Contents lists available at GrowingScience

Management Science Letters

homepage: www.GrowingScience.com/msl

Evaluation of software architecture using fuzzy colored Petri nets

Vahid Abroshan[a*], Ali Harounabadi[b] and Seyed Javad Mirabedini[b]

[b]Department of Computer, Science and Research Branch Bushehr, Islamic Azad University, Bushehr, Iran
[a]Department of Computer, Central Tehran Branch, Islamic Azad University, Tehran, Iran

C H R O N I C L E

Article history:
Received September 29, 2012
Received in revised format
19 November 2012
Accepted 29 November 2012
Available online
December 1 2012

Keywords:
Performance Evaluation
Software Architecture
Evaluation
Fuzzy Colored Petri Nets
Fuzzy UML

A B S T R A C T

Software Architecture (SA) is one of the most important artifacts for life cycle of a software system because it incorporates some important decisions and principles for the system development. On the other hand, developing the systems based on uncertain and ambiguous requirement has been increased, significantly. Therefore, there have been significant attentions on SA requirements. In this paper, we present a new method for evaluation of performance characteristics based on a use case, response time, and queue length of SA. Since there are some ambiguities associated with considered systems, we use the idea of Fuzzy UML (F-UML) diagrams. In addition, these diagrams have been enriched with performance annotations using proposed Fuzzy-SPT sub profile, the extended version of SPT profile proposed by OMG. Then, these diagrams are mapped into an executable model based on Fuzzy Colored Petri Nets (FCPN) and finally the performance metrics are calculated using the proposed algorithms. We have implemented CPN-Tools for creating and evaluating the FCPN model.

1. Introduction

Software Architecture (SA) is one of the most important artifacts for life cycle of a software system because it incorporates some important decisions and principles for the system development. SA deals with structural issues, which are becoming more important as the size and complexity of software systems increase, substantially over the past two decades. SA can be defined as structure or structures of some system(s), which comprise software elements, the externally visible properties of those elements and the relationships among them (Bass et al., 2003).

This definition concentrates only on the internal aspects of a system and most of the analysis methods are based on this definition (Balsamo & Maraolla, 2005). Another brief definition establishes SA as "the structure of components in a program or system, their interrelationships, and the principles and guides that control the design and evolution in time". According to these definitions, it is obvious that

*Corresponding author.
E-mail addresses: v.abroshan@yahoo.com (V. Abroshan)

doi: 10.5267/j.msl.2012.12.001

SAs describe software system structures at a high level of abstraction. By SA, we mean the components into which a system is categorized at the level of system organization, and the ways in which those components communicate, interact, and coordinate with each other.

For SA documentation, various ADLs have been designed, however, most ADLs are conceptually based on the structural architecture primitives of components, connectors, interfaces and configurations. ADLs can be grouped into two categories including the specially designed ADLs such as C2, Unicon or ACME, and those ADLs based on Unified Modeling Language (UML) (Perez-Palcin & Merseguer, 2010). Since UML is an effective diagrammatic notation used to capture high-level designs of systems, it can represent SAs in various diagrams, thus in this paper we use UML as an ADL. UML can represent architectural styles and architectural properties by using its extension mechanisms (e.g., tagged values, stereotypes, constraints and profiles). Although UML has extended substantially but it suffers from some issues, which are as follows:

a- Because uncertain information is widely implemented in software systems, it must be investigated as a critical problem in modeling SAs. Unfortunately, UML cannot handle imprecise and uncertain information.
b- Because UML is not a formal model, we cannot create an executable model of SA from UML diagrams so SAs evaluation in not possible by it, directly. Therefore, for performance evaluation of SAs, we must transform pragmatic model to formal model.

To conquest the first issue, we have entered Fuzzy Logic concepts in UML and we have imparted UML as F-UML. With F-UML, we can use imprecise and uncertain information to model SAs. In addition, we attempt to handle the second one based on a formal model named CPN (Jensen & Kristensen, 2009). CPN is high-level Petri net implemented to generate an executable model of SAs. Since fuzzy concepts for modeling SAs in this paper is applied, we can implement an extension version of CPNs called FCPN. With FCPN, it is possible to implement fuzzy rules in modeling SAs.

One of the major problems many designers face with is on how to choose an appropriate SA among different SAs. Analysis of non-functional factors such as performance, reliability, security, accessibility, reusability, etc. of software systems at the SA design level has received much attention as a means to determine a system's potential problems such as system's incompleteness and inconsistency. This would help reduce development cost and time while improving quality. Therefore, it is especially necessary to make an assessment on SA based on quality attributes to make sure that the resulting software satisfies all of the stakeholders' requirements as much as possible. Among non-functional factors, performance is one of the most influential ones, which must be taken into consideration. Performance evaluation of software system at SA design level can help software architecture find out the responses on different questions such as: (i) which components are critical to the performance of the application? and (ii) how is the application performance influenced by performance of individual components?

During the past few years, the quantitative analysis of SAs has been a topic of interest. For this purpose, many researchers (Cooper et al., 2005; Balsamo et al., 2002; Balsamo & Maraolla, 2005) have used different methods to analyze non-functional parameters of SAs, especially performance. In such works, various methods have been used for description of SAs such as UML diagrams, Labeled Transition System, ADLs, etc. In addition, various works have been imparted different formal methods including Petri Nets(PNs), CPN, Queuing Network Model(QNM), process algebras, Markov Chain, etc. for deriving performance models, which could be analyzed to performance evaluation of SAs. For example, Balsamo et al. (2002) has indicated how QNM with blocking could be applied into the performance evaluation and prediction of SAs. For this purpose, after that UML message sequence chart has been implemented as behavioral SA description, a QNM model has derived from the SA description and then the authors have used a scenario-based technique to parameterize and evaluate the obtained QNM model. Cooper et al. (2005) tried to model and to analyze performance aspects of SA with a UML based approach called FDFA. This approach integrates part of UML and a set of existing formal methods, Rapide and Armani, into an aspect-oriented framework at the architecture design level. It presents the results of using Armani to analyze the resource utilization aspect in a UML based design as well as using Rapide to simulate the response time aspect of SA. Balsamo and Maraolla (2005) proposed an approach based on QNM for performance prediction of SA. The approach starting from UML use case, Activity, Deployment diagrams annotated with UML-SPT profiles and then it has derived performance models based on QNM, which could then be analyzed using standard solution techniques

In addition, the works mentioned above, several works have been proposed to transform automatically UML-SPT models, i.e. UML models enriched with performance annotations, into performance models using various kinds of Petri Nets. For instance, Bernardi and Merseguer (2007) used performance annotated UML diagrams (e.g. State Machine, Sequence Diagram, Interaction Overview Diagrams) for deriving performance models using Stochastic Well-formed Nets. Perez-Palcin and Merseguer (2010) tried to evaluate performance of Self-reconfigurable Service-oriented software systems with Stochastic Petri Nets (SPNs). In this work, the authors have used UML Component and Deployment diagrams. Lopez Grao et al. (2004) used UML activity diagram to derived performance model using SPN.

Besides, Staines (2008) proposed a solution to conquest the problem of translating UML activity diagram into PNs. The objective was to translate the UML activity diagram into a fundamental modeling concept PN diagram compact notation, this can be converted to a CPN for execution and validation. Lian-Zhang and Fang-Sheng (2012) used an intermediate model to transform UML-SPT models to CPN for software performance evaluation. Hong-Xia and Lial-Zhang (2009) proposed a UML-CPN transformation method to build dynamic model in UML using CPN. This work aims to develop efficient transformation methods, and then uses CPN models to simulate, verify and evaluate the systems. Akbari et al. (2010) and Motameni et al. (2008) tried to convert some UML diagrams using FCPN.

