Contents lists available at GrowingScience

Management Science Letters

homepage: www.GrowingScience.com/msl

Evaluating software architecture using fuzzy formal models

Payman Behbahaninejad[a], **Ali Harounabadi**[b*] and **Sayed Javad Mirabedini**[b]

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

A R T I C L E I N F O

Article history:
Received June 2, 2011
Received in Revised form
October, 19, 2011
Accepted 8 January 2012
Available online
11 January 2012

Keywords:
Fuzzy-UML
Software architecture
Evaluation
Architecture model

A B S T R A C T

Unified Modeling Language (UML) has been recognized as one of the most popular techniques to describe static and dynamic aspects of software systems. One of the primary issues in designing software packages is the existence of uncertainty associated with such models. Fuzzy-UML to describe software architecture has both static and dynamic perspective, simultaneously. The evaluation of software architecture design phase initiates always help us find some additional requirements, which helps reduce cost of design. In this paper, we use a fuzzy data model to describe the static aspects of software architecture and the fuzzy sequence diagram to illustrate the dynamic aspects of software architecture. We also transform these diagrams into Petri Nets and evaluate reliability of the architecture. The web-based hotel reservation system for further explanation has been studied.

1. Introduction

Unified Modeling Language (UML) is a semi-formal modeling and standard language, for easy describing software architecture (Object Management Group, 2002). This language has developed a powerful set of predefined modeling elements, diagrams and structure to describe the structural and behavioral properties of software architecture and the introduction of appropriate tools to support it. Unfortunately, this language is only capable of modeling some specific information systems, where there is no uncertainty in the model. However, when we consider uncertainty in UML, the extended version named Fuzzy-UML will be produced (Haroonabadi & Teshnehlab, 2008; Ma et al., 2011). Fuzzy-UML includes both o structure and behavior sights, which would be explained later in this paper. The next section is proposed method, and its subsections included fuzzy data model, fuzzy sequence diagram, evaluating software architecture using fuzzy Petri Nets, reliability using fuzzy Petri Net, case study, and the last section is conclusion and future works.

* Corresponding author.
E-mail addresses: payman.a907@gmail.com (P. Behbahaninejad)

doi: 10.5267/j.msl.2012.01.010

To evaluate software architecture, both structural and behavioral aspects must be considered including use case, deployment and class diagram used to display structural aspect and activity, sequence and state diagram stand for behavioral aspect.

In this paper, we study Fuzzy-UML to display uncertainty in the systems, in this paper we use fuzzy class diagram to show the structural aspect of the software system, and to display behavioral aspect, we use fuzzy sequence diagram. During the past few years, there have been several studies on fuzzy UML, fuzzy logic, Petri Nets, etc. (Ma, 2005; Motameni et al., 2008; Motameni & Ghassempouri, 2011). Ma has contributed on the idea of Fuzzy-UML by discussing and developing some new ideas with complete explanation of fuzzy programming (Ma et al., 2005-2011). Haroonabadi and Teshnehlab (2009) used stereotypes to describe software architecture and added fuzzy features to transform use case and sequence diagram respectively to fuzzy use case and fuzzy sequence diagram. Behavioral modeling system based on the fuzzy sequence diagram was also explained in details. With using these diagrams, one is able to describe the architecture of uncertain information systems and then analyze them.

2. Fuzzy data model

The class diagrams in UML are the logical frameworks, which describe the nature of the main structure. The classes and the relationships among classes are the integrated elements of the class diagram. Fuzzy-UML is created by adding uncertainty as part of system integration. According to MA et al. (2011) in the context of classes, there are three levels of fuzziness defined as follows,

-Fuzziness in the extent where the class belongs to the data model as well as fuzziness on the content of the class in term of attributes.

-Fuzziness represents whether some instances belong to a particular class; even though the structure of a class is crisp, it is possible that an instance of the class belongs to the class with degree of membership.

-The third level of fuzziness is on attribute values of the instances of the class; an attribute in a class defines a value domain, and when this domain is a fuzzy subset or a set of fuzzy subset, the fuzziness of an attribute value appears.

