# Simulation Modeling of Fuzzy Systems for Model Contiuity

**Embedded software engineers have often relied on the use of modeling and simulation techniques in order to make software development tasks manageable. However, they often embed external components in simulation models, which may cause model continuity problems. That is, such models and their simulation artifacts could be abandoned during the design phase since the models may not work properly on target platforms or be transformed into other forms of models or languages. In this paper, we first propose an approach to formally model Takagi-Sugeno-Kang fuzzy systems without the use of any external components. In order to keep the model continuity, the formal simulation model for a TSK fuzzy system is comprised of three types of reusable sub-models involving primitive operations. Thus, the model can be executed even on a limited computational platform. We next describe our implementation of simulation models for fuzzy systems including two other inference models, and of a visual modeling tool for the fuzzy systems.**

**Keywords: modeling and simulation, model continuity, fuzzy logic, discrete event system specification, embedded systems**

## I. Introduction

*Modeling and simulation* (M&S) technologies have been widely used in industry to assist in system development [1]. M&S technologies let engineers experiment with 'virtual' systems, allowing them to explore changes and test dynamic conditions in risk-free environments [2]. One particular use of these technologies is in the development of embedded software systems since they usually have time constraints [3]. The use of M&S-based software design and implementation combined with hardware-in-the-loop simulation techniques results in a faster product development cycle, lower development costs, and higher overall product quality [1]. Integration of M&S techniques with embedded software development must provide *model continuity* [4], a seamless development where the same simulation model is used, with minimal change, for both property analysis and real time execution [5]. This approach is one of the most beneficial and time saving applications of M&S [2].

The world of embedded control systems is experiencing a push into the realm of fuzzy logic due to its simplicity and effectiveness in solving control problems [6]. Fuzzy logic-based approaches allow building robust and smooth control systems starting from heuristic knowledge and qualitative models, considering imprecise, vague, and unreliable information, and can integrate symbolic reasoning and numeric processing in the same framework [7]. Also, fuzzy-based approaches are well suited for limited computational platforms (e.g., embedded platforms), as they are intrinsically modular and computationally simple [8]. Thus, even household machines are advertised as being intelligent with the help of built-in fuzzy logic [6].

When modelers build simulation models for embedded

fuzzy control systems, they typically embed external fuzzy components, such as a Fuzzy Logic Toolbox [9], in their models [10]-[11]. These models, however, may not be used throughout all of the design phases since M&S environments do not support the use of some external components [12]; as shown Fig. 1(a), simulation models may be executed even on the final target platforms for products [13]. That is, their continuity may not be kept through the design phase. Therefore, in order to keep the continuity of models, they need to be built without the use of external components. If not, the early models and simulation artifacts could be abandoned when the development tasks switch towards target platforms [14]. Keeping model continuity is an effective way to manage software complexity and maintain consistency throughout the design phase [4]. Also, the use of external components may make the transformation of simulation models difficult or impossible [12]. M&S-based approaches have gained popularity due to the fact that they enable not only interactive simulations but also a 'smooth' transformation, as shown in Fig. 1(b), from one model to other forms of models or languages [15]. For example, simulation models can be automatically transformed into hardware description languages [16]. However, it would be very difficult or impossible to properly transform external components into codes. Therefore, simulation models should not contain any external components to keep their continuity.

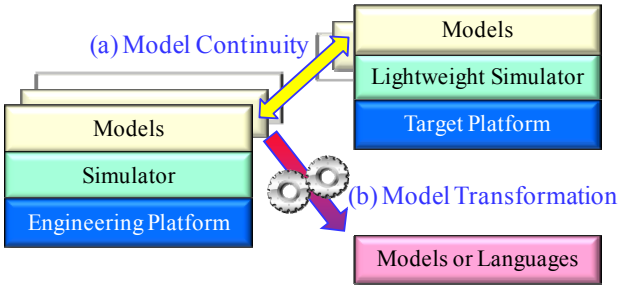


Fig. 1. Model continuity and transformation.

Several research efforts [12,17,18] have been made to build 'pure' simulation models, instead of the use of external components. In [17], Jamshidi *et al.* proposed an approach to model Mamdani fuzzy systems [18] together with several soft computing techniques, in order to support their M&S environment, V-Lab. The models are constructed in a hierarchical and modular fashion based on the parallel discrete event system specification (P-DEVS), which is a formal M&S methodology invented by Zeigler [19]. The modeling approach proposed by Lee and Kim [20] can reduce the complexity of the Mamdani P-DEVS models. The standard additive model (SAM) fuzzy systems [21] can be built with P-DEVS models based on the approach proposed in [12]. However, simulation modeling of Takagi-Sugeno-Kang (TSK) fuzzy systems [22], [23] has not been addressed yet. TSK fuzzy systems have a great advantage over other models in terms of representative power [24]. Also, the existing efforts do not provide user-friendly visual modeling environments. Thus, the building of simulation models for fuzzy systems could be a time-consuming process.

In this paper, we first propose an approach to build simulation models for TSK fuzzy systems based on P-DEVS. A P-DEVS model of a TSK fuzzy system is a coupled model consisting of three types of sub-models: an input membership function model, rule model, and defuzzification model. Since the models are all pure simulation models involving only addition and multiplication, they could be executed even on embedded platforms. Consequently, their continuity can be maintained. Compared to the existing approaches for the modeling of fuzzy systems, the proposed approach can model a TSK fuzzy system with a smaller number of sub-models. Also, P-DEVS supports the hierarchical and modular model-composition, so that the sub-models can be easily reused to implement simulation models of new fuzzy systems.

We next introduce the implementation of simulation models for fuzzy systems, including Mamdani, SAM, and TSK, for our M&S environment. For Mamdani and SAM fuzzy systems, the sub-models were implemented based on [12], and [20]. We also developed a tool that facilitates the simulation modeling process through a user-friendly graphical user interface (GUI). Within the modeling tool, fuzzy systems can be built as P-DEVS models using the sub-models provided in the tool. The models of fuzzy systems, generated by the tool, can be directly used in our simulation environment.