Although most of the researches have tried to model SAs with UML diagrams enriched with SPT profiles and create an execution model with formal models to performance evaluation of SAs, but none of them used any uncertain and ambiguous performance information. It means that the mentioned researches did not have any imparted fuzzy logic theory subjects such as fuzzy rules, linguistic variables, etc. for modeling UML-SPT and formal models. The things that distinct our work from other works is to use fuzzy logic for performance analyze of SAs. Therefore, in this paper, for performance evaluation of SAs, first we have described SAs using UML diagrams, use case diagram (UCD), sequence diagram (SD), and F-UML extensions mechanisms. Then we use a fuzzy version of UML Profile for Schedulability, Performance and Time Specification (SPT) (Object Management Group, 2005), called F-SPT, for representation of performance properties. After that, with FCPN we create an executable model from evolved F-UML model and finally we evaluate performance characteristics of SA using proposed algorithm. We also use CPN Tools to model and evaluate SA.

The rest of this paper is organized as follows. In section 2, some background information, including UML and SPT, fuzzy UML, fuzzy concepts, a brief overview of CPNs and FCPNs are presented. In section 3, fuzzy use case diagram and fuzzy sequence diagram have been investigated. Also, two algorithms to mapping each of these diagrams to FCPN model are presented. Section 4 and 5 include the proposed algorithms to calculate response time and queue length. Section 6 provides one illustrative example. Finally, Section 7 concludes the paper.

2. Background

In this section we review some information on UML, F-UML and SPT, fuzzy logic concepts, CPNs and FCPNs.

2.1. UML and SPT overview

The Unified Modeling Language (UML) is a general concept, object oriented, visual modeling language designed to specify, visualize, construct and document the artifacts of a software system, rapidly becoming an official language for modeling object-oriented systems. One of the reasons motivating the success of UML is its flexibility, which allows the model designer to take advantage of, to arrange the diagrams in multiple ways and to consider various levels of abstraction from multiple points of view. This consideration regards in particular those diagrams defined for the description of the dynamic behavioral aspects of the system.

UML can be extended or adapted to a specific method, organization, or users and it provides different solutions to extend itself. Profiles are stereotyped packages that contain model elements customized for a specific domain or purpose, by extending the metamodel with stereotypes, tagged definitions and constraints. More specifically, the UML Profile for Schedulability, Performance and Time Specification (SPT) (Object Management Group, 2005) is an OMG standard profile adopted for the performance annotations in the UML model. This profile is concerned with time properties and aspects related to time, such as schedulability and performance.

2.2. Fuzzy UML Overview

The technology implemented in UML is useful for certain problems and uncertainly is considered in many software systems. These systems resolve the user’s requirements and uncertainly can be considered as user requirements natural essence. Therefore, if we enter the uncertainly in UML, the causes of better exploitation will be prepared.

Ma et al. (2011) extended UML class diagram to model fuzzy information. The class diagrams in UML are the logical models. They described the system's main structure. The classes and the relationships among them consist of the elements in class diagram. By entering the uncertainty into these elements, the F-UML data model is produced. In the context of classes, three levels of fuzziness are defined as follows (Haroonabadi & Teshnehlab, 2008):

1-Fuzziness in some extent where class belongs to some data model as well as fuzziness on the content (in term of attributes) of the class,
2-Fuzziness related to whether some instances belong to a class; even though the structure of a class is crisp, it is possible that an instance of the class belongs to the class with special degree of membership
3-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 should be described by the phrase of *WITH mem DEGREE* where, $0 \leq mem \leq 1$. This value demonstrates the degree of membership 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 where it belongs to the class should be specified. So an additional attribute in the class is defined for representation of the instance membership degree to the class where its domain is [0, 1]. This special attribute is specified with μ. The classes with the second level of fuzziness have specified by a rectangle where its lines are dash. In the third level, a *fuzzy* keyword is appeared in

front of the attribute. 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 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*"

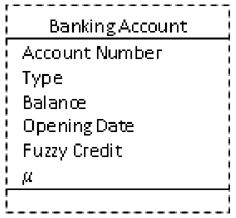


Fig.1. A fuzzy class of banking account

Finally, μ attribute specifies the membership degree of a class instance to the class (the second level of fuzziness). The relationships among the classes are divided into four categories and they are propounded fuzzily (Ma et al., 2011): fuzzy generalization, fuzzy association, fuzzy aggregation and fuzzy dependency.

2.3. Fuzzy logic, fuzzy sets and linguistic variables

Fuzzy logic is an approach for computing based on "degrees of truth" rather than the usual "true or false". In this approach, variables can have a true value, which ranges in degree between 0 and 1. Fuzzy logic approach has some important concepts like if-then rules, linguistic variables, fuzzy sets, etc.

Fuzzy systems are knowledge-based or rule-based systems. The heart of a fuzzy system is a knowledge base constructed using fuzzy if-then rules. A fuzzy if-then rule is an if-then phrase. In this phrase, some words are specified using membership functions. These words are known as linguistic variables. Linguistic variables are variables whose values are not numbers but words or sentences in a natural or artificial language. For example, variable *Speed* is a linguistic variable, which can choose some values in a fuzzy set: {Slow, Medium, Fast}. Each linguistic variable has a degree of membership, which determines amount of belonging of that variable to a fuzzy set. These memberships are determined using membership functions, which attempt to describe vagueness and ambiguity. In addition to the mentioned concepts about fuzzy logic, a fuzzy set F can be described as follows (Ma et al., 2011):

Let U be a universe of discourse, then a fuzzy value on U is characterized by a fuzzy set F in U. A membership function $\mu_F\colon U \rightarrow [0,1]$ is defined for the fuzzy set F, where $\mu_F(U)$, for each $u \in U$, denotes the degree of membership of u in the fuzzy set F. Thus, the fuzzy set F is described as follows,

$$\mathrm{F} = \left\{\frac{\mu(u_1)}{u_1}, \frac{\mu(u_2)}{u_2}, \ldots, \frac{\mu(u_n)}{u_n}\right\}, \tag{1}$$

where the $\mu_F(U)$ above is explained to be a measure of the possibility that a variable X has the value u in this approach, where X takes values in U, a fuzzy value is described by a possibility distribution $\pi_\chi = F$.

2.4. Coloured Petri Nets

Coloured Petri Nets (CP-nets or CPNs) are classes of high-level nets, which extend ordinary Petri nets. CPNs is a graphical language for constructing models of concurrent systems and analyzing their properties. CP-nets is a discrete-event modeling language combining the capabilities of Petri nets with the capabilities of a high-level programming language. Petri nets incorporate the basis of the graphical notation and the basic primitives for modeling concurrency, communication, and synchronization. In CPNs, tokens can carry arbitrarily complex data, arcs can be annotated with input inscriptions influencing the enabling of a transition, or output inscriptions stating the production rule of tokens when a transition fires (Jensen, 1993). A coloured Petri net is a 9-tuple, $CPN = (\Sigma, P, T, A, N, C, G, E, I)$, where,

- Σ is a finite set of non-empty types, also called color sets.
- P is a finite set of places.
- T is a finite set of transitions.
- A is a finite set of arcs such that: $P \cap T = P \cap A = T \cap A = \emptyset$.
- N is a node function. It is defined from A into $P \times T \cup T \times P$.
- C is a colour function. It is defined from P into Σ.
- G is a guard function. It is defined from T into expressions such that: $\forall t \in T$: $[Type\big(G(t)\big) = B \wedge Type(Var\big(G(t)\big) \subseteq \sum]$, where B to denote the Boolean type.
- E is an arc expression function. It is defined from A into expressions such that: $\forall a \in A$: $[Type\big(E(a)\big) = C(p)_{MS} \wedge Type(Var\big(E(a)\big)) \subseteq \sum]$, where p is the place of $N(a)$.
- I is an initialization function. It is defined from P into closed expressions such that: $\forall p \in P$: $[Type\big(I(p)\big) = C(p)_{MS}]$.

