World Applied Sciences Journal 3 (3): 514-521, 2008
ISSN 1818-4952
© IDOSI Publications, 2008

Mapping to Convert Activity Diagram in Fuzzy UML to Fuzzy Petri Net

[1]*H. Motameni,* [2]*A. Movaghar,* [3]*I. Daneshfar,*[3]*H. Nemat Zadeh and* [3]*J. Bakhshi*

[1]Department of Computer Engineering, Islamic Azad University, Sari Branch, Iran
[2]Department of Computer Engineering, Sharif University of Technology, Tehran, Iran
[3]Department of Computer Engineering, University of Science and Technology of Mazandaran, Babol, Iran

Abstract: UML is known as one of the most common methods in software engineering. since this language is semi formed, many researches and effort have been performed to transform this language in to formal methods including Petri nets. Thus, the operation of verification and validation of the qualitative and non functional parameters could be achieved with more ability. Since the majority of the real world information are uncertain, there fore fuzzy UML diagram has been extensively used by system analyzer this paper is attempt to transform activity diagrams created in fuzzy UML into fuzzy Petri net.so that the verification and performance evaluation operation could be performed formally, rather than exact visual analysis.

Key words: Software engineering . fuzzy UML . fuzzy petri net . fuzzy activity diagrams

INTRODUCTION

Nowadays, UML diagrams are extensively used in software design. However, the semi-formal characteristic of this method is a limitation for verification operations and predicting non-functional parameters of the software, especially in the first cycle of the software production. This problem is more critical for control, critical, reactive and real time systems. On the other hand, since the majority of the real world information is uncertain, therefore fuzzy UML diagrams have been extensively used by system analyzers. Several researches have been performed to tackle with the semi-formal problem of UML. Some of these researches have only used a transformation algorithm, which transforms the created UML model into a Petri net as a mathematical and formal model that, in turn, contains the visual aspect of modeling and pursues the verification operations with further ability [1-8]. Some of the researches in this field besides representing a transformation algorithm (or without representing an algorithm and only by using the available Algorithm); evaluate the capability of the non-operational parameters and commonly qualitative parameters on the obtained Petri nets of the UML model created [9-12]. It is obvious that the lack of this important ability in UML models remains the needs of the costumer and the market unsatisfied. So, this is the reason that makes this type of researches important. In our previous researches [13-17] besides of studying and presenting transformational patterns for some kinds of usual UML diagrams, especially state diagrams and activity diagrams, we presented methods for evaluating some qualitative parameters. In this paper, due to the growing process of using UML diagrams in fuzzy model, we centralized on this kind of diagrams and with the significant Ability of Petri nets in semi-formal UML model formalization we present a pattern to transform fuzzy Activity diagrams to fuzzy Petri nets. First, we introduce fuzzy Activity diagrams briefly. Then, we describe the transform algorithm. At the end, as a case study, we will study the usage of this model for a car sharer service system.

FUZZY UML

UML is known as one of the most important tools in extending object oriented systems. This language makes visual modeling possible so that the system developers will be able to standardize and make understandable the ideas and establish more effective mechanism in relations with other patterns. In a proposed general pattern [18, 19]. Since the real world information is mostly uncertain, in many case these type of information can not be modeled by UML. Recently, a model named fuzzy Mulches been introduced [20-22] which has the UML characteristics, is also able to model uncertain concepts.

Fuzzy activity diagram: Activity diagram is one of the most important UML diagrams, since it has got momentous efficiency in designing stage of software [17]. This diagram helps us to define operations better. It gives the programmer the ability of implementing

Corresponding Author: Dr. H. Motameni, Department of Computer Engineering, Islamic Azad University , Sari Branch, Iran

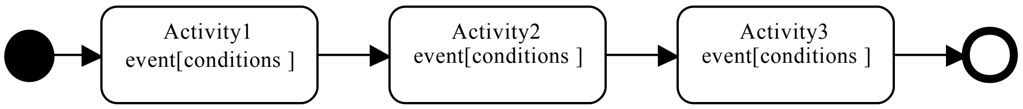


Fig. 1: Sample activity diagram

efficient classes. This diagram has got its activities, states and transitions. A simple activity diagram is shown Fig. 1.

A simple activity diagram comprises 6 sections:

- Start state
- Different activities
- Events
- Conditions
- Transitions
- Final state

As it is shown start state is depicted with ● that indicates the entry of diagram. The final state is depicted with ◉ . On each activity diagram there is only and only one start state whereas it may have more than one final state. Each activity is shown by a rectangle that has circular corners. These activities show the work sequence of software.