The remainder of the paper is organized as follows: Section II briefly describes TSK fuzzy systems and P-DEVS. Section III introduces the proposed approach for simulation modeling of TSK systems in detail. Section IV discusses the implementation status of simulation models and our visual modeling tool. Finally, conclusions and future work are discussed in section V.

## II. Background

In this section, we briefly describe the backgrounds of TSK fuzzy systems and P-DEVS.

### 1. TSK fuzzy systems

TSK, which is an additive rule model, was introduced by Takagi and Sugeno [22]. Later, Sugeno and Kang also worked on the identification of this type of fuzzy model [23]. In general, a rule in a TSK model has the following form:

IF $x_1$ is $A_{i1}$ and $x_2$ is $A_{i2}$ and ⋯ and $x_k$ is $A_{ik}$
THEN $y = a_{i0} + a_{i1} \times x_1 + \cdots + a_{ik} \times x_k$,

where $x_1, x_2, \cdots, x_k$ are input parameters, $A_{i1}, A_{i2}, \cdots, A_{ik}$ are the membership functions of $i$-th rule, $a_{i0}, a_{i1}, \cdots, a_{ik}$ are real-valued parameters, and $y$ is the output parameter. The total output, $y$, of the model is given by (1), where $\alpha_i$ is the matching degree of the $i$-th rule.

$$y = \frac{\sum_{i=1}^{j} \alpha_i (a_{i0} + a_{i1} x_1 + \cdots + a_{ik} x_k)}{\sum_{i=1}^{j} \alpha_i}. \quad (1)$$

The great advantage of the TSK model is its representative power [24]: it can describe a highly nonlinear system using a small number of rules. Moreover, due to the explicit functional representation form, it is convenient to identify its parameters using learning algorithms.

### 2. Parallel DEVS (P-DEVS)

P-DEVS formalism [20] is a theoretically well-grounded means of expressing modularly discrete event simulation models. The basic (the atomic) formalism of a P-DEVS model is

$$M = <X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta>,$$

where

$X$ is the set of input events,
$Y$ is the set of output events,
$S$ is the set of sequential states,
$\delta_{ext}$: $Q \times X^b \rightarrow S$ is the external transition function,
where $X^b$ is a set of bags over the elements in $X$,
$\delta_{int}$: $S \rightarrow S$ is the internal transition function,
$\delta_{con}$: $Q \times X^b \rightarrow S$ is the confluent transition function, subject to $\delta_{con}(s, \emptyset) = \delta_{int}(s)$,
$\lambda$: $S \rightarrow Y^b$ is the output function, and
$ta$ is the time advanced function,
where
$Q = \{(s, e) \mid s \in S, 0 < e < ta(s)\}$, and
$e$ is the elapsed time since the last state transition.

A coupled (digraph) model is defined as follows:

$$DN = <X, Y, D, \{M_i\}, \{I_i\}, \{Z_{ij}\}>,$$

where,

$X$ is the set of input events,
Y is the set of output events,
$D$ is the set of component names,
for each $i$ in $D$,
$M_i$ is an atomic model,
$I_i$ is the set of influencees for $I$,
for each $j$ in $I_i$, and
$Z_{i,j}$ is the $i$-to-$j$ output translation function.

P-DEVS environments support hierarchical and modular model-building, where the term 'modular' describes a model that has recognized input and output ports through which all interactions with the external world are mediated. This property enables the hierarchical construction of models, so that complex models can be easily developed.

## III. P-DEVS Modeling of TSK Fuzzy Systems

In the proposed approach, a TSK fuzzy system containing $i$ input membership functions and $j$ rules, with $k$ inputs and a single output is represented as a P-DEVS coupled model with $k$ input ports and a single output port. The coupled model contains $i + j + 1$ P-DEVS atomic models: *$i$ input membership function models*, *$j$ rule models* and a single *defuzzification model*. Each input membership function computes a membership degree for every input. A rule model produces conclusions of the corresponding rule, based on the input values of the fuzzy system and membership degrees. The defuzzification model finally generates outputs of the fuzzy system using the collection of conclusions of the rules. Figure 2 shows the P-DEVS model of a fuzzy system containing four input membership functions and four rules with two inputs and a single output (i.e., $i = 4, j = 4, k = 2$).

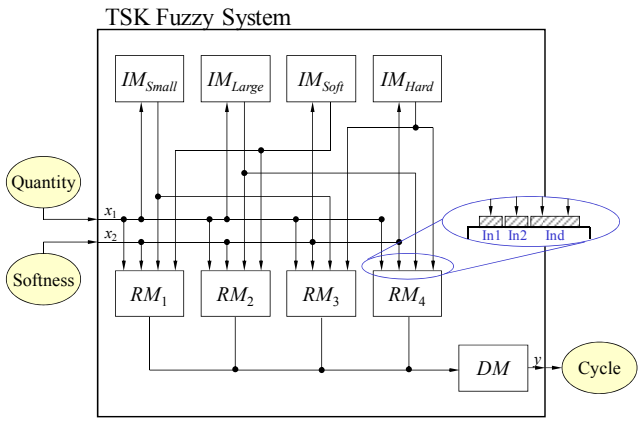


Fig. 2. A model structure for a TSK fuzzy system.

### 1. Input membership function models (IMs)

Each input membership function of the fuzzy system is represented as an input membership function model (IM) $M$ that is defined as

$$M = < X_M, Y_M, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta >,$$

where

$InPorts = \{\text{“In”}\}$,
$X_{\text{In}} = \Re$,
$X_M = \{(p, v) \mid p \in InPorts, v \in X_p\}$,
$OutPorts = \{\text{“Out”}\}$,
$Y_{\text{Out}} = [0, 1]$,
$Y_M = \{(p, v) \mid p \in OutPorts, v \in X_p\}$,
$S = \{\text{“passive”}, \text{“active”}\} \times \Re$,
$\delta_{ext}(\text{“passive”}, d, e, (\text{“In”}, x)) = (\text{“active”}, \mu(x))$,
$\delta_{int}(\text{“active”}, d) = (\text{“passive”}, d)$,
$\delta_{con}(s, ta(s), x) = \delta_{ext}(\delta_{int}(s), 0, x)$
$\lambda(\text{“active”}, d) = (\text{“Out”}, d)$,
$ta(phase, d) = \begin{cases} 0 & \text{if } phase = \text{“active”}; \\ \infty & \text{otherwise.} \end{cases}$

Every IM produces a matching degree of the corresponding membership function for each input value. It initially starts with its state = (“passive”, $d$), where $d$ is an arbitrary real value. When the IM for an input membership function $I$ receives a real value $x$ as an input, it transitions its state from (“passive”, $d$) to (“active”, $\mu_I(x)$). Immediately, the IM generates the membership degree of $x$ in $I$ (i.e., $\mu_I(x)$) as its output and transitions to a passive state. In short, the IM directly generates $\mu_I(x)$ as an output for an input value $x$, as shown in Fig. 3(a).