In the definition of CPNs, the concrete syntax for writing net expressions is not fixed. Declarations and net inscriptions can possibly be expressed in many various languages, e.g., by means of standard mathematical notations or by means of ordinary high- level programming languages.

CPN Tools (Jensen & Kristensen, 2009) is a well-known tool, which enables modeling, verifying and analyzing of CPNs. CPN Tools is an industrial strength computer facility for building and analyzing CPN different models. CPN Tools makes it possible to study the behavior of the modeled system using a simulation to verify properties by means of state space methods and model checking, and to conduct a simulation-based performance analysis.

As we mentioned above, CPNs provide a powerful formal modeling method based on a solid mathematical structure while having graphical representation of system models as net diagrams. However, CPNs have a lot of limitations requiring the provision of exact and precise description of the system. It may not be able to model incomplete, uncertain, and approximate information or states. As the popularity of fuzzy reasoning grows in certain kinds of manufacturing processes, it is necessary to extend CPNs to incorporate fuzzy logic in such processes. Therefore, Fuzzy Coloured Petri Nets (FCPNs), a model, which is able to represent the fuzzy production rules of a rule based system, is the ideal tool to aid such type of manufacturing system development. A formal definition of a FCPN is as follows (Yeung et al., 1996):

A generalized non-hierarchical FCPNs is defined as 12-tuple FCPN = (Σ, P, T , D, A, N, C, G, E, β, f , I) where:

- $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_l\}$ denotes a finite set of non-empty types, called color sets where $l \geq 0$.
- $P = \{P_C, P_F\}$ denotes a finite set of places;

- $P_C = \{pc_1, pc_2, \ldots, pc_m\}$ denotes a finite set of places that model the dynamic control behaviour of a system, and is called control places where m ≥ 0;
- $P_F = \{pf_1, pf_2, \ldots, pf_n\}$ denotes a finite set of places that model the fuzzy production rules, and is called fuzzy places where n ≥ 0, and $P_C \cap P_F = \emptyset$.
- $T = \{T_C, T_F\}$ denotes a finite set of transitions;
- $T_C = \{tc_1, tc_2, \ldots, tc_i\}$ denotes a finite set of transitions that are connected to and from control places, and is called control transition where i ≥ 0;
- $T_F = \{tf_1, tf_2, \ldots, tf_j\}$ denotes a finite set of transitions that are connected to or from fuzzy places, and is called fuzzy transition where j ≥ 0, and $T_C \cap T_F = \emptyset$.
- $D = \{d_1, d_2, \ldots, d_h\}$ denotes a finite set of propositions, $|P_F| = |D|$.
- $A = \{a_1, a_2, \ldots, a_k\}$ denotes a finite set of arcs, k ≥ 0, and $P \cap T = P \cap A = T \cap A = \emptyset$.
- $N: A \rightarrow P \times T \cup T \times P$ denotes a node function, and it maps each arc to a pair, where the first element is the source node and the second element is the destination node; the two nodes have to be of different kinds;
- In: an input function that maps each node, x, to the set of nodes that are connected by an input arc(x) → x;
- Out: an output function that maps each node, x, to the set of its nodes that are connected to x by output arc(x) → x.
- $C: (P \cup T) \rightarrow \Sigma_{ss}$ is a color function, which maps each place and transition to a super-set of colour sets.
- $G: T \rightarrow expression$ which denotes a guard function, $\forall t \in T : [Type(G(t)) = Boolean \wedge Type(Var(G(t))) \subseteq \Sigma]$, where Type (Vars) denotes the set of types, $\{Type(v) | v \in Vars\}$. $Vars$ denotes the set of variables, and $Var(G(t))$ denotes the set of variables used in $G(t)$.
- $E: A \rightarrow expression$ which denotes an arc expression function, $\forall a \in A : [Type(E(a)) = C(p(a))MS \wedge Type(Var(E(a))) \subseteq \Sigma]$, where $p(a)$ is a place in $N(a)$, and MS stands for multi-set.
- $\beta: PF \rightarrow D$ denotes a bijective mapping from fuzzy places to a proposition.
- $f : T \rightarrow [0, 1]$ denotes an association function, which assigns a certainty value to each color used in each fuzzy transition.
- I: denotes an initialization of double (δ, α),
- $\delta: P \rightarrow expression$ which denotes an initialization function: $\forall p \in P : [Type(\delta(p)) = C(p)MS]$.
- α: denotes an association function, which assigns a certainty value in the range [0, 1] to each token in the fuzzy places.

3. Fuzzy use case and fuzzy sequence diagrams

In this section, fuzzy use case diagram (FUCD) and fuzzy sequence diagram (FSD) are introduced. In addition, the algorithm to convert each of these diagrams to FCPN model will be presented.

3.1. The role of UCD concerning evaluation of SA

A UCD as first view in design of SA, models user usage from a system. Lots of UCs are used in designing of SA, but for performance evaluation, software architect must choose a subset of them. This choice is accomplished according to type of system and recognition of important UCs. A UC can use for describing requirements of a system, a subsystem or a class and it can describe functionality of them. However, since in fuzzy systems the requirements are uncertain and ambiguous, the services are expressed as fuzzy and therefore fuzzy use cases (FUC) are proposed. For representation of a FUC we use a dotted line ellipse as shown in Fig. 2.

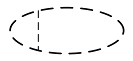


Fig.2. FUC symbol

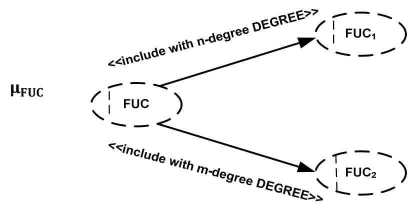

μFUC
<<include with n-degree DEGREE>>
FUC₁
FUC
<<include with m-degree DEGREE>>
FUC₂

Fig.3. Include relationship in FUCD

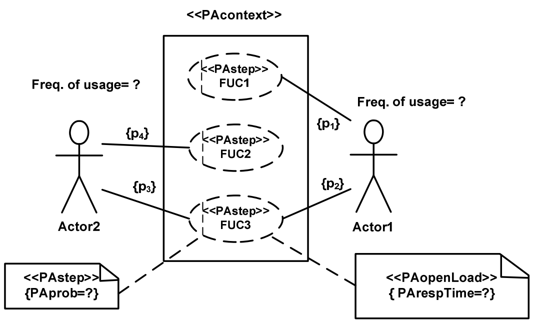


Fig.4. fuzzy use case diagram

In a FUCD, each UC has a membership degree (μ_{FUC}) showing percentage of user usage from provided services of the UC. Fig. 3 represents a FUCD that has been constructed according to three fuzzy rules as follows:

IF d_1 THEN d_2 : (μ_i) → Fuzzy use case d_2 = FUC
IF d_2 AND d_3 THEN d_4 : (μ_j) → Fuzzy use case d_4 = FUC_1
IF d_2 AND d_5 THEN d_6 : (μ_k) → Fuzzy use case d_6 = FUC_2

In fig. 3, fuzzy use case *FUC* uses *n* degree from fuzzy use case *FUC_1* and *m* degree from fuzzy use case *FUC_2*. Suppose to have a UCD with *m* users and *n* UCs. Let *pi(i=1,2,…,m)* be the *i-th* user makes use of the UC *j(j=1,2,…,n)*. The probability of a SD corresponding to the UC *x* to be executed is (Merseguer, 2003):

$$P(x) = \sum_{i=1}^{m} p_i \cdot p_{ix} \tag{2}$$

In fuzzy systems, the above probabilities are considered as linguistic variables, which can choose a value from a set of linguistic values. In addition, tagged values can behavior like mentioned variables. Therefore, the probability of user usage from a FUC is expressed as linguistic terms (Fig. 4). Suppose that the probability of user1 usage from FUC3 and user2 usage from FUC3 is represented by P_2 and P_3 , respectively. Also assume that user2 and user1 frequency of usage of system are represented by FOU_1 and FOU_2 , respectively. Also, suppose that one of the fuzzy rules for FUC3 in Fig. 4 is like following rule:

IF FOU1 is FS1 AND p2 is FS2 AND FOU2 is FS3 AND P3 is FS4, THEN PAprob is X.