Each activity has got its events and conditions. Transition which is depicted with → is a moving from one activity to another or sometimes one state to an activity or vice versa. Normally a transition occurs when an activity is done. Conditions which are limited in bracket control whether an activity can happen or not. It is important to consider that happening an activity means the event is done.

According to these explanations, a fuzzy activity diagram is a graphical model in fuzzy uml which shows the different levels of a fuzzy object in its real world life cycle. This diagram uses fuzzy rules for transforming the state of an object to another state. A fuzzy rule is shown as below:

```
<on event list <event threshold>>
    if condition list <EC coupling>
        Then action
```

Fuzzy rules are used to show the real world rules for an object in which these rules can be active or deductive. As an example, the above mentioned rule is an active one. If the on part is omitted, then it becomes a deductive rule. If in the on part, the threshold is omitted in active rules, the threshold is assumed to be an exact matching with a value of 1.

Each section of the activity diagram can be transformed to a fuzzy activity diagram. Table 1 shows these transformations, clearly.

Table 1: Transformation of an activity diagram into its fuzzy state

Fuzzy activity diagram	Activity diagram
Action of rule	activity
[Fuzzy condition]	[condition]
Fuzzy event	Event

FUZZY PETRI NETS

We introduce the following fuzzy Petri net (FPN) structure to model fuzzy ruler [23-25]:
$(P, P_S, P_e, T, T_F, TRTF, A, I, O, TT, TTF, AEF, PR, PPM, TV)$,
where

1. 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.
2. T is a finite set of fuzzy transitions. They use the values provided by input places and produce values for output places.
3. TF is a finite set of transition functions, which perform activities of fuzzy inference.
4. TRTF:$T \rightarrow TF$ is transition type function, mapping each transition $\in T$ to a transition function $\in TF$.
5. $A \subseteq (P \times T \cup T \times P)$ 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:
   - $I: P \rightarrow T$ is an input mapping.
   - $O: T \rightarrow P$ is an output mapping.
6. 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.
7. $O: T \rightarrow PLs$ token type function, mapping each fuzzy place $\in P$ to a fuzzy token type $\in TT$. A token in a place is characterized by the property of the place and a level to which it possesses that property.
8. AEF:Arc$\rightarrow$ Expression is arc expression function mapping each arc to an expression, which carries the information (token values).
9. PR is a finite set of propositions, corresponding to either events or conditions or actions/conclusions.
10. PPM:$P \rightarrow PR$, is a fuzzy place to proposition mapping, where$|PR| = |P|$.

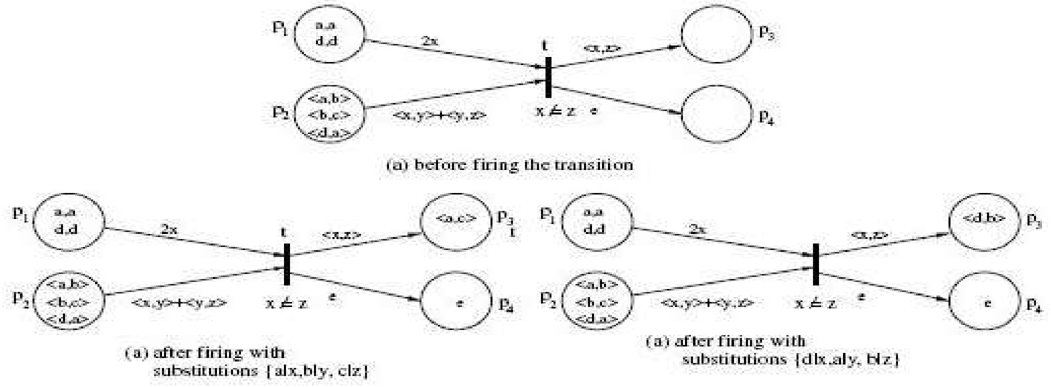
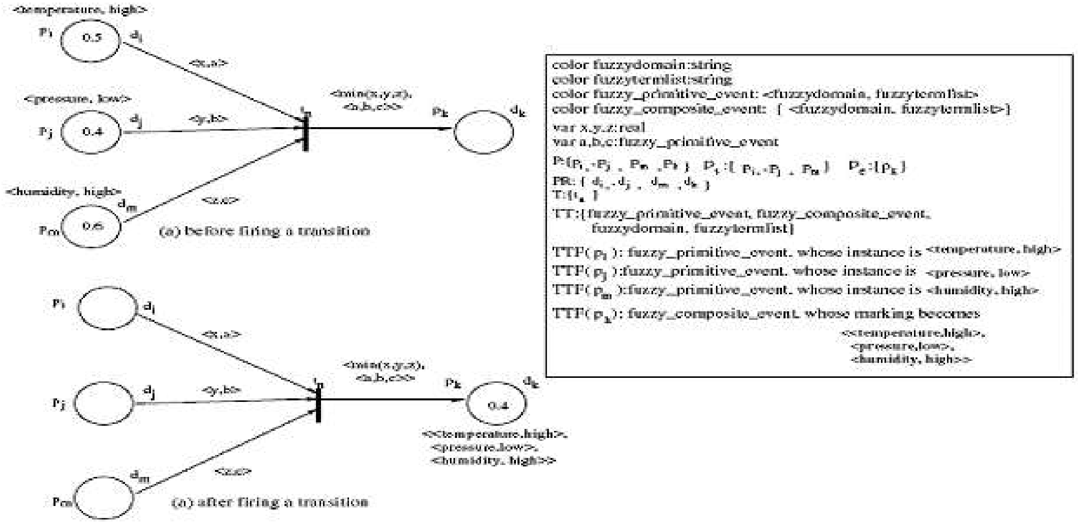