Once an IM for a membership function type (e.g., the triangular membership function type) has been implemented, it can be easily reused just by setting the parameters of the membership functions (e.g., $a$, $b$ and $c$ in an IM for the triangular membership function) in the same type. Even if any IM for a certain type (e.g., a sigmoid membership function type) does not exist, it can be implemented simply by redefining the external transition function, $\delta_{ext}$, of the existing one; the only difference between IMs for different membership function types is the definition of the external transition function. IMs are independent from fuzzy inference models; TSK, SAM, and Mamdani use the same IMs in the proposed approach [12], [20].

## 2. Rule models (RMs)

Each if-then rule of the fuzzy system corresponds to a rule model (RM). An RM is defined as

$$M = < X_M, Y_M, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta >,$$

where

$InPorts = \{\text{“In1”}, \text{“In2”}, \cdots, \text{“In}k\text{”}, \text{“Ind”}\}$,
$X_{\text{In1}} = X_{\text{In2}} = \cdots = X_{\text{In}k}, X\text{Ind} = \Re$,
$X_M = \{(p, v) \mid p \in InPorts, v \in X_p\}$,
$OutPorts = \{\text{“Out”}\}$,
$Y_{\text{Out}} = \Re^2$,
$Y_M = \{(p, v) \mid \text{p} \in OutPorts, \text{v} \in X_p\}$,
$S = \{\text{“passive”}, \text{“active”}\} \times \Re^{k+3}$,
$\delta_{ext}(\text{“passive”}, a_0, a_1, \cdots, a_k, b, c, e, ((\text{“In1”}, x_1), (\text{“In2”}, x_2), \cdots, (\text{“In}k\text{”}, x_k), (\text{“Ind”}, d_1), (\text{“Ind”}, d_2), \cdots, (\text{“Ind”}, d_k))$
$= (\text{“active”}, a_0, a_1, \cdots, a_k, a_0 + a_1 \times x_1 + \cdots + a_k \times x_k, \min(d_1, d_2, \cdots, d_k))$
or
$= (\text{“active”}, a_0, a_1, \cdots, a_k, a_0 + a_1 \times x_1 + \cdots + a_k \times x_k, d_1 \times d_2 \times \cdots \times d_k)$,
$\delta_{int}(\text{“active”}, a_0, a_1, \cdots, a_k, b, c) = (\text{“passive”}, a_0, a_1, \cdots, a_k, b, c)$,
$\delta_{con}(s, ta(s), x) = \delta_{ext}(\delta_{int}(s), 0, x)$
$\lambda(\text{“active”}, a_0, a_1, \cdots, a_k, b, c) = (\text{“Out”}, (b \times c, c))$,
$ta(\text{phase}, a_0, a_1, \cdots, a_k, b, c) = \begin{cases} 0 & \text{if } phase = \text{“active”}; \\ \infty & \text{otherwise.} \end{cases}$

Each RM produces a conclusion of the corresponding rule, based on input values: all input values of the fuzzy system and the membership degrees from the associated IMs. It has $k$ input ports, “In1”, “In2”, ···, “In$k$”, used to receive the $k$ input values, $x_1, x_2, \cdots, x_k$, of the fuzzy system; an additional input port, “Ind”,

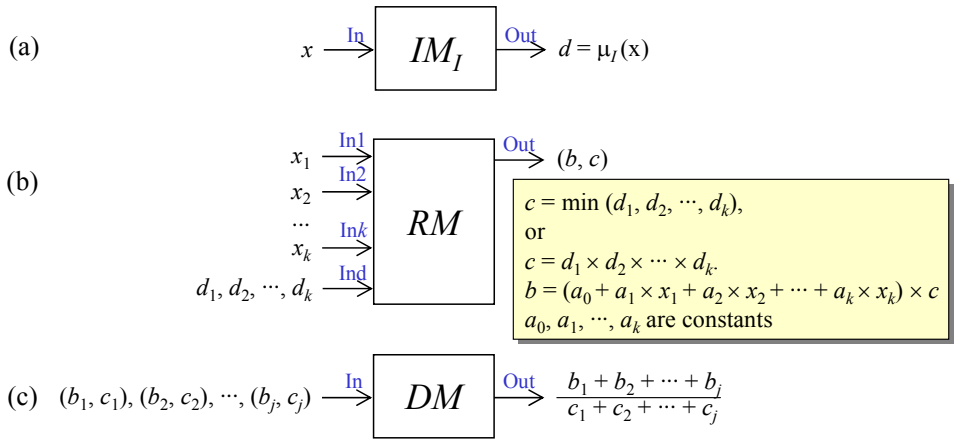


Fig. 3. Sub-models for TSK fuzzy systems.

used for $k$ membership degrees, $d_1$, $d_2$, ···, $d_k$, from the associated IMs; and a single output port, "Out," as shown in Figs. 2 and 3(b). The RM corresponding to rule $R$ starts with the initial state = ("passive", $a_0$, $a_1$, ···, $a_k$, $b$, $c$), where $a_0$, $a_1$, ···, $a_k$ are the constant values defined in the consequent part (i.e., then-part) of $R$, and $b$ and $c$ are arbitrary real values. When the RM receives $x_1$, $x_2$, ···, $x_k$, through the ports "In1", "In2", ···, "In$k$", respectively, and $d_1$, $d_2$, ···, $d_k$ through the port "Ind", it stores $b = a_0 + a_1 \times x_1 + \cdots + a_k \times x_k$ and $c = \min(d_1, d_2, \cdots, d_k)$. Typically, its rule matching degree, $c$, is computed using the 'min' operator, but the product operator can be also used. Finally, the RM outputs ($b \times c$, $c$) via the output port and transitions to a passive state.