For calculate amount of *X*, we use from Eq. (2). Therefore:

$$X = \mu_{FS_1}(FOU_1).\mu_{FS_2}(P_2) + \mu_{FS_3}(FOU_2).\mu_{FS_4}(P_3) \tag{3}$$

After specifying all fuzzy rules of the system, we can calculate output of fuzzy systems (PAprob) using product inference engine, single fuzzification and center of average defuzzification with following,

$$f(x) = \frac{\sum_{l=1}^{M} y^{-l}\left(\prod_{i=1}^{n} \mu_{A_i^l}(x_i)\right)}{\sum_{l=1}^{M}\left(\prod_{i=1}^{n} \mu_{A_i^l}(x_i)\right)} \tag{4}$$

3.2. Mapping UCD to FCPN

To convert a UCD to FCPN, all actors, FUCs as well as relationship between actors and FUCs should be mapped to FCPN model. In FCPN model, UCs and actors are represented with places. Transitions guards determine those conditions that specify when an actor can call a UC. After execution of a UC the results should return to that UC or actor call executed UC, for this purpose a transition is added to the model.

3.3. The role of SD concerning evaluation of SA

Architecture is the structure of the components of a program or system, their interrelationships, and the principles and guidelines governing their design and evolution over time. On the other hand, the main elements in a SD includes objects (components), messages that determine how system's components are associated with each other and other information that add to objects and components using different profiles or Object Constraint Language (OCL). Therefore, a SD has all the properties to represent SA. In addition, this diagram can cover process view in the 4+1 view model of SA. If a UC is fuzzy, the corresponding SD is considered as fuzzy. In a fuzzy SD (FSD) we have two levels of fuzzyness: 1) level one: the method belongs to an object with μ membership degree. 2) level two: the method in essence is fuzzy.

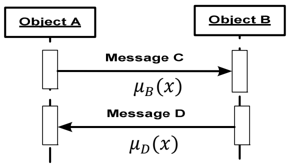


Fig.5. Different levels of method fuzziness

As an example, in Fig. 5, message C may belong to object B with $\mu_B(x)$ membership degree (first level of fuzziness) or message D can be defined as fuzzy with $\mu_D(x)$ membership degree (second level of fuzziness). As mentioned earlier, a SD can show message passing between those objects have participated in a scenario. These objects can reside in the same machine or in different machines in the case of distributed systems. In the first case it can be assumed that the time spent to send the message is not significant in the scope of the modeled system. For the second case, those messages that travel through the net, it is considered that they spend time, then supposing a load for the system that should be modeled.

3.4. Mapping a FSD to FCPN model

Since a SD is a collection of fuzzy or non-fuzzy methods, which exchange among those objects have participated in a scenario. With regard to this rule that a fuzzy method can execute when some conditions are true, first each method is mapped to FCPN with regard to corresponding fuzzy if-then rules and then all the resultant nets from each method are connected to each other and form a single net. The mapping algorithm for convert fuzzy method to FCPN is accomplished in three steps (Akbari et al., 2010):

3.4.1 First step:

This step includes specifying all linguistic variables that are required for calculating the studying metrics and determine all membership function for them. Then, software architect can form a fuzzy system consisting of fuzzy if-then rules with regard to system's requirements. These rules are defined for each fuzzy message (method). In addition, system states are specified after the conditions are true. For example, some conditions need for sending method *m_1* is represented in Table 1.

Table 1
Conditions for sending method m1

Rule	Event	Condition	State
R_1	v_1 is fs_{1-1}	C_1	S_1
R_2	v_1 is fs_{1-1} v_2 is fs_{2-1}	C_1 AND C_2	S_2
R_3	v_1 is fs_{1-3}	C_3	S_3

The if-then rules for Table 1 are as follows:

C_1: IF v_1 is fs_{1-1} THEN …
C_2: IF v_2 is fs_{2-1} THEN …
C_3: IF v_1 is fs_{1-3} THEN …

Since the algorithm considers two places and one transition for each simple fuzzy rule for mapping above rules to FCPN, we determine one place for each linguistic variable. Also for each fuzzy set that the variable can choose a value from them, one place is specified. For example, in above rules, variable v1 is represented with place P_{v1}. In addition, fuzzy sets fs_{1-1}, fs_{1-2}, fs_{1-3} corresponding to this variable show with three different places (P_{fs1-1}, P_{fs1-2}, P_{fs1-3}). For simplification, we add an initial place $P_{initial}$ to the net. In this place, for each linguistic variable exists in each method we consider one token. These tokens carry two colors: a fuzzy value and a crisp value. These tokens will add to the places related to linguistic variable with regard to those conditions that are specified in exp functions of outputs arcs from the initial place. Fig. 6 shows the FCPN model according to above rules.

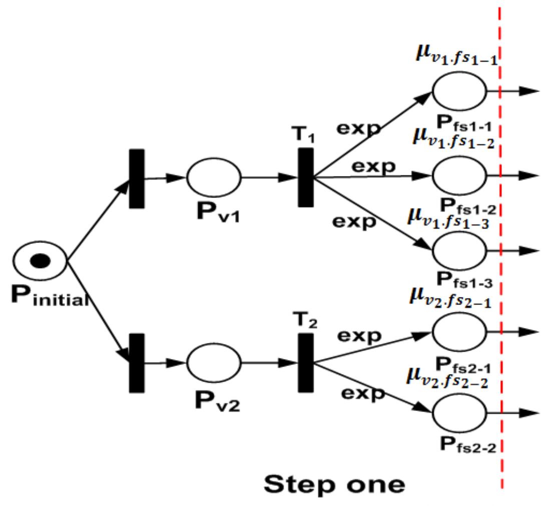


Fig.6. First step of mapping algorithm

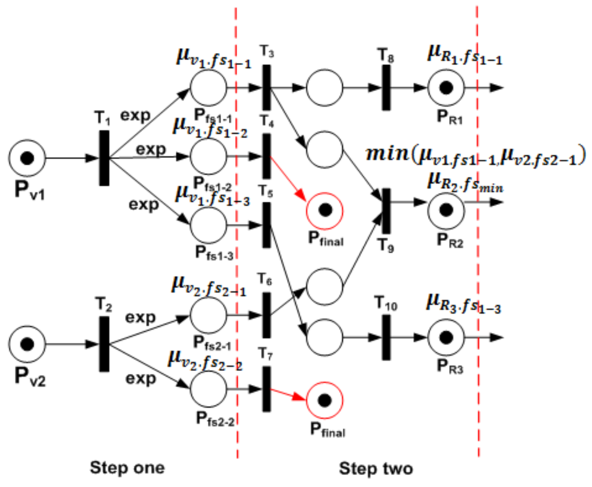


Fig.7. Second step of mapping algorithm

Each linguistic variable represents with a token in place its own. This token carries a crisp value as a color. After firing the transition related to linguistic variable, the tokens that exist in linguistic variable's place are removed. Then with regard to membership function placed in exp function on output arc from the transition, a membership degree for a crisp value will be calculated and will be added to the token that is placed in places related to fuzzy sets as a new color.