P1
a,a
d,d
2x
P2
<a,b>
<b,c>
<d,a>
<x,y>+<y,z>
t
<x,z>
x ≠ z
e
P3
P4
(a) before firing the transition
<a,c>
(a) after firing with substitutions {a|x,b|y, c|z}
<d,b>
(a) after firing with substitutions {d|x,a|y, b|z}

Fig. 2: Firing a Petri net


<temperature, high>
0.5
<pressure, low>
0.4
<humidity, high>
0.6
<x,a>
<y,b>
<z,c>
<min(x,y,z), <a,b,c>>
(a) before firing a transition
(a) after firing a transition
<<temperature,high>, <pressure,low>, <humidity, high>>

```
color fuzzydomain:string
color fuzzytermlist:string
color fuzzy_primitive_event: <fuzzydomain, fuzzytermlist>
color fuzzy_composite_event: { <fuzzydomain, fuzzytermlist>}
var x,y,z:real
var a,b,c:fuzzy_primitive_event
P:{Pi , Pj , Pm ,Pk }  PI :{ Pi , Pj , Pm }   PE :{ Pk }
PR: { di , dj , dm , dk }
T:{tn }
TT:{fuzzy_primitive_event, fuzzy_composite_event,
     fuzzydomain, fuzzytermlist}
TTF( pi ): fuzzy_primitive_event, whose instance is <temperature, high>
TTF( pj ):fuzzy_primitive_event, whose instance is <pressure, low>
TTF( pm ):fuzzy_primitive_event, whose instance is <humidity, high>
TTF( pk ): fuzzy_composite_event, whose marking becomes
                                        <<temperature,high>,
                                         <pressure,low>,
                                         <humidity, high>>
```

Fig. 3: Firing the fuzzy Petri net

11. TV:P$\rightarrow$[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.

A token value in place $p_i \in P$ is denoted by TV (p_i) $\in$ [0, 1]. If TV (p_i) = μ_i, $\mu_i \in$ [0, 1] and PPM (p_i) = d_i. This states that the degree of the truth of proposition d_i is $\in \mu_i$. A transition t_i is enabled if $\forall$ $p_i \in$ I (t_i), $\mu_i > 0$. If this transition t_i is fired, tokens are removed from input places I (t_i) and a token is deposited onto each of the output places O (t_i). Since we provide parameter passing, the token value of an output place $p_k \in$ O (t_i) is calculated from that of the input places I (t_i) using the transition function TF_i, where TF_i = TRTF (t_i). This token's membership value to the place p_k, (i.e., μ_k = TV (p_k)), is part of the token and gets calculated within the transition function TF_i, where $\mu_k = TF_i$ (I (t_i)).