Consider a RM corresponding to the following rule:

IF $x_1$ is $P$ AND $x_2$ is $Q$
THEN $y = a_0 + a_1 \times x_1 + a_2 \times x_2$.

When the RM receives $x_1$, $x_2$, $\mu_P(x_1)$ and $\mu_Q(x_2)$ as inputs, it promptly generates $((a_0 + a_1 \times x_1 + a_2 \times x_2) \times \min(\mu_P(x_1), \mu_Q(x_2)), \min(\mu_P(x_1), \mu_Q(x_2)))$, or $((a_0 + a_1 \times x_1 + a_2 \times x_2) \times \mu_P(x_1) \times \mu_Q(x_2), \mu_P(x_1) \times \mu_Q(x_2))$ as an output. The computation of a conclusion is done within the external transition function.

The RM can be reused repeatedly once it has been implemented; no further implementation for every RM is necessary. The reuse can be done simply through the creation of RM instances and assigning $k + 1$ parameters $a_0, a_1, \cdots, a_k$, of each instance.

## 3. Defuzzification model (DM)

A defuzzification model (DM) is an application-independent atomic-model that generates the outputs of a fuzzy system. It is formally defined as

$$M = < X_M, Y_M, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta >,$$

where

$InPorts$ = {"In"},
$X_{\text{In}} = \Re^2$
$X_M = \{(p, v) \mid p \in InPorts, v \in X_p\}$,
$OutPorts$ = {"Out"},
$Y_{\text{Out}} = \Re$,
$Y_M = \{(p, v) \mid p \in OutPorts, v \in X_p\}$,
$S$ = {"passive", "active"} ,
$\delta_{ext}(phase, y, e, (b_1, c_1), (b_2, c_2), \cdots, (b_j, c_j))$
$= (\text{"active"}, (b_1 + b_2 + \cdots + b_j) / (c_1 + c_2 + \cdots + c_j))$,
$\delta_{int}(\text{"active"}, y) = (\text{"passive"}, y)$,
$\lambda(\text{"active"}, y) = (\text{"Out"}, y)$,
$ta(phase, y) = 0$ if $phase$ = "active";
$\infty$ otherwise.

The DM produces a final conclusion of the fuzzy system based on the collection of conclusions of the rules. It starts with the passive state = ("passive", $y$), where $y$ is an arbitrary real value. When the DM receives ($b_1$, $c_1$), ($b_2$, $c_2$), ···, ($b_j$, $c_j$) from all RMs, it transitions its state to ("active", $(b_1 + b_2 + \cdots + b_j) / (c_1 + c_2 + \cdots + c_j)$). That is, the final conclusion is computed when the external transition occurs. Then, the DM outputs $y$ and transitions its state back to the passive state, as shown in Fig. 3(c).

Since any implementation of the DM is application-independent, it is reused for every TSK fuzzy system. Also, it is identical to the DM of SAM described in [12], so that simulation models for TSK and SAM use the same DM.

## 4. Model couplings

In the proposed approach, a TSK fuzzy system is represented as a P-DEVS coupled model consisting of atomic models (sub-models): $j$ IMs, $k$ RMs, and a DM. Each of them is coupled with other atomic models within the coupled model or the coupled model, based on fuzzy if-then rules. The coupled model has $i$ input ports and a single output port. Each of the input ports is connected with the associated input ports (e.g., input port "In1" for input $x_1$, "In2" for $x_2$, ···) of all RMs. The port is also coupled with the input ports of the associated IMs. Consider the following fuzzy if-then rules of a TSK fuzzy system that receives quantity $x_1$ and softness $x_2$:

IF $x_1$ is Small and $x_2$ is Soft
THEN $y = 1 + x_1 + x_2$,
IF $x_1$ is Large and $x_2$ is Soft
THEN $y = 1 + 2 \cdot x_1 + 2 \cdot x_2$,
IF $x_1$ is Small and $x_2$ is Hard
THEN $y = 1 + x_1 + 2 \cdot x_2$,
IF $x_1$ is Large and $x_2$ is Hard
THEN $y = 1 + 2 \cdot x_1 + 4 \cdot x_2$.

In the above example, the port that receives $x_1$ (quantity value) would be connected with the input ports of $IM_{Small}$ and $IM_{Large}$. Note that each IM has a single input port "In". The output port "Out" of each IM is coupled with input port "Ind" of each associated RMs. In the example, "Out" of $IM_{Small}$ would be connected with "Ind" of $RM_1$ and $RM_3$, which have a linguistic variable 'Small' in the if-part. The output port of every RM is coupled with the input port of the DM. And the output port of the DM is connected with that of the coupled model.

## 5. Fuzzy inference example

Figure 4 shows a simulation model of a toy washing

machine controller based on the TSK inference model. When the coupled model (corresponding to the fuzzy controller) receives two normalized values $x_1$ (*Quantity*) and $x_2$ (*Softness*), they would be delivered to the associated IMs and all RMs according to the couplings of the model. Each IM produces matching degrees for the inputs. As shown in Fig. 4(a), for $x_1$, $IM_{Small}$ and $IM_{Large}$ would generate $\mu_{Small}(x_1)$ and $\mu_{Large}(x_1)$, respectively. For $x_2$, $\mu_{Soft}(x_2)$ and $\mu_{Large}(x_1)$ are produced by $IM_{Soft}$ and $IM_{Hard}$, respectively. Every matching degree produced by an IM is delivered to the associated RMs, based on the couplings.