3.4.2 Second step: Create fuzzy if-then rules

In this step, with regard to the predicates that have constructed in previous step, the rules will be formed. For accomplishment of this work, first the tokens that can continue their life are specified. For example, however fuzzy set fs_{1-2} exist in Table 1 but there is no rules that use from this fuzzy set. Therefore, the token that is placed in P_{fs2-1} cannot continue its life and will carry to final place P_{final}

after firing transition T_4. The tokens resided in places associated with rules, have a new color that specifies accuracy of a rule. If fuzzy predicates connect to each other using AND operator, for calculating accuracy of a rule we use t-norm of the fuzzy sets associated with these predicates. In addition, if fuzzy predicates connect to each other using OR operator, it is possible to use s-norm to calculate the accuracy. Fig. 7 shows the second step of the proposed algorithm.

3.4.3 Third step: calculating output of fuzzy system

With regard to this fact that in this paper we have used from center of average defuzzification, the crisp output of fuzzy system calculates using following formula:

$$y = \frac{\bar{y}_1\mu_{R1} + \bar{y}_2\mu_{R2} + \bar{y}_3\mu_{R3}}{\mu_{R1} + \mu_{R2} + \mu_{R3}} \tag{5}$$

$$\bar{y}_1 = \text{center of } \mathrm{fs}_{1-1} \quad , \bar{y}_2 = \text{center of } \mathrm{fs}_{\min} \quad , \quad \bar{y}_3 = \text{center of } \mathrm{fs}_{1-3}$$

Fig. 8 shows the third step of the algorithm.

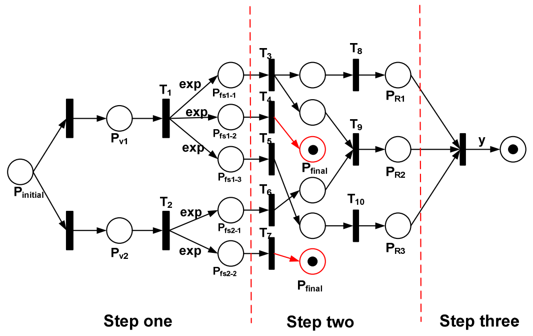


Fig.8. Third step of mapping algorithm

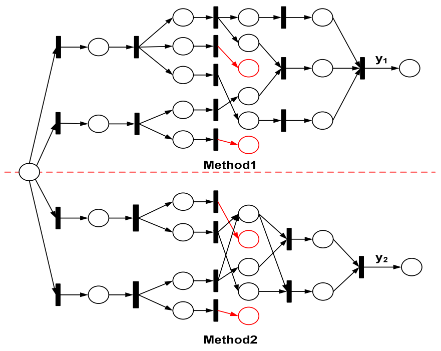


Fig.9. FCPN model of a FSD contains two methods

Note that in this paper, the output of fuzzy systems is called Message Time. For forming a single net, all the nets that have constructed for each method will be connected to each other. As an example, suppose a SD consists of two fuzzy methods, one like the second rule given in Table 1 and another one is like a hypothetical method. Fig. 9 represents the FCPN model of mentioned diagram.

4. Proposed algorithm for calculating response time

Response time in a FSD is defined as the time needed for executing a scenario in a FSD. First, we assume that all the messages transform in sequential mode. Since the studying system is distributed, the time that a message needs for sending and execution (message time) are affected by the following parameters:

1- Think time: the time spent in an interactive system by a user or objects to determine the next request.
2- The time needed for transmission a message from object A to object B.
3- The time that a method must spend in a queue for execution.
4- The time for executing a method.

With regard to above parameters, message time is calculated using following formula:

Message Time = Think Time + Transmission Time + Queuing Time + Execution Time (6)

Note that transmission time is affected by two parameters, which are message size and network speed. According to the definition of a message time in a FSD, for calculating this time the following linguistic variables are considered:

1-Message size 2-Network speed 3- Queuing Time 4-Execution time 5-Thinking Time

The five parameters (linguistic variables) mentioned above will be annotated to FSD by stereotype *<<PAstep>>* using *PAsize*, *PAnetSpeed*, *PAqTime*, *PAqTime*, *PAtinkTime* tagged values shown in (Fig. 17). After calculating message time for each method in a FSD, the response time will be calculated using the following,

Response time = sum of all message time in a FSD (7)

In FCPN model has been represented in Fig. 10, the value of final token is the response time value.

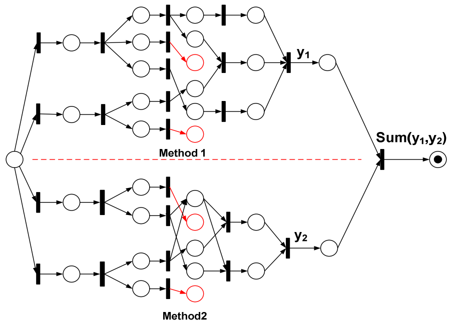


Fig.10. FCPN model of two methods in a FSD for calculating response time

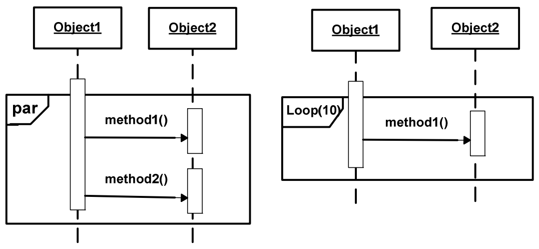


Fig.11. par and loop alternatives in a FSD

As mentioned, we have assumed that all messages are sent in sequential mode. In following, we consider how message time is calculated in parallel and loop alternatives in a FSD.

1-Parallel execution of several messages

When several messages are sent in parallel mode, for calculating message time in Par segment of a FSD we choose the maximum calculated message time in the segment shown in Fig. 11.

2-Existance of loops in FSD

When a message is sent several times in Loop segment of a FSD (Fig. 11), for calculating message time we product number of iterations in the segment by message time of the message. Note that when a self-message exists in a FSD, we must sum up the times for this message by all messages that transmit between different machines in distributed system.

5. Proposed algorithm for calculating queue length

Since a FUD cannot execute countless requests at a time, some tokens (requests) may reside in an entrance place of a FUD in the FCPN model. The tokens (requests) that exist in this place are waiting for execution. Number of tokens in this place will determine queue length of the FUD.

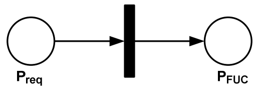


Fig.12. Part of a FUC

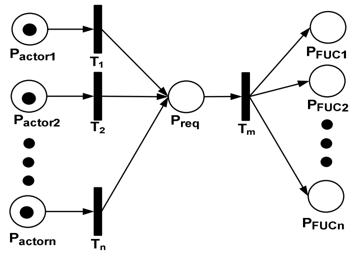


Fig.13. Part of a FCPN model created Equivalent to a FUCD

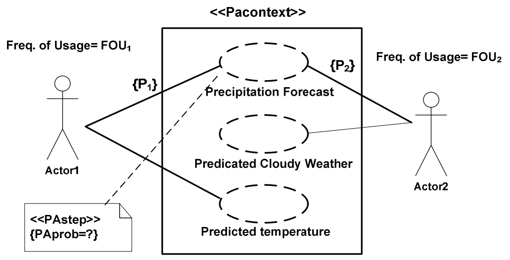


Fig.14. A FUCD with performance annotations

As represented in Fig. 12, place p_{req} contains those requests that will execute by FUC P_{FUC}. Numbers of tokens in this place specify queue length. As we can calculate queue length for a FUC, we can calculate this parameter for the whole system. In Fig. 13, which represents a part of a FCPN model of a FUC, *n* actors have modeled with places P_{actor1}, P_{actor2}, … , P_{actorn}. These actors send their own requests to FUC. These requests are shown with tokens. After firing transitions T_1, T_2, … , T_n, these tokens will depart to place p_{req} and wait for execution. Then with firing transition T_m, the tokens will transmit to appropriate places (P_{FUC1}, P_{FUC2}, …, P_{FUCn}) with regard to exp functions that exist on input arc to P_{req}. Number of tokens in place P_{req} will show queue length of whole system.