Example. The fuzzy deductive rule (IF d_i and d_j and d_m THEN d_k) can be modeled as shown in Fig. 3. In this example, PPM (p_i) = d_i, PPM (p_j) = dj, PPM (p_m) = d_m, PPM (p_k) = d_k, TV (p_i) = μ_i = 0.5, TV (p_j) = μ_j = 0.4 and TV (p_m) = μ_m = 0.6. Since $\mu_i > 0$, $\mu_j > 0$ and $\mu_m > 0$, transition t_n is enabled and fired. Tokens are removed from I (t_n), which are p_i, p_j, p_m and deposited onto O (t_n), which is p_k. Suppose that the transition function of t_n, which is TF_n = TRTF (t_n), is defined as a min operator. Then the truth value of the output token (membership degree) is calculated as

$$TV(P_k) = TF_n\,(I(t_n)) = \min(\mu_i, \mu_j, \mu_m) = 0.4$$

TRANSFORMATION ALGORITHM

Before starting the meaning of transformation Algorithm it is necessary to introduce the meaning of

Table 2: The events and conditions calculated for the dream activity

Rule	Event	Condition
R1	e_1 is e_{11}	C_1

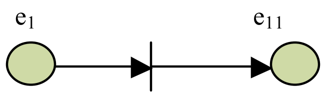


Fig. 4: Event representation of a R1

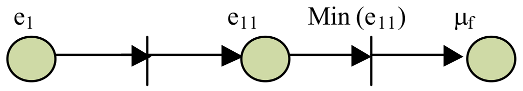


Fig. 5: Calculation of μ_{ef}

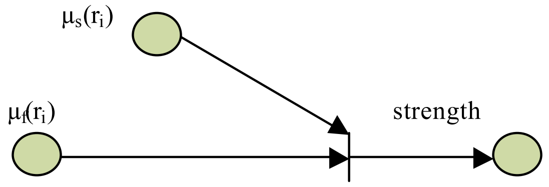


Fig. 6: Calculation of strength for R1

scenario. Scenario is a parameter that can divide the rules [23]. Only one of the states of this parameter can be active at a time. The substitution of the scenario is specified by the user. In the deduction cycle of fuzzy the strength of event e for rule, r in scenario s is calculated with formula (1) [23]:

$$\text{Strength}(e, r, s) = \mu_S(r) * \mu_{ef}(\text{value}(e_c)) \text{ formula} \qquad (1)$$

This uses scalar multiplication. Where value (e_c) is the value of the event (fuzzy or crisp) occurs. μ_{ef} is a function for the fuzz event e_f and $\mu_s(r)$ is like rule r for the current scenario s. $\mu_s(r)$ is defined as formula (2): $\mu_s(r) = \max([\ \min(\max(\mu(A_s, A_r)), \max(\mu(C_s, C_r))) * RLV_{rs} / RLV_{max}])$ formula (2) in which:

$$A_s \in S, \forall A_r \in R_i, C_s \in S, \forall C_r \in R_i, RLV_{rs}, RLV_{max} \in S$$

A comprises an event and condition of a rule and C includes an action and the result of a rule. A_S is the event and condition of the current scenario rule and A_r is the event and the condition of R_I rules (the rules which are analyzed). C_S is the action and the result of the current rule and C_R is the action and the result of the rules which are analyzed. RLV_{rs} is the amount of relation between the meta rule with the current scenario and RLV_{max} is the maximum amount from the relation amount.The Fuzzy UML activity diagram created will be transformed to a fuzzy Petri net according to the step below:

Step1: First for each activity change in this diagram, its event and conditions must be found. For each activity we derive its events and conditions. The events and conditions calculated for the dream activity is represented in Table 2.

Step 2: The highest level of division in the rules concluded is found and will be selected as the scenario. So, the rules are classified according to the scenario. In case of no scenario we assume it one. Thereby in our example we assume we don't have any scenario so it will be quantified by one.

Step 3: For each parameter defined in the rule we create a place where these parameters can't be repeated {and also can't be a scenario parameter}. Then for different kinds of state which these parameters can have in all of the rules we create a place. These places are joined to the proper places with a transition, as shown in Fig. 4.

Step 3: For each rule we provide a transition and then the events of each rule should be conducted to the provided transition, the function of this transition should be MIN. this function calculates μ_{ef} for each rule. This aspect is shown in Fig. 5.

Step 4: To calculate the strength of each event on the specified rule in an active scenario, first we have to calculate value of $\mu_s(r)$ using the formula (2). For each rule we create a transition which one of its inputs is a place which is initialized by the value $\mu_s(r_i)$ and the other input of the transition is the previous output of the transition that is μ_{ef} and its output is another place that comprises the event strength as an amount. This aspect is shown in Fig. 6.