The attribute or the class name in the first level must be described by the phrase *WITH membership DEGREE* where, $0 \leq membership \leq 1$. This value shows the belonging degree of the attribute to the class or the class to the data model. The second level of fuzziness, the membership degree in an Instance of the class, which belongs to the class should be specified. So an additional attribute in the class is defined for representing of the instance membership degree to the class where its domain is [0, 1]. The classes with the second level of fuzziness are specified by a rectangle where its lines are in dash form. In the third level, a *fuzzy* keyword is appeared in front of the attribute. Ma et al. (2011) presents an example of banking account using the concept of fuzzy class and Fig. 1 shows the *banking account* fuzzy class. In the mentioned class, the *credit* attribute could have the fuzzy value (the third level of fuzziness). In the other hand, the credit attribute is a linguistic variable, and it has a domain like fuzzy sets (for example: little / much). The account *type* specifies the membership degree of *credit* attribute to the class (the first level of fuzziness): "*Credit WITH 0.8 membership DEGREE*"

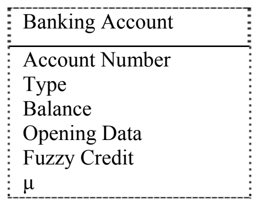


Fig. 1. A fuzzy class of banking account.

The relationships among the classes are divided into four categories and they are propounded in fuzz terms (Larman, 1998) including fuzzy generalization, fuzzy association, fuzzy aggregation and fuzzy dependency.

3. Fuzzy system sequence diagram

In UML sequence diagram is implemented for materializing the practical cases and if the case is uncertain, the sequence diagram will be uncertain, too. A system sequence diagram could be created very easily by representing input and output events. System sequence diagrams such as use cases and system contracts describe the role of a system without explaining how it accomplishes. A simple system sequence diagram consists of actor, system, and messages and each message itself has events and conditions. This diagram incorporates fuzzy rules for transforming the state of an object into another state and a fuzzy rule can be expressed as follows,

Rule = If <condition list> Then <event list>

According to Haroonabadi and Teshnehlab (2009) uncertainty in the method has two levels of fuzzification. First, we use a degree method second we must determine the logic. Fig. 2 shows a sample of fuzzy concept where a message C belongs to object B with the membership function of $\mu_B(x)$ and a message D belongs to object A with the membership function of $\mu_A(x)$.

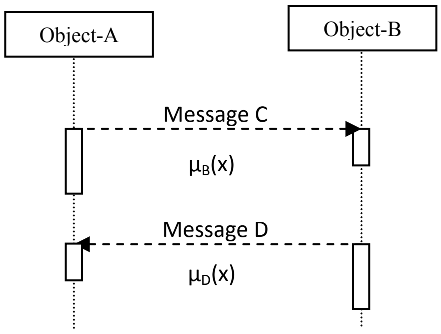


Fig. 2. Fuzzy message in fuzzy sequence diagram

4. Fuzzy Petri Net

According to Motameni et al. (2006) the following fuzzy Petri Net (FPN) structure can be used to model fuzzy rules,

$(P, P_s, P_e, T, TF, TRTF, A, I, O, TT, TTF, AEF, PR, PPM, TV)$, where :

i. *P*: is a finite set of fuzzy places. Each place has a property associated with it, in which:
- $P_s \subset p$ is a finite set of input places for primitive events.
- $P_e \subset p$ is a finite set of output places for actions or conclusions.

ii. *T*: is a finite set of fuzzy transitions. They use the values provided by input places and produce values for output places.

iii. *TF*: is a finite set of transition functions, which perform activities of fuzzy inference.

iv. $TRTF: T \rightarrow TF$ is transition type function, mapping each transition $\in T$ to a transition function $\in TF$.

v. $A \subseteq (P\times T \cup T\times P)$ is a finite set of arcs for connections between places and transitions. Connections Between the input places and transitions $(P \times T)$ and connections between the transitions and output places $(T \times P)$ are provided by arcs. In that:

- *I: P →T* is an input mapping.
- *O: T→ P* is an output mapping.

vi. *TT* is a finite set of fuzzy token (color) types. Each token has a linguistic value (i.e., low, medium and high), which is defined with a membership function.

vii. *O: T→P* is token type function, mapping each fuzzy place ∊ *P* to a fuzzy token type ∊ *TT*. A token in a place is characterized by the property of the place and a level to which it possesses that property.

viii. *AEF: Arc →* Expression is arc expression function mapping each arc to an expression, which carries the information (token values).

ix. *PR* is a finite set of propositions, corresponding to either events or conditions or actions/conclusions.

x. *PPM: P →[0, 1]* is a fuzzy place to proposition mapping, where$|PR| = |P|$.

xi. *TV: P →[0, 1]* is truth values of tokens *(μ_i)* assigned to places. It holds the degree of membership of a token to a particular place.

