrance token and transition token are taken into the 2-tuple, such as (token1, token2). Therefore, if a transition has two or more than two entrances with different tokens, for any different token, there has to be defined separate 2-tuple.

Stage 8: After identification of these 2-tuples, the logical entities associated with *requirement nets* in the 2-tuples should be determined. If there is a logical entity that is in the set of logical entities of two *requirement nets* belonging to a 2-tuple, that logical entity is considered as an entity that has tangling problem. The *concern net* which has this *requirement net* in their set of *requirement nets*, considered as crosscutting concern (aspect). However, a *concern net* may be has transition with several entrance, but while the tokens are similar, the transition will not explain any meaning.

Implementation of the method by Petri Nets provides a number of collections that includes logical entities and imposing aspects. It is possible that a share collection exist within them. With extraction of this logical entities and aspects, we will reach to structure like Table 1. In the Table 1, name of logical entities and imposing aspects are specified.

5. Case Study

In this section, a case study for description of proposed method is offered. The case study is a hotel management system [11] which is explained in following concerns (stage 1):

- C_1: Reserve Room: To reserve a room, you check the room availability, and if a room is available, you create a reservation.
- C_2: Reserve Room: To check in a customer, you assign him to a room and consume his reservation. At the same time, you create an initial bill for the customer.
- C_3: Reserve Room: To check out a customer, you collect the payment for the bill. Once the bill has been paid, the customer is removed from the room.
- C_4: Reserve Room: To log, the system checks operations and if there are changes, it loges them.

Now that identifying concerns of the system is done, we should determine requirements of each concern (stage 2).

The requirements of any concern are depicted in Figure 4. In Figure 4, every concern and its requirements are illustrated in the same simple structure with viewpoints [13, 16]. After the associated requirements for each concern are specified, we must constitute *requirement nets* (stage 3). In our case study, there are ten requirements therefore

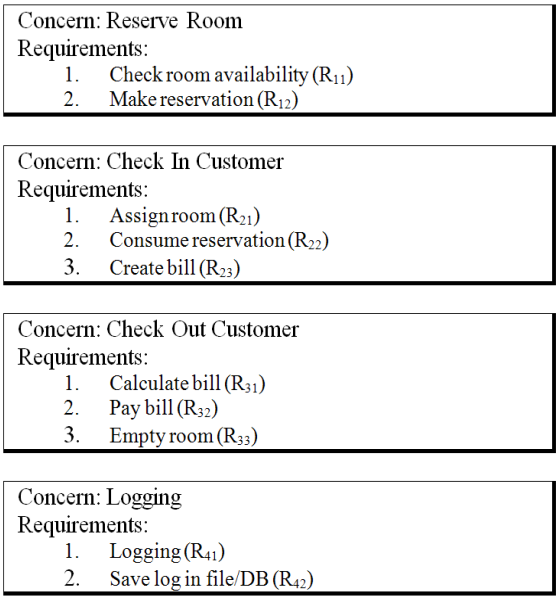

Figure 4: Concerns and associated requirements for hotel management system.

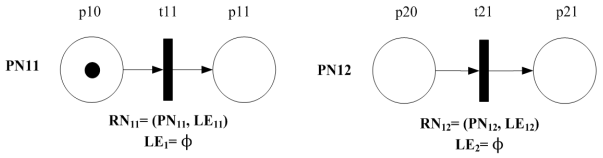

Figure 5: Requirement nets for R_{11}, R_{12}.

we have to constitute ten *requirement nets*. For instance, the *requirement nets* RN_{11}, RN_{12} for the requirements R_{11}, R_{12} are depicted in Figure 5 respectively. These *requirement nets* just are constituted based on the definition in Section 3. In other words, we must follow the definition in order to constitute these *requirement nets*. But important point is that, in this stage, we do not identify logical entities (set of logical entities is considered null) for each *requirement net* because this action will be performed afterwards. The remaining *requirement nets* of hotel management system will constitute in the same way.

In stage 4, we specify execution orders for the concerns. The execution orders for each concern are depicted in Table 2.

For instance, execution order of concern C_1 is EO_{11} that RN_{11}, RN_{12} have to execute respectively. This means that check room availability concern has to satisfy before making reservation concern. In stage 5, we make *concern nets*. There is an execution order for each concern there-

Table 2: The execution orders for concerns of hotel management system.