6. Case study

In this section, to investigate proposed algorithm, first a weather system is modeled with a FCPN model and then it will be analyzed with CPN Tools.

6.1. FUCD and calculating amount of usage of a FUC

In the studying weather system, we have chosen three UCs. Fig. 14 represents a FUCD includes three UCs and performance information.

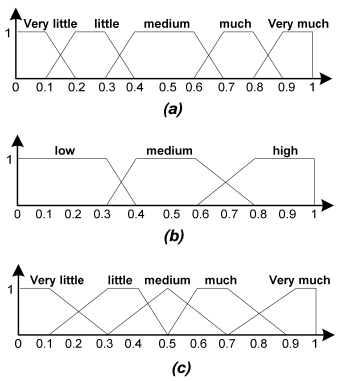


Fig.15. membership functions for: (a) user frequency of usage of the weather system. (b) Probability of user frequency of usage of a FUC. (c) Probability of a SD corresponding to the UC to be executed (amount of UC usage).

Fig. 15 illustrates membership functions needed for calculating amount of usage of a FUC. Assume that we want to calculate the amount of usage of FUC "Precipitation Forecast". Also, assume that fuzzy if-then rules are as follows:

1- **IF** FOU_1 is medium **AND** P_1 is high **AND** FOU_2 is very much **AND** P_2 is high **THEN** PAprob is much.
2- **IF** FOU_1 is very much **AND** P_1 is high **AND** FOU_2 is much **AND** P_2 is high **THEN** PAprob is very much.
3- **IF** FOU_1 is much **AND** P_1 is low **AND** FOU_2 is much **AND** P_2 is low **THEN** PAprob is little.
4- **IF** FOU_1 is little **AND** P_1 is medium **AND** FOU_2 is medium **AND** P_2 is medium **THEN** PAprob is medium.
5- **IF** FOU_1 is medium **AND** P_1 is medium **AND** FOU_2 is little **AND** P_2 is low **THEN** PAprob is little.

6- IF FOU$_1$ is very much **AND** P$_1$ is low **AND** FOU$_2$ is medium **AND** P$_2$ is low **THEN** PAprob is medium.
7- IF FOU$_1$ is much **AND** P$_1$ is medium **AND** FOU$_2$ is medium **AND** P$_2$ is medium **THEN** PAprob is medium.
8- IF FOU$_1$ is very little **AND** P$_1$ is low **AND** FOU$_2$ is very little **AND** P$_2$ is low **THEN** PAprob is very little.
9- IF FOU$_1$ is medium **AND** P$_1$ is medium **AND** FOU$_2$ is medium **AND** P$_2$ is medium **THEN** PAprob is medium.
10- IF FOU$_1$ is very much **AND** P$_1$ is high **AND** FOU$_2$ is very much **AND** P$_2$ is high **THEN** PAprob is very much.
11- IF FOU$_1$ is little **AND** P$_1$ is low **AND** FOU$_2$ is very much **AND** P$_2$ is high **THEN** PAprob is much.
12- IF FOU$_1$ is much **AND** P$_1$ is high **AND** FOU$_2$ is medium **AND** P$_2$ is medium **THEN** PAprob is much.
13- IF FOU$_1$ is very little **AND** P$_1$ is low **AND** FOU$_2$ is much **AND** P$_2$ is medium **THEN** PAprob is little.
14- IF FOU$_1$ is much **AND** P$_1$ is high **AND** FOU$_2$ is little **AND** P$_2$ is low **THEN** PAprob is little.

Using Fuzzy toolbox in Matlab we can calculate amount of usage of the FUC. For this purpose we have used from single fuzzification and center of averages defuzzification.

Table 2
Amount of usage from FUC "Precipitation Forecast"

FOU$_1$	**P$_1$**	**FOU$_2$**	**P$_2$**	**PAprob**
0.365	0.391	0.155	0.342	**0.305**
0.145	0.312	0.155	0.183	**0.137**
0.845	0.847	0.885	0.738	**0.865**
0.935	0.936	0.925	0.906	**0.894**
0.405	0.856	0.965	0.748	**0.684**

Table 2 represents amount of usage of the FUC with different crisp values with regard to fuzzy if-then rules have shown in Fig. 16.

6.2. FSD and calculating response time and queue length

Fig. 17 represents a FSD realizing FUC “Precipitation Forecast”.

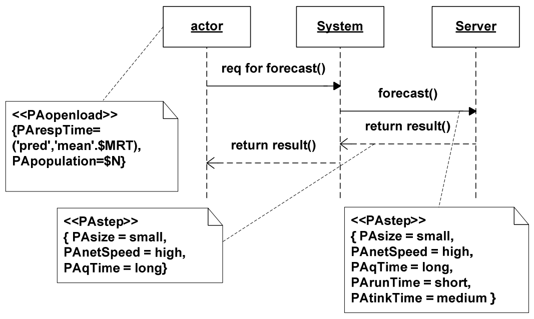
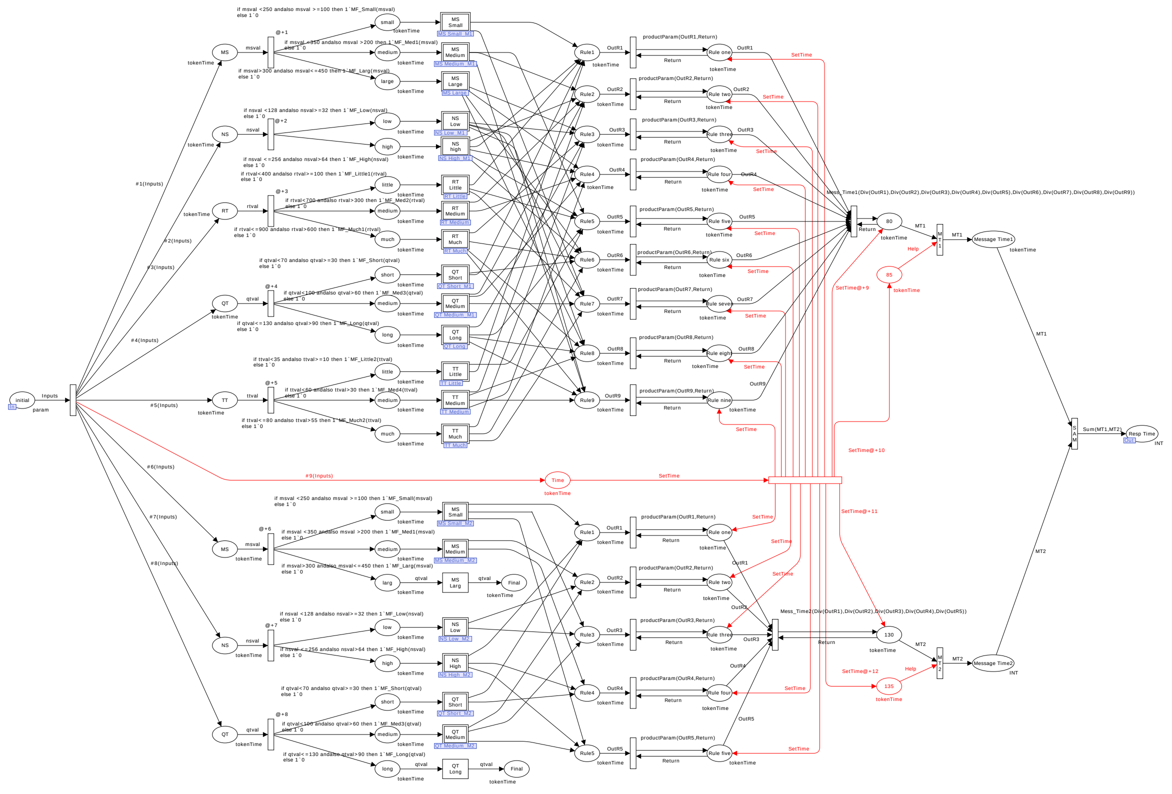

actor
System
Server
req for forecast()
forecast()
return result()
return result()
<<PAopenload>> {PArespTime= ('pred','mean'.$MRT), PApopulation=$N}
<<PAstep>> { PAsize = small, PAnetSpeed = high, PAqTime = long}
<<PAstep>> { PAsize = small, PAnetSpeed = high, PAqTime = long, PArunTime = short, PAtinkTime = medium }

Fig.17. FSD for FUC “Precipitation Forecast”

In above SD, after a user sends his/her request to the weather system for precipitation forecast, with regard to information that exist in a server it has been connected with, the server calculates the amount of precipitation and sends it to the system and finally the system will send the final result to the user. Because message 1 and 4 are transformed in a centralized system, we can disregard their transformation time. For calculating response time, message times of two fuzzy methods *forecast()* and *return result()* are added together.