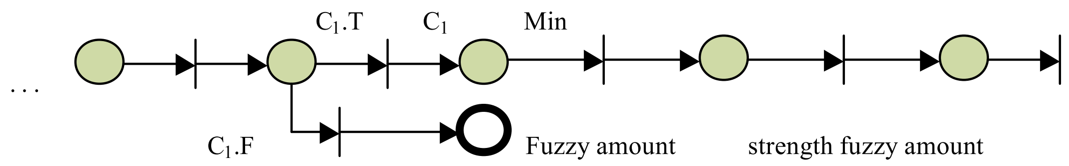


Fig. 7: Condition representation of a R1

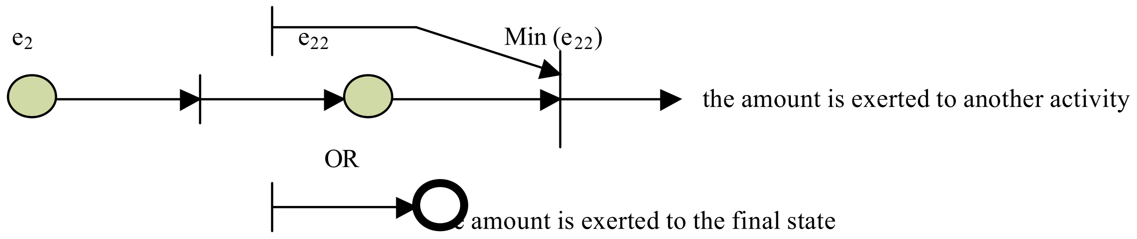

e2
e22
Min (e22)
the amount is exerted to another activity
OR
amount is exerted to the final state

Fig. 8: Final state representation for example

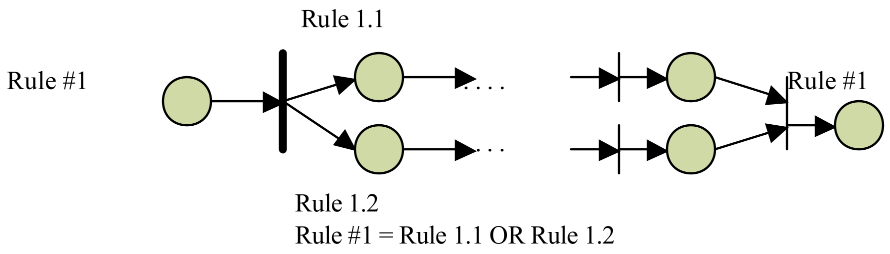


Fig. 9: Representation of OR Logic for a certain Rule

Step 5: We create a place for the condition of each rule and we valuate each condition with the fuzzy values calculated. This aspect is shown in Fig. 7.

Step 6: The result of this step which is a strength fuzzy amount will be exerted to the next activity or next state (final state for example).this aspect is shown in Fig. 8.

Sometimes we encounter OR logic, we solve this problem with the presented Fig. 9.

CASE STUDY

In this case study we are going to analyze a car sharer service system (Fig. 10)

The general activity diagram which yet has not fuzzy aspects is shown in Fig. 11:

Now gradually we are going transform each activity into fuzzy activity, now let's see how we can transform the first ordinary activity into fuzzy activity. The software should be able to match potential member requirements with all Car Match services in a particular geographic area. If the customer is not able to provide the requirements, it will sent out. This aspect is shown by an invalid exception in the transformation algorithm. It is obvious that a class is needed here when programming. The event and conditions of the first activity are derived as Fig. 12:

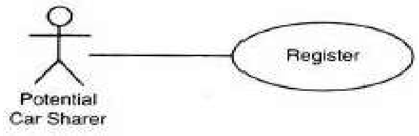


Fig. 10: Representation of user system transaction

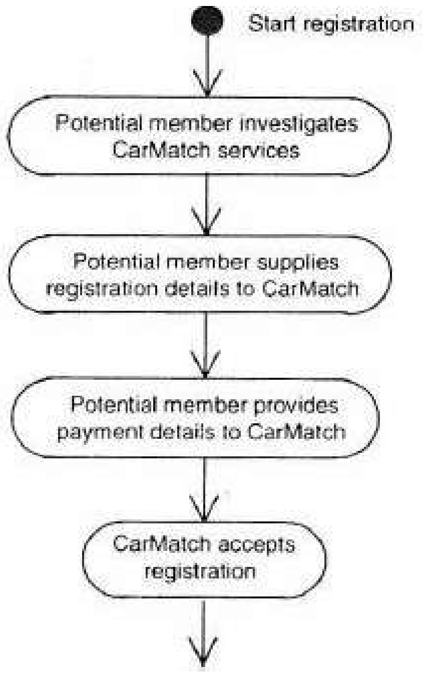
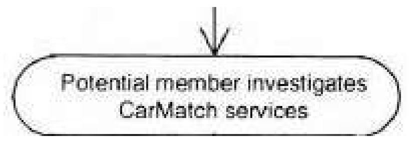