Concern Name	Name of execution order	Execution order
C_1	EO_{11}	RN_{11},RN_{12}
C_2	EO_{21}	RN_{21},RN_{22},RN_{23}
C_3	EO_{31}	RN_{31},RN_{32},RN_{33}
C_4	EO_{41}	RN_{41},RN_{42}

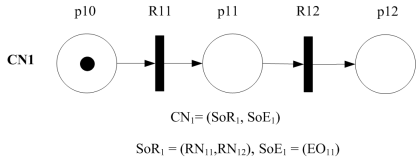

Figure 6: Concern net for Reserve Room (C_1).

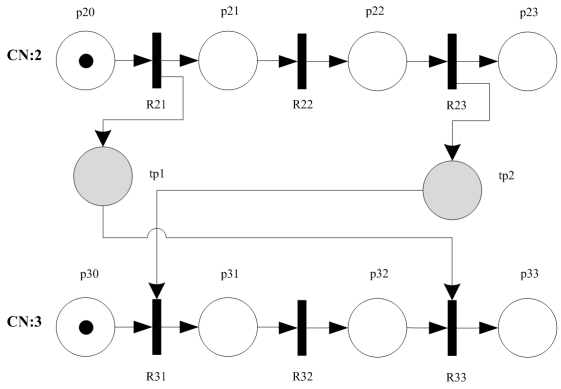

Figure 7: Dependencies between concern nets CN_2 and CN_3.

fore in the first place of every *concern net* one token has to be placed. For example, *concern net* CN_1 for concern C_1 (Reserve Room) is depicted in Figure 6. The *concern net* CN_1 composes of two *requirement nets* and an execution order. In the *concern net* CN_1 because of existing an execution order, one token is placed in place p_{10}. Other *concern nets* are constituted in the same way.

After constitution of all *concern nets*, we can identify dependencies between *concern nets* and *requirement nets* (stage 6). In many cases, dependencies and relationships exist between *requirement nets* of *concern nets*. Also, it is possible that some *concern nets* have dependencies with other *concern nets*. The dependencies of the hotel management system are a kind of restriction of execution order and interpretive relationships. For example, in the system, we should create bill and then calculate it, and also first assigning room concern should performed and then room should be emptied. These dependencies for two *concern nets*, CN_2 (check in customer) and CN_3 (check out customer) is depicted in Figure 7.

In Figure 8, tp_1 and tp_2 are two places with gray color. These places are regarded to establish dependencies between *requirement nets* (RN_{21}, RN_{33}) and (RN_{23}, RN_{31}). In the temporary places for any exit arc, a token must be placed in it by related enter arc, because a *requirement net* may have dependencies (more than one) with other *requirement nets*. Therefore, adequate tokens must exist in the temporary places for applying dependencies and execution of the model. Now we have a final Petri Nets model for hotel management system which is depicted in Figure 9. The final Petri Nets model must be executed then its transitions should be examined (stage 7). This action can be implemented with CPN/Tools and its monitoring capability.

The monitoring output of final Petri Nets model of hotel management system for four transitions R_{21}, R_{22}, R_{33}, R_{41} is depicted in Figure 9.

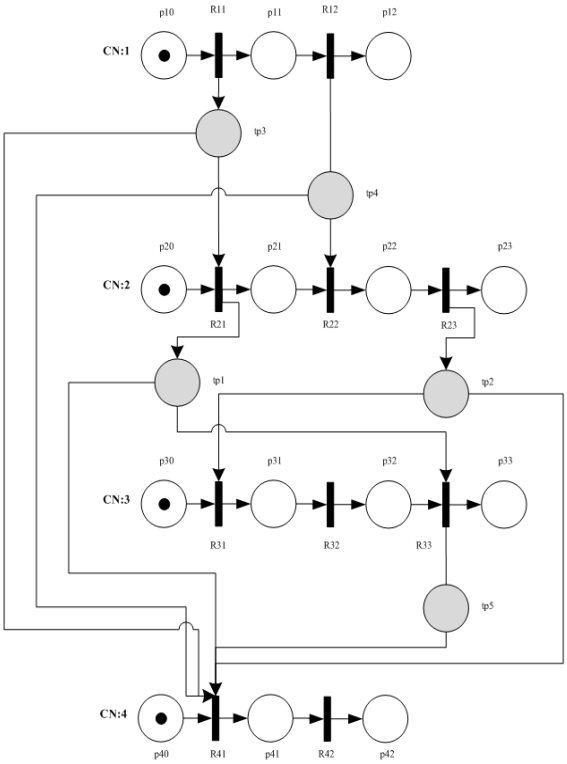

Figure 8: Final Petri nets Model for hotel management system.