5. Proposed model

To evaluate software architecture, both structural and behavioral aspects should be considered. Use case, deployment and class diagram are implemented to display structural aspect and activity, sequence and state diagram stand for behavioral aspect. To display uncertainty in the systems, in this paper we use fuzzy class diagram to show the structural aspect of the software system, and to display behavioral aspect, we use fuzzy sequence diagram as follows,

Step1. First for each message in this diagram, its event and conditions must be found. The events and conditions calculated for the dream activity is represented in Table 1.

Step2. First we need to check the correctness of the conditions. Therefore, we should provide a mapping to do that. For each condition, we provide a transition, which is responsible to validate the result and the result will go to another place. That means the token with the given fuzzy amount continues its lifecycle and the token may have the value from 0 to 1, which means: $0 \leq$ token value ≤ 1. The result at the end of analyzing condition C will be a fuzzy amount based on the condition. Sometimes, we have more than one condition depending on fuzzy logical operations and we choose the appropriate function. The concept is represented in Eq. (1) as follows,

$$\text{OR} = \mu AB(x) = \max [\mu A(x), \mu B(x)] , \qquad (1)$$
$$\text{AND} = \mu AB(x) = \min [\mu A(x), \mu B(x)].$$

In Fig. 3 and Fig. 4, two different common cases are depicted. Similarly, we can play with the rules and create more complicated and complex logical operation by extending the figure based on the number of rules and order of logical operation.

Step3. The next step is to run the event/events if the conditions are met. Here the antecedent fuzzy amount is used to evaluate the consequent. The transition here is to apply the antecedent truth value to the consequent membership function. Aggregation occurs here as in Fig 5. The defuzzification process will be accomplished here because the final output of a fuzzy system should be a crisp number. The defuzzification process is done based on center of gravity test. Center of gravity can be expressed as equation (2).

$$\text{COG} = \int_a^b \mu\,(A)\; x\; dx \,/\, \int_a^b \mu\,(A)\;\; dx \qquad (2)$$

Table 1 shows details of the events and conditions for rule 1 and 2.

Table 1
Event and condition for dream activity

Condition	Event	Rule
C1	E is E1	R1
C2	E is E2	R2

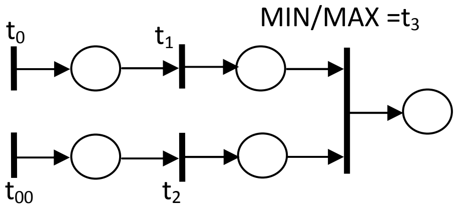


Fig. 3. If (C is C1) And/OR THEN…

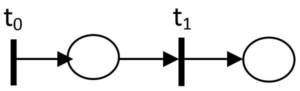


Fig. 4. If C is C1 THEN…

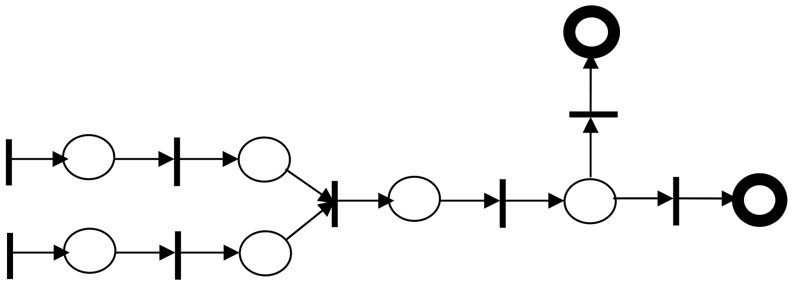
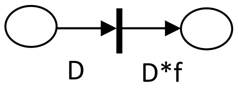


Fig. 5. Modeling consequent of fuzzy rule with aggregation

Here we use fuzzy Petri Net to calculate the data flow and a brief description of Fig. 3 is as follows: We can say t1 uses C1 to adjust and clip consequent membership function of E1, on the other hand, it calculates the consequent of rule R1. T2 does same but for E2 , t3 aggregates the two rules , t4 does defuzzification , t5 and t6 both are responsible to check to see if the correct condition that needs to trigger the system occurs or not. If it does not occur t5 passes the token to its end place and if it occurs t6 passes it to the system. The result of this step (the amount in the output place of t6) which is a strength fuzzy amount will be exerted from actor to the system as a black box or from system to the actor as a result of a message.