Start registration
Potential member investigates CarMatch services
Potential member supplies registration details to CarMatch
Potential member provides payment details to CarMatch
CarMatch accepts registration

Fig. 11: Normal activity diagram

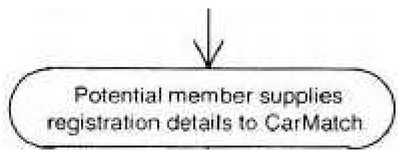
The registration form comprises some information that should be fulfilled by the customer, so these events and conditions are deduced by the designer (Fig. 13):

This stage is to some extent general so by dividing it, a new form of activity diagram will be derived, this rule comprises two parts because it has got two ways. This aspect is shown in Fig. 14. And in algorithm of transforming it is easy to implement it by using OR logic. The following rules and their events and conditions are deduced by the software designer:



Rule #1
Event: registration form will be given to the customer
Conditions: If the desired car is available
If the proposed money is enough
If the proposed period of time is valid
If the area is authorized
Action: the process of registration continues
[going to the next stage] ELSE invalid exception

Fig. 12: First activity with its Fuzzy rule

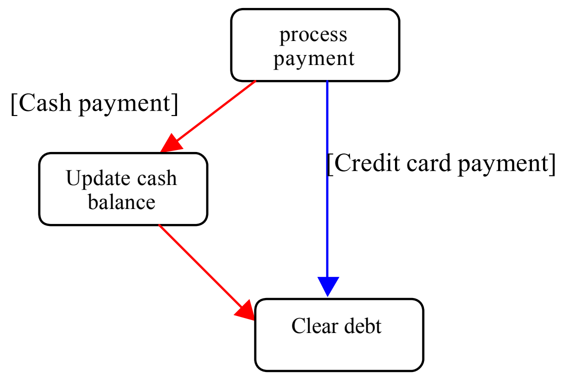


Rule #2
Event: fee form will be given to the customer
Condition: if the registration form has been
Completed by the customer successfully
Action: the process of registration continues
[going to the next stage] ELSE invalid exception

Fig. 13: Second activity with its Fuzzy rule

Rule #3.1	Rule #3.2
Event: clear debt process will be started Condition: if the amount is paid by a credit card Action: the process of registration continues [going to the next stage] ELSE invalid exception	Events: cash balance will be updated, clear debt process will be started Condition: if the amount is paid in cash Action: the process of registration continues [going to the next stage] ELSE invalid exception

Rule #3 =Rule #3.1 + Rule# 3.2



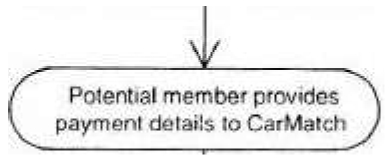
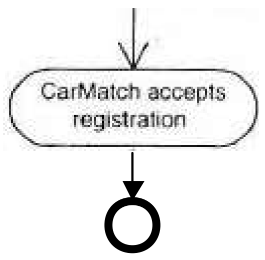


Fig. 14: Third activity with its rules



Rule #4
Event: final state will be started
Conditions: If the services are enabled for the member
If the member is notified
If the member details is added to the member list
If confirmation of payment is received
Action: exit