Table 3: Logical entities of requirement nets for hotel management system.

Requirement nets	Logic entity
RN_{11}	Room
RN_{12}	Reservation
RN_{21}	Room
RN_{22}	Reservation
RN_{23}	Bill
RN_{31}	Bill
RN_{33}	Room
RN_{41}	Room, Reservation, Bill

In Figure 9, there is an output like $<C_2, R_{21}><C_1, R_{11}>$. This 2-tuple indicates that if minimum a share logical entity exist in the *requirement nets* RN_{21} and RN_{11} then two *concern nets* CN_1, CN_2 can be considered as aspect, because the *requirement nets* has tangling problem. In here, this share logical entity that cause tangling problem is "Room". Action for identifying logical entities must be performed for *requirement nets* which appear in monitoring output. Logical entities for monitoring outputs *requirement nets* are listed in Table 3 (stage 8). We continue this survey (monitoring output) until it is determined that there is share logical entities or not. When there are sharing entities, face to face concerns can be considered as aspect. In the hotel management system because of existing share entities in all requirements, four concerns are viewed as aspect and we call them A_1, A_2, A_3 and A_4. Any concern in the system as aspect has a series of logical entities that aspect is imposed to them. These logical entities are sharing entities and are depicted in Table 4.

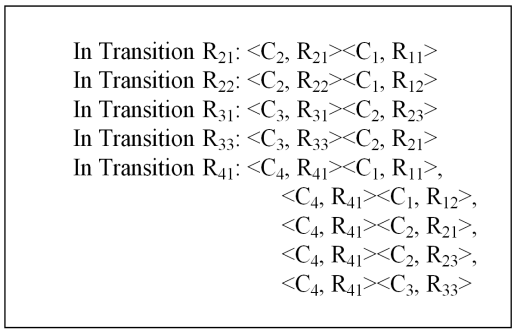

Figure 9: The monitoring output in CPN/Tools for hotel management system.

Table 4: Relating aspect with logical entities in hotel management system.

	Room	Reservation	Bill
A_1	√	√	√
A_2	√	√	√
A_3	√		
A_4	√	√	√

6. Related Works

Rashid [14] provide the AORE model and ARCaDe tools for describing components and requirements-level aspects. Examples of these aspects are compatibility, availability. This work build on ViewPoints model [6], which is intended to support the integration of heterogeneous requirements specified from multiple perspective. In AORE model, concern identification relies on domain knowledge of developers, and also concerns are non-functional properties. We use a similar means for identification of concerns in stage one of proposed method and are considered concerns as functional and non-functional properties. Our method uses a formal method for identification of aspects but AORE model use an informal approach.

Elisa [2] proposed a Theme approach for viewing the relationships between behaviors in a requirements document, identifying and isolating aspects in the requirements, and modeling those aspects using a design language. In Theme approach, Theme provides support for aspect-oriented development at two levels. At the requirements level, Theme/Doc and at the design level Theme/UML. Theme/Doc can be used as a method for identifying concerns in the stage one of our method. Also, Theme approach presents an informal method for identification of aspects.

D. Xu [21] firstly incorporated the features of AOP into Petri Net and extended Petri Nets to support AOM. His work is based on Predicate/Transition Nets (PrT nets). Lianwei Guan [9] presents a Petri Net-based approach to support aspect-oriented modeling. In this approach, software systems are modeled as aspect nets and base net, then a woven mechanism is given to compose the aspect nets and base net. They also give four mechanisms to model the order constraints and dependencies among aspects that supposed on the Same Joint Point (SJP), and give a solution to detect conflict relations among the aspects. Two method based on Petri Nets are not present a method for identification of aspects. However, these methods can be used in our method as a complementary approach. In [19] authors proposed an approach based on matrix for identification of aspects. This approach and our method using common concepts for identifying aspects but there is no useable model for this approach in order to can be used in the design level but the design model of our method can be used in the design level.