Conclusions of the rules for the inputs (including the matching degrees) are computed by RMs. When $RM_1$ corresponding to Rule 1, "IF $x_1$ is *Small* and $x_2$ is *Soft* THEN $y = a_{10} + a_{11}{\cdot}x_1 + a_{12}{\cdot}x_2$," receives $x_1$ via the port "In1," $x_2$ via the port "In2," and $\mu_{Small}(x_1)$ and $\mu_{Soft}(x_2)$ via the port "Ind," it then generates $(b_1, c_1) = ((a_{10} + a_{11}{\cdot}x_1 + a_{12}{\cdot}x_2) \times \mu_{Small}(x_1) \times \mu_{Soft}(x_2), \mu_{Small}(x_1) \times \mu_{Soft}(x_2))$ as the output (Fig. 4(b)). Also, $(b_2, c_2) = ((a_{20} + a_{21}{\cdot}x_1 + a_{22}{\cdot}x_2) \times \mu_{Large}(x_1) \times \mu_{Soft}(x_2), \mu_{Large}(x_1) \times \mu_{Soft}(x_2))$, $(b_3, c_3) = ((a_{30} + a_{31}{\cdot}x_1 + a_{32}{\cdot}x_2) \times \mu_{Small}(x_1) \times \mu_{Hard}(x_2), \mu_{Small}(x_1) \times \mu_{Hard}(x_2))$, and $(b_4, c_4) = ((a_{40} + a_{41}{\cdot}x_1 + a_{42}{\cdot}x_2) \times \mu_{Large}(x_1) \times \mu_{Hard}(x_2), \mu_{Large}(x_1) \times \mu_{Hard}(x_2))$ are generated by $RM_2$, $RM_3$, and $RM_4$, respectively. Note that multiplication is used for the computation of rule matching degrees in this example. However, a 'min' operator can be also be used to compute the rule matching degrees. The conclusions are then forwarded to the DM.

As shown in Fig. 4(c), the DM collects the conclusions of the RMs and generates $y = (b_1 + b_2 + b_3 + b_4) / (c_1 + c_2 + c_3 + c_4)$ as the output, which is equal to the results given by (1). Finally, the coupled model outputs $y$, as the fuzzy system's product.

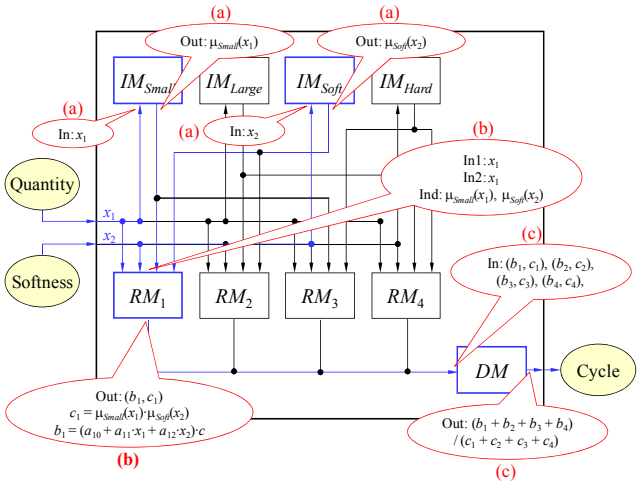


Fig. 4. Fuzzy inference through sub-models.

## 6. Overhead analysis

Table 1 shows an overhead analysis for the proposed approach and the three existing approaches [12], [17], [20]. Each fuzzy system consists of $i$ input membership functions, $j$ rules, and $l$ output membership functions (in SAM and Mamdani fuzzy systems), with $k$ inputs and a single output. In the proposed approach, a coupled model for a fuzzy system contains $i + j + 1$ atomic models, while a higher number of sub-models is required to build a coupled model for a fuzzy system in other approaches. The complex couplings among the sub-models in the proposed approach make the communications overhead (i.e., the number of messages generated for inter-communications) increase. However, the overhead of the proposed approach is still smaller than that of [17]. Moreover, TSK can describe a highly nonlinear system using a small number of rules [24]. That is, $j$ of TSK could be much smaller than that of SAM or Mamdani. While Mamdani fuzzy systems are widely used, they usually involve complex operations, such as the clipping and merging of membership functions and finding their centroids. Such complex operations might be too heavy on resource-constrained systems. Similar to [12], the proposed approach can model TSK fuzzy systems, which involve only primitive operations. Thus, the proposed approach will be suitable for the M&S-based engineering of embedded software systems.

Table 1. Overhead in four modeling approaches.

| Overhead | TSK (Proposed) | SAM [12] | Mamdani [17] | Mamdani [20] |
|---|---|---|---|---|
| Sub-Models | $i+j+1$ | $i+j+l+1$ | $j \times k+2j+2$ | $i+j+l+2$ |
| Communications | $i+2j \times k+$ $+j+1$ | $i+j \times k+$ $j+k+1$ | $2j \times k+2j$ $+2$ | $i+j \times k+j$ $+l+2$ |
| Inference | Multiply + Add | Multiply | Find a minimum + Clip or scale a MF | |
| Combining | Add | | Merge MFs | |
| Defuzzification | Multiply | | Find the mean of the maximum or the centroid of an area | |

## IV. Implementation Status

In this section, we will describe the implementation status of the simulation models and our visual modeling tool prototype.

### 1. Simulation model implementation

The P-DEVS models for TSK systems described in section III were implemented in C++ for our simulation environment, the DEVS Object C++ (DOC++) environment. The IM for the triangular membership function type was implemented as an `IMTraingle` class. The RM and DM were implemented as

TSKRM and SAMDM classes, respectively. As shown in Fig. 5, these classes inherit the atomic of DOC++ class, which corresponds to the basic model of P-DEVS. The essential member functions of atomic are ext_tn_fn (the implementation of $\delta_{ext}$), int_tn_fn ($\delta_{int}$), and output ($\lambda$). By overriding these functions, the behavior of a subclass is determined. The FuzzyMessage class is used for internal communications between the atomic models of fuzzy systems.