6. Reliability using Petri Net

For calculating reliability of the system, we may define a success rate, f, for the transition in CPN. Success rate specifies the probability of firing transition if something wrong does not happen. On the other hand, 1-f is the failure rate, the probability of data loss. The token in the input of the transition *t* is assumed that carries the amount, which stands for the accumulation of the success rate up to that place. When transition fires the amounts which represents the success rate will be changed to D*f, i.e. the token has got a new probability of firing; D*f instead of D. Fig 6 shows the concept.



D: accumulation of success rate = Reliability
Fig. 6. Reliability using fuzzy Petri Net

7. Case study

The following case study is chosen because it is familiar, but rich architectural problems, and thus allows one to concentrate on how to do analysis, rather than explain the problem and domain. A RRS system is a computerized application used to record rents and handle payments; it is typically used in web based room renting systems. It includes hardware components such as a computer and software to run the system. It interfaces to various service applications. These systems must be relatively faulttolerant; that is, even if remote services are temporarily unavailable, they must still be capable of capturing data and handling at least cash payments. After studying the use cases and use case diagram, the main successful scenario is chosen by the analyzer for SSD. Analyzer chose the SSD in fig 7. Now gradually we are going to transform each sequence into fuzzy message. The customer should be able to initiate a new rent if two requirements are provided. The event and conditions of the first message are derived. The scenario considered by the analyst as followed: for received money, we have three fuzzy values, named: not enough, almost enough, indeed enough, and for date we have two fuzzy values, named: valid and invalid. Combining the inputs and their corresponding fuzzy values, we have six conditions, accordance with table 2. Like the first actor-system message, other actor-system messages and the fuzzy rules are created by fuzzy specialist. For system- actor messages we don't need to do the same mapping because in SSD, system is black box, so this is just the result that system produces is important and we don't need to know how it is created. This message is based on the result from the previous step. System reaction to the rule is done via a simple message but we should define two important things; first value(s) that should be returned from system to actor is associated with the previous message and in case of no proper result from the previous message, customer should resend the previous message. Fuzzy rules for message one, are in Fig 8. Fig 9. shows the output for two fuzzy input variables. Accordance to it, the output for mentioned inputs are 5.65 out of 15. Fig10. shows the final fuzzy system sequence diagram modeled with fuzzy Petri Net. To calculate the reliability of each incoming message or actor-system message, we can follow the formula given in D. With the input money s $5000 and the date 20th, we may have the following fuzzy values:

μ(money=indeed enough) =0.0 μ(date=valid) =0.48 μ(money=almost enough) =0.09 μ(date=invalid) =0.39 μ(money=not enough) =0.88

We assume all the transitions have the probability of firing 0.97 except t0 and t00 with the reliability of 1. path: The tokens may pass all the transitions except t15 and they may pass the rest once. One important point here is that in case of many inputs and one output as in t3 we assume D in fig .6 the minimum of the input reliabilities. This is because t3 should have it's both inputs if it wants to fire. Therefore the reliability of a token after passing each transition for message one in the fuzzy SSD for the given inputs is as follow:

t0 =1, t00 =1, t1 = 0.97, t2 = 0.97, t8 = 0.97, t9 = 0.97, t11=0.97, t12 = 0.97,

t3 =0.94, t10 =0.94, t13 = 0.94, t4 = 0.91, t5 = 0.91, t6 = 0.91, t7 = 0.88, t14 = 0.85, t16 = 0.83

The final reliability of message one with mentioned input is 0.83 because it is the last transition, token passes. For each message, we can calculate the reliability based on the path a token may move and success rate of the transitions token encounters in its lifecycle. And the reliability of SSD is the reliability of its messages individually.