7. Conclusions

This paper has offered a formal method based on Petri Nets for specification of crosscutting concerns and identification of aspects. In the proposed method, a software system was considered as sets of *concern nets*. Each *concern net* is constituted as a set of *requirement nets* and execution orders. The *requirement nets* have logical entities which will be used for identification of aspects and may have dependencies with other *requirement nets*. For identification of aspects, we execute the final Petri Nets model and then monitoring each transition. In the monitoring process, if a transition has following conditions: (1) each transition has two or more than two entrances; (2) token of entrances are different; (3) There is a share entity between two various entrances, we consider its dominant *concern net* of transition as aspect. Offering a formal method based on Petri Nets for identification of aspects provides some advantages. One advantage is that we make sure of what was considered as an aspect is crosscutting concern. Therefore, an aspect with high finality can be considered in the next development phases.

There is still shortcoming in the proposed method. This method identify aspects using Petri Nets but cannot determine join-points with more detail and the circumstances of imposing aspects (after, before, around) to logical entities, therefore, still there are works have to be done to extend the method in the future.

References

[1] Aspect-Oriented Software Association. Aspect-oriented software development. AOSD Web page. `http://aosd.net/`, 2002.

[2] E. Baniassad and S. Clarke. Theme: An approach for aspect-oriented analysis and design. In *26th International Conference on Software Engineering (ICSE 2004)*, pages 158–167. IEEE CS, 2004.

[3] R. V. Binder. *Testing Object-Oriented Systems: Models, Patterns and Tools*. Addison-Wesley, 2000.

[4] R. Chitchyan, A. Rashid, P. Rayson, and R. Waters. Semantics-based composition for aspect-oriented requirements engineering. In *6th International Conference Aspect-Oriented Software Development (AOSD 2007)*, pages 36–48. ACM, 2007.

[5] E. W. Dijkstra. *A Discipline of Programming*. Englewood Cliffs, NJ:Prentice Hall, 1976.

[6] A. Finkelstein and I. Sommerville. The viewpoints faq. *BCS/IEE Software Engineering Journal*, 11(1), 1996.

[7] J. D. Gradecki and N. Lesiecki. *Mastering AspectJ: Aspect-Oriented Programming in Java*. Wiley, ISBN: 0-471-43104-4, 2003.

[8] J. Grundy. Aspect-oriented requirements engineering for component-based software systems. In *4th IEEE Int'l Symp. on RE*, pages 84–91. IEEE CS, 1999.

[9] L. Guan, X. Li, and H. Hu. A petri net-based approach for supporting aspect-oriented modeling. In *2nd IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering*, pages 83–90. IEEE CS, 2008.

[10] IEEE. IEEE recommended practice for architectural description of software-intensive systems. Technical Report IEEE Std 1471-2000, Software Engineering Standards Committee of the IEEE Computer Society, 2000.

[11] I. Jacobson and P.-W. Ng. *Aspect-Oriented Software Development with Use Cases*. Addison-Wesley, 2004.

[12] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In M. Aksit and S. Matsuoka, editors, *Proc. of 11th European Conference on Object-Oriented Programming (ECOOP'97)*, LNCS 1241, Jyväskylä, Finland, June 1997. Springer.

[13] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[14] A. Rashid, A. Moreira, and J. Araújo. Modularisation and composition of aspectual requirements. In *2nd International Conference on Aspect-Oriented Software Development (AOSD 2003)*, pages 11–20. ACM, 2003.

[15] A. Rashid, P. Sawyer, A. Moreira, and J. Araújo. Early aspects: A model for aspect-oriented requirements engineering. In *Proceedings of the IEEE Joint International Conference on Requirements Engineering (REŠ02)*. IEEE, 2002.

[16] I. Sommerville. *Software Engineering, Seventh edition*. Addison-Wesley, 2005.

[17] S. M. Sutton and I. Rouvellou. Modeling of software concerns in cosmos. In *1st International Conference on Aspect-Oriented Software Development (AOSD 2002)*, pages 127–133. ACM, 2002.

[18] P. Tarr, H. Ossher, W. Harrison, and S. Sutton. N degrees of separation: Multi-dimensional separation of concerns. In *21st International Conference on Software Engineering (ICSE)*, pages 107–119, Los Angeles, 1999. IEEE.

[19] K. van den Berg, J. M. Conejero, and J. Hernández. Identification of crosscutting in software design. In *8th International Workshop on Aspect-Oriented Modeling*, 2006.

[20] W. van der Aalst. The application of petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998.

[21] D. Xu and K. E. Nygard. Threat-driven modeling and verification of secure software using aspect-oriented Petri nets. *IEEE Transactions on Software Engineering*, 32(4):265–278, April 2006.