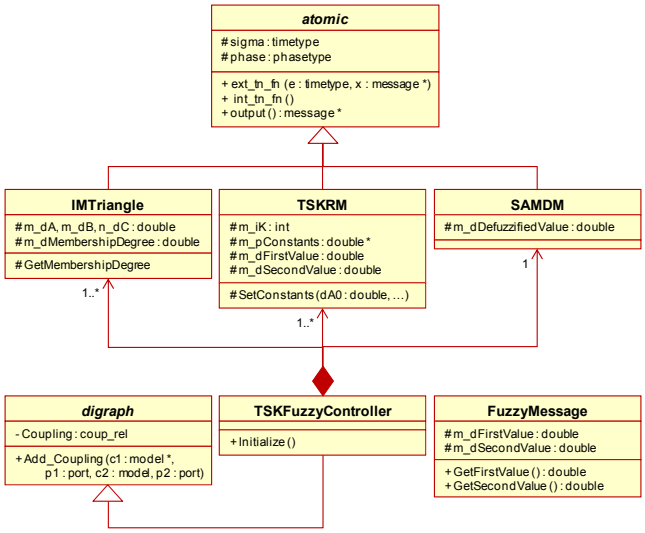


Fig. 5. Simplified UML diagrams of TSK simulation models.

```
void IMTriangle::ext_tn_fn (timetype e, message * x)
{
 // compute 'mu' value for input value (x)
 GetMembershipDegree
 (static_cast <FuzzyMessage *> (x->read (0)->Get_Val ())->GetFirstValue ());
 Hold_In ("active", 0);
}

void IMTriangle::int_tn_fn ()
{ Hold_In ("passive", INFINITY); }

message * IMTriangle::output ()
{
 // output 'mu' value
 message * y = new message ();
 y->Add_Content
 (new content (& m_oOutPort, this, new FuzzyMessage (m_dMembershipDegree)));
 return (y);
}
```

Fig. 6. Simplified implementation of IM behavior.

A triangular membership function is determined by three parameters: *a*, *b* and *c*. The internal state of an IM is comprised of 'phase' (e.g., "passive" or "active") and membership degree, described in section III. Therefore, IMTriangle class, an implementation of an IM, has four member variables: three for *a*, *b* and *c*, and one for membership degree. Note that phase is defined as a member variable of atomic since it is a common state of P-DEVS atomic models. Figure 6 shows a simplified definition of the essential functions for IMTriangle. When an instance of IMTriangle receives an input value, ext_tn_fn is executed. That is, an external transition occurs. During the transition, the membership degree for the input is computed by the member function GetMembershipDegree. The instance then transitions to an "active" state for 0 time units. That is, after 0 time units have elapsed (i.e., immediately), output and int_tn_fn are sequentially executed. The output produces the membership degree as an output. Finally, the instance transitions into a "passive" state by int_tn_fn. The state continues until the instance receives other inputs.

```
void SAMDM::ext_tn_fn (timetype e, message * x)
{
 int iCount = x->Get_Count ();
 double dNumerator = 0, dDenominator = 0;
 FuzzyMessage * pMessage;
 // collect rules' conclusions (b1, c1), (b2, c2), ...
 // compute final conclusion based on rules' conclusions
 while (iCount --)
 {
  pMessage = static_cast <FuzzyMessage *> (x->read (iCount)->Get_Val ());
  dNumerator += pMessage->GetFirstValue ();
  dDenominator += pMessage->GetSecondValue ();
 }
 m_dDefuzzifiedValue = dNumerator / dDenominator;
 Hold_In ("active", 0);
}

void SAMDM::int_tn_fn ()
{ Hold_In ("passive", INFINITY); }

message * SAMDM::output ()
{
 // output final conclusion (y)
 message * y = new message ();
 y->Add_Content
 (new content (& m_oOutPort, this, new FuzzyMessage (m_dDefuzzifiedValue)));
 return (y);
}
```

Fig. 7. Simplified implementation of DM's behavior.

The SAMDM class is the implementation of the DM. The DM has a single member variable to store a final conclusion of the fuzzy system. No additional member function is required. Figure 7 shows a simplified definition of the essential functions for SAMDM. Its instance can be created without any initial parameters and is used for all TSK and SAM fuzzy systems. When ext_tn_fn of an instance is executed (i.e., when the instance receives some inputs), the final conclusion ($\Sigma b_i$ / $\Sigma c_i$) of the fuzzy system is computed and stored in the member variable. Then, the instance transitions to an "active" state. An output is immediately called after the external transition and produces the conclusion as an output. Finally, int_tn_fn makes the instance transition into a "passive" state. The instance does nothing until the arrival of the next inputs.

The RM was implemented as a TSKRM class. The class has four membership variables: one to store the number of inputs (i.e., *k*), one as a pointer for the parameters of the rule (i.e., $a_0$, $a_1$, ⋯, $a_k$), and two, as the numerator and denominator, for the conclusion of the rule. A simplified implementation of RM is

shown in Fig. 8. In `ext_tn_fn`, the conclusion of a rule as a vector is computed from the collected inputs: the external input values of the fuzzy system and the membership degrees from the associated IMs. The conclusion is immediately produced as the output of the rule by `output`. The member variables are then initialized for the next input arrivals in `int_tn_fn`. Finally, the instance transitions into a passive state.

```
void TSKRM::ext_tn_fn(timetype e, message * x)
{
 int iCount = x -> Get_Count (), i;
 while (iCount --)
 {
  // collect 'mu' values and compute 'b' value (multiplication of 'mu' values)
  if (x->is_on_port (iCount, & m_oInPortD))
  {
   m_iDReceived ++;
   m_dSecondValue *=
   static_cast <FuzzyMessage *> (x->read (iCount)->Get_Val ())->GetFirstValue ();
  }
 // collect input values (x1, x2, ...)
  else
  {
   for (i = 0; i < m_iK; i ++)
   {
    if (x->is_on_port (iCount, m_pInPortX + i))
    {
     m_iXReceived ++;
     m_dFirstValue +=
     m_pConstants [++ i] *
     static_cast <FuzzyMessage *> (x->read (iCount)->Get_Val ())->GetFirstValue ();
     break;
    }
   }
  }
 }
 // transitions when received all
 if (m_iXReceived == m_iK && m_iDReceived == m_iK)
 { Hold_In ("active", 0); }
}

void TSKRM::int_tn_fn ()
{
 m_iXReceived = m_iDReceived = 0;
 m_dFirstValue = m_pConstants [0];
 m_dSecondValue = 1;
 Hold_In ("passive", INFINITY);
}

message * TSKRM::output ()
{
 // output rule's conclusion (b, c)
 message * y = new message ();
 y->Add_Content
 (new content (& m_oOutPort, this,
 new FuzzyMessage (m_dFirstValue * m_dSecondValue, m_dSecondValue)));
 return (y);
}
```

Fig.8. Implementation of RM's behavior.