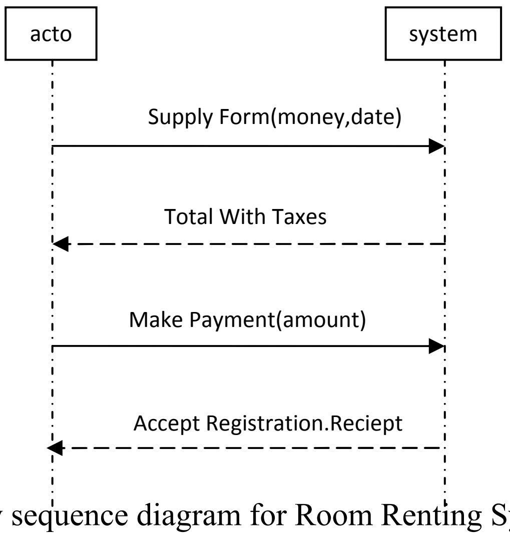


Fig. 7. Fuzzy sequence diagram for Room Renting System

Table 2
Event and condition for 1st message in F-SSD

Rule	Event	Condition	State
R1	Money is not enough Date is invalid	If Money is **not enough And** Date is **invalid**	Reception is **unacceptable**
R2	Money is not enough Date is valid	If Money is **not enough And** Date is **valid**	Reception is **unacceptable**
R3	Money is almost enough Date is invalid	If Money is **almost enough And** Date is **invalid**	Reception is **almost acceptable**
R4	Money is almost enough Date is valid	If Money is **almost enough And** Date is **valid**	Reception is **almost acceptable**
R5	Money is indeed enough Date is invalid	If Money is **indeed enough And** Date is **invalid**	Reception is **Indeed acceptable**
R6	Money is indeed enough Date is valid	If Money is **indeed enough And** Date is **valid**	Reception is **Indeed acceptable**

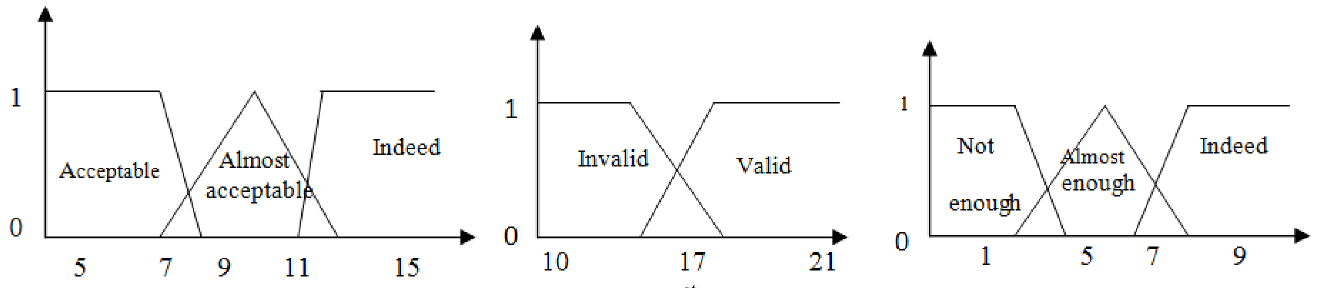


Fig. 8. Fuzzy rule for 1st actor-system message

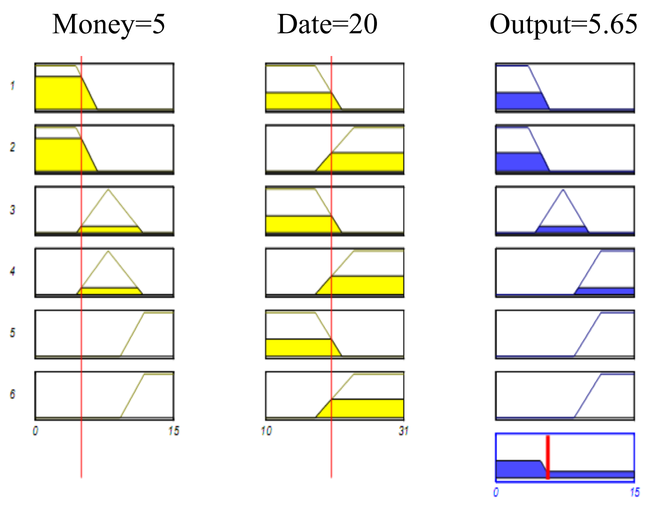


Fig. 9. Output for the two fuzzy inputs

8. Conclusion

The final reliability of the 1st message is 0.83, which is low and it is because the reliability of each transition is 0.97. The final reliability of SSD depends on the success rate of each transition in each incoming message. The developer of the system should create more reliable components in other word transitions with better success rate. In order to enhance the reliability more precisely, software classes must be programmed. In RUP, these kinds of shortcomings will be solved during the different iterations. Reliability plays an important role in purchasing software and it must be considered by system analysts. This is very significant while dealing with crucial controlling systems. Analysts of the software can benefit from this research. The study concludes to model and evaluate the UML system sequence diagram. System sequence diagram first modeled to fuzzy Petri net and then the model was analyzed based on reliability. Through the formalism of the Petri Net, analysis of SSDs in terms of nonfunctional parameters is done. At the end the result of these experiments were assessed.