Membership functions for linguistic variables: *Message Size(MS), Network Speed(NS), Queuing Time(QT), Run Time(RT), Think Time(TT) and Message Time(MT)* are represented in Fig. 19.



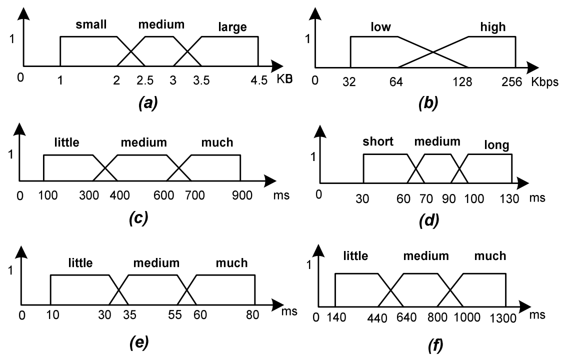


Fig.19. Membership functions for linguistic variables: (a) MS, (b) NS, (c) QT, (d) RT, (e) TT, (f) MT

The fuzzy if-then rules for calculating message time of fuzzy method *forecast()* are as follows:

*1-**IF** MS is small **AND** NS is high **AND** RT is little **AND** QT is short **AND** TT is little, **THEN** MT is little.*
*2-**IF** MS is medium **AND** NS is high **AND** RT is much **AND** QT is short **AND** TT is much, **THEN** MT is medium.*
*3-**IF** MS is large **AND** NS is low **AND** RT is much **AND** QT is long **AND** TT is much, **THEN** MT is much.*
*4-**IF** MS is large **AND** NS is low **AND** RT is medium **AND** QT is medium **AND** TT is medium,* THEN *MT is medium.*
*5-**IF** MS is medium **AND** NS is low **AND** RT is little **AND** QT is long **AND** TT is little, **THEN** MT is little.*
*6-**IF** MS is large **AND** NS is high **AND** RT is much **AND** QT is short **AND** TT is much, **THEN** MT is medium.*
*7-**IF** MS is small **AND** NS is low **AND** RT is little **AND** QT is medium **AND** TT is much, **THEN** MT is little.*
*8-**IF** MS is large **AND** NS is low **AND** RT is much **AND** QT is long **AND** TT is medium, **THEN** MT is much.*
*9-**IF** MS is medium **AND** NS is low **AND** RT is much **AND** QT is long **AND** TT is medium, **THEN** MT is much.*

Because method *return result()* from server to the weather system does not consume any time for thinking and running, for calculating message time of this method, these two parameters are disregard.

Table 3
Results of executing 15 requests in FCPN model

	Forecast Method						Return result Method				
	MS	NS	RT	QT	TT	Message Time (ms)	MS	NS	QT	Message Time (ms)	Response Time (ms)
Request1	100	207	802	65	37	580	322	199	39	290	870
Request2	340	211	330	81	32	720	109	88	34	290	1010
Request3	270	125	660	99	34	907	270	32	100	580	1487
Request4	225	64	800	61	55	580	210	60	80	376	956
Request5	270	129	500	79	41	720	263	122	71	720	1440
Request6	245	182	346	97	33	442	220	128	65	290	732
Request7	302	65	325	125	80	580	398	218	95	580	1160
Request8	335	349	650	61	34	580	227	117	37	290	870
Request9	115	143	527	45	15	580	270	35	68	720	1300
Request10	352	53	692	93	531	580	217	127	62	293	837
Request11	321	182	635	98	35	720	225	116	87	378	1098
Request12	102	37	223	47	16	580	195	51	69	290	870
Request13	323	95	635	98	35	1001	225	116	87	378	1379
Request14	450	32	100	30	10	580	270	128	91	580	1160
Request15	387	51	681	96	33	1150	217	167	82	290	1440

The fuzzy if-then rules have been used for calculating message time of method *return result()* are as follows:

*1-**IF** MS is small **AND** NS is high **AND** QT is short, **THEN** MT is little.*

*2-**IF** MS is medium **AND** NS is low **AND** QT is medium, **THEN** MT is medium.*

*3-**IF** MS is small **AND** NS is low **AND** QT is medium, **THEN** MT is little.*

*4-**IF** MS is medium **AND** NS is high **AND** QT is short, **THEN** MT is little.*

*5-**IF** MS is small **AND** NS is high **AND** QT is medium, **THEN** MT is little.*

When some requests are sent from users for execute FUC "Precipitation Forecast". The relevant FCPN model of these requests has been shown in Fig. 20.

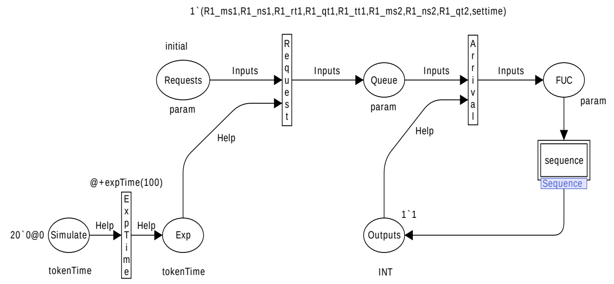


Fig.20. FCPN model for calling fuzzy use case FUC

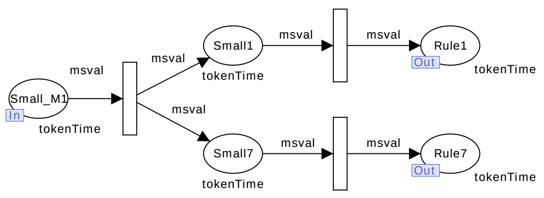
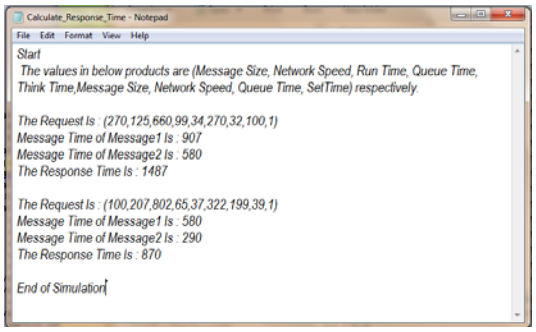


Fig.21. Subpage "MS Small_M1"

In Fig. 20, FUC "Precipitation Forecast" has been represented with place P_{FUC}. It is possible that the time between two successive requests be lower from the time needed for executing one request by the system. Therefore, a queue, which includes several requests will be created. For representing this queue, a place named Queue is added to the FCPN model. Numbers of tokens in this place will show queue length. In Fig. 20, place $P_{Request}$ contains the requests (tokens) that carry linguistic variables values as colors for calculating response time. In addition, places $P_{Simulate}$ and P_{Exp} are used for modeling entrance of the requests to place P_{Queue}. After firing transition *ExpTime* a random time will create using exponential function and then it will add to token exist in place $P_{Simulate}$ and finally it will reside in place P_{Exp}. After residing a token in place P_{Exp}, transition *Request* can fire. With firing this transition, the requests with time stamps created by mentioned exponential function will enter to queue. These time stamps determine which requests must execute first.