The implementation for the frame of a TSK fuzzy system is `TSKFuzzyController`, which inherits the `digraph` class of DOC++. The `diagraph` corresponds to the P-DEVS coupled model. Thus, it has the coupling information as a member variable. An instance of `TSKFuzzyController` has *j* instances of IM implementation, *k* instance of `TSKRM`, and an instance of `SAMDM`. These sub-models are initialized in the membership function `Initialize` of `TSKFuzzyController` and logically connected based on the coupling information inherited from the superclass. P-DEVS models for other fuzzy inference models (i.e., Mamdani and SAM) and the IM for the trapezoid membership function were also implemented in DOC++. Consequently, `MAMCM`, `MAMOM`, `MAMUM`, `MAMDM`, `SAMRM`, `SAMOM`, and `IMTrapezoid` classes were written.

The simulation modeling of fuzzy systems can be done with ease using our modeling tool described in the next section. However, they can also be manually constructed without the use of the tool, thanks to the hierarchical and modular model-composition provided by the P-DEVS environments. A P-DEVS coupled model *M* for a new TSK fuzzy system containing *i* input membership functions and *j* rules with *k* inputs and a single output can be constructed using the following steps:

1. Make *i* instances of the IM implementation based on the membership function types of the inputs (e.g., `IMTriangle`) and set their parameters if necessary. Then put them into *M*.
2. Make *j* instances of `TSKRM`. Set the parameters of the instance and put them into *M*.
3. Put an instance of `SAMDM` into *M*.
4. Couple these models based on the if-then rules of the fuzzy system.

## 2. Modeling tool prototype implementation

In order to facilitate the modeling process, we have also developed a prototype of a visual modeling tool. Figure 9 shows a screenshot of the prototype. The atomic models for fuzzy systems and input/output stubs can be placed into coupled models through a GUI. The user can assign or modify their parameters in dialogs. The coupling process can be easily done within the tool. As shown in Fig. 10, the tool can also generate models (codes) for the DOC++ simulation environment from the modeled fuzzy systems. The generated models can be directly executed on the DOC++ environment. Within the visual modeling tool and DOC++ environment, the user can construct the target system models that employ fuzzy logic by coupling the system model with the fuzzy models generated by the tool.

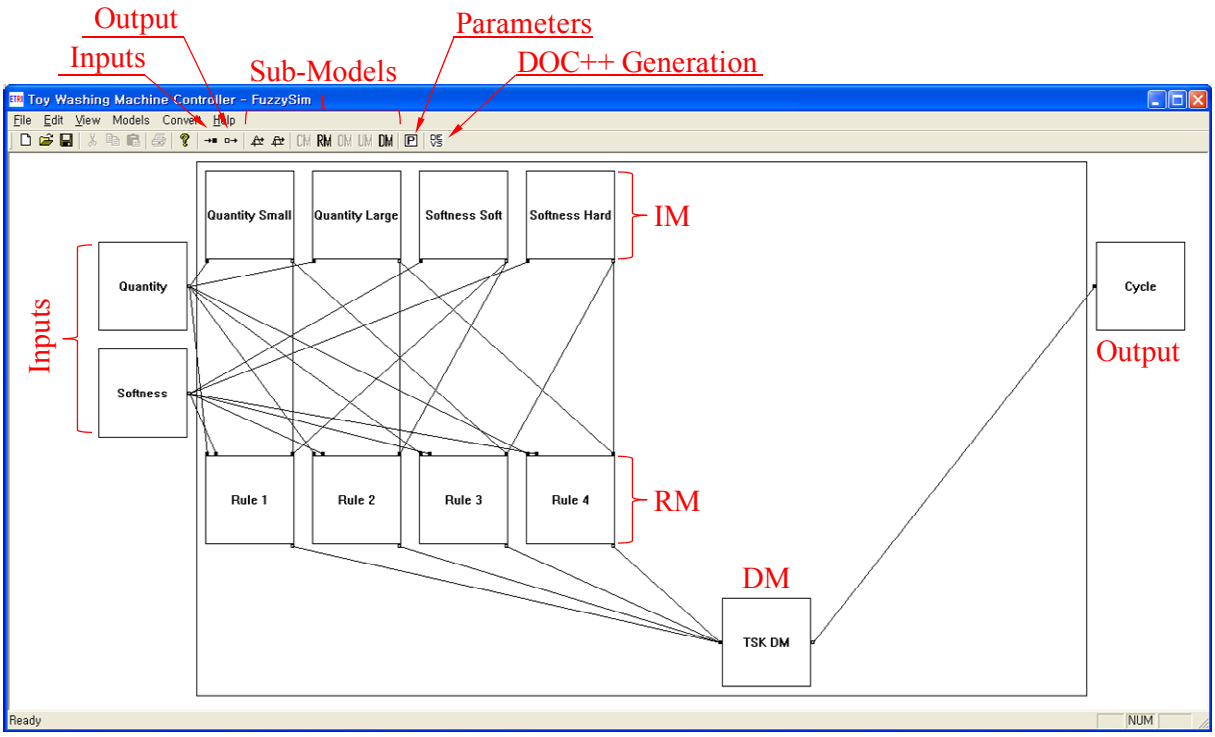


Fig. 9. Fuzzy system modeling tool prototype.