Fig. 15: Final activity with its Fuzzy rule

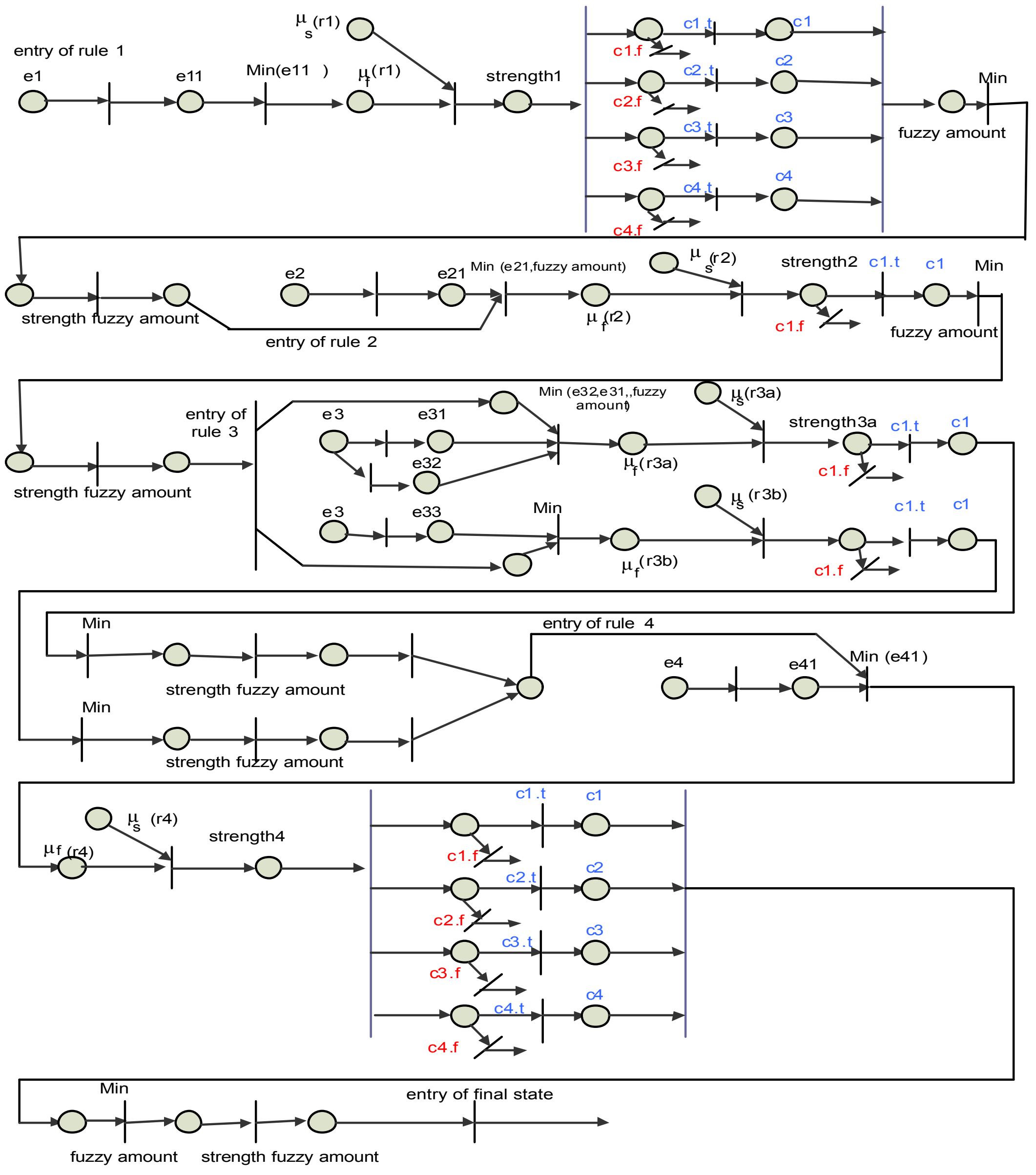


Fig. 16: Fuzzy Petri net for activity diagram in fuzzy UML

And at last the final activity will be done if its event occurs (Fig. 15):

The entire mapping is provided in Fig. 16:

CONCLUSION

In this paper we propose a method to transform activity diagram in fuzzy UML into fuzzy Petri net. Since activity diagram plays an important role in making and analyzing software by transforming it to Petri net which is a graphical and formal tool and adding fuzzy aspect, we will be able to analyze and sometimes debug the whole software better. By this approach following the work sequence of the software and sometimes the life cycle of an object in the software will be easier and through it some non-functional parameters of the software will be derivable [21,26].

FUTURE WORK

In our future work we will transform some other important UML diagrams to fuzzy Petri net such as sequence diagram and we will discuss some facts and aspects about them and we will pose new challenges on them.

