Modeling Self-Adaptive Software Systems by Fuzzy Rules and Petri Nets

Zuohua Ding, Member, IEEE, Yuan Zhou, and Mengchu Zhou, Fellow, IEEE

Abstract—A self-adaptive software system is one that can autonomously modify its behavior at runtime in response to changes in the system and its environment. It is a challenge to model such a kind of systems since it is hard to predict runtime environmental changes at the design phase. In this paper, a formal model called intelligent Petri net (I-PN) is proposed to model a self-adaptive software system. I-PN is formed by incorporating fuzzy rules to a regular Petri net. The proposed net has the following advantages. 1) Since fuzzy rules can express the behavior of a system in an interpretable way and their variables can be reconfigured by the runtime data, the proposed model can model runtime environment and system behavior. 2) Since a fuzzy inference system with well-defined semantics can be used in a complementary way with other model languages for the analysis, the proposed model can be analyzed, even though it is described in two different languages: component behaviors in Petri nets while logic control in fuzzy rules. 3) The proposed model has self-adaptation ability and can make adaptive decisions at runtime with the help of fuzzy inference reasoning. We adopt a manufacturing system to show the feasibility of the proposed model.

Index Terms—Adaptive software system, fuzzy rule, Petri net (PN), requirement modeling.

I. INTRODUCTION

A SELF-ADAPTIVE software system can autonomously modify itself to satisfy certain objectives in response to changes in the system and its environment. It has many applications, e.g., in fault-tolerant computing, distributed systems, distributed artificial intelligence, integrated management, robotics, knowledge-based systems, smart grids, and control theory [8], [30], [32].

Although we can get plenty of benefits from such systems, their development has shown to be more challenging than that of traditional software systems. A general discussion about the challenges for the systematic software engineering of self-adaptive systems can be found in [8]. Modeling of adaptive systems is one of hard problems because it is hard to model runtime environment changes and system behavior changes at design phase with limited knowledge of the system.

There are a few studies to formally model self-adaptive software systems [8], [11], [15], [22], [24], [33], [34], [48], [49]. For example, in order to meet a system’s nonfunctional requirements [21], a finite state automaton with probabilities is used to model the adaptation of a system among alternative possible paths; dynamic decision networks are used to model the adaptation for the situation in which the conditional probabilities change for different design alternatives over nonfunctional requirements [3]; context Petri nets (PNs), a PN-based programming model, is used to model the behavioral adaptation based on the sensed context of execution [7]; Rainbow, an architecture-based framework, is used to model the adaptation when the operational conditions are unfavorable [19]; MARS [38], which uses labeled state transition systems, is used to model embedded adaptive systems, and adaptation is done by changing the active configuration. External software mechanisms to maintain a form of closed-loop control allow a system to self-adapt dynamically [4], [34].

Regarding their expressiveness, these models are either hard to be interpreted, e.g., statistical model, or not able to deal with data and variables, e.g., architecture model, or not able to manage soft transitions, e.g., control model. Regarding the adaptation ability, they fall into two categories [39]: structure and behavior-based adaptations. The former adapts system behavior via changing a system’s architecture, while the latter modifies the functionalities of computational entities. However, the above models have at least one of the two serious drawbacks: 1) the adaption mechanisms are designed as a control center, and thus, the runtime data do not receive enough consideration to be used for learning, reconfiguring, and collaborative adaptation, e.g., mode switching takes place at an architectural level in [11]; and 2) the adaption leads to computation overhead since a local change often requires computation involving many components in the system, which decreases the system performance. For example, the overlap adaptation [48] requires a sequence of adaptation transitions.

To address the above issues, this work proposes a new formal model to describe the behavior of adaptive systems. The proposed model is an extension to a PN by adding fuzzy rules to it. This is due to the following considerations.

1) Since we have limited knowledge about a system at the design phase, we need to model unpredicted environments and behavior. Some requirements are often described in fuzzy linguistic terms, such as higher, lower, and far.
Fuzzy rules can well express the behavior of a system in an interpretable way.

2) The requirements usually contain two parts describing component behavior and logic control. We use PNs to characterize the discrete event behavior since they have been successfully used to model concurrent systems and control systems, while fuzzy rules, after quantifying the fuzzy requirements, are used for logic control. The rules behave like the industrial process supervision for managing soft transitions between two states. Since a fuzzy inference system has well-defined semantics, it can be used in a complementary way with other model languages for the property analysis.

3) Since the rules provide an interface between data and variable symbols, the variables can be reconfigured at runtime. Thus, the runtime data can be incorporated into the model. Its adaptation is realized through the fuzzy inference reasoning.

The rest of this paper is organized as follows. Section II gives a motivating example of self-adaptive systems. Section III introduces fuzzy rules and gives their PN representations. Section IV presents intelligent Petri net (I-PN), to model self-adaptive systems, and an algorithm to build an evolution graph (EG). Section V gives the adaptation analysis of the proposed PN. Section VI studies the motivating example with I-PN and its properties. Section VII summarizes the related work. Section VIII concludes this paper.

II. MOTIVATING EXAMPLE

We use a manufacturing system shown in Fig. 1 as a motivating example. It consists of three components: the customer component for order placing and product acceptance, the manufacturing factory component for product production, and the supplier component for raw material supply.

The manufacturing factory may have many branch factories to fulfill the orders, and the supplier component may have many suppliers to select to provide the required raw material. The selection processes should be able to respond to market changes quickly. In this paper, we assume that there are three branch factories, denoted as B1–B3, and two suppliers, denoted as S1 and S2, can be selected. B1–B3 and S2 are in one city, while S1 is in another city. S2 asks for a higher price than S1.

The system should select one appropriate branch factory and one supplier each time. In practice, many factors can affect the selections of branch factories and suppliers. However, to simplify the problem, we only select the following three factors to analyze the selection of a branch factory: the quantity of order (x1), the price of raw material (x2), and the cost of transport for production (x3), while the following two factors to analyze the selection of suppliers: the demanded amount of raw material (y1) and the cost of transport for material (y2).

These five factors form the environment of the entire system. The decision to select a supplier and factory is made based on the environment. We need to find another appropriate combination of branch factory and supplier to complete product manufacturing in response to environmental changes.

Initially, we know that the five factors affect the selections, but we do not know how they affect the selections. This should be learnt and adapted continuously during runtime. Besides, in the requirement specifications, users are likely to give fuzzy requirements, such as “When there is a large amount of orders and the average price of the raw material is low, then a branch factory with higher production capacity is better to be selected to fulfill the orders. When the amount of the orders is not very large and the price of the raw material is low but the transport cost is very high, then we select a branch factory that is near the customers.” Thus, in order to develop a robust system, the system needs to reach the following performance requirements: R1) It can process the fuzzy requirements; R2) it can be refined during runtime (Adaption 1); and R3) it should dynamically find the best solution to complete the manufacturing in response to the market changes (Adaption 2).

Thus, in this paper, we propose a new formal model to describe such a system, where fuzzy rules are used to quantify fuzzy requirements, the behavior can be adapted at runtime in order to refine the system, and the adaptive selection is realized through fuzzy reasoning.

III. FUZZY RULES AND PETRI NETS

A. Fuzzy Numbers and Fuzzy Rules

The following definitions are due to Zadeh [46].

Definition 1: Let X be the domain of discourse and x ∈ X be an element of X. A fuzzy set ̄u in X is characterized by a real-value function μ̄u : X → [0, 1] that assigns each element in X with a real number in the interval [0, 1]. The value μ̄u(x) represents the “the degree of membership” of x in ̄u. The function μ̄u is called the membership function of ̄u in X.

Definition 2: A fuzzy number is a convex normalized fuzzy set of real line R whose membership function is at least segmentally continuous and has the functional value at precisely one element.

For example, the fuzzy number ̄u with the following membership function is a fuzzy number, called a triangular fuzzy number:

\[ ̄u(x) = \begin{cases} \frac{x + b - a}{b}, & a - b \leq x \leq a \\ \frac{x + a - b}{b}, & a < x \leq a + b, (a, b \in R) \\ 0, & \text{otherwise} \end{cases} \] (1)

Its graph is shown in Fig. 2.
We need the following concepts to define fuzzy rules.

A linguistic variable is characterized by a quintuple \((X, T, U, G, M)\), where \(X\) is the name of the variable, \(T\) is the set of terms of \(X\), \(U\) is the domains of discourse, \(G\) is a syntactic rule for generating the name of the terms, and \(M\) is a semantic rule for associating each term with its meaning, that is, a fuzzy set defined on \(U\).

A single fuzzy IF–THEN rule (or simply fuzzy rule) has the form

\[
\text{Form 1: } R_i : \text{IF } x \text{ is } A \text{ THEN } y \text{ is } B(CF = \varphi_i)
\]

where \(x\) and \(y\) are two linguistic variables whose domains of discourse are \(U_x\) and \(U_y\), respectively; \(A\) and \(B\) are two fuzzy numbers on \(U_x\) and \(U_y\), respectively. \(\varphi_i\) is the value of a certainty factor \((CF), \varphi_i \in [0, 1]\), which represents the strength of the usefulness in the rule.

The premise “\(x\) is \(A\)” and the consequence “\(y\) is \(B\)” are actually two fuzzy propositions. The activation degree of a fuzzy proposition is characterized by the membership degree of its fuzzy number. For example, the activation degree of “\(x\) is \(A\)” can be computed as \(A(x)\). For convenience, we denote the premise and consequence propositions by \(d_j\) and \(d_k\), respectively, where \(d_j\) is called the antecedent proposition of \(R_i\) and \(d_k\) the consequent proposition of \(R_i\). The sets of antecedent and consequent propositions of \(R_i\) are denoted by \(A(R_i)\) and \(C(R_i)\), respectively. For example, the following is a fuzzy rule:

\[
R_i : \text{IF the weather is hot THEN the humidity is low } (CF = 0.90)
\]

where the antecedent proposition \(d_j = \text{“the weather is hot,”}\) the consequent proposition \(d_k = \text{“the humidity is low,”} A(R_i) = \{d_j\}, \text{ and } C(R_i) = \{d_k\}.