After firing transition *Arrival* a request enters to place P_{FUC} for execution. Then the substitution transition *Sequence* that modeles a subpage (*Sequence*) will fire. In this subpage fuzzy if-then rules have modeled and with regard to single fuzzification, product inference engine and center of average defuzzification, message times of methods *forecast()* and *return result()* are calculated and finally the response time will send as final result to $P_{Outputs}$ place. Fig. 18 shows subpage *Sequence*.

The subpages that exist in subpage "*Sequence*" show second step of the algorithm for mapping a FSD to a FCPN model. As an example subpage *"MS Small_M1"* has been represented in Fig. 21. Places and arcs with red color in subpage "*Sequence*" are for adjusting priority between firing transitions. As shown in Fig. 18, after calculating message times of methods *forecast()* and *return result()*, these results will reside in places $P_{Message\ time1}$ and $P_{message\ time2}$ as a new token color. Since these methods are in sequential mode in FSD, for calculating response time of scenario, the two message times will add together. This event will accomplish after firing transition SAM and then final result that shows response time will be sent to output place ($P_{Resp\ Time}$). For calculating messages' time and response time, assume that 15 requests with different times have called a FUC as shown in Table 3. Note that, because CPN Tools cannot support real values, we have multiplied message size by 100. After execution these requests within the FCPN model, the results as a report has been illustrated in Fig. 22 using monitor capability of CPN Tools.



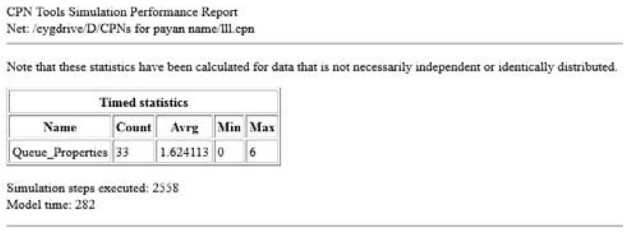
```
Start
 The values in below products are (Message Size, Network Speed, Run Time, Queue Time,
Think Time,Message Size, Network Speed, Queue Time, SetTime) respectively.

The Request Is : (270,125,660,99,34,270,32,100,1)
Message Time of Message1 Is : 907
Message Time of Message2 Is : 580
The Response Time Is : 1487

The Request Is : (100,207,802,65,37,322,199,39,1)
Message Time of Message1 Is : 580
Message Time of Message2 Is : 290
The Response Time Is : 870

End of Simulation
```

Fig. 22. Simulation reports after execution of request 1 and 3


CPN Tools Simulation Performance Report
Net: /cygdrive/D/CPNs for payan name/lll.cpn

Note that these statistics have been calculated for data that is not necessarily independent or identically distributed.

Timed statistics				
Name	Count	Avrg	Min	Max
Queue_Properties	33	1.624113	0	6

Simulation steps executed: 2558
Model time: 282

Fig. 23. Queue properties after execution of FCPN model

The report results have been shown in Table 3 with green highlight. Also, minimum and average of queue length have been shown in Fig. 23.

7. Conclusion

In this paper, a novel method has been presented for evaluation of SA in the systems, which work with uncertain information explained. Because this research focused on uncertain and ambiguous system, we have used from F-UML to describe SA. In addition, for enriching F-UML diagrams with performance information, we have used F-SPT profile. In this paper, after mapping F-UML diagrams to a formal model (FCPN), we proposed an algorithm to calculate amount of usage from a FUC as well as two algorithms for calculating response time and queue length in a FSD. Using proposed method, software architect can enter uncertainty in system modeling and evaluate performance of SA of the system.

References

Akbari, E., Noorian Talooki, R., & Motameni, H.(2010). Mapping sequence diagram in Fuzzy UML to fuzzy Petri Net. *Iranian Journal of Optimization*, 3, 492–505

Balsamo, S., Person, V., & Inverardi, P. (2002). A review on queueing network models with finite capacity queues for software architectures performance prediction. *Performance Evaluation,* 974, 1–20.

Balsamo, S., & Maraolla, M. (2005). Performance Evaluation of UML Software Architectures with Multiclass Queueing Network Models. *WOSP '05 Proceedings of the 5th international workshop on Software and performance*, 37–42.

Bass, L., Clements, P., & Kazman, R. (2003). *Software Architecture in Practice*. Addison Wesley.

Bernardi, S., & Merseguer, J. (2007). Performance evaluation of UML design with stochastic well-formed nets. *The Journal of Systems and Software,* 80, 1843–1865.

Cooper, K., Dai, L., & Deng, Y.(2005). Performance modeling and analysis of software architectures: An aspect oriented
UML based approach. *International Workshop on Systems and Software Architecting, Science of Computer Programming,* 57, 89–108

Dobrica, L., & Niemela, E. (2002). Survey on software architecture analysis methods. *IEEE Transactions on Software Engineering,* 28(7), 638–653.

Haroonabadi, A., & Teshnehlab, M. (2008). A novel method for behavior modeling in uncertain information systems. *International Journal of Electrical and Electronics Engineering*, 2(7).

Hong-Xia, Z., & Lian-Zhang, Z. (2009). Building dynamic model in UML using colored Petri Nets. *IEEE, Computer Network and Multimedia Technology*.

Jensen, K. (1993). An introduction to the theoretical aspects of coloured Petri nets. *A Decade of Concurrency, in: Lecture Notes in Computer Science, vol. 803, Springer-Verlag*, 230–272.

Jensen, K., & Kristensen, L.M.(2009). *Coloured Petri Nets*. Springer.

Lian-Zhang. Z., & Fan-Sheng, K.(2012). Automatic Conversion from UML to CPN for Software Performance Evaluation. *2012 International Workshop on Information and Electronics Engineering (IWIEE), Procedia Engineering,* 29, 2682 – 2686

Lopez Grao, J.P., Merseguer, J., & Campos, J.(2004). From UML activity diagrams to stochastic Petri nets: application to software performance engineering. *Proceedings of the Fourth International Workshop on Software and Performance (WOSP'04). ACM, Redwood City, CA, USA*. 25–36.

Ma. Z.M., Zhang, F., & Yan, L.(2011). Fuzzy information modeling in UML class diagram and relational database models. *Applied Soft Computing,* 11, 4236–4245.

Medvidovic, N., & Taylor, J. (2000). A classification and comparison framework for software architecture description
languages. *IEEE Transaction on Software Engineering*, 26(1), 70–92.

Merseguer, J. (2003). *Software Performance Modeling Based on UML and Petri Net.* Ph.D thesis.

Motameni, H., Movaghar, A., Daneshfar, I., Nemat Zadehand, H., & Bakhshi, J.(2008). Mapping to convert activity diagram in Fuzzy UML to fuzzy Petri Net. *World Applied Sciences*. 3(3), 514–521.

Object Management Group. (2005). *UML profile for schedulability, performance and time specification version 1.1*.

Perez-Palcin, D., & Merseguer, J. (2010). Performance evaluation of self-reconfigurable service-oriented software with stochastic Petri Nets. *Electronic Notes in Theoretical Computer Science,* 261, 181–201.

Staines, T. (2008). Intuitive Mapping of UML 2 Activity Diagrams into Fundamental Modeling Concept Petri Net Diagrams and Colored Petri Nets. *15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based System.*

Yeung, D.S., Liu, J.N.K., Shiu, S.C.K,. & Fung, G.S.K.(1996). fuzzy coloured petri nets in modelling flexible manufacturing systems. *ISAI/IFIS 1996. Mexico-USA Collaboration in Intelligent Systems Technologies, IEEE.*