REFERENCES

1. Faul M.B., 2004. Verifiable Modeling Techniques Using a Colored Petri Net Graphical Language. Technology Review Journal, spring/summer.
2. Shin, M., A. Levis and L. Wagenhals, 2003. Transformation of UML-Based System Model into CPN Model for Validating System Behavior. In Proc. of Compositional Verification of UML Models, Workshop of the UML'03 Conference, California USA, Oct. 21
3. Bernardi, S. S. Donatelli and J. Merseguer, 2002. From UML Sequence Diagrams and Statecharts to Analysable Petri Net Models. ACM Proc. Int'l Workshop Software and Performance, pp: 35-45.
4. Eshuis, R., 2002. Semantics and Verification of UML Activity Diagrams for Workflow Modelling. Ph.D Thesis, University of Twente.
5. Pettit, R.G. and H. Gomaa, 2002. Validation of dynamic behavior in UML using colored Petri nets' UML. (2000 , Zaragoza, Spain, pp: 295-302.
6. Saldhana, J. and S.m. Shatz, 2000. UML Diagrams to Object Petri Net Models: An Approach for Modeling and Analysis. Proc. of the Int. Conf. on Software Eng. and Knowledge Eng. (SEKE), Chicago10-103.
7. Elkoutbi, M. and Rodulf K. Keller, 1998. Modeling Interactive Systems with Hierarchical Colored Petri Nets. 1998 Advanced Simulation Technologies Conf., Boston, MA, pp: 432-437.
8. Bernardinello, L. and F. De Cindio, 1992. A Survey of Basic Net Models and Modular Net Classes. LNCS, Springer-Verlag, 609: 609.
9. Balsamo, S. *et al.*, 2004. Model-Based Performance Prediction in Software Development: A Survey. IEEE Transactions on Software Engineering, 30 (5): 295.
10. Merseguer, J., J.P. L´opezGrao and J. Campos, 2004. From UML Activity Diagrams To Stochastic Petri Nets:Application To Software Performance Engineering. ACM, WOSP 04 January 1416, 2004.
11. Fukuzawa, K. *et al.*, 2002. Evaluating Software Architecture by Colored Petri Net. Dept. of Computer Sience, Tokyo Institute of Technology Ookayama 2-12-1, Meguro, UK, Tokyo 152-8552 Japan 2002
12. Merseguer, J., S. Bernardi, J. Campos and S. Donatelli, 2002. A Compositional Semantics for UML State Machines Aimed at Performance Evaluation. Silva, M., A. Giua and J.M Colom (Eds.). Proc. of the th Intl. Workshop on Discrete Event Systems) WODES'02), Zaragoza, Spain, pp: 295-302.
13. Motameni, H. *et al.*, 2006. Mapping State Diagram to Petri Net: An Approach Tousemarkov Theory For Analyzingnon-Functional Parameters. IEEE, International Conferenceon Computer, Information and System Science, December 4_14 2006, University of Bridgport, USA (presented).
14. Motameni, H. *et al.*, 2006. Using Markov Theory For Deriving Non-Functional Parameters On Transformed Petri Net From Activity Diagram. Proc of software engineering conference (Russia), 16-17 November 2006, Moscow, Russi, (Presented).
15. Motameni, H., M. Zandakbari and Movaghar, 2006. Deriving performance parameters from the activity diagram using gspn and markov chain. ICCSA 2006 Proceeding of 4th International Conference on Computer Science and Its Aapplications, San Ddiego,California.
16. Motameni, H. *et al.*, Evaluating UML State Diagrams Using Colored Petri Net" SYNASC'05.
17. Motameni, H. *et al.*, 2005. Verifying and Evaluating UML Activity Diagram by Converting to CPN. Proc. of SYNASC'05, Romania, Sep 2005, (presented).
18. Object Management Group, UMLTM Profile for Schedulability, Performance and Time Specification, OMG Document, Version 1.1, January 2005.
19. Rumbuaugh, J., M. Blaha, W. Premerlani, F. Eddy and W. Lorensen, 1991. Object-Oreinted Modeling and Design. Prentice hall, Englewood Cliffs, NJ, USA.
20. Wang, lu, 2005. Fuzzy UML. Seminararbeit, Sommersemester.
21. Zongmin, Ma., 2005. Fuzzy Information Modeling With the Uml. Idea.
22. Ma, Z.M., 2004. Extending UML For Fuzzy Information Modeling In Object_Oriented Database. Theories and Practices, Idea Group Publishing.
23. Burcin Bostan-Korpeoglu and Adnan Yazici, 2006. A Fuzzy Petri Net Model For Intelligent Database, Data and Knowledge Engineering (2006), Elsevier.
24. Nihal, Y. Ö., 2007. On the Numbers of the Form n = x2 + Ny2, World Applied Sciences Journal, 2(1): 45-48.
25. Erçetin, S.S., Çetin, B. and N. Potas, 2007. Multi-Dimensional Organizational Intelligence Scale (Muldimorins), World Applied Sciences Journal, 2(3): 151-157.
26. Hayati, M., Karami, B. and M. Abbasi, 2007. Numerical Simulation of Fuzzy Nonlinear Equations by Feedforward Neural Networks. World Applied Sciences Journal, 2 (3): 229-234.