## V. Conclusions and Future Work

In this paper, we presented an approach for representing P-DEVS models of TSK fuzzy systems without the use of any external components. Exclusion of external components from simulation models would improve the continuity of the models so that the user can efficiently manage software complexity and maintain consistency throughout the design phase. A P-DEVS model of a fuzzy system is comprised of easy-to-reuse atomic models: IMs, RMs, and a DM. Since each atomic model involves primitive operations, such as addition or multiplication, the model works on target platforms and can be smoothly transformed into other forms of models or languages. A coupled model for a TSK fuzzy system requires a smaller number of sub-models, compared to that of a Mamdani or SAM fuzzy system. Thus, it will be more compatible with embedded platforms. We implemented P-DEVS models for fuzzy systems, including TSK, SAM, and Mamdani, for a DOC++ environment. To facilitate the modeling of fuzzy systems, a GUI-based modeling tool prototype was developed. The tool supports the generation of models that can be directly used in the environment. We will implement the models for other DEVS environments, such as eCD++ [3], [13]. We will also study the simulation modeling of other artificial intelligence techniques.

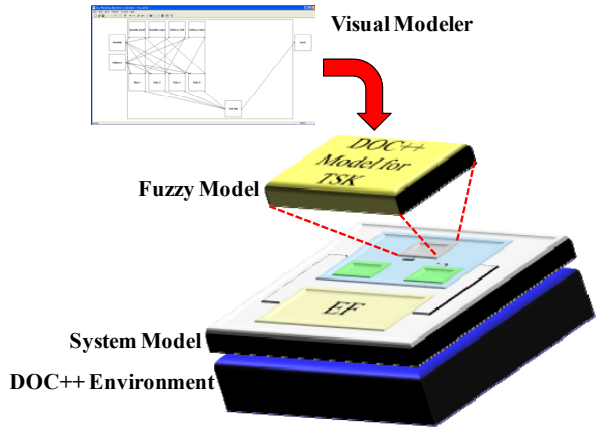


Fig. 10. Model generation for a DOC++ simulation environment.

## References

[1] X. Hu, *A Simulation-Based Software Development Methodology for Distributed Real-Time Systems*, Doctoral Dissertation, The University of Arizona, 2004.

[2] M. Moallemi and G. Wainer, "A Simplified Real-Time Embedded DEVS Approach Towards Embedded and Control Design," *Proc. WinterSim*, 2009.

[3] M. Moallemi, J.M. Gutierrez-Alcaraz, and G. Wainer, "ECD++ A DEVS Based Real-Time Simulator for Embedded Systems," *Proc. SpringSim*, 2008.

[4] X. Hu and B.P. Zeigler, "Model Continuity to Support Software Development for Distributed Robotic Systems: A Team Formation Example," Journal of Intelligent and Robotic Systems, vol. 39, no. 1, 2004, pp. 71-87.
[5] A. Furfaro and L. Nigro, "A Development Methodology for Embedded Systems Based on RT-DEVS," Innovations in Systems and Software Engineering: A NASA Journal, vol. 5, no. 2, 2009, pp. 117-127.
[6] F. Farkas and S. Halasz, "Embedded Fuzzy Controller for Industrial Applications," Acta Polytechnica Hungarica, vol. 3, no. 2, 2006, pp. 41-63.
[7] I. Baturone, F.J. Moreno-Velo, S. Sanchez-Solano, *et al*., "Embedded Fuzzy Controllers on Standard DSPs," *Proc. IEEE-ISIE*, 2006, 1197-1202.
[8] N. Zhang, D. Beetner, D.C. Wunsch II, *et al*., "An Embedded Real-Time Neuro-Fuzzy Controller for Mobile Robot Navigation," *Proc. FUZZ-IEEE,* 2005, pp. 319-323.
[9] The Fuzzy Logic Toolbox. http://www.mathworks.com/products/fuzzylogic/
[10] A.M. Garcia, B. Baumgartner, U. Schreiber, *et al*., "AutoMedic: Fuzzy Control Development Platform for a Mobile Heart-Lung Machine," IFMBE Proceedings, vol. 25, no. 7, 2009, pp. 685-688.
[11] M. Muruganandam and M. Madheswaran, "Modeling and Simulation of Modified Fuzzy Logic Controller for Various Types of DC Motor Drives," *Proc. INCACEC*, 2009, pp. 1-6.
[12] H.Y. Lee, S.M. Park, and T.H. Cho, "Simulation Modeling of SAM Fuzzy Logic Controllers," IEICE Transactions on Information and Systems.
[13] Y.H. Yu and G. Wainer, "eCD++: An Engine for Executing DEVS Models in Embedded Platforms," Proc. SCSC, 2007, 323-330.
[14] T. Pearce, "Simulation-Driven Architecture in the Engineering of Real-Time Embedded Systems," *Proc. RTSS-WIP*, 2003.
[15] H. Shang and G. Wainer, "Dynamic Structure DEVS: Improving the Real-Time Embedded Systems Simulation and Design," *Proc. ANSS*, 2008, pp. 271-278.
[16] Y.M. Lee, H.B. Kim, J.S. Hong, *et al*., "Translation from DEVS Models to Synthesizable VHDL Programs," *Proc. IEEE TENCON*, 1996, pp. 252-255.
[17] M. Jamshidi *et al*., "V-LAB – A Distributed Intelligent Discrete-Event Environment for Autonomous Agents Simulation," Intelligent Automation and Soft Computing, vol. 9, no. 3, 2003, pp. 181-214.
[18] E.H. Mamdani, "Application of Fuzzy Algorithms for Control of Simple Dynamic Plant," IEEE Proceedings, vol. 121, 1974.
[19] B.P. Zeigler, T.G. Kim, and H. Praehofer, *Theory of Modeling and Simulation*, 2nd Ed., Academic Press, 2000.
[20] H.Y. Lee and H.J. Kim, "Reducing the Complexity of DEVS-Based Mamdani Models for Enhancing Privacy," *Proc. ISIS*, 2009.
[21] B. Kosko, *Fuzzy Engineering*, Prentice Hall, 1997.
[22] T. Takagi and M. Sugeno, "Fuzzy Identification of Systems and Its Application to Modeling and Control," IEEE Transactions on Systems, Man, and Cybernetics, vol. 15, 1985.
[23] M. Sugeno and K.T. Kang, "Structure Identification of Fuzzy Model," Fuzzy Sets and Systems, vol. 28, 1988.
[24] J. Yen and R. Langari, *Fuzzy Logic: Intelligence, Control, and Information*, Prentice Hall, 1999.