Based on [31], there are three types of fuzzy rules that are useful in practice:

- **Type 1:** IF \(d_{j_1} \land d_{j_2} \land \cdots \land d_{j_n} \text{ THEN } d_k (CF = \varphi_i)\).
- **Type 2:** IF \(d_j \text{ THEN } d_{k_1} \land d_{k_2} \land \cdots \land d_{k_n} (CF = \varphi_i)\).
- **Type 3:** IF \(d_{j_1} \lor d_{j_2} \lor \cdots \lor d_{j_n} \text{ THEN } d_k (CF = \varphi_i)\).

\[\text{B. Modeling Fuzzy Rules With Petri Nets}\]

In this subsection, we discuss how to model fuzzy rules with PNs. We first give some basic concepts of PNs. A PN is a quadruple \((P, T, A, w)\), where \(P\) is a nonempty set of places, \(T\) is a nonempty set of transitions such that \(P \cap T = \emptyset\), \(A \subset (P \times T) \cup (T \times P)\) is the set of arcs, \(w\) is a set of weights assigned to each arc. A marking of a PN, denoted as \(M\), is a function assigning each place with a value, and the marking of \(p \in M\) is denoted as \(M(p)\). The nodes, including the places and transitions, can be divided into discrete and continuous. A discrete (respectively, continuous) place, represented by a single circle (respectively, a double circle), is a place whose marking is a nonnegative integer (respectively, a nonnegative real number). A discrete (respectively, continuous) transition, represented by a single line (respectively, a hollow rectangle), is a transition that moves a integer number (respectively, a real number) of markings.

To model the fuzzy rules and their inference with the values in the universe of discourse, we introduce two new types of transitions, i.e., \(\alpha\)- and \(\beta\)-transitions, and a new type of places, i.e., \(f\)-place. Each \(\alpha\)-transition is associated with a membership function that maps the element of the universe of discourse to a fuzzy number with a certain membership degree, while each \(\beta\)-transition is associated with a value that represents the certainty factor of the fuzzy rule and is used to describe the reasoning of a fuzzy rule. Thus, \(\alpha\)-transitions have higher priority than \(\beta\)-transitions to fire. Each \(f\)-place represents a proposition \(d_i\) and its marking represents the \(d_i\)'s activation degree, which is a real value in \([0, 1]\), in terms of the input value. Their graph representations are shown in Fig. 3.

We also need the following definitions before modeling fuzzy rules with PNs.

**Definition 3:** Let \(FR\) be a finite set of fuzzy rules, \(\bar{A} = \{A(R_i)|R_i \in FR\}\), and \(\bar{C} = \{C(R_i)|R_i \in FR\}\).

1) The set of input propositions of \(FR\), denoted as \(I_R\), is defined as \(I_R = \bar{A} - \bar{C}\).

2) The set of output propositions of \(FR\), denoted as \(O_R\), is defined as \(O_R = \bar{C} - \bar{A}\).

3) The set of internal propositions of \(FR\), denoted as \(IN_R\), is defined as \(IN_R = \bar{A} \cap \bar{C}\).

Now, we model fuzzy rules with PN. Given a set of fuzzy rules \(FR\), 1) For the rules with input propositions, \(\alpha\)-transitions are used. First, each \(\alpha\)-transition translates the crisp value to the membership degrees of corresponding fuzzy numbers. Then, \(\beta\)-transitions are used to model the reasoning of fuzzy rules. 2) For the rules whose antecedent propositions are not in \(I_R\), i.e., the activation degrees of their antecedent propositions can be obtained by the reasoning of other rules, only \(\beta\)-transitions are needed to model these fuzzy rules. The details are given next.

1) **Model Type 1 fuzzy rule:** The net to model a Type 1 fuzzy rule is shown in Fig. 4. Fig. 4(a) shows the model for a rule whose antecedent propositions are input propositions, where \(p_{i_1} - p_{i_n}\) are \(C\)-places used to model the environment and their markings are the current environment value, and \(p_{j_1} - p_{j_n}\) are \(f\)-places whose markings are the membership degrees of the corresponding fuzzy numbers. Each of \(t_{i_1}^n - t_{i_n}^n\) is associated with a membership function. They are used to translate the environment to
the corresponding membership degree of the fuzzy number, i.e., the activation degree of the antecedent proposition of the rule. \(t^1\) is a \(\beta\)-transition associated with the certainty factor of the rule. It models the reasoning of the fuzzy rule and its firing generates the activation degree of its consequent propositions. The evolution of this net is described below. First, \(\alpha\)-transitions are enabled and fired, such that \(p_{j_1} - p_{j_2}\) obtain markings: \(M(p_{j_1}) = \tilde{A}_i(M(p_{j_2}^r))\), where \(\tilde{A}_i\) is the membership functions assigned to \(t^j\), \(i \in \{1, 2, \ldots, n\}\). Then, \(t^2\) is enabled and fired, and place \(p_k\) obtains a marking equal to \(M(p_k) = \varphi_i \times \min(M(p_{j_1}^r), M(p_{j_2}^r), \ldots, M(p_{j_n}^r))\). Fig. 4(b) shows the net of the rule whose propositions are not input propositions. In this case, the propositions represented by \(p_{j_1} - p_{j_n}\) must be the consequent propositions of other rules, and thus, the markings of \(p_{j_1} - p_{j_n}\) can be obtained by the reasoning of other rules. Once these places obtain their markings, \(t^3\) can fire and \(p_k\) obtains a marking \(M(p_k) = \varphi_i \times \min(M(p_{j_1}^r), M(p_{j_2}^r), \ldots, M(p_{j_n}^r))\).

2) Model a Type 2 fuzzy rule: Fig. 5 shows the net to model a Type 2 fuzzy rule. Fig. 5(a) and (b) shows the models for the rules whose antecedent propositions are input propositions and internal propositions, respectively. For the former, \(t^0\) fires first and the \(f\)-place \(p_j\) obtains a marking \(M(p_j) = \hat{A}(M(p_j^r))\), i.e., the activation degree of the antecedent proposition of the rule. Then, \(t^2\) fires and \(p_{j_1} - p_{j_n}\) obtain their markings: \(M(p_{j_1}) = M(p_{j_2}) = \cdots = M(p_{j_n}) = \varphi_i \times M(p_j)\). The firing of the \(\beta\)-transition in Fig. 5(b) causes the same result with that of the \(\beta\)-transition in Fig. 5(a).

3) Model a Type 3 fuzzy rule: The net in Fig. 6 is used to model a Type 3 fuzzy rule. In Fig. 6(a), \(t^0_n - t^1_n\) are \(\alpha\)-transitions. After their firing, \(p_{j_1} - p_{j_n}\) obtain markings \(M(p_{j_i}) = \tilde{A}_i(M(p_{j_2}^r)), i \in \{1, 2, \ldots, n\}\). \(t^2_1 - t^2_n\) are then enabled and fired to perform the reasoning and place \(p_k\) gets the maximum marking, i.e., \(M(p_k) = \max_{i \in \{1, 2, \ldots, n\}} \{\varphi_i \times M(p_{j_i})\}\). In Fig. 6(b), the markings of \(p_{j_1} - p_{j_n}\) are obtained from the firing of other rules, while \(p_k\) gets the marking \(M(p_k) = \max_{i \in \{1, 2, \ldots, n\}} \{\varphi_i \times M(p_{j_i})\}\) by the firing of \(t^3_{1-n}\).

Based on the above metamodels, we can make the adaptive decisions by the evolution of the metamodels. For example, the following rules are the fuzzy rules for the selection of an appropriate supplier, where \(y_1\) is the amount of the raw material, \(y_2\) is the cost of transport for material, \(z_1\) is the benefit from the low price of raw material, and \(z_2\) is the benefit from the short transport distance.

\begin{align*}
R_1: & \ \text{IF} \ y_1 \ \text{is high, THEN} \ z_1 \ \text{is great} (CF = \varphi_1). \\
R_2: & \ \text{IF} \ y_2 \ \text{is median and} \ y_2 \ \text{is high, THEN} \ z_1 \ \text{is great} (CF = \varphi_2). \\
R_3: & \ \text{IF} \ y_2 \ \text{is median and} \ y_2 \ \text{is low, THEN} \ z_2 \ \text{is great} (CF = \varphi_3). \\
R_4: & \ \text{IF} \ y_1 \ \text{is median and} \ y_2 \ \text{is low, THEN} \ z_2 \ \text{is great} (CF = \varphi_4). \\
R_5: & \ \text{IF} \ y_1 \ \text{is low, THEN} \ z_2 \ \text{is greater} (CF = \varphi_5). \\
R_6: & \ \text{IF} \ z_1 \ \text{is greater, THEN the selection of} \ S_1 \ \text{is good} (CF = \varphi_6). \\
R_7: & \ \text{IF} \ z_1 \ \text{and} \ z_2 \ \text{are similar, THEN the selection of} \ S_1 \ \text{is good} (CF = \varphi_7). \\
R_8: & \ \text{IF} \ z_1 \ \text{and} \ z_2 \ \text{are similar, THEN the selection of} \ S_2 \ \text{is good} (CF = \varphi_8). \\
R_9: & \ \text{IF} \ z_2 \ \text{is greater, THEN the selection of} \ S_2 \ \text{is good} (CF = \varphi_9). \\
\end{align*}

The net for the nine rules is shown in Fig. 7. In the net, \(p_{j_1}^1\) and \(p_{j_2}^2\) are two continuous places representing two environmental factors \(y_1\) and \(y_2\), respectively. \(p_{j_1}^1 - p_{j_n}^1\) are \(f\)-places representing six input propositions: \(y_1\) is high, \(y_1\) is median, and \(y_1\) is low; \(y_2\) is high, \(y_2\) is median, and \(y_2\) is low. Hence, there are six
C. Fuzzy Petri Net

1) Definition of Fuzzy Petri Net: In this subsection, a new PN, called F-net, is introduced. It can model the adaptive decision making by fuzzy reasoning.

Some notations are used. Let \( \langle P, T, A, w \rangle \) be a PN. \( \forall p \in P \) (respectively, \( t \in T \)), \( \bullet p \) (respectively, \( \bullet t \)) denotes the set of its input transitions (respectively, places), i.e., \( \bullet p = \{ t \in T \mid (p, t) \in A \} \) (respectively, \( \bullet t = \{ p \in P \mid (p, t) \in A \} \)), and \( \forall p \in P \) (respectively, \( t \in T \)), \( p^* \) (respectively, \( t^* \)) denotes the set of its output transitions (respectively, places), i.e., \( p^* = \{ t \in T \mid (p, t) \in A \} \) (respectively, \( t^* = \{ p \in P \mid (p, t) \in A \} \)). \( | \bullet | \) denotes the cardinality of a set.

Definition 4: A fuzzy PN, called F-net, is a tuple \( PNF = \langle PF, TF, A, w \rangle \), where:

1) \( PF = P^c_\alpha \cup P^*_\beta \cup P_d \cup \{ p_{\text{ctl}} \} \) is a finite set of places, where \( P^c_\alpha \) is a set of continuous places, \( P^*_\beta \) is a set of discrete places, and \( p_{\text{ctl}} \) is a special discrete place called control place.

2) \( TF = T_\alpha \cup T_\beta \cup T_d \) is a finite set of transitions, where \( T_\alpha \), \( T_\beta \), and \( T_d \) are sets of \( \alpha \)-transitions, \( \beta \)-transitions, and discrete transitions, respectively. The transitions satisfy:

(a) \( p_{\text{ctl}} = p_{\text{ctl}}^* = T_d \); (b) \( \forall t \in T_\alpha \), \( t \in P^c_\alpha \), \( t^* \subset P^*_\beta \), and \( |t^*| = |t^*| = 1 \); and (c) \( P^c_\alpha = \cup \{ t \} \).

3) \( A \) is the set of arcs such that \( A \subset (P \times T) \cup (T \times P) \).

4) \( w : A \rightarrow \{ 1 \} \) is a mapping that assigns each arc with a weight equal to 1.

In this definition, the places in \( P^c_\alpha \) denote the input variables, and the corresponding markings denote the crisp values of these variables; each place in \( P^*_\beta \) represents a fuzzy proposition, and its marking represents the activation degree of the corresponding proposition; the places in \( P_d \) represent the possible output decisions based on the reasoning of the fuzzy rules. The transitions in \( T_\alpha \) perform fuzzification, while the transitions in \( T_d \) process defuzzification; transitions in \( T_\beta \) perform the fuzzy reasoning of the fuzzy rules.

Indeed, a discrete transition in \( T_d \) connects fuzzy places and discrete places, and its firing is controlled by \( p_{\text{ctl}} \). Fig. 7 describes a fuzzy PN, where \( P^c_\alpha = \{ p_1^c, p_2^c \} \), \( P^*_\beta = \{ p_1^\beta, p_2^\beta \} \), \( P_d = \{ t_1, t_2 \} \), \( T_d = \{ t_1, t_2 \} \), \( T_\alpha = \{ t^\alpha_i \mid i = 1, 2, \ldots, 11 \} \), \( T_\beta = \{ t_3, t_4 \} \), \( T_\beta = \{ t^\beta_i \mid i = 1, 2, \ldots, 9 \} \). Each \( t^\alpha_i \) is assigned with a membership function, and each \( \beta \)-transition is assigned with the certainty factor of the rule.

For convenience, we sometimes use a set of filled rectangles to represent an F-net. The number of the rectangles is equal to the number of the output propositions of fuzzy rules. We call the filled rectangles \( f \)-transitions. Each \( f \)-transition is assigned with a tuple \( (t, l) \), where \( t \) is the name of the transition and \( l \) is the label of the transition. The set of \( f \)-transitions with the same label represents one F-net. For example, the F-net in Fig. 7 can be simplified as the net shown in Fig. 8. Note that a single \( f \)-transition is just a symbol to make the display of the net simple, and it has no semantic.

2) Fuzzy Reasoning in the F-Net: In this part, we introduce the fuzzy reasoning algorithm in terms of the reachability computation of F-nets. Here, the reasoning algorithm is inspired from [10] and [20], and it is a breadth-first algorithm. Some applications of fuzzy reasoning using PNs are also given in [9], [23], and [41].

Let \( I(p) \) and \( O(p) \) be the sets of the input and output transitions of place \( p \), and \( I(t) \) and \( O(t) \) be the set of input and output places of transition \( t \), respectively. \( P^c_\alpha \subseteq P^c \) is the set of fuzzy places that represent the input fuzzy propositions of the fuzzy rule set, and \( P^c_\beta \subseteq P^c \) is the set of places that represent fuzzy places, \( P_d \) is a set of discrete places, and \( p_{\text{ctl}} \) is a special discrete place called control place.
the output propositions of the fuzzy rule set; place \( p_d \) can be
reached immediately from \( p_d \) iff there exists a transition \( t \) such
that \( t \in \text{O}(p_d) \) and \( p_d \in \text{O}(t) \), and the set of the fuzzy
places that can be reached immediately from \( p \) is denoted as \( \text{IRS}(p) \),
i.e., \( \text{IRS}(p) = \bigcup_{t \in \text{O}(p)} \text{O}(t) \). Two places \( p_x \) and \( p_k \) are
called adjacent if there exists a transition \( t \) such that \( p_x \in I(t) \) and
\( p_k \in I(t) \). Suppose \( p_x \) and \( p_y \in \text{IRS}(p_x) \) are connected by \( t \), i.e., \( \text{O}(p_x) \cap \text{I}(p_y) = \{ t \} \), the set of adjacent places of \( p_x \) with
respect to \( p_y \) is defined as \( \text{AP}_{xy} = \{ p_k | t \in \text{O}(p_k) \} \).

The following are the formulas to compute the marking evolution
of F-nets.

1) If \( t \) is an \( \alpha \)-transition associated with membership function \( \tilde{A} \), then
\[
M_{i+1}(p) = \begin{cases}
\tilde{A}(M_i(p)), & p \in t^* \\
M_i(p), & \text{otherwise}
\end{cases}
\] (2)

2) If \( t \) is an \( \beta \)-transition associated with certainty factor \( \varphi \), then
\[
M_{i+1}(p) = \begin{cases}
\max\{ M_i(p), \varphi \times \min\{ M_i(t)\} \}, & p \in t^* \\
M_i(p), & \text{otherwise}
\end{cases}
\] (3)

where \( M_i(t) \) is the set of markings of \( t \)'s input places at \( M_i \).

3) If \( t \) is a \( D \)-transition, then
\[
M_{i+1}(p) = \begin{cases}
M_i(p) - 1, & p \in t^* \cap P_D \\
0, & p \notin t^* \cap P_f \\
M_i(p) + 1, & p \in t^* \cap P_D \\
M_i(p), & \text{otherwise}
\end{cases}
\] (4)

Now, we are ready to give the fuzzy reasoning and decision
making algorithm for the F-net, as shown in Algorithm 1.

For example, we can obtain the marking of each fuzzy
place and the reasoning tree shown in Fig. 9. Since \( M(p) > M(p') \), condition \( c_1 \) is satisfied and \( t \) fires, causing the final
making to be \( M(p_1) = 1 \) and \( M(p_2) = 0 \). The parameters in
the net are to be given later.

IV. INTELLIGENT PETRI NETS

Based on the discussion in the above section, we define a
new type of PNs to model self-adaptive systems, namely I-
PNs, which has learning and adaption ability, and can model
environmental changes.

A. Definition of Intelligent Petri Net

Definition 5: A Learning Net is an extension of an F-
net \( PNF = \{P, T, F, A, w\} \) by adding a set of places \( P^E = \{p_d\} \cup P \) and a set of transitions \( T^E = \{t_d\} \cup T \), where:

1) \( P_d \) is a set of continuous places and \( p_d \) is a discrete place
such that \( *_{pd} = 0 \).

2) \( T_d \) is a set of continuous transitions such that \( \cup_{t \in T} *_t = P \), and \( t_d \) is a discrete transition such that \( *_{td} = \{pd\} \) and \( t_d \) is a discrete transition such that \( *_{td} = \{pd\} \) and
\( \text{AP}_{xy} \).

3) For each transition \( t \in T_d \), there exists a unique continuous
place \( p \in P^E \) such that \( *t = \{p\} \).

Algorithm 1: Fuzzy reasoning algorithm of F-net.

Input: An F-net \( \{P^E, T^E, A, w\} \) and an initial marking \( M_0 \) for the
fuzzy reasoning.

Output: The marking of discrete places: \( M_0 \).

1) Initialization: \( FRS = \emptyset \), NewLeaf = OldLeaf = \emptyset , and create a
root node with marking \( M_0 \).

2) Compute \( T_{\text{enable}} \): the set of transitions that can be enabled by \( M_0 \),
\( T_{\text{enable}} = T_{FS} \);

while \( T_{\text{enable}} \neq \emptyset \) do

3) select a transition \( t \in T_{\text{enable}} \), \( T_{\text{enable}} = T_{\text{enable}} - \{t\} \);

4) fire \( t \) and compute the marking \( M(p) \) based on Eq. 2, where
\( p = O(t) \);

5) create a node \( (p, M(p), IRS(p)) \) and an arc from the root
node to this node:
\( \text{NewLeaf} = \text{NewLeaf} + \{ (p, M(p), IRS(p)) \} \);

while \( \text{NewLeaf} \neq \emptyset \) do

6) \( \text{OldLeaf} = \{ (p, M(p), IRS(p)) \} \); \( \text{NewLeaf} = \{ (p, M(p), IRS(p)) \} \);

if \( p \in P_f \) then

7) If \( p \notin \text{FS} \), \( FRS = \text{FS} + \{ (p, M(p)) \} \);

else update the marking of \( p \) in \( FRS \);

while \( \text{NC} = IRS(p) \); \( \text{while} \ NC \neq \emptyset \) do

9) select a place \( p \notin NC \); \( NC = NC - \{ p \} \);

if \( \exists p \in AP_{xy} \), \( \exists \{ p \} = 0 \) then

10) continue;

else compute the marking \( M(p) \) according to Eqn.
3;

11) create a node \( (p, M(p), IRS(p)) \), and
generate an arc directed from
\( (p, M(p), IRS(p)) \) to \( (p, M(p), IRS(p)) \);

12) add \( (p, M(p), IRS(p)) \) to NewLeaf;

13) \( \text{Condition} \) \( A(p, w, n, t) \) \( \{ \text{FRS} \} \) do

14) \( \text{Condition} \) \( A(p, w, n, t) \) \( \text{is the condition assigned to} \ A(p, w) \).

15) fire \( t \) and compute marking \( M_t \) based on Eqn. 4;

16) return \( M_t \).

Fig. 9. (a) Reasoning tree of the net in Fig. 7. (b) Marking set of the fuzzy
places in Fig. 7.

4) \( A \) is the set of arcs such that \( A \subset (P \times T) \cup (T \times P) \).

5) \( w \) is the set of weights assigned to each arc \( \forall p \in t \), \( w(t_d, p) \in R^+ \), and for each transition \( t \in T \), satisfying
\( \forall p \in t \), \( t', p \in t^* \), \( w(t, p) \in R^+ \) and \( w(t^*, p') \in R^+ \),
where \( R^+ \) is a set of nonnegative real numbers. Other
weights are equal to 1.
For convenience, a learning net is denoted as $PN^C = (P^E \cup P^E, T^F \cup T^E, A, w)$. In the definition, the place $p_i$ is an interface to a classical PN, and the transition $t_d$ is used to initialize the learning process; the places in $P_e$ denote the initialization of the input variables, i.e., the five environment factors in this paper; transitions in $T_e$ monitor the values of the input variables and transmit to their output places, i.e., the places in $P_o^r$ in the F-net.

Definition 6: A Branch Net is a tuple $PN^B = (P, T, F)$, where:
1) $P$ is a finite set of places, for each $p \in P$, $|p| \geq 1$, and $|p^*| \geq 1$.
2) $T$ is a finite set of transitions, for each $t \in T$, $|t^*| = 1$, and $|t^*| = 1$.
3) $A$ is the set of arcs such that $A \subset (P \times T) \cup (T \times P)$.

Definition 7: A Learning Branch Net $PN^{BE}$ is a combination of a Learning Net $PN^C = (P^E \cup P^E, T^F \cup T^E, A, w)$ and a Branch Net $PN^B = (P, T, A)$, where there exists a transition $t \in T$ such that $t^* = \{p_0\} \subset P^E$, and $p \in P_d \subset P^F$, there exists a transition $t \in T$ such that $p^* = \{t\}$.

Definition 8 (see [17]): A Closed Process Net is a tuple $PN^C = (P \cup \{p_s, p_e\}, T \cup \{t_e\}, A)$, where:
1) $P$ is a finite set of internal places, for each $p \in P$, $|p| \geq 1$, and $|p^*| \geq 1$.
2) $T$ is a finite set of internal transitions, for each $t \in T$, $|t^*| = 1$, and $|t^*| = 1$.
3) $p_s$ is a start place, $|p_s| = 1$, and $|p^*_s| = 1$.
4) $p_e$ is an end place, $|p_e| = 1$, and $|p^*_e| = 1$.
5) $t_e = \{p_s\}, t_e = \{p_e\}$.
6) $A$ is the set of arcs such that $A \subset (P \times T) \cup (T \times P)$.

Definition 9: A Closed Learning Process Net $PN^{CL}$ is a combination of a Closed Process Net $PN^C = (P \cup \{p_s, p_e\}, T \cup \{t_e\}, A)$ and a Learning Net $PN^C = (P^E \cup P^E, T^F \cup T^E, A, w)$, where there exists a transition $t \in T$ such that $p_0 = \{t_1\}$, and $p \in P_d \subset P^F$, there exists a transition $t \in T$ such that $p^* = \{t\}$.

A concurrent or distributed system usually consists of a set of processes. Each closed process net can be used to model a process, and a closed learning process net can model a process with learning ability. Thus, such systems can be modeled by a set of closed (learning) process nets, resulting in good scalability of the model.

It is easy to see that start place $p_s$ of a closed (learning) process net leads one or many (learning) branch nets that end at $p_e$ exemplified in Fig. 10.

A process needs to communicate with others for message processing. In this paper, the communications among processes are through rendezvous communication, as shown in Fig. 11. Next, we give a PN model for this communication mechanism between two processes.

Definition 10 (see [17]): A Rendezvous Communication Mechanism is a tuple $CM^Q = \langle t_1, t_2, t_3, t_4, p_{t12}, p_{t14}, p_{send}, p_{ack}, p_{mcp} \rangle$, where:
1) $p_{t12}, p_{t14}, p_{send},$ and $p_{ack}$ are places such that $|p_{t12}| = \{t_2\} = 1$, $|p_{t14}| = \{t_4\} = 1$, $|p_{send}| = \{t_{send}\} = 1$, and $|p_{ack}| = \{t_{ack}\} = 1$; and $p_{mcp}$ is a place such that $|p_{mcp}| = |p_{mcp}|$.
2) $t_1$ is a transition such that $|t_1| = |t_1^*| = 2$, $p_{t12} = \{t_1\}$, $p_{send} = \{t_1\}$, and $t_1 \in \{p_{mcp}\}$.
3) $t_2$ is a transition such that $|t_2| = |t_2^*| = 2$, $p_{t12} = \{t_2\}$, and $t_2 \in \{p_{mcp}\}$.
4) $t_3$ is a transition such that $|t_3| = 2$, $|t_3^*| = 1$, $p_{t14} = \{t_3\}$, and $p_{send} = \{t_3\}$.
5) $t_4$ is a transition such that $|t_4| = 1$, $|t_4^*| = 2$, $p_{t14} = \{t_4\}$, and $p_{ack} = \{t_4\}$.

$p_{t12}$ and $p_{t14}$ are called intermediate places, $p_{send}$ and $p_{ack}$ are called Buffer places, and $p_{mcp}$ is called a Control place; $t_1$ and $t_4$ are called output communication transition, and $t_2$ and $t_3$ are called input communication transition.

Definition 11 (see [17]): Let $PN^C_1$ and $PN^C_2$ be two closed process nets. If there exists a rendezvous communication mechanism $CM^Q = \{t_1, t_2, t_3, t_4, p_{t12}, p_{t14}, p_{send}, p_{ack}, p_{mcp}\}$ such that $PN^C_1$ and $CM^Q$ have common parts $\{t_1, t_2, p_{t12}\}$, and $PN^C_2$ and $CM^Q$ have common parts $\{t_3, t_4, p_{t14}\}$, then $PN^C_1$ is Communicating with $PN^C_2$ through communication mechanism $CM^Q$, or simply Communicating.

The above two definitions are also true if $PN^C_1$ and $PN^C_2$ are replaced by two closed learning process nets $PN^{CLC}_1$ and $PN^{CLC}_2$. Let $A, B,$ and $C$ be three closed (learning) process nets. If $A$ and $B$ are communicating with $C$ at the same time, then their rendezvous communication mechanisms must have the same $\{p_{send}, p_{ack}, p_{mcp}\}$, i.e., only $A$ or $B$ can send a message to $C$, but not both at the same time. For example, Fig. 12 shows that two closed process nets $A$ and $B$ mutually send messages to the same closed process net $C$.

Definition 12: An I-PN is a PN, consisting of a set of closed process nets $\{PN^C_o, o = 1, 2, \ldots, g\}$ and closed learning
process nets \{PN^i_C, i = 1, 2, \ldots, k\} and a set of Rendezvous communication mechanisms \{CM_j^Q, j = 1, 2, \ldots, l\} such that:

1) Each closed (learning) process net \(PN^i_C\) (\(PN^i_C\)) is communicating through at least one communication mechanism.

2) Each communication mechanism \(CM_j^Q\) has been used for one and only one pair of process nets.

An I-PN is denoted by \(PN^T\).

In an I-PN, places are classified as idle, denoted as \(P^D\), buffer, denoted as \(P^B = \{p_{end}, p_{ack}\}\), control, denoted as \(P^C = \{p_{mpc}\}\), and activity for the rest, denoted as \(P^A\), while transitions are classified into activity ones, \(T^A\), input communication ones, \(T^I = \{t|\exists p \in P^B, s.t. p \in ^t\}\), and output communication ones, \(T^O = \{t|\exists p \in P^B, s.t. p \in ^t\}\). Thus, in an I-PN, we can say that a closed process net has input and output communication transitions.

Definition 13: A marked I-PN is a tuple \((PN^T, M_0)\), where:

1) \(PN^T\) is an I-PN.

2) \(M_0 : P \rightarrow \{0, 1\}\) is the initial marking: \(\forall p \in P^D, M_0(p) = 1\); \(\forall p \in P^A, M_0(p) = 0\); \(\forall p \in P^B, M_0(p) = 0\); and \(\forall p \in P^C, M_0(p) = 1\).

From the I-PN construction, the I-PN can be used to model a system consisting of a set of processes that communicate through Rendezvous Communication Mechanisms.

Next, we show how to model a manufacturing system by the I-PN. Let \(x_1\) denote the quantity of the order, \(x_2\) the price of the raw material, and \(x_3\) the cost of transport for products; \(y_1\) is the required amount of raw material and \(y_2\) is the cost of transport; \(z_1\) is the benefit from high production capacity, \(z_2\) is the benefit from short transport distance, \(z_3\) is the benefit from the low price of raw material, and \(z_2\) is the benefit from short transport distance. Next, suppose the following fuzzy rules are abstracted from the requirements and domain experts.

\(R_7:\) **IF** \(x_1\) is high and \(x_2\) is middle and \(x_3\) is high, **THEN** \(z_2\) is crucial \((CF = \varphi_7)\).

\(R_8:\) **IF** \(x_1\) is middle and \(x_2\) is high and \(x_3\) is high, **THEN** \(z_2\) is greater than \(z_2\) \((CF = \varphi_8)\).

\(R_9:\) **IF** \(x_1\) is middle and \(x_2\) is middle and \(x_3\) is middle, **THEN** \(z_2\) is crucial \((CF = \varphi_9)\).

\(R_{10}:\) **IF** \(x_1\) is middle and \(x_2\) is high and \(x_3\) is low, **THEN** \(z_2\) is crucial \((CF = \varphi_{10})\).

\(R_{11}:\) **IF** \(x_1\) is middle and \(x_2\) is middle and \(x_3\) is low, **THEN** \(z_2\) is crucial \((CF = \varphi_{11})\).

\(R_{12}:\) **IF** \(x_1\) is middle and \(x_2\) is middle and \(x_3\) is middle, **THEN** \(z_2\) is crucial \((CF = \varphi_{12})\).

\(R_{13}:\) **IF** \(x_1\) is middle and \(x_2\) is middle and \(x_3\) is middle, **THEN** \(z_2\) is greater than \(z_2\) \((CF = \varphi_{13})\).

\(R_{14}:\) **IF** \(x_1\) is middle and \(x_2\) is low and \(x_3\) is low, **THEN** \(z_2\) is greater than \(z_2\) \((CF = \varphi_{14})\).

\(R_{15}:\) **IF** \(x_1\) is middle and \(x_2\) is low and \(x_3\) is middle, **THEN** \(z_2\) is greater than \(z_2\) \((CF = \varphi_{15})\).

\(R_{16}:\) **IF** \(x_1\) is middle and \(x_2\) is low and \(x_3\) is high, **THEN** \(z_2\) is greater than \(z_2\) \((CF = \varphi_{16})\).

\(R_{17}:\) **IF** \(x_1\) is low, **THEN** \(z_2\) is similar with \(z_2\) \((CF = \varphi_{17})\).

\(R_{18}:\) **IF** \(z_2\) is crucial, **THEN** the selection of \(B_1\) is good \((CF = \varphi_{18})\).

\(R_{19}:\) **IF** \(z_2\) is greater than \(z_2\), **THEN** the selection of \(B_1\) is good \((CF = \varphi_{19})\).

\(R_{20}:\) **IF** \(z_2\) is greater than \(z_2\), **THEN** the selection of \(B_2\) is good \((CF = \varphi_{20})\).

\(R_{21}:\) **IF** \(z_2\) is similar with \(z_2\), **THEN** the selection of \(B_1\) is good \((CF = \varphi_{21})\).

\(R_{22}:\) **IF** \(z_2\) is similar with \(z_2\), **THEN** the selection of \(B_2\) is good \((CF = \varphi_{22})\).

\(R_{23}:\) **IF** \(z_2\) is similar with \(z_2\), **THEN** the selection of \(B_3\) is good \((CF = \varphi_{23})\).

\(R_{24}:\) **IF** \(z_2\) is greater than \(z_2\), **THEN** the selection of \(B_3\) is good \((CF = \varphi_{24})\).

\(R_{25}:\) **IF** \(z_2\) is greater than \(z_2\), **THEN** the selection of \(B_3\) is good \((CF = \varphi_{25})\).

\(R_{26}:\) **IF** \(z_2\) is crucial, **THEN** the selection of \(B_3\) is good \((CF = \varphi_{26})\).

\(R_{27}:\) **IF** \(y_1\) is high, **THEN** \(z_1\) is great \((CF = \varphi_{27})\).

\(R_{28}:\) **IF** \(y_1\) is middle and \(y_2\) is high, **THEN** \(z_1\) is great \((CF = \varphi_{29})\).

\(R_{29}:\) **IF** \(y_1\) is middle and \(y_2\) is middle, **THEN** \(z_1\) and \(z_2\) are similar \((CF = \varphi_{29})\).

\(R_{30}:\) **IF** \(y_1\) is middle and \(y_2\) is low, **THEN** \(z_2\) is great \((CF = \varphi_{28})\).

\(R_{31}:\) **IF** \(y_1\) is low, **THEN** \(z_2\) is greater \((CF = \varphi_{31})\).

\(R_{32}:\) **IF** \(z_1\) is greater, **THEN** the selection of \(S_1\) is good \((CF = \varphi_{32})\).

\(R_{33}:\) **IF** \(z_1\) and \(z_2\) are similar, **THEN** the selection of \(S_1\) is good \((CF = \varphi_{33})\).

\(R_{34}:\) **IF** \(z_1\) and \(z_2\) are similar, **THEN** the selection of \(S_2\) is good \((CF = \varphi_{34})\).

\(R_{35}:\) **IF** \(z_2\) is greater, **THEN** the selection of \(S_2\) is good \((CF = \varphi_{35})\).

In the rules, there are two kinds of parameters: fuzzy numbers and certainty factors. The fuzzy membership functions of the
DING et al.: MODELING SELF-ADAPTIVE SOFTWARE SYSTEMS BY FUZZY RULES AND PETRI NETS 975

input propositions are given in Fig. 13, and certainty factors of fuzzy rules are shown in Table I.

The I-PN of the manufacturing system is shown in Fig. 14, where the detailed F-nets of $l_1$ f-transitions and $l_2$ f-transitions are shown in Figs. 15 and 16, respectively. In the model, $p_{27}$-$p_{29}$ are used to model the facts $x_1$, $x_2, x_3$, respectively, while $p_{32}$ and $p_{33}$ are used to model the facts $y_1$ and $y_2$, respectively. $A_{1}$-$A_{3}$, $i = 1, 2, \ldots, 5$, represent the membership functions of the fuzzy variables “high,” “middle,” and “low,” respectively. $p_{1}^1$, $p_{1}^2$, $p_{1}^3$ denote that $B_1$, $B_2$, $B_3$ are selected as the production factory, respectively, and $p_{2}^1$, $p_{2}^2$ denote that $S_1$ and $S_2$ are selected as the material suppliers.

In order to give the semantics of the I-PN, we classify the nodes into three kinds: discrete, continuous, and fuzzy. Hence, the places can be divided into discrete places $P_D$, continuous places $P_C$, and fuzzy places $P_F$, while the transitions can be divided into discrete transitions ($T_D$), continuous transitions ($T_C$), fuzzy transitions ($T_\alpha$, and $T_\beta$).

### B. Semantics of $PN^F$

A transition can fire if it is enabled at marking $M$. It has the following enabling conditions.

1. A $D$-transition $t$ can be enabled by $M$, if $\forall p \in \bullet t \cap P_D, M_i(p) \geq w(p, t)$ and $\forall p \in \bullet t \cap P_F, M_i(p) > 0$.
2. A $C$-transition $t$ can be enabled by $M$, if $\forall p \in \bullet t, M_i(p) > 0$.
3. An $\alpha$-transition $t$ can be enabled by $M$, if $\forall p \in \bullet t, M_i(p) > 0$.
4. An $\beta$-transition $t$ can be enabled by $M$, if $\forall p \in \bullet t, M_i(p) > 0$.

The enabling degree of transition $t$ at marking $M$, denoted by $q(t, M)$ or $q$, can be described as follows [14].

1. If $t \in T_D$, its enabling degree is an integral number such that
   $$q \leq \min_{p_i, p, e, t, p_D} \frac{M(p_i)}{w(p, t)} < q + 1.$$ $w$ is removed and a making $\alpha$ or $t$ are two discrete places, and $p$.

2. If $t \in T_C$, the enabling degree is a real number such that
   $$q = \min_{p_i, p, e, t, p_C} \frac{M(p_i)}{w(p, t)}.$$ $w$ is removed and a making

Based on the enabling degree, the semantics of $PN^F$ can be defined based on the following basic net structures. According to the definition of $PN^F$, we have the following structures, of which the first three are the common structures studied in [36].

1. $D$-places are connected by $D$-transitions: In this structure, the input places and output places are both $D$-places. The structure is the same as that in classical PNs. For example, in Fig. 17, $p_1$ and $p_2$ are two discrete places, and $t$ is a $D$-transition. $M(p_1) = 1$ and $w(p_1, t) = 1$. After $t$ fires, $M(p_1) = 0$ and $M(p_2) = 1$.

2. $D$-places and $C$-places are connected by $D$-transitions: In this structure, the input and output places are $D$-places and $C$-places, respectively. It is used to translate an integer number of tokens to a real number of markings, and it can model the initialization of the environment in self-adaptive systems. When a $D$-transition fires, an integer number of tokens are removed from its input places and a real number of marking is added to its output places. The number of removed tokens is the product of its enabling degree and weight of its input arc, and the number of added marking is the product of its enabling degree and weight of its output arc. For example, in Fig. 18, $p_1$ is a $D$-place and $p_2$ is a $C$-place. $t$ is a $D$-transition. Assume that $p_1$ has a token and the weight $w(t, p_1) = 1$, $t$ can be enabled. When $t$ fires, the token in $p_1$ is removed and a making $\alpha$ is added to $p_2$ since $w(t, p_2) = 1.5$ and $w(p_2, t) = 0$.

3. $C$-places are connected by a $C$-transition: In this case, the input places and output places are $C$-places. The structure is used to model the inputs from an environment. When the transition fires, a marking with the value equal to the product of firing quantity and the weight of the input arc is removed from each input place; meanwhile, a marking with the value equal to the product of firing quantity and the weight of the output arc is added to each output place. For example, in Fig. 19, $p_1$-$p_3$ are $C$-places, and $t$ is a $C$-transition. The initial marking is $(1, 5, 1, 2, 0)$. According to the enabling condition, $t$ can be enabled with the enabling degree 1.2. Thus, its firing generates a new marking $(0.3, 0, 1.44)$.

4. Fuzzy places are connected with $\alpha$-transitions or $\beta$-transitions or $D$-transitions: Indeed, these structures build an F-net, and the semantics has been described in Section III-C.

The above semantics can be summarized as follows. Suppose that $t$ is enabled and fires at $M$. We have the following.
Fig. 14. $PN^T$ for the manufacturing system.

Fig. 15. Net of $t_i$ $f$-transitions.
1) If \( t \in T_D \) with the enabling degree \( q \), then
\[
M_{i+1}(p) = \begin{cases} 
    M_i(p) - q w(p, t), & p \in t \cap P_D \\
    0, & p \in t \cap P_F \\
    M_i(p) + q w(t, p), & p \in t^* \cap P_D \\
    M_i(p), & \text{otherwise.}
\end{cases}
\]
(5)

2) If \( t \in T_C \) with enabling degree \( q \), then
\[
M_{i+1}(p) = \begin{cases} 
    M_i(p) - q w(p, t), & p \in t^* \cap P_C \\
    0, & p \in t \cap P_C \\
    M_i(p) + q w(t, p), & p \in t^* \cap P_C \\
    M_i(p), & \text{otherwise.}
\end{cases}
\]
(6)

3) If \( t \in T_\alpha \) associated with membership function \( \tilde{A} \), then
\[
M_{i+1}(p) = \begin{cases} 
    \tilde{A}(M_i(*t)), & p \in t^* \\
    M_i(p), & \text{otherwise.}
\end{cases}
\]
(7)

4) If \( t \in T_\beta \) associated with certainty factor \( \varphi \), then
\[
M_{i+1}(p) = \begin{cases} 
    \max\{M_i(p), \varphi \min\{M_i(*t)\}\}, & p \in t^* \\
    M_i(p), & \text{otherwise.}
\end{cases}
\]
(8)

Remark 1: Note that in an I-PN, the current environment is modeled by \( C \)-transitions. \( D \)-transitions can be regarded as bridges between discrete and continuous values. \( \alpha \)-transitions with the marking evolutionary equation (7), which uses the membership function to translate the real number of the environment to a membership degree of the fuzzy value, can be regarded as bridges between continuous and fuzzy values. The defuzzification is modeled by a set of conflicting \( D \)-transitions and control places.

### C. Building Evolution Graph

Like building a reachability graph (RG) for a traditional PN [47], we can also build a behavioral EG for an I-PN. The algorithm is shown in Algorithm 2.

The main idea to compute the reachable states is to identify the enabled transitions enabled by the markings in \( \text{new\_states} \), and then fire them to result in new states and update the set \( \text{new\_states} \). Then, repeat this process until there is no new state generated. Since there exists \( C \)-transitions, the number of reachable states is unfortunately infinity.
However, we have the following observation: If we consider the simplified form of the net and overlap the nodes by firing the same sequence of transitions regardless of the marking at the nodes, we obtain a new graph, namely EG, which has the same structure as that of RG of a classical PN. To be more precisely, we have the following.

**Proposition 1:** In a given I-PN, replacing $C$-transitions and $F$-transitions by $D$-transitions and setting the weights of the arcs connected with $C$-transitions to 1 leads to a classical PN. Then, the EG of I-PN has the same structure as that of RGs of PN, and the difference is that in EG a node may have an infinity number of markings, while in RG, a node has only one marking.

The following is a sketch of the proof.

First, consider the classical algorithm for producing RG [47]. It has the following steps. Step 1: identify all the enabled transitions. Step 2: fire all of them resulting in "new" markings. Step 3: For each marking, if it is an old or dead marking, go to the next marking. Otherwise, identify all the enabled transitions and fire them to generate all the "new" markings, and then go to next marking until the markings at the same level are exhausted. Then, start the next level of markings.

Compared with our algorithm, we find that the process to generate RG and EG are similar except that in Algorithm 2, there are no stops since the number of reachable states is infinity. But once we overlap the nodes, both of them are determined by the firing sequences of transitions only. Since we only replace the $C$-transitions and $f$-transitions rather than the structure of the PN, they have the same number of transitions and same enabling and firing conditions; thus, they have the same firing sequences of transitions. The difference is that repeating of a sequence of firing transitions produces the same marking in the classical PN, while that in our net produces new markings due to the function of $C$-transitions.

The EG of the manufacturing system shown in Fig. 14 is shown in Fig. 20. It has a same structure as RG except that every node contains an infinity number of markings.

**V. ADAPTATION ANALYSIS OF $PN^I$**

In this section, we analyze the adaptivity of $PN^I$, i.e., $PN^I$ can respond to the changes in the environment. To guarantee the adaptivity of $PN^I$, we need to prove two properties. The first one is the correctness, including completeness and determinacy. This means it can actually describe the behavior of the system, and there are no conflicts in the output of the model. The second one is the adaptivity, i.e., the model can switch to a designed behavior autonomously when the environment changes. Next, we give the detailed analysis.

**Definition 14:** A model is complete if for any point in the input domain of a system, there exists a designed output based on the model. A model is deterministic if for any point in the input domain of a system, there exists unique output based on the model.

To guarantee the completeness and determinacy of a model for a system, the system itself should be well defined. Thus, we need the following assumption.

(H) Every decision is made under an environment, and every environment can only activate one decision.

**Definition 15 (E-Vector):** For $PN^I$, let $P_e = \{p_{e1}, p_{e2}, \ldots, p_{er}\}$ be the set of the input places of all $F$-transitions. Then, $e = (p_{e1}, p_{e2}, \ldots, p_{er})$ is called an $E$-vector of $PN^I$.

**Definition 16 (E-Space):** The marking range of an $E$-vector is called the $E$-space of $PN^I$, denoted as $E$.

An $E$-space is an input domain of a software system and each $E$-vector is a concrete input. Next, we define a binary relation in the $E$-space $E$. First, we need some notations. Let $T^f = \{T_{i1}, \ldots, T_{ik}\}$ be the set of all $f$-transitions in an I-PN, where $l_1$–$l_k$ are labels and $T_{il}$ is the set of $f$-transitions whose label is $l_i$. $T_{il}(e)$ denotes the final marking of the discrete places in the $l_i$ F-net according to Algorithm 1 with the current $E$-vector $e$. 

![Fig. 20. EG of the manufacturing system.](image-url)
Definition 17 (A binary relation in E): $R^E$ is a binary relation in E if it satisfies: $\forall e_1, e_2 \in E$, $(e_1, e_2) \in R^E \iff \forall T^E_i \in T^E, T^E_i(e_1) = T^E_i(e_2)$.

Lemma 1: Let $R^E$ be the binary relation defined in Definition 17, $R^E$ is an equivalence relation.

Proof: We need to prove that $R^E$ satisfies the following properties.

1) Reflexivity: It is clear that $\forall T^E_i \in T^E, e \in E$, $T^E_i(e) = T^E_i(e)$, so $(e, e) \in R^E$.

2) Symmetry: Suppose $(e_1, e_2) \in R^E$; then, we have $\forall T^E_i \in T^E, T^E_i(e_1) = T^E_i(e_2)$. Clearly, $T^E_i(e_2) = T^E_i(e_1)$ is also satisfied. Thus, $(e_2, e_1) \in R^E$.

3) Transitivity: Let $(e_1, e_2) \in R^E$ and $(e_2, e_3) \in R^E$; then, $\forall T^E_i \in T^E E$, $T^E_i(e_1) = T^E_i(e_2)$ and $T^E_i(e_2) = T^E_i(e_3)$, so $T^E_i(e_1) = T^E_i(e_3)$, which means $(e_1, e_3) \in R^E$.

According to Lemma 1, we can partition the $E$-space $E$ based on the equivalence relation $R^E$. Clearly, this partitioning is an equivalence partitioning, and the set of all the equivalence classes is denoted as $\{ [E_i] \}$. Then, we have $E = \bigcup [E_i]$ satisfying $\forall i \neq j, E_i \cap E_j = \emptyset$. Based on the definition of the binary relation, all $E$-vectors in the same class have the same decision, and we say they are the same in terms of equivalence classes.

Definition 18 (Behavior): Let $p$ be a place of $PN^E$ and $m$ is a marking of place $p$. $(p, m)$ represents a local state of $PN^E$. An execution of $PN^E$ is a sequence of firing transitions such that local states $\{ (p_j, m_{i_j}), j = 1, 2, \ldots, l \}$ change in order. If in the ordered local states, some or all $p_j$ can be repeatedly visited, then set the ordered local states is called a behavior of $PN^E$.

Lemma 2: A behavior is a subgraph of the EG of $PN^E$ containing one or more cycles.

Proof: From the definition of a behavior, it contains a set of local states, and all the local states in the subset of $PN^E$ can be reached, and thus, all the markings of the places in the subset are reachable markings. Hence, these markings must be in the nodes in the EG. All such nodes form a subgraph. Since the local states in the subset can be repeatedly reached, then the markings of those places are repeatedly visited. It means the nodes in this subgraph should be visited repeatedly. Hence, the subgraph at least has a cycle. If the subgraph of $PN^E$ contains some concurrent transitions, then there are branches in EG because of the different firing orders of these transitions, but the branches eventually end at a same node. Again since the local states can be repeatedly reached, at least one order of the corresponding transitions can be executed. If only one is repeatedly executed, then the subgraph contains only one cycle; if more orders can be repeatedly executed, then this subgraph contains more cycles.

Remark 2: Different cycles in a subgraph are generated by firing the same set of transitions, but with different orders.

Such a subgraph is named b-graph. If a b-graph has at most one cycle, then we say the behavior is a simple behavior; otherwise, we call it a complex behavior. For example, in Fig. 20, there are six b-graphs as shown in Fig. 21 representing six complex behaviors.

Definition 19: Two nodes of different subgraphs are said to be the same if the markings in the two nodes are the same in terms of the equivalence class. Two subgraphs $G_i$ and $G_j$ are said to be the same if they have the same nodes.

Based on this definition, a node in the EG may represent different nodes in different subgraphs.

Lemma 3:

1) One equivalence class of $E$-space corresponds to one and only one subgraph in the EG.

2) Different equivalence classes of the $E$-space correspond to different subgraphs in the EG.

3) Let $G$ be the EG; then, $G = \bigcup G_i$, $G_i \cap G_j = \emptyset$.

Proof:

1) Let $E_1$ be an equivalence class of $E$-space. For any $e_i \in E_1$, it must be in part of some reachable marking; thus, there is a node $q$ corresponds to $e_i$. From Lemma 2, $q$ is in some subgraph, say $G_1$. We say that there is only one subgraph for it. If not, there is another subgraph, say $G_2$, that contains $q$. So, both subgraphs have an overlap. Since $G_1$ and $G_2$ are different, there are choices from some node, say $\omega$, in both subgraphs. First, we say that $\omega$ must be the node where the decision of an $F$-net is made. Otherwise, the choices are generated because of the different firing orders of the concurrent transitions, and thus, the nodes following $\omega$ would be in $G_1$, so $G_1 \subset G_i$ and this is a contradiction. Then, since $\omega$ has two different choices, thus the environment $e_i$ will cause at least different decisions, which is a contradiction to assumption (H).

2) Let $E_i$ and $E_j$ be two different equivalence classes. From (1), let $G_i$ and $G_j$ be their corresponding subgraphs in the EG. If $G_i = G_j$, $\forall e_i \in E_i$ and $\forall e_j \in E_j$, we have $T^E_i(e_i) = T^E_j(e_j)$. Based on Definition 17, $e_i \in E_i$ and $e_j \in E_j$, so $E_i \cap E_j \neq \emptyset$. But $E_i$ and $E_j$ are two different equivalence class, so $E_i \cap E_j = \emptyset$. Hence, $G_i \neq G_j$. What is more, $G_i \cap G_j = \emptyset$. This is because the markings in each node of $G_i (G_j)$ contain the environmental value, which is in $E_i (E_j)$. Thus, the markings in the nodes of $G_i$ and $G_j$ are different in terms of equivalence class, and thus, they have no same nodes.

3) Let $E$ be the $E$-space. Then, based on the equivalence partitioning, we have $E = \bigcup E_i$ and $E_i \cap E_j = \emptyset$. From 1) and 2), $G_i$ is the corresponding subgraph of $E_i$. Thus, $\bigcup G_i \subset G$. We say that all these $G_i$ s would cover $G$. In fact, if not, assume that some node in $G$ cannot be covered. This means that the marking in this node cannot be reached in any environments. It is a contradiction to assumption (H). Moreover, $G_i \cap G_j = \emptyset$ can be gotten from (2) directly.

Theorem 1: $PN^E$ is complete and deterministic.

Proof: It can be implied from Lemma 3. Let $E$ be the $E$-space of $PN^E$. First, for any environment vector, there exists an equivalence class $E_i$ such that $e \in E_i$. Based on the first statement of Lemma 3, there exists a subgraph $G_i$ corresponding to $E_i$. According to Lemma 2, there exists a behavior that can respond to $e$. Thus, $PN^E$ is complete. Second, based on the property of an equivalence relation, each $e$ can only belong
Fig. 21. Six subgraphs of EG of manufacturing system. (a) Subgraph $G_1$ of the EG. (b) Subgraph $G_2$ of the EG. (c) Subgraph $G_3$ of the EG. (d) Subgraph $G_4$ of the EG. (e) Subgraph $G_5$ of the EG. (f) Subgraph $G_6$ of the EG.

The result of this theorem means that for each environment input, the system has one and only one behavior corresponding to it. For example, let $E$ be the $E$-space of $PN^T$ in Fig. 20, and $E = E_1 \cup E_2 \cup E_3 \cup E_4 \cup E_5 \cup E_6$, $E_i \cap E_j = \emptyset$. For any environment vector $e$, if it takes values in some equivalent class, say $E_1$, then there is one and only one behavior in Fig. 21 to match the values of $e$.

**Theorem 2:** For $PN^T$, let $E_i$ and $E_j$ be two equivalence classes, and $G_i$ and $G_j$ be their behaviors, respectively. If $E_i$ changes to $E_j$, then $G_i$ changes to $G_j$.

**Proof:** From Lemma 3, $G_i$ and $G_j$ are the unique subgraphs corresponding to $E_i$ and $E_j$, respectively. If $E_i$ changes to $E_j$, then the reachable marking has to be in some node in $G_j$. Once the model is under environment $E_j$, the reachable markings form the nodes in $G_j$. Thus, the behavior of the model changes to $G_j$ from $G_i$ autonomously.

The result of this theorem means that once the environment changes, the system behavior changes. For example, let $E$ be the $E$-space of $PN^T$ in Fig. 20, and $E = E_1 \cup E_2 \cup E_3 \cup E_4 \cup E_5 \cup E_6$, $E_i \cap E_j = \emptyset$. For any environment vector $e$, if it takes values in $E_1$, and its corresponding behavior is $G_1$, then if $e$ changes its domain from $E_1$ to $E_2$, its corresponding behavior changes from $G_1$ to a different behavior, say $G_2$ in Fig. 21.

Thus, based on Theorems 1 and 2, we can immediately obtain the following corollary.

**Corollary 1:** The model $PN^T$ can be adaptive to the changes in the environment. Thus, it can be used to describe adaptive systems.
VI. ILLUSTRATING EXAMPLE

A. Generating Fuzzy Rules for the Manufacturing System

In practice, there are two ways to generate fuzzy rules. The first one is from requirement specifications and the other is from historical data.

When we analyze the requirement specifications, we can extract some rules. For example, in the manufacturing system, a part of the requirement specifications can be described as follows:

“If the production needs a large amount of raw material, then it is better to select S1, since in this case, the price of raw material has great influence on the cost of production.”

From this specification, we can extract the following rules.

1) IF the demand quantity of raw material is big, THEN the effect on the price of raw material is great.
2) IF the effect on the price of raw material is great, THEN the selection of S1 is good.

On the other hand, we can also generate fuzzy rules from sample data that can be collected during the real-time execution. There are many methods to generate fuzzy rules from data [13], [25], [26], [42], [43], [45]. In this paper, the fuzzy rules

\[ R_1 : \text{IF } x \text{ is } A \text{ THEN } y \text{ is } B(\text{CF} = \varphi) \]

in Form 1 are generated by using a modified method proposed in [25], [26], and [45].

First, we give a brief description of the concept of confidence of a linguistic associate rule, which is discussed in detail in [26]. Consider a fuzzy rule:

\[ \text{IF } x_1 \text{ is } A_{q1} \text{ and } \ldots \text{ and } x_n \text{ is } A_{qn} \text{ THEN } y \text{ is } B_q, \]

where \( A_{qi} \) (\( i = 1, 2, \ldots, n \)) is an antecedent fuzzy number and \( B_q \) is a consequence fuzzy number, \( x = (x_1, \ldots, x_n) \) is the \( n \)-dimensional input vector and \( y \) is an output variable.

This rule can be viewed as an association rule \( A_q \Rightarrow B_q \), where \( A_q = (A_{q1}, \ldots, A_{qn}) \). Besides, assume that in the sample data \( D \), there are total \( m \) input-output pairs \((x_p, y_p)\) \((p = 1, \ldots, m)\), where \( x_p = (x_{p1}, \ldots, x_{pn}) \) is the \( n \)-dimensional input vector and \( y_p \) is the output value corresponding to \( x_p \). Let \( A_q(x_p) = A_{q1}(x_{p1}) \times \cdots \times A_{qn}(x_{pn}) \). Then, the confidence of rule \( A_q \Rightarrow B_q \) is defined as

\[ Cf(A_q \Rightarrow B_q) = \frac{\sum_{p=1}^{m} A_q(x_p)B_q(y_p)}{\sum_{p=1}^{m} A_q(x_p)}. \]

Next, we give the method to generate fuzzy rules based on the sample data. This method is a combination of Wang’s method [45] and Ishibuchi’s [26]. It can also be viewed as an extension of the latter [26]. Note that [26] just considers the case of pattern classification, and thus, the membership of the consequence is only 0 or 1.

Step 1: Dividing the input and output spaces into fuzzy regions: Assume that the domain of discourse of \( x_i \) is \([x_{iL}, x_{iU}]\) \((i = 1, 2, \ldots, n)\) and that of \( y \) is \([y_L, y_U]\). Divide each domain into \( N \) regions (note that one region may overlap with another; different input variables may have different regions), and each region is assigned with a fuzzy number. Assume that the domain of discourse of \( x_i \) is divided into \( N_i \) regions and the domain of discourse of \( y \) is divided into \( N_{n+1} \) regions. Thus, \( x_i \) has \( N_i \) fuzzy numbers and \( y \) has \( N_{n+1} \) fuzzy numbers.

Step 2: Generating all possible antecedent propositions (i.e., the IF part) of the rule base: Based on the partition of the input spaces, we obtain all the possible combinations of the antecedent fuzzy numbers of different input variable, and the total number is \( \prod_{i=1}^{N_i} \). Among all combinations, consider a combination \( A_q \) that satisfies \( \sum_{p=1}^{m} A_q(x_p) = 0 \). That means that \( \forall x_p \in D, A_q(x_p) = \prod_{i=1}^{n} A_{qi}(x_{pi}) = 0 \). Hence, at least \( \exists x_p, s.t. A_{qi}(x_{pi}) = 0 \). Therefore, this combination is useless based on the sample data and should be removed. Suppose that the set of all possible combinations is denoted as \( C_q \). Then, \( C_q \) can be reduced to: \( C_q = C_q - \{A_q \sum_{p=1}^{m} A_q(x_p) = 0 \} \).

Step 3: Determining candidate fuzzy rules: In Step 2, we have obtained possible IF parts, and next, we should examine each combination and determine the possible consequence proposition. Thus, the confidence defined in (9) can be used. Suppose that \( B = \{B_1, \ldots, B_{N_{n+1}} \} \) is the set of fuzzy numbers of the output variable \( y \). For each combination \( A_q \in C_q \), the resulting fuzzy number is specified as

\[ Cf(A_q \Rightarrow B_q) = \max_{h=1, \ldots, N_{n+1}} Cf(A_q \Rightarrow B_h). \]

If \( Cf(A_q \Rightarrow B_q) > 0 \), then a fuzzy rule is generated and the confidence can be directly used as the certainty factor of the rule, i.e.,

IF \( x_1 \) is \( A_{q1} \) and \( \ldots \) and \( x_n \) is \( A_{qn} \), THEN \( y \) is \( B_q \) \((CF = \text{Conf}(A_q \Rightarrow B_q))\).

Remark 3: In practice, we only consider the rules whose antecedent propositions are combined by word “and,” i.e., the conditions of IF part must be met simultaneously in order for the result of THEN part to occur. If the antecedent propositions are combined by word “or,” then the rule can be decomposed into two rules without “or.” For example, the rule “IF \( x_1 \) is \( A_1 \) or \( x_2 \) is \( A_2 \) THEN \( y \) is \( B \) \((CF = \varphi)\)” can be decomposed into two rules: “IF \( x_1 \) is \( A_1 \) THEN \( y \) is \( B \) \((CF = \varphi)\)” and “IF \( x_2 \) is \( A_2 \) THEN \( y \) is \( B \) \((CF = \varphi)\)”.

Remark 4: As described before, different rules may generate different F-nets, so do different I-PNs. In this paper, we only focus on the situation that the structure of the net is determined before implementation and cannot be changed at runtime. Thus, the adaption of fuzzy rules we consider in this paper is the adaption of the membership functions of the fuzzy numbers and the certainty factors of the rules. This adaption can help one refine a model.

B. Simulation

In this section, we give the simulation results of the manufacturing system described in Sections II and IV.

First, we give an overview of the decision made in the environment space according to the fuzzy rules. Take the F-net shown in Fig. 16, which is obtained from rules \( R_{277} \sim R_{355} \), as an example.
Assume that the current environment consists of the environment factors \((0.2, 0.1, 0.3, 0.5, 0.1)\) and the selection is \(B_2, S_2\); the environmental factors \((0.9, 0.1, 0.5, 0.5, 0.1)\) and the selection is \(B_3, S_2\).

1) **Runtime environment**: The markings of the continuous places \(p_{27}, p_{28}, p_{29}, p_{32}\), and \(p_{33}\) consist of the environmental vector, i.e., \(e = (m_{27}, m_{28}, m_{29}, m_{32}, m_{33})\). The continuous transitions \(t_1^1-t_2^2\) get the current value of \(e\).

2) **Fuzzy reasoning**: Assume that the current environmental vector changes from \(e_0\) to \(e_1 = (0.2, 0.1, 0.3, 0.5, 0.1)\). The reasoning results are \(M(p_{16}') > M(p_{17}') > M(p_{15}')\) and \(M(p_{28}') > M(p_{27}')\). \(t_1^1-t_2^2\) are selected to be fired and \(B_2\) and \(S_2\) are selected for the factory to produce and the supplier to provide material. We see that the computing is done locally, while the system reaches a global decision. The system is running with behavior \(B_1\) \(b\)-graph \(G_1\) in Fig. 21(d).

3) **Adaptation**: Now, assume that the environment changes from \(e_1\) to \(e_2 = (0.9, 0.1, 0.5, 0.5, 0.1)\). After detecting the change of environment, the system preforms the reasoning again and obtains the outputs as \(M(p_{15}') = 0.171, M(p_{16}') = 0.576, M(p_{17}') = 0.608, M(p_{27}') = 0.162, M(p_{28}') = 0.612\). So \(B_3\) and \(S_2\) are selected. This means that the system is switched to run under \(b\)-graph \(G_6\), as shown in Fig. 21(f). Thus, the system changes from behavior \(B_1\) \(b\)-graph \(G_1\) to another behavior \(B_2\), i.e., \(b\)-graph \(G_6\). Fig. 25 displays the simulation of adaptation. At the beginning, the selection position is at point \((B_1, S_1)\). When time goes to 20, the selection position is adapted to point \((B_2, S_2)\), then to \((B_3, S_2)\).

Remark 5: Our adaption is based on the behavior switching. However, this switching is through fuzzy rules with certainty factors, which is different from most of the existing deterministic mode switch methods. Hence, in our self-adaptation, the learning algorithm is used to make adaptation decisions at
The complexity of our adaptation is the complexity of the online learning algorithm, i.e., the reasoning of the fuzzy rules, which is $O(N)$, where $N$ is the number of fuzzy propositions in the fuzzy rule base.

VII. RELATED WORK

After enhancing PN with fuzziness, or the applications of fuzzy logic [12], [18], [37], the resulting fuzzy PN can be used as a formal language to model systems with fuzzy elements, or used as a computing machine because of its semantics. For examples, Chen et al. [10] proposed a fuzzy PN to model fuzzy rules. In the model, each fuzzy proposition is modeled as a place, the execution of a rule is modeled as the firing of a transition, and the marking in a place indicates the activation degree of the corresponding propositions. Bugarin and Barro [2] proposed an improved PN formulation, which additionally attains the level of expressive power, to represent the knowledge base of fuzzy production systems with rule chaining. This model facilitates the design of algorithms for more efficient and flexible executions of the knowledge bases. Wang et al. [44] modeled fuzzy rules by logical PNs, based on which the concurrent reasoning algorithms we applied. Gao et al. [20] presented a fuzzy PN to model a fuzzy production rule-based system. It is shown that this net can represent and reason about rules containing negative literals and can improve the efficiency of fuzzy reasoning.

Cao and Chen [5] proposed a kind of fuzzy PNs for computing with linguistic words. It is a concurrency model of computing with words. Its feature is that the transitions are labeled by some special words modeled by fuzzy sets.

Shen et al. [40] developed supervised and unsupervised learning algorithms for the machine learning Petri net (MLPN) models in order to make them fully trainable and to remedy the difficulties encountered by artificial neural networks (ANNs). Compared with ANN, MLPN has some advantages. Konar [29] presented a scheme for supervised learning on a fuzzy PN that provided semantic justification of the hidden layers of feed-forward neural networks and was capable of approximate reasoning and learning from noisy training instances. This model can be used in fuzzy pattern recognition.

Pal and Konar [35] presented a new model for cognitive reasoning using fuzzy neural nets. Its analysis yields guaranteed stability of the temporal fuzzy inferences. The model has a wide range of applications, especially in intelligent decision support systems. Konar [27], Konar and Chakraborty [28] presented two distinct models of unsupervised learning on a special type of cognitive maps, realized with fuzzy PNs. In the first model, a Hebbian-type learning algorithm with a natural decay in weights is employed to study the dynamic behavior of the algorithm. It may be used in complex decision making and learning such as automated car driving in an accident-prone environment. The second model employs a minor variation of Hebbian learning with no passive decay in weights.

Different from the existing work, the proposed net by us can be used to model adaptive software systems. We model the fuzzy rules and their inference by a new fuzzy PN, named F-net, that can perform the reasoning with respect to the crisp values of fuzzy parameters, rather than the membership degrees. F-nets have the interfaces to the classical PNs and thus can then be integrated directly into a PN to construct a new integrated PN, named I-PN, to model self-adaptive systems. Moreover, we give a rigorous proof on its adaption ability.

VIII. CONCLUSION

A new type of PNs has been proposed in this paper to model self-adaptive software systems. It combines fuzzy rules with PNs. The rules can model runtime environmental changes, and their fuzzy reasoning can help make decisions for behavioral adaption to respond to such changes. Based on fuzzy rules, the complexity of the adaption algorithm is linear with the number of rules. The model language can be used to describe component-based software systems.

In the future, we plan to consider the model-checking methods and their implementation for the properties of an I-PN, such as safety, liveness, and rules’ completeness, consistence, and correctness. More importantly, we need to check the adaption property of the model. In order to do, we need to map I-PN to the input of some existing model-checking tool, or develop new algorithms. We also need to study how to generate test cases from an I-PN model, such that they can be used to test the adaptive software system.

REFERENCES

[1] M. Amin and D. Shebl, “Reasoning dynamic fuzzy systems based on
adaptive fuzzy higher order Petri nets,” Inf. Sci., vol. 286, pp. 161–172,
Dec. 2014.


intelligence for tackling uncertainty in self-adaptive systems: The case of
dynamic decision networks,” in Proc. 2nd Workshop Realizing Artif. Intell.

loops,” in Software Engineering for Self-Adaptive Systems (Lecture Notes
New York, NY, USA: Springer-Verlag, 2009, pp. 48–70.