As a future work, other UML diagrams specially the behavioral diagrams of UML and other nonfunctional parameters or can be evaluated through Petri Net. Besides UML diagrams, software architectures will be studied later through the same technique.

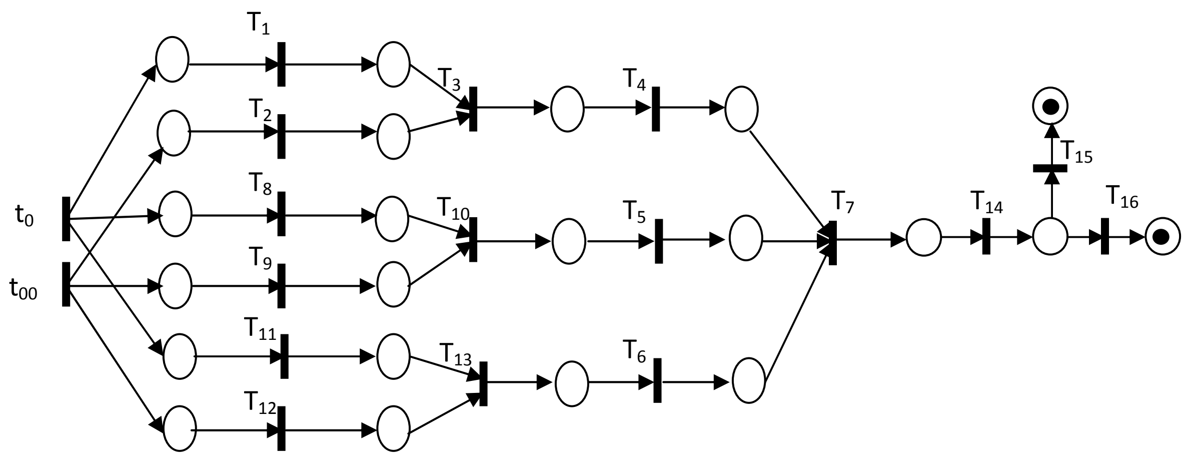


Fig. 10. Fuzzy Petri Net for SSD in fuzzy-UML

References

Bostan-Korpeoglu, B., & Yazici, A. (2006). A Fuzzy Petri Net Model for Intelligent Database. *Data & Knowledge Engineering*, 8, 112-122.

Haroonabadi, A., & Teshnehlab, M. (2008). A novel method for behavior modeling in uncertain information systems. *World Academy of Science, Engineering and Technology*, 41, 959-966.

Haroonabadi, A., & Teshnehlab, M. (2009). Behavior Modeling in uncertain information by Fuzzy-UML. International journal of soft computing 4(1), 32-38.

Larman, C. (1998). Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and the Unified Process. 2nd ed. Prentice Hall.

Ma, Z. (2005). Fuzzy Information Modeling With the UML. *Idea Group Publishing*, 153-176.

Ma, Z.M., Zhang, F., & Yan, L. (2011a). Fuzzy Information Modeling In UML Class Diagram And Relational Database Models. *Applied Soft Computing*, 11, 4236-4245.

Ma, Z.M., Zhang, F., Yan, F., & Cheng, J. (2011b). Representing and reasoning on fuzzy UML models: A description logic approach. *Expert Systems With Applications*, 38, 2536-2549.

Ma, Z.M., Yan, L., & Zhang, F. (2011c). Modelling fuzzy information in UML class diagrams and object-oriented database models. *Fuzzy Sets & Systems*, 186, 26-46.

Motameni, H. Movaghar, A., Daneefar, I., Nematzadeh, H., & Bakhshi, J. (2008). Mapping to convert activity diagram in fuzzy UML to fuzzy Petri Net. *World Applied Sciences Journal*, 3(3), 514-521.

Motameni, H., Daneefar, I., Bakhshi, J., & Nematzadeh, H. (2009). Transforming fuzzy state diagram to fuzzy Petri net. *Journal of Computer Engineering*, 1(1), 29-44.

Motameni, H., & Ghassempouri, T. (2011). Transforming fuzzy communication diagram to fuzzy Petri net. *American Journal of Scientific Research*, 16, 62-73.

Object Management Group. (2002). UML Profile for Schedulability. Performance and Time Specification. http:/www.